# PicoMite

# Benutzerhandbuch

## MMBasic-BASIC-Interpreter
## Ver 6.02.01

Für den
Raspberry Pi Pico
Raspberry Pi Pico 2
Raspberry Pi Pico W
Raspberry Pi Pico 2 W
und
Module mit den Prozessoren RP2040 und RP2350

*Die Übersetzung wurde mithilfe von deepl.com erstellt.*
*Anschließend hat Matthias Hollatz den Text, insbesondere die Befehle und Programmteile, manuell überarbeitet bzw. wiederhergestellt.*
*Version 0 (29. März 2026)*

Revision 0
( 25. März 202629 März 2026 )

Aktualisierungen zu diesem Handbuch und weitere Details zu MMBasic findest du unter

http://geoffg.net/picomite.html

und http://mmbasic.com

# Über

Peter Mather (matherp im Back Shed Forum) leitete das Projekt, portierte MMBasic auf den Raspberry Pi Pico und schrieb die Treiber für dessen Hardwarefunktionen. Der MMBasic-Interpreter und dieses Handbuch wurden von Geoff Graham ( http://geoffg.net ) verfasst. Darüber hinaus haben viele andere das Projekt mit spezialisiertem Code, Tests und Vorschlägen unterstützt.

# Support

Fragen zum Support solltest du im Back Shed Forum ( http://www.thebackshed.com/forum/Microcontrollers ) stellen, wo es viele begeisterte MMBasic-Nutzer gibt, die dir gerne weiterhelfen. Auch die Entwickler der PicoMite-Firmware sind regelmäßig in diesem Forum unterwegs.

# Copyright und Danksagungen



# Dieses Handbuch

# Inhalt

# Einführung

Der PicoMite ist eine Betriebs-Firmware für alle Versionen des Raspberry Pi Pico einschließlich Pico, Pico 2, Pico W und Pico 2 W.

Es enthält einen BASIC-Interpreter (MMBasic), eine Microsoft BASIC-kompatible Implementierung der BASIC-Sprache mit Gleitkomma-, Ganzzahl- und Zeichenfolgenvariablen, Arrays, langen Variablennamen, einem integrierten Programmeditor und vielen weiteren Funktionen.

Es gibt Versionen der PicoMite-Firmware, die für eingebettete Steuerungsanwendungen (wie Heizungssteuerungen, Einbruchmeldeanlagen usw.) geeignet sind, sowie Versionen mit VGA/HDMI- und Tastaturunterstützung für den Aufbau eines eigenständigen Computers.

Mit MMBasic kannst du die I/O-Pins steuern und Kommunikationsprotokolle wie I²C oder SPI nutzen, um Daten von verschiedenen Sensoren abzurufen. Du kannst Daten auf kostengünstigen Farb-LCD-Displays anzeigen, Spannungen messen, digitale Eingänge erkennen und Ausgangspins ansteuern, um Lichter, Relais usw. einzuschalten. Und mit dem Raspberry Pi Pico W kannst du auf das Internet zugreifen und einen Webserver auf diesem kostengünstigen Modul aufbauen.

Die PicoMite-Firmware kann völlig kostenlos heruntergeladen und genutzt werden.

Zusammenfassend sind die Funktionen der PicoMite-Firmware:

- **Der BASIC-Interpreter ist voll ausgestattet** mit Gleitkommazahlen mit doppelter Genauigkeit, 64-Bit-Ganzzahlen und Zeichenfolgenvariablen, langen Variablennamen, mehrdimensionalen Arrays aus Gleitkommazahlen, Ganzzahlen oder Zeichenfolgen, umfangreicher Zeichenfolgenverarbeitung sowie benutzerdefinierten Strukturen, Unterprogrammen und Funktionen. Darüber hinaus ermöglicht MMBasic die Einbettung kompilierter C-Programme für Hochleistungsfunktionen. Der Schwerpunkt liegt auf Benutzerfreundlichkeit und einfacher Entwicklung.
- **Unterstützung für alle Raspberry Pi Pico-Ein-/Ausgangspins**. Diese können unabhängig voneinander als digitaler Ein- oder Ausgang, analoger Eingang, Frequenz- oder Periodenmessung sowie Zählung konfiguriert werden. Interrupts können verwendet werden, um zu melden, wenn ein Eingangspin seinen Zustand geändert hat. PWM-Ausgänge können genutzt werden, um verschiedene Töne zu erzeugen, Servos zu steuern oder computergesteuerte Spannungen zu generieren.
- **Unterstützung für LCD-/OLED-Displays** über parallele, SPI- und I²C-Schnittstellen, sodass das BASIC-Programm Text anzeigen und Linien, Kreise, Rechtecke usw. in bis zu 16 Millionen Farben zeichnen kann. Resistive Touch-Controller auf diesen Displays werden ebenfalls unterstützt, sodass sie als komplexe Eingabegeräte genutzt werden können.
- **Unterstützung für Internet- und Web-Protokolle mit dem Raspberry Pi Pico W, Pico 2 W und Pimoroni Pico Plus 2W**. Dazu gehört ein Webserver mit TCP und HTML sowie der Zugriff auf andere Ressourcen über TCP und HTTP. Das MQTT-Protokoll für die Verbindung über einen Message Broker. Das NTP-Protokoll zum Abrufen von Datum und Uhrzeit von einem Zeitserver. Telnet für den Fernzugriff auf die Konsole und TFTP für die schnelle Dateiübertragung.
- **Unterstützung für PS2- und USB-Tastaturen sowie HDMI- und VGA-Videoausgang**. Dazu gehört die vollständige Unterstützung für Grafik, Audio (Soundeffekte und Musik), internen Programmspeicher, Gamecontroller und mehr. Dadurch wird der Raspberry Pi Pico zu einem eigenständigen Computer, ähnlich wie der Apple II oder der Tandy TRS-80 von früher. Ideal zum Schreiben von Spielen, zum Erlernen von BASIC oder einfach zum Führen deines Haushaltsbuchs
- **Flexibler Programm- und Datenspeicher**. Programme und Daten können von einem internen Dateisystem gelesen und geschrieben werden, das aus dem Flash-Speicher des Pico erstellt wurde, oder auf eine extern angeschlossene SD-Karte mit bis zu 32 GB, die als FAT16 oder FAT32 formatiert ist. Dazu gehören das Öffnen von Dateien zum Lesen, Schreiben oder für den wahlfreien Zugriff sowie das Laden und Speichern von Programmen.
- In der Firmware ist **ein Vollbild-Editor** integriert, mit dem du das gesamte Programm in einem Durchgang bearbeiten kannst. Er bietet erweiterte Funktionen wie farblich hervorgehobene Syntax, Suchen sowie Kopieren, Ausschneiden und Einfügen in die bzw. aus der Zwischenablage.

- **Programme lassen sich ganz einfach** von einem Desktop- oder Laptop-Computer (Windows, Mac oder Linux) über die serielle Konsole oder eine SD-Karte **übertragen**.
- Es ist **eine umfassende Auswahl an Kommunikationsprotokollen** implementiert, darunter I²C, asynchrone serielle Schnittstelle, RS232, SPI und 1-Wire. Diese können zur Kommunikation mit vielen Sensoren (Temperatur, Luftfeuchtigkeit, Beschleunigung usw.) sowie zum Senden von Daten an Testgeräte verwendet werden.
- **Integrierte Befehle** für die direkte Anbindung an Infrarot-Fernbedienungen, den DS18B20-Temperatursensor, LCD-Anzeigemodule, eine batteriegepufferte Uhr, numerische Tastaturen und mehr.

# Firmware-Versionen und Dateien

Die PicoMite-Firmware kann je nach geladener Version für zwei deutlich unterschiedliche Aufgaben genutzt werden. Diese Aufgaben sind: ein eigenständiger Computer und ein eingebetteter Controller:

## Eigenständiger Computer

Versionen mit VGA- oder HDMI-Videoausgang sind für den Einsatz als eigenständiger Computer vorgesehen. Diese starten hoch und zeigen die Ausgabe des BASIC-Interpreters auf dem an den Videoausgang angeschlossenen Monitor an. Sie werden mit einer PS2- oder USB-Tastatur verbunden, und mithilfe der Tastatur und des Videoausgangs kannst du ein Programm eingeben und bearbeiten, es ausführen, Optionen einstellen usw.

Da der eigenständige Computer mit der Anzeige der BASIC-Eingabeaufforderung startet, werden sie oft als „Boot-to-BASIC"-Computer bezeichnet. Sie sind einfach und machen Spaß und waren in den 70er- und 80er-Jahren beliebt, zum Beispiel der Apple II, Tandy TRS-80, Commodore 64 und andere.

Wenn ein Programm läuft, wird die gesamte Ausgabe (Text und Grafiken) über den Videoausgang angezeigt. Die Textausgabe wird außerdem an die serielle Konsole gesendet. Dies ist ein sekundärer Kommunikationskanal, der den USB-Anschluss des Raspberry Pi Pico nutzt, und stellt eine weitere Möglichkeit dar, über einen Desktop- oder Laptop-Computer mit dem MMBasic-Interpreter zu kommunizieren. Details zur Verwendung findest du im nächsten Kapitel: *Serielle Konsole*.

## Embedded-Controller

Firmware-Versionen ohne Videoausgang sind in erster Linie für den Einsatz als eingebetteter Controller gedacht. Hier wird der Raspberry Pi Pico oder Pico 2 als „Gehirn" in einem Gerät verwendet. Zum Beispiel in einer Alarmanlage, einem Heizungsregler, einer Wetterstation usw. Oft ist ein berührungsempfindliches LCD-Panel angebracht, über das der Benutzer das Gerät steuern und die Ausgabe beobachten kann.

Es gibt auch eine Version der Firmware, die die drahtlose Schnittstelle des Raspberry Pi Pico W (und 2 W) unterstützt. Damit kannst du einen eingebetteten Controller erstellen, auf dem ein Miniatur-Webserver läuft, der über das Internet die Uhrzeit abrufen, E-Mails versenden usw. kann.

Um Programme einzugeben, Optionen festzulegen und den Raspberry Pi Pico generell als eingebetteten Controller zu verwalten, verbindest du dich über die serielle Konsole mit einem Desktop- oder Laptop-Computer. Im Gegensatz zu dem oben beschriebenen eigenständigen Computer ist dies die einzige Möglichkeit, mit dem BASIC-Interpreter zu kommunizieren; daher ist es wichtig, dass du eine Verbindung herstellen kannst. Eine Beschreibung der seriellen Konsole findest du unter der Überschrift *„Serielle Konsole"* in diesem Handbuch.

## Prozessorunterstützung

Die PicoMite-Firmware unterstützt die ursprünglichen RP2040-Prozessoren, die im Raspberry Pi Pico verwendet werden, sowie den neueren RP2350, der im Raspberry Pi Pico 2 zum Einsatz kommt. Die Firmware ist zudem so konzipiert, dass sie mit Modulen anderer Hersteller funktioniert, die dieselben Chips verwenden.

Während es vom RP2040 nur eine Version gibt, gibt es vom RP2350 vier Unterversionen mit den Bezeichnungen RP2350A, RP2350B, RP2354A und RP2354B. Der RP2350B entspricht dem RP2350A, verfügt jedoch über 18 zusätzliche I/O-Pins (Pins GP30 bis GP47), die in MMBasic automatisch zur Verfügung stehen. Beide Chips werden von derselben PicoMite-Firmware unterstützt und funktionieren identisch. In diesem Handbuch gelten daher alle Verweise auf den RP2350 gleichermaßen für die Varianten A und B, und es kann dieselbe Firmware verwendet werden.

Der RP2354A und der RP2354B werden derzeit nicht unterstützt (könnten es aber in Zukunft sein).

In diesem Handbuch beziehen sich alle Verweise auf den Raspberry Pi Pico auch auf den Raspberry Pi Pico 2, sofern dies nicht ausdrücklich ausgeschlossen wird. Sollten Unterschiede bestehen, wird die Teilenummer des Prozessors (RP2040 oder RP2350) verwendet, um den Unterschied deutlich zu machen.

## PicoCalc-Unterstützung

Die PicoMite-Firmware unterstützt den von clockwork angebotenen PicoCalc. Lade einfach die richtige Firmware – PicoMiteRP2040, PicoMiteRP2350, WebMiteRP2040 oder WebMiteRP2350 – auf einen leeren Pico, und der PicoCalc wird automatisch konfiguriert und ist einsatzbereit.

## Dateinamen

Die ZIP-Datei mit der Firmware-Distribution enthält zwölf Firmware-Dateien.

Ein typischer Dateiname für ein Firmware-Image sieht so aus:

PicoMiteRP2350VGAUSBV6.01.00.uf2

Wobei (in diesem Beispiel):

- **RP2350** der Prozessor ist, für den die Firmware kompiliert wurde.
- **VGAUSB** steht für die unterstützten Funktionen (VGA und USB).
- **V6.01.00** ist die Versionsnummer. Diese wird in zukünftigen Versionen erhöht.
- **.uf2** ist die Erweiterung, die auf ein ladbares Raspberry Pi Pico-Firmware-Image hinweist.

Die folgende Tabelle listet das Präfix für jede Firmware-Datei und die damit verbundenen Funktionen auf.

| **Name der Firmware-Datei**<br>Zum Beispiel: PicoMiteRP2040V6.01.00.uf2 | CPU | Touch-LCD-Panel | Tastatur/Maus | | Videoausgang | | WLAN-Internet |
|---|---|---|---|---|---|---|---|
| | | | PS2 | USB | VGA | HDMI | |
| PicoMiteRP2040 | RP2040 | ✓ | ✓ | | | | |
| PicoMiteRP2350 | RP2350 | ✓ | ✓ | | | | |
| PicoMiteRP2040USB | RP2040 | ✓ | | ✓ | | | |
| PicoMiteRP2350USB | RP2350 | ✓ | | ✓ | | | |
| PicoMiteRP2040VGA | RP2040 | | ✓ | | ✓ | | |
| PicoMiteRP2350VGA | RP2350 | | ✓ | | ✓ | | |
| PicoMiteRP2040VGAUSB | RP2040 | | | ✓ | ✓ | | |
| PicoMiteRP2350VGAUSB | RP2350 | | | ✓ | ✓ | | |
| PicoMiteHDMI | RP2350 | | ✓ | | | ✓ | |
| PicoMiteHDMIUSB | RP2350 | | | ✓ | | ✓ | |
| WebMiteRP2040 | RP2040 | ✓ | ✓ | | | | ✓ |
| WebMiteRP2350 | RP2350A | ✓ | ✓ | | | | ✓ |

## Laden der Firmware

Der Raspberry Pi Pico und der Pico 2 verfügen über einen eigenen, integrierten Firmware-Loader, der einfach zu bedienen ist.

Um die PicoMite-Firmware zu laden, gehe wie folgt vor:

- Lade die PicoMite-Firmware von http://geoffg.net/picomite.html herunter, entpacke die Datei und suche die für deinen Einsatzzweck passende Firmware heraus (siehe die vorherigen Abschnitte).
- Schließe den Raspberry Pi Pico mit einem USB-Kabel an deinen Computer (Windows, Linux oder Mac) an, **während du die weiße BOOTSEL-Taste** oben auf dem Modul **gedrückt hältst**.
- Der Raspberry Pi Pico sollte sich mit deinem Computer verbinden und ein virtuelles Laufwerk erstellen (genau wie beim Anschließen eines USB-Sticks). Du kannst alle Dateien ignorieren, die sich möglicherweise auf diesem „Laufwerk“ befinden.
- Kopiere die Firmware-Datei (mit der Endung .uf2) auf dieses virtuelle Laufwerk.
- Sobald der Kopiervorgang abgeschlossen ist, startet der Raspberry Pi Pico neu und erstellt einen virtuellen seriellen Port über USB auf deinem Computer. Details zur Verwendung findest du im Abschnitt *„Serielle Konsole“* weiter unten.
- Die LED am Raspberry Pi Pico blinkt langsam und zeigt damit an, dass die PicoMite-Firmware mit MMBasic nun läuft.

Das vom Raspberry Pi Pico erstellte virtuelle Laufwerk sieht zwar aus wie ein USB-Stick, ist aber keiner. Die Firmware-Datei verschwindet nach dem Kopieren, und wenn du versuchst, eine andere Art von Datei zu kopieren, wird diese ignoriert.

Das Laden der PicoMite-Firmware kann den gesamten Flash-Speicher löschen, einschließlich des aktuellen Programms, aller Dateien auf Laufwerk A: und aller gespeicherten Variablen. Stelle daher sicher, dass du diese Daten sicherst, bevor du die Firmware aktualisierst.

Es ist möglich, dass der Flash-Speicher beschädigt wird, was zu ungewöhnlichem und unvorhersehbarem Verhalten führen kann. In diesem Fall solltest du die unten aufgeführte passende Firmware herunterladen und wie oben beschrieben auf den Pico laden. Dadurch wird der Raspberry Pi Pico auf den Werkszustand zurückgesetzt, und du kannst die PicoMite-Firmware erneut laden:

| | |
|---|---|
| Raspberry Pi Pico (RP2040) | https://geoffg.net/Downloads/picomite/Clear_Flash.uf2 |
| Raspberry Pi Pico 2 (RP2350) | https://geoffg.net/Downloads/picomite/Clear_Flash_RP2350.uf2 |

# Serielle Konsole

Die serielle Konsole ist eine Möglichkeit, deinen Desktop- oder Laptop-Computer mit dem Raspberry Pi Pico und der MMBasic-Konsole zu verbinden. Mit Zugriff auf die Konsole kannst du Programme eingeben und bearbeiten, sie ausführen usw. Die meisten Firmware-Versionen richten die serielle Konsole automatisch als virtuellen seriellen Port über USB ein. In diesem Kapitel wird beschrieben, wie das funktioniert und wie du sie nutzt.

Bei einem eigenständigen Computer (wie oben beschrieben) ist die serielle Konsole eine sekundäre Kommunikationsmethode, aber wenn du den Raspberry Pi Pico als eingebetteten Controller verwendest, ist sie die einzige verfügbare Kommunikationsmethode, daher ist es wichtig, dass du eine Verbindung herstellen kannst.

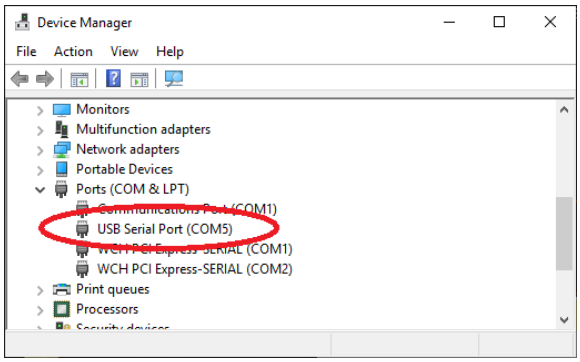
Versionen der PicoMite-Firmware, die eine USB-Tastatur/Maus unterstützen, können den seriellen Port nicht über USB erstellen. Für diese Firmware solltest du daher das Kapitel *„Tastatur/Maus/Gamepad“* lesen, um eine Alternative zu finden.

## Virtueller serieller Port

Der von der PicoMite-Firmware über USB erstellte virtuelle serielle Port nutzt das CDC-Protokoll (Communication Device Class) und verhält sich wie ein normaler serieller Port, funktioniert jedoch über USB. Windows 10 und 11 enthalten einen Treiber dafür, bei anderen Betriebssystemen musst du möglicherweise einen Treiber laden (siehe nächste Seite).

Wenn du den USB-Anschluss des Raspberry Pi Pico an deinen Desktop- oder Laptop-Computer anschließt (nachdem du die PicoMite-Firmware geladen hast), wird die Verbindung sofort hergestellt.

Du solltest dir dann die Portnummer notieren, die dein Computer für die virtuelle serielle Verbindung erstellt hat. In Windows kannst du dies tun, indem du den Geräte-Manager startest und unter dem Eintrag „Anschlüsse (COM & LPT)“ nach einem neuen COM-Port suchst, wie rechts dargestellt.



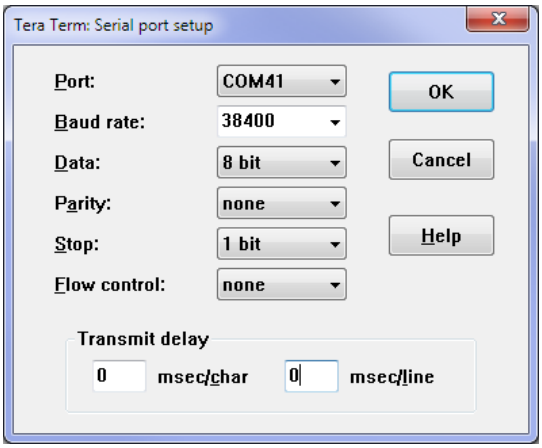

## Terminalemulator

Du benötigst außerdem einen Terminalemulator, der auf deinem Desktop- oder Laptop-Computer läuft. Dies ist ein Programm, das wie ein altmodisches Computerterminal funktioniert: Es zeigt den von einem Remote-Computer empfangenen Text an, und alle Tasteneingaben werden über die serielle Verbindung an den Remote-Computer gesendet. Der von dir verwendete Terminalemulator sollte VT100-Emulation unterstützen, da dies vom in der PicoMite-Firmware integrierten Editor benötigt wird.

Für Windows-Nutzer wird empfohlen, Tera Term zu verwenden, da dieses Programm über einen guten VT100-Emulator verfügt und bekanntermaßen mit dem XModem-Protokoll funktioniert, über das du Programme zum und vom PicoMite übertragen kannst. Tera Term kann hier heruntergeladen werden: http://tera-term.en.lo4d.com .

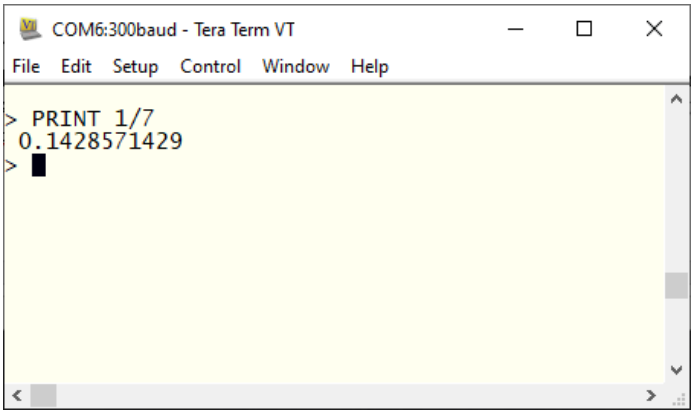
Der Screenshot rechts zeigt die Einstellungen für Tera Term. Beachte, dass die Einstellung unter „Port:“ davon abhängt, an welchem USB-Anschluss dein Raspberry Pi Pico angeschlossen ist.

Die PicoMite-Firmware ignoriert die Baudrateneinstellung, sodass diese auf eine beliebige Geschwindigkeit eingestellt werden kann (außer 1200 Baud, da dies den Pico in den Firmware-Upgrade-Modus versetzt).



Wenn du Tera Term verwendest, stelle keine Verzögerung zwischen den Zeichen ein, und wenn du Putty verwendest, stelle die Rücktaste so ein, dass sie das Rücktastenzeichen erzeugt.

## Die Konsole




Sobald du den virtuellen seriellen Port identifiziert und deinen Terminalemulator damit verbunden hast, solltest du die Eingabetaste auf deiner Tastatur drücken können und die MMBasic-Eingabeaufforderung sehen, die aus dem Größer-als-Zeichen besteht (z. B. „>“).

Dies ist die Konsole, über die du Befehle eingibst, um MMBasic zu konfigurieren, das BASIC-Programm zu laden, zu bearbeiten und auszuführen. MMBasic nutzt die Konsole auch, um etwaige Fehlermeldungen anzuzeigen.

## Windows 7 und 8.1

Der serielle USB-Anschluss nutzt das CDC-Protokoll; die Treiber dafür sind in Windows 10 und 11 standardmäßig enthalten und werden automatisch geladen.

Die Raspberry Pi Foundation listet Windows 7 oder 8.1 als „nicht unterstützt“ auf, du kannst jedoch ein Tool wie Zadig ( https://zadig.akeo.ie ) verwenden, um einen generischen Treiber für ein „usbser“-Gerät zu installieren, wodurch diese Computer verbunden werden sollten. Dieser Beitrag beschreibt den Vorgang: https://github.com/raspberrypi/pico-feedback/issues/118

## Apple Macintosh

Der Apple Macintosh (OS X) ist etwas einfacher, da er über einen integrierten Gerätetreiber und Terminalemulator verfügt. Starte zunächst die Anwendung „Terminal“ und liste an der Eingabeaufforderung die angeschlossenen seriellen Geräte auf, indem du Folgendes eingibst:

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

Der USB-zu-Seriell-Konverter wird etwa als `/dev/tty.usbmodem12345` aufgeführt. Während du dich noch in der Terminal-Eingabeaufforderung befindest, kannst du den Terminalemulator mit 115200 Baud starten, indem du den folgenden Befehl eingibst:

```
screen   /dev/tty.usbmodem12345 115200
```

Standardmäßig sind die Funktionstasten für die Verwendung im integrierten Programmeditor des PicoMite nicht korrekt definiert, sodass du die im Kapitel *„Vollbild-Editor“* dieses Handbuchs beschriebenen Steuerungssequenzen verwenden musst. Um dies zu vermeiden, kannst du den Terminalemulator so konfigurieren, dass er diese Codes generiert, wenn die entsprechenden Funktionstasten gedrückt werden.

Die Dokumentation zum Befehl „screen“ findest du hier: https://www.systutorials.com/docs/linux/man/1-screen/

## Linux

Für Linux siehe diese Beiträge:

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

und

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

## Android

Für Android-Geräte schau dir diesen Beitrag an:

https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=17476&LastEntry=Y#230521#230517

# Erste Schritte

Sobald du Zugriff auf die Konsole und die MMBasic-Eingabeaufforderung hast, kannst du ein paar Dinge tun, um zu überprüfen, ob dein Computer funktioniert. Alle diese Befehle sollten an der Eingabeaufforderung (d. h. „>“) eingegeben werden.

Was du eingibst, wird fett dargestellt, und die MMBasic-Ausgabe wird in normaler Schrift angezeigt.

Versuche eine einfache Berechnung:

```
> PRINT 1/7
0,1428571429
```

Schau nach, wie viel Speicher du hast:

```
> MEMORY
Programm:
   0K (0 %) Programm (0 Zeilen)
 180K (100 %) Frei

Gespeicherte Variablen:
  16K (100 %) Frei

RAM:
   0K (0 %) 0 Variablen
   0 KB (0 %) Allgemein
 228K (100 %) frei 112K (100 %) frei
```

Wie spät ist es gerade? Beachte, dass die interne Uhr beim Einschalten auf Mitternacht zurückgesetzt wird.

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

Stelle die Uhr auf die aktuelle Uhrzeit ein:

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

Überprüfe die Uhrzeit noch einmal:

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

Zähle bis 20:

```
> FOR a = 1 to 20 : PRINT a; : NEXT a
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

## Ein einfaches Programm

Um ein Programm einzugeben, kannst du den Befehl EDIT verwenden, der später in diesem Handbuch beschrieben wird. Im Moment reicht es jedoch zu wissen, dass alles, was du eingibst, an der Cursorposition eingefügt wird, die Pfeiltasten den Cursor bewegen und die Rücktaste das Zeichen vor dem Cursor löscht.

Um einen schnellen Eindruck davon zu bekommen, wie MMBasic funktioniert, probier diese Schritte aus:

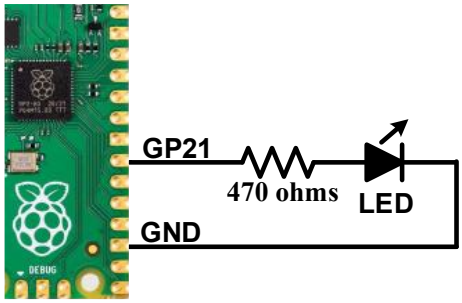
- Gib an der Eingabeaufforderung `EDIT` ein und drücke die ENTER-Taste.
- Der Editor sollte starten und du kannst diese Zeile eingeben: `PRINT „Hello World“`
- Drücke die F1-Taste in deinem Terminalemulator (oder STRG-Q, was dasselbe bewirkt). Dadurch wird der Editor angewiesen, dein Programm zu speichern und zur Eingabeaufforderung zurückzukehren.
- Gib an der Befehlszeile `RUN` ein und drücke die Eingabetaste.
- Du solltest die Meldung `„Hello World“` sehen

Herzlichen Glückwunsch. Du hast gerade dein erstes Programm in BASIC geschrieben und ausgeführt.

Wenn du erneut EDIT eingibst, gelangst du zurück in den Editor, wo du dein Programm ändern oder ergänzen kannst.

## Eine LED zum Blinken bringen

Schließe eine LED an Pin GP21 (auf der Unterseite der Platine markiert) und einen Masse-Pin an, wie in der Abbildung rechts gezeigt.




Gib dann mit dem Befehl EDIT das folgende Programm ein:

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

Wenn du dieses Programm gespeichert und ausgeführt hast, sollte die LED blinken. Es ist kein großartiges Programm, aber es veranschaulicht, wie die PicoMite-Firmware über deine Programmierung mit der physischen Welt interagieren kann.

Das Programm selbst ist einfach. Die erste Zeile legt den Pin GP21 als Ausgang fest. Dann geht das Programm in eine Endlosschleife, in der der Ausgang dieses Pins auf „High“ gesetzt wird, um die LED einzuschalten, gefolgt von einer kurzen Pause (300 Millisekunden). Der Ausgang wird dann auf „Low“ gesetzt, gefolgt von einer weiteren Pause. Das Programm wiederholt dann die Schleife.

Wenn du es so belässt, bleibt der Raspberry Pi Pico für immer in diesem Zustand, während die LED blinkt. Wenn du etwas ändern möchtest (zum Beispiel die Blinkgeschwindigkeit), kannst du das Programm unterbrechen, indem du STRG-C in die Konsole eingibst, 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 du möchtest, dass dieses Programm bei jedem Einschalten automatisch startet, kannst du den folgenden Befehl an der Eingabeaufforderung oder im Programm verwenden:

```
OPTION AUTORUN ON
```

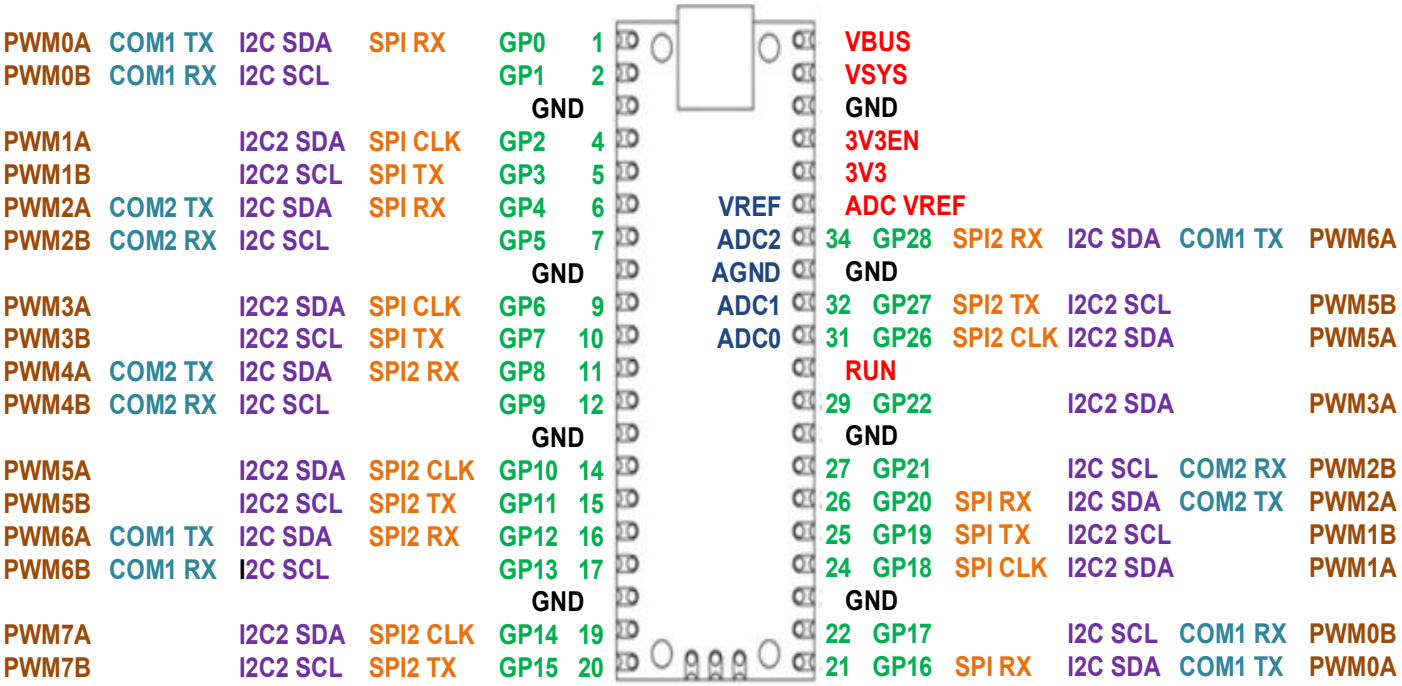
Um dies zu testen, kannst du das Gerät vom Stromnetz trennen und dann wieder anschließen. Das Gerät sollte starten und die LED blinken lassen.

## Tutorial zur Programmierung in der Sprache BASIC

Wenn du noch keine Erfahrung mit der Programmiersprache BASIC hast, wäre jetzt ein guter Zeitpunkt, um einen Blick in Anhang J (*Programmieren in BASIC – Ein Tutorial*) am Ende dieses Handbuchs zu werfen. Dies ist ein umfassendes Tutorial zur Sprache, das dich in einem leicht verständlichen Format mit vielen Beispielen durch die Grundlagen führt.

# Details zur Hardware-

Dieses Diagramm zeigt die möglichen Verwendungszwecke innerhalb von MMBasic für jeden I/O-Pin auf dem Raspberry Pi Pico und Pico 2:




Bei Versionen mit VGA-Videoausgang sind sechs Pins (GP16 bis GP21) für diese Funktion reserviert. Ebenso sind bei HDMI-Versionen acht Pins (GP12 bis GP19) für diese Funktion reserviert. Weitere Informationen findest du im Kapitel *„Videoausgang“*.

Die Firmware-Version mit USB-Tastatur-/Maus-Unterstützung reserviert außerdem Pin 11 (GP8) für die serielle Konsole Tx und Pin 12 (GP9) für Rx. Weitere Informationen findest du im Kapitel *„Tastatur/Maus/Gamepad“*.

Die Notation lautet wie folgt:

| | |
|---|---|
| GP0 bis GP28 | Kann für digitale Ein- oder Ausgänge verwendet werden. |
| COM1, COM2 | Können für asynchrone serielle E/A verwendet werden (Pins UART0 und UART1 im Pico-Datenblatt). |
| I2C, I2C2 | Kann für $I^2C$-Kommunikation verwendet werden (I2C0- und I2C1-Pins im Pico-Datenblatt). |
| SPI, SPI2 | Kann für SPI-I/O verwendet werden (siehe Anhang D). (SPI0- und SPI1-Pins im Pico-Datenblatt). |
| PWMnx | Kann für PWM-Ausgänge verwendet werden (siehe die PWM- und SERVO-Befehle). |
| GND | Gemeinsame Masse. |
| VBUS | 5-V-Versorgung direkt vom USB-Anschluss. |
| VSYS | 5-V-Versorgung, die vom SMPS zur Bereitstellung von 3,3 V genutzt wird. Kann als 5-V-Ausgang oder -Eingang verwendet werden. |
| 3V3EN | Aktiviert den 3,3-V-Regler (Low = aus, High = aktiviert). |
| RUN | Reset-Pin, Low hält das Gerät im Reset-Zustand. |
| ADCn | Diese Pins können zur Spannungsmessung verwendet werden (Analogeingang). |
| ADC VREF | Referenzspannung für die Spannungsmessung. |
| AGND | Analoge Masse. |

Im MMBasic-Programm kannst du auf I/O-Pins mit der physikalischen Pin-Nummer (d. h. 1 bis 40) oder der GP-Nummer (d. h. GP0 bis GP28) verweisen. Die folgenden Beispiele beziehen sich beispielsweise auf denselben Pin und funktionieren identisch:

```
      SETPIN 32, DOUT
und   SETPIN GP27, DOUT
```

Die Pin-Nummern (d. h. 1 bis 40) gelten nur für die Standard-Formfaktoren Pico und Pico 2; andere Module von Drittanbietern können ein anderes Pin-Nummerierungsschema haben. Aus diesem Grund wird dringend empfohlen, bei der Bezugnahme auf einen I/O-Pin nur die GP-Nummer zu verwenden.

Du kannst für GP-Pin-Nummern auch String-Literale und String-Variablen verwenden (außer beim PORT-Befehl und bei Funktionen). Zum Beispiel:

```
CONST outpin$ = "GP27"
SETPIN outpin$, DOUT
PIN(outpin$) = 1
```

In der PicoMite-Firmware sind On-Chip-Funktionen wie die SPI- und I2C-Schnittstellen nicht festen Pins zugeordnet, anders als (zum Beispiel) beim Micromite. Die PicoMite-Firmware nutzt den SETPIN-Befehl ausgiebig, nicht nur zur Konfiguration von I/O-Pins, sondern auch zur Konfiguration der Pins, die für Schnittstellen wie Serial, SPI, I2C usw. verwendet werden.

Die Pins müssen gemäß dieser Zeichnung zugewiesen werden. Beispielsweise kann der SPI-TX den Pins GP3, GP7 oder GP19 zugewiesen werden, jedoch nicht dem Pin GP11, der ausschließlich dem SPI2-Kanal zugewiesen werden kann. Die Zuweisungen müssen nicht im selben „Block“ erfolgen, sodass du beispielsweise SPI2-TX dem Pin GP11 und SPI2-RX dem Pin GP28 zuweisen könntest.

## Module von Drittanbietern

Auf Pins, die beim Raspberry Pi Pico nicht freiliegen, kannst du mit MMBasic trotzdem über ihre GPn-Nummer zugreifen. Dadurch lässt sich MMBasic auch auf anderen Modulen nutzen, die mit den Prozessoren RP2040 oder RP2350 arbeiten.

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

- GP23 ist ein digitaler Ausgang, der auf den Wert von OPTION POWER gesetzt ist. (ON=PWM, OFF=PFM).
- GP24 ist ein digitaler Eingang, der auf High steht, wenn VBUS anliegt.
- GP25 ist auch PWM4B. Es ist ein Ausgang, der mit der integrierten LED verbunden ist.
- GP29 ist auch ADC3, ein analoger Eingang, der ⅓ von VSYS misst.

Auf Modulen von Drittanbietern, die diese Pins zur Verfügung stellen, können sie jedoch wie folgt verwendet werden:

- GP23: DIGITAL_IN: DIGITAL_OUT, SPI TX, I2C2 SCL, PWM3B
- GP24: DIGITAL_IN: DIGITAL_OUT, SPI2 RX, COM2 TX, I2C SDA, PWM4A
- GP25: DIGITAL_IN: DIGITAL_OUT, COM2 RX, I2C SCL, PWM4B
- GP29: DIGITAL_IN: DIGITAL_OUT, ANALOG_IN, COM1 RX, I2C SCL, PWM6B

## WebMite-Version für den Raspberry Pi Pico W oder Pico 2 W

Die WebMite-Version verfügt außerdem über einige GPIO-Pins, die für interne Board-Funktionen genutzt werden:

- GP29 (Eingang/Ausgang) im drahtlosen SPI-CLK/ADC-Modus (ADC3) misst VSYS/3
- GP25 SPI CS (Ausgang) – wenn High, aktiviert dies auch den GPIO29-ADC-Pin zum Auslesen von VSYS
- GP24 (Eingang/Ausgang) Wireless-SPI-Daten / IRQ
- GP23 (Ausgang) – Signal für das Einschalten der drahtlosen Verbindung

Die WebMite-Firmware erlaubt keine Neuzuweisung dieser Pins.

Im Gegensatz zum Standard-Raspberry Pi Pico ist die integrierte LED des Pico W nicht mit einem Pin des RP2040 verbunden, sondern mit einem GPIO-Pin des Wireless-Chips und kann nicht von einem BASIC-Programm aus angesteuert werden.

Die Antenne befindet sich auf der Leiterplatte am gegenüberliegenden Ende des USB-Anschlusses und sollte für eine optimale Leistung frei bleiben – lege kein Metall unter oder in die Nähe der Antenne.

## CPU-Varianten

Die von der Raspberry Pi Foundation für den Pico und den Pico 2 veröffentlichten Chips sind:

RP2040

Dieser ist in einem 60-Pin-Gehäuse untergebracht und wird im Original-Raspberry Pi Pico sowie in vielen anderen Modulen von Drittanbietern verwendet. Die Pinbelegung und Funktionen der I/O-Pins entsprechen der oben dokumentierten Beschreibung.

RP2350A

Dieser ist ebenfalls in einem 60-Pin-Gehäuse untergebracht und wird im Raspberry Pi Pico 2 sowie in Modulen von Drittanbietern verwendet. Die Pinbelegung und Funktionen der I/O-Pins entsprechen denen des RP2040 und sind oben dokumentiert.

RP2350B

Der RP2350B ist in einem 80-Pin-Gehäuse untergebracht und verfügt über 18 zusätzliche GPIO-Pins. Die PicoMite-Firmware unterstützt diese zusätzlichen Pins, einschließlich der PWM-Kanäle 8–11 (wodurch maximal 24 gleichzeitige PWM-Ausgänge möglich sind). Die Konfiguration für den RP2350B erfolgt automatisch, wodurch alle zusätzlichen Pins und drei PIO-Kanäle verfügbar sind (die VGA-Version verfügt über zwei freie PIO-Kanäle). Die Pin-Definitionen für den RP2350B sollten die GP- -Nomenklatur verwenden (d. h. GP0 bis GP47). Die Pins GP40 bis GP47 können für analoge Eingänge verwendet werden, die Pins GP26–GP29 unterstützen keine analogen Eingänge.

Die Pin-Belegung für die Pins GP0 bis GP29 beim RP2350B entspricht der des RP2040 und RP2350A. Die Pins GP30 bis GP47 können wie folgt verwendet werden:

GP30: DIGITAL_IN: DIGITAL_OUT, SPI2 SCK, I2C2 SDA, PWM7A
GP31: DIGITAL_IN: DIGITAL_OUT, SPI2 TX, I2C2 SCL, PWM7B
GP32: DIGITAL_IN: DIGITAL_OUT, COM1 TX, SPI RX, I2C SDA, EXT_PWM8A
GP33: DIGITAL_IN: DIGITAL_OUT, COM1 RX, I2C SCL, PWM8B
GP34: DIGITAL_IN: DIGITAL_OUT, SPI SCK, I2C2 SDA, PWM9A
GP35: DIGITAL_IN: DIGITAL_OUT, SPI TX, I2C2 SCL, PWM9B
GP36: DIGITAL_IN: DIGITAL_OUT, COM2 TX, SPI RX, I2C SDA, PWM10A
GP37: DIGITAL_IN: DIGITAL_OUT, COM2 RX, I2C SCL, PWM10B
GP38: DIGITAL_IN: DIGITAL_OUT, SPI SCK, I2C2 SDA, PWM11A
GP39: DIGITAL_IN: DIGITAL_OUT, SPI TX, I2C2 SCL, PWM11B
GP40: DIGITAL_IN: DIGITAL_OUT, ANALOG_IN, COM2 TX, SPI2 RX, I2C SDA, PWM8A
GP41: DIGITAL_IN: DIGITAL_OUT, ANALOG_IN, COM2 RX, I2C SCL, PWM8B
GP42: DIGITAL_IN: DIGITAL_OUT, ANALOG_IN, SPI2 SCK, I2C2 SDA, PWM9A
GP43: DIGITAL_IN: DIGITAL_OUT, ANALOG_IN, SPI2 TX, I2C2 SCL, PWM9B
GP44: DIGITAL_IN: DIGITAL_OUT, COM1TX, ANALOG_IN, SPI2 RX, I2C SDA, PWM10A
GP45: DIGITAL_IN: DIGITAL_OUT, COM1RX, ANALOG_IN, I2C SCL, PWM10B
GP46: DIGITAL_IN: DIGITAL_OUT, ANALOG_IN, SPI2 SCK, I2C2 SDA, PWM11A
GP47: DIGITAL_IN: DIGITAL_OUT, ANALOG_IN, SPI2 TX, I2C2 SCL, PWM11B

RP2354A und RP2354B

Diese werden derzeit von der PicoMite-Firmware nicht unterstützt.

## PSRAM

Der RP2350 unterstützt PSRAM, und einige kommerzielle Angebote haben Boards mit dem 80-Pin-RP2350B um 8 MB PSRAM erweitert (z. B. Pimoroni PGA2350).

Der Zugriff auf das PSRAM erfolgt über denselben Quad-SPI-Bus, der auch vom Flash-Speicher genutzt wird, weshalb es vergleichsweise langsam ist, obwohl es über einen Cache gepuffert wird, was dieses Problem abmildert. Wenn ein PSRAM vorhanden und konfiguriert ist, fügt MMBasic es dem allgemeinen RAM-Pool hinzu, sodass Programme über eine enorme Menge an allgemeinem RAM verfügen, auch wenn dieses vielleicht etwas langsamer ist.

Um auf einen PSRAM zuzugreifen, wird ein zusätzlicher Pin für die Chip-Select-Funktion benötigt, der mit dem Befehl OPTION PSRAM PIN ausgewählt wird. Gültige Pins für den PSRAM-Chip-Select sind GP0, GP8, GP19 und GP47.

Nach dem Einschalten ist der Inhalt des PSRAM unbestimmt – er ist nicht auf Null gesetzt. Du musst RAM ERASE verwenden, um ihn vor der Verwendung zu löschen. Dieses Verhalten ist beabsichtigt, damit der Inhalt einen Reset übersteht.

## I/O-Pin-Grenzwerte

Die maximale Spannung, die an einen beliebigen I/O-Pin des Raspberry Pi Pico mit RP2040-Prozessor angelegt werden kann, beträgt 3,6 V. Der Raspberry Pi Pico 2 mit RP2350-Prozessor kann 5 V vertragen, solange der Chip mit Strom versorgt wird.

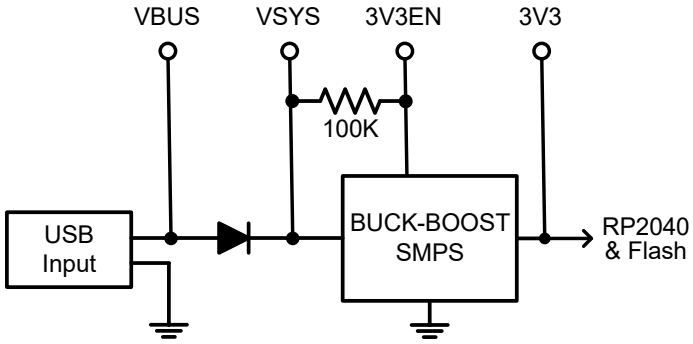
Als Ausgänge können alle I/O-Pins einzeln bis zu 8 mA liefern oder aufnehmen. Bei dieser Last sinkt die Ausgangsspannung auf etwa 2,3 V. Eine praktischere Last ist 5 mA, bei der die Ausgangsspannung typischerweise 3 V beträgt. Um eine rote LED mit 5 mA zu betreiben, beträgt der empfohlene Widerstand 220 Ω. Andere Farben erfordern möglicherweise einen anderen Wert.

Die maximale Gesamt-I/O-Strombelastung für den gesamten Chip beträgt 100 mA.

## Stromversorgung

Der Raspberry Pi Pico und Pico 2 verfügt über ein flexibles Stromversorgungssystem .

Die Eingangsspannung von entweder dem USB- oder dem VBUS-Eingang wird über eine - Schottky-Diode an das Buck-Boost-SMPS (Switch Mode Power Supply) mit einer Ausgangsspannung von 3,3 V weitergeleitet. Das SMPS unterstützt Eingangsspannungen von 1,8 V bis 5,5 V, sodass das Gerät 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 in der Regel eine externe Stromquelle (außer USB) erforderlich, die über eine „ “-Schottky- -Diode an VSYS angeschlossen werden kann. Dadurch kann der Raspberry Pi Pico von der Quelle mit der höheren Spannung (USB oder VSYS) gespeist werden. Die Dioden verhindern eine Rückkopplung in die Versorgung mit der niedrigeren Spannung.

Um Störgeräusche der Stromversorgung zu minimieren, kannst du 3V3EN erden, um den SMPS auszuschalten. Beim Herunterfahren hört der Wandler auf zu schalten, die interne Steuerelektronik wird ausgeschaltet und die Last getrennt. Du kannst das Board dann über einen 3,3-V-Linearregler versorgen, der in den 3V3-Pin eingespeist wird.

## Taktfrequenz

Standardmäßig beträgt die Taktrate des im Raspberry Pi Pico verwendeten RP2040 200 MHz und die des im Raspberry Pi Pico 2 verwendeten RP2350 150 MHz. Das sind die empfohlenen Höchstwerte.

Mit dem Befehl OPTION CPUSPEED lassen sich die meisten RP2040-CPUs auf bis zu 420 MHz und der RP2350 auf bis zu 396 MHz übertakten (nur PicoMite und WebMite). Sie können auch langsamer laufen, bis auf mindestens 48 MHz. Diese Einstellung wird gespeichert und beim Einschalten wieder angewendet. Wenn du die Taktrate änderst, wird die PicoMite-Firmware zurückgesetzt und anschließend neu gestartet, sodass die USB-Verbindung unterbrochen wird.

Bei VGA-/HDMI-Versionen mit einer Videoauflösung von 640x400 wird die Taktrate auf 252 MHz eingestellt; dies kann jedoch über OPTION RESOLUTION auf 252 MHz, 315 MHz oder 378 MHz geändert werden. Bei anderen Bildschirmauflösungen ist die Taktrate je nach gewählter Auflösung auf 283,2 MHz, 324 MHz, 360 MHz, 372 MHz oder 375 MHz festgelegt und kann nicht geändert werden.

Nahezu alle getesteten Raspberry Pi Picos haben bei 380 MHz oder mehr einwandfrei funktioniert, daher kann Übertaktung sinnvoll sein. Falls der Prozessor bei der neuen Taktrate nicht neu startet, kannst du ihn zurücksetzen, indem du Clear_flash.uf2 lädst, um den Pico auf den Werkszustand zurückzusetzen (siehe den Abschnitt *„Laden der Firmware“* oben).

## Stromverbrauch

Der Stromverbrauch des hängt von der Taktrate ab, bei der Standardtaktrate (200 MHz für den RP2040 und 150 MHz für den RP2350) liegt der typische Stromverbrauch jedoch bei 25 mA. Darin sind keine Ströme enthalten, die von den I/O-Pins oder dem 3V3-Pin zugeführt oder abgeleitet werden:

Der Stromverbrauch der WebMite-Version für den Raspberry Pi Pico W beträgt bei deaktiviertem WLAN ebenfalls 20 mA, bei aktiviertem WLAN steigt er jedoch auf 40 bis 70 mA.

# Verwendung von MMBasic

## Befehle und Programmeingabe

An der Befehlszeile kannst du einen Befehl eingeben, der dann sofort ausgeführt wird. Meistens wirst du dies tun, um der PicoMite-Firmware zu sagen, dass sie etwas tun soll, wie zum Beispiel ein Programm ausführen oder eine Option einstellen. Diese Funktion ermöglicht es dir aber auch, Befehle an der Befehlszeile auszuprobieren.

Um ein Programm einzugeben, ist es am einfachsten, den Befehl EDIT zu verwenden. Dadurch wird der Vollbild-Programmeditor aufgerufen, der in die PicoMite-Firmware integriert ist und später in diesem Handbuch beschrieben wird. Er enthält erweiterte Funktionen wie Suchen und Kopieren sowie Ausschneiden und Einfügen in die bzw. aus der Zwischenablage.

Du kannst das Programm auch auf deinem Desktop-Computer mit einem Programm wie Notepad erstellen und es dann über die XModem- oder YModem-Protokolle (siehe den Befehl XMODEM oder YMODEM) oder durch Streaming über die serielle Konsolenverbindung (siehe den Befehl AUTOSAVE) auf den Raspberry Pi Pico übertragen.

Eine dritte und bequeme Methode zum Schreiben und Debuggen eines Programms ist die Verwendung von MMEdit. Dabei handelt es sich um ein Programm, das auf deinem Windows-Computer läuft und es dir ermöglicht, dein Programm auf dem Computer zu bearbeiten und es dann mit einem einzigen Mausklick auf den PicoMite zu übertragen. MMEdit wurde von Jim Hiley geschrieben und kann kostenlos heruntergeladen und genutzt werden. Weitere Informationen findest du unter: https://geoffg.net/mmedit.html

Eine Sache, die du nicht tun kannst, ist die alte BASIC-Methode zur Eingabe eines Programms zu verwenden, bei der jeder Zeile eine Zeilennummer vorangestellt wurde. Zeilennummern sind in MMBasic optional, du kannst sie also weiterhin verwenden, wenn du möchtest, aber wenn du eine Zeile mit einer Zeilennummer an der Eingabeaufforderung eingibst, führt MMBasic sie einfach sofort aus.

## Programmstruktur

Ein BASIC-Programm beginnt in der ersten Zeile und läuft so lange, bis es das Ende des Programms erreicht oder auf einen END-Befehl stößt – an diesem Punkt zeigt MMBasic die Eingabeaufforderung (>) auf der Konsole an und wartet auf eine Eingabe.

Ein Programm besteht aus einer Reihe von Anweisungen oder Befehlen, von denen jede den BASIC-Interpreter dazu veranlasst, etwas auszuführen (die Begriffe „Anweisung“ und „Befehl“ bedeuten im Allgemeinen dasselbe und werden synonym verwendet). Normalerweise steht jede Anweisung in einer eigenen Zeile, aber du kannst auch mehrere Anweisungen in einer Zeile haben, die durch das Doppelpunktzeichen (:) getrennt sind. Zum Beispiel.

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

Anhang J (*Programmieren in BASIC – Ein Tutorial*) am Ende dieses Handbuchs bietet ein umfassendes Tutorial zur Sprache, das dich in einem leicht verständlichen Format mit vielen Beispielen durch die Grundlagen führt.

## Bearbeiten der Befehlszeile

Wenn du eine Zeile an der Befehlszeile eingibst, kannst du sie bearbeiten, indem du mit den Pfeiltasten nach links und rechts innerhalb der Zeile navigierst, mit der Entf-Taste ein Zeichen löschst, mit der Rücktaste vor dem Cursor löschst und mit der Einfügen-Taste zwischen Einfüge- und Überschreibmodus wechselst. Mit Home/End springst du an den Anfang/das Ende der Zeile, und durch zweimaliges Drücken von Home beendest du die Bearbeitung. Zu jedem Zeitpunkt sendet die Enter-Taste die Zeile an MMBasic, das sie ausführt.

Mit den Aufwärts- und Abwärtspfeiltasten kannst du durch den Verlauf zuvor eingegebener Befehlszeilen blättern, die bearbeitet und wiederverwendet werden können.

## Tastenkombinationen

Die Funktionstasten der Tastatur, die für die Konsole verwendet werden, können an der Eingabeaufforderung genutzt werden, um gängige Befehle automatisch einzugeben. Diese Funktionstasten fügen den Text ein, gefolgt von der Eingabetaste, sodass der Befehl sofort ausgeführt wird:

| | |
|---|---|
| F2 | RUN |
| F3 | LIST |

F4 EDIT
F5 Sendet eine ESC-Sequenz, um den VT100-Bildschirm zu löschen. Löscht auch die Konsole.
F10 AUTOSAVE
F11 XMODEM EMPFANGEN
F12 XMODEM SENDEN

Die Funktionstasten F1 sowie F5 bis F9 können mit benutzerdefiniertem Text belegt werden. Siehe den Befehl OPTION FNKey.

## Ein laufendes Programm unterbrechen

Ein Programm wird mit dem Befehl RUN gestartet. Du kannst MMBasic und das laufende Programm jederzeit unterbrechen, indem du STRG-C in die Konsoleneingabe eingibst; MMBasic kehrt dann zur Befehlszeile zurück.

## Optionen einstellen

Viele Optionen können mit Befehlen eingestellt werden, die mit dem Schlüsselwort OPTION beginnen. Sie sind in einem eigenen Abschnitt dieses Handbuchs aufgeführt. Bei einigen Firmware-Versionen kannst du beispielsweise die CPU-Taktfrequenz mit dem Befehl ändern:

```
OPTION CPUSPEED speed
```

## Gespeicherte Variablen

Oftmals besteht die Notwendigkeit, Daten zu speichern, die bei Wiederherstellung der Stromversorgung wiederhergestellt werden können. Zum Beispiel Programmoptionen, Kalibrierungseinstellungen usw. Dies kann mit dem Befehl VAR SAVE erfolgen, der die in seiner Befehlszeile aufgeführten Variablen in einem nichtflüchtigen Flash-Speicher speichert. Der für gespeicherte Variablen reservierte Speicherplatz 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 das Programm bereitstehen.

Diese Funktion ist zum Speichern von Kalibrierungsdaten, vom Benutzer gewählten Optionen und anderen Elementen gedacht, die sich nur selten ändern. Sie sollte nicht für schnelle Speichervorgänge verwendet werden, da du sonst den Flash-Speicher verschleißen könntest. Der für den Raspberry Pi Pico verwendete Flash-Speicher hat eine hohe Lebensdauer, diese kann jedoch durch ein Programm überschritten werden, das wiederholt Variablen speichert. Wenn du Daten häufig speichern möchtest, solltest du einen Echtzeituhr-Chip hinzufügen. Die RTC-Befehle können dann verwendet werden, um Daten im batteriegepufferten Speicher der Echtzeituhr zu speichern und abzurufen. Weitere Details findest du unter dem RTC-Befehl.

Gespeicherte Variablen werden gelöscht, wenn ein neues Programm geladen wird.

## Watchdog-Timer

Eine der Anwendungsmöglichkeiten für den Raspberry Pi Pico ist der Einsatz als eingebetteter Controller. Er kann in MMBasic programmiert werden, und wenn das Programm debuggt und einsatzbereit ist, kann die Konfigurationseinstellung OPTION AUTORUN aktiviert werden. Das Modul führt dann sein Programm automatisch aus, sobald es mit Strom versorgt wird, und fungiert als benutzerdefinierte Schaltung, die eine bestimmte Aufgabe ausführt. Der Benutzer muss nichts darüber wissen, was im Inneren abläuft.

Es besteht jedoch die Möglichkeit, dass ein Fehler im Programm dazu führt, dass MMBasic einen Fehler ausgibt und zur Eingabeaufforderung zurückkehrt. Das wäre in einer eingebetteten Umgebung wenig nützlich, da das Gerät nicht an eine Konsole angeschlossen 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 nicht mehr laufen, bis der Strom aus- und wieder eingeschaltet wird.

Um dem vorzubeugen, kann der Watchdog-Timer verwendet werden. Das ist ein Timer, der bis Null herunterzählt, und wenn er Null erreicht, werden die Prozessoren automatisch neu gestartet (genau wie beim ersten Einschalten). Das passiert auch dann, wenn MMBasic gerade an der Befehlszeile lief. Nach dem Neustart wird die automatische Variable MM.WATCHDOG auf „true“ gesetzt, um anzuzeigen, dass der Neustart durch ein Watchdog-Timeout verursacht wurde.

Der Befehl WATCHDOG sollte an strategischen Stellen im Programm platziert werden, um den Timer immer wieder zurückzusetzen und so zu verhindern, dass er bis Null herunterzählt. Tritt dann ein Fehler auf, wird der Timer nicht zurückgesetzt, er zählt bis Null herunter und das Programm wird neu gestartet (vorausgesetzt, die Option AUTORUN ist gesetzt).

## PIN-Sicherheit

Manchmal ist es wichtig, die Daten und das Programm in einem eingebetteten Controller vertraulich zu behandeln. In der PicoMite-Firmware kann dies mit dem Befehl OPTION PIN erreicht werden. Dieser Befehl legt eine PIN-Nummer fest (die im Flash-Speicher abgelegt wird), und wann immer die PicoMite-Firmware (aus welchem Grund auch immer) zur Befehlszeile zurückkehrt, wird der Benutzer an der Konsole aufgefordert, die PIN-Nummer einzugeben. Ohne die richtige PIN gelangt der Benutzer nicht zur Befehlszeile und hat nur die Möglichkeit, die richtige PIN einzugeben oder die PicoMite-Firmware neu zu starten. Nach dem Neustart benötigt der Benutzer weiterhin die richtige PIN, um auf die Befehlszeile zuzugreifen.

Da ein Eindringling die Befehlszeile nicht erreichen kann, kann er kein Programm auflisten oder kopieren, er kann das Programm nicht ändern und auch keine Aspekte von MMBasic oder der PicoMite-Firmware verändern. Einmal festgelegt, kann die PIN nur durch Eingabe der ursprünglich festgelegten korrekten PIN entfernt werden. Wenn die Nummer verloren geht, besteht die einzige Möglichkeit zur Wiederherstellung darin, die PicoMite-Firmware neu zu laden (wodurch das Programm und alle Optionen gelöscht werden).

Es gibt andere zeitaufwändige Möglichkeiten, auf die Daten zuzugreifen (z. B. die Untersuchung des Flash-Speichers mit einem Programmiergerät), daher sollte dies nicht als ultimative Sicherheit angesehen werden, aber es wirkt als erhebliche Abschreckung.

## Die Library-

Mit der LIBRARY-Funktion kannst du BASIC-Funktionen, Unterprogramme und eingebettete Schriftarten erstellen und sie zu MMBasic hinzufügen, um sie dauerhaft und zu einem Teil der Sprache zu machen. Du hast vielleicht zum Beispiel eine Reihe von Unterprogrammen und Funktionen geschrieben, die komplexe Bitmanipulationen durchführen; diese könnten als Bibliothek gespeichert werden, Teil von MMBasic werden und genauso funktionieren wie andere integrierte Funktionen, die bereits Teil der Sprache sind. Eine eingebettete Schriftart kann auf die gleiche Weise hinzugefügt und wie eine normale Schriftart verwendet werden.

Um Komponenten in die Bibliothek zu installieren, musst du die Routinen schreiben und testen, wie du es bei normalen BASIC-Routinen tun würdest. Wenn sie korrekt funktionieren, kannst du den Befehl LIBRARY SAVE verwenden. Dadurch werden die Routinen (so viele, wie du möchtest) in einen nicht sichtbaren Teil des Flash-Speichers übertragen, wo sie für jedes BASIC-Programm verfügbar sind, aber nicht angezeigt werden, wenn der Befehl LIST verwendet wird, und nicht gelöscht werden, wenn ein neues Programm geladen oder NEW verwendet wird. Die gespeicherten Unterprogramme und Funktionen können jedoch aus dem Hauptprogramm heraus aufgerufen und sogar an der Befehlszeile ausgeführt werden (genau wie ein integrierter Befehl oder eine integrierte Funktion).

Einige Punkte, die du beachten solltest:

- Bibliotheksroutinen verhalten sich genau wie normaler BASIC-Code und können aus einer beliebigen Anzahl von Unterprogrammen, Funktionen, eingebetteten C-Routinen und Schriftarten bestehen. Der einzige Unterschied besteht darin, dass sie bei der Auflistung eines Programms nicht angezeigt werden und beim Laden eines neuen Programms nicht gelöscht werden.
- Bibliotheksroutinen können globale Variablen erstellen und darauf zugreifen und unterliegen denselben Regeln wie das Hauptprogramm – zum Beispiel müssen sie OPTION EXPLICIT beachten, wenn diese Option gesetzt ist.
- Wenn die Routinen in die Bibliothek übertragen werden, komprimiert MMBasic sie, indem Kommentare, überflüssige Leerzeichen, Leerzeilen und die Hex-Codes in eingebetteten C-Routinen und Schriftarten entfernt werden. Dadurch wird der Speicherplatz in der Bibliothek effizient genutzt, insbesondere beim Laden großer Schriftarten. Nach dem Speichern wird der Programmbereich gelöscht.
- Du kannst den Befehl LIBRARY SAVE mehrmals verwenden. Bei jedem Speichern wird der neue Inhalt des Programmbereichs an den bereits vorhandenen Code in der Bibliothek angehängt.
- Du kannst Zeilennummern in der Bibliothek verwenden, aber du kannst keine Zeilennummer in einer ansonsten leeren Zeile als Ziel für ein GOTO usw. verwenden. Das liegt daran, dass der Befehl LIBRARY SAVE alle Leerzeilen entfernt.
- Du kannst READ-Befehle in der Bibliothek verwenden, aber standardmäßig lesen sie DATA-Anweisungen im Hauptprogrammspeicher. Wenn du DATA-Anweisungen in der Bibliothek lesen möchtest, musst du vor dem ersten READ-Befehl den Befehl RESTORE verwenden. Dadurch wird der Zeiger auf den Bibliotheksspeicher zurückgesetzt.

- Die Bibliothek wird im Programm-Flash-Speicher-Slot 3 gespeichert, der bei Verwendung von LIBRARY SAVE nicht mehr zum Speichern eines Programms zur Verfügung steht.
- Du kannst den Inhalt der Bibliothek mit dem Befehl LIBRARY LIST anzeigen, der den Inhalt des Bibliotheksbereichs auflistet.
- Der Inhalt der Bibliothek kann mit LIBRARY DISK SAVE fname$ auf Diskette gespeichert und mit LIBRARY DISK LOAD fname$ wiederhergestellt werden

Um die Routinen im Bibliotheksspeicher zu löschen, verwendest du den Befehl LIBRARY DELETE. Dadurch wird der Speicherplatz gelöscht und der von der Bibliothek belegte Flash-Slot 3 wieder für die Speicherung normaler Programme verfügbar gemacht. Die einzige andere Möglichkeit, eine Bibliothek zu löschen, ist die Verwendung von OPTION RESET.

## Programminitialisierung

Die Bibliothek kann auch Code enthalten, der nicht in einer Subroutine oder Funktion enthalten ist. Dieser Code (falls vorhanden) wird automatisch ausgeführt, bevor ein Programm gestartet wird (d. h. über den Befehl RUN). Diese Funktion kann verwendet werden, um Konstanten zu initialisieren oder MMBasic auf irgendeine Weise einzurichten. Wenn du beispielsweise einige Konstanten festlegen möchtest, könntest du die folgenden Zeilen in den Bibliothekscode einfügen:

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

Im Grunde genommen wurden die Bezeichner TRUE und FALSE der Sprache hinzugefügt und stehen jedem Programm zur Verfügung, das ausgeführt wird.

## MM.STARTUP-

Möglicherweise musst du beim ersten Einschalten etwas Code ausführen, etwa um Hardware zu initialisieren, Optionen festzulegen oder ein benutzerdefiniertes Startbanner anzuzeigen. Dies kannst du erreichen, indem du eine Subroutine mit dem Namen MM.STARTUP erstellst. Wenn die PicoMite-Firmware zum ersten Mal eingeschaltet oder zurückgesetzt wird, sucht sie nach dieser Subroutine und führt sie, falls gefunden, einmal aus.

Wenn am Raspberry Pi Pico beispielsweise eine Echtzeituhr angeschlossen ist, könnte das Programm den folgenden Code enthalten:

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

Dadurch wird die interne Uhr in MMBasic bei jedem Einschalten oder Zurücksetzen auf die aktuelle Uhrzeit eingestellt.

Nachdem der Code in MM.STARTUP ausgeführt wurde, fährt MMBasic mit der Ausführung des restlichen Programms im Programmspeicher fort. Wenn kein weiterer Code vorhanden ist, kehrt MMBasic zur Eingabeaufforderung zurück.

Beachte, dass du MM.STARTUP nicht für allgemeine Einstellungen von MMBasic (wie das Dimensionieren von Arrays, das Öffnen von Kommunikationskanälen usw.) vor dem Ausführen eines Programms verwenden solltest. Der Grund dafür ist, dass MMBasic beim Ausführen des Befehls RUN zunächst den Zustand des Interpreters löscht, um einen Neuanfang zu ermöglichen.

## MM.PROMPT

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

Beachte, dass MMBasic alle Variablen und I/O-Pin-Einstellungen löscht, wenn ein Programm ausgeführt wird; daher gelten alle in dieser Subroutine festgelegten Einstellungen nur für Befehle, die an der Befehlszeile eingegeben werden (d. h. im Immediate-Modus).

Als Beispiel zeigt das Folgende eine benutzerdefinierte Eingabeaufforderung an:

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

Beachte, dass Konstanten zwar definiert werden können, aber nicht sichtbar sind, da eine innerhalb einer Subroutine definierte Konstante lokal auf diese Subroutine beschränkt ist. Mit DIM werden jedoch globale Variablen erstellt, die stattdessen verwendet werden sollten.

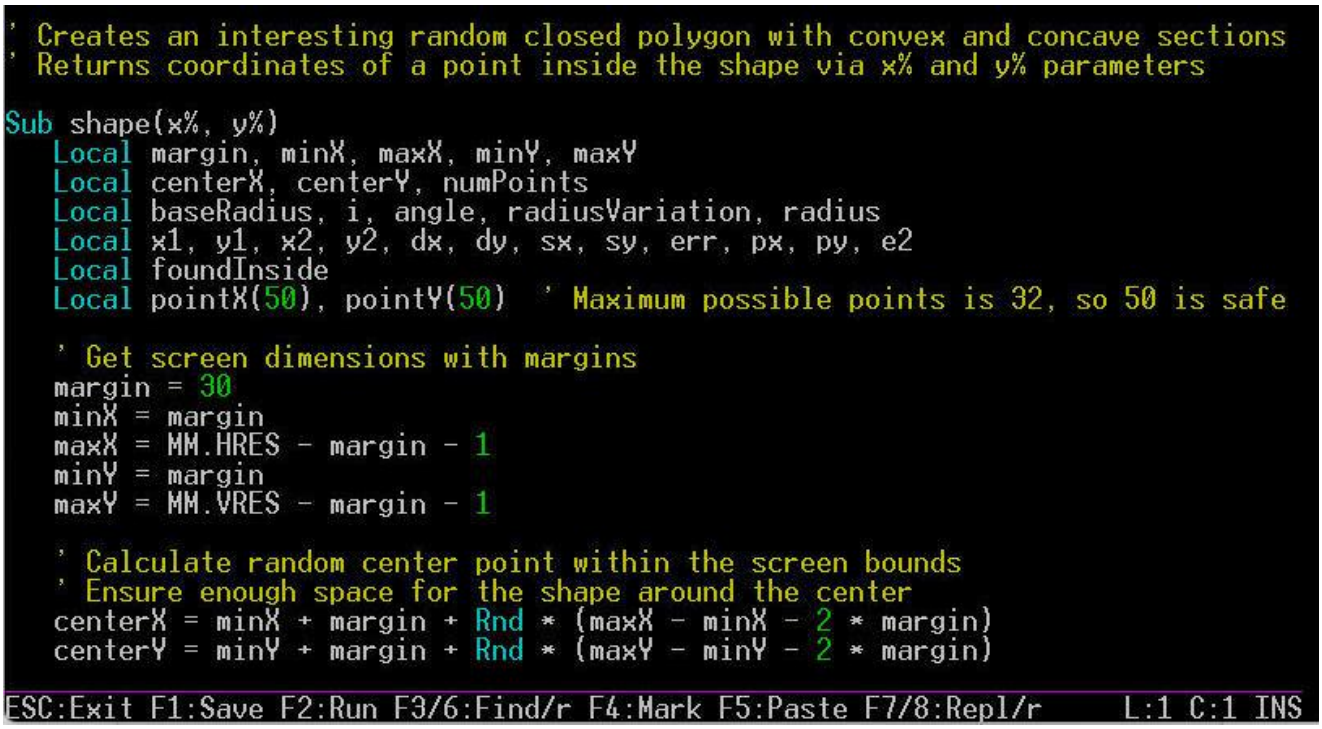
## MM.END

Wenn im Programm eine Subroutine namens MM.END vorhanden ist, wird sie immer dann ausgeführt, wenn das Programm mit einem tatsächlichen oder impliziten END-Befehl endet. Sie wird nicht ausgeführt, wenn das Programm mit Strg-C beendet wird.

Der optionale Parameter „noend“ des END-Befehls kann verwendet werden, um die Ausführung der Subroutine MM.END bei Bedarf zu blockieren (weitere Informationen findest du beim END-Befehl),

# Vollbild-Editor

Eine wichtige Funktion zur Steigerung der Produktivität ist der integrierte Vollbild-Editor. Im Betrieb sieht er so aus:




Der Editor arbeitet mit:

- der seriellen Konsole unter Verwendung eines VT100-kompatiblen Terminalemulators (wie z. B. Tera Term) auf allen Versionen.
- Der VGA- oder HDMI-Videoausgang (bei Versionen mit dieser Funktion).
- Ein angeschlossenes LCD-Panel, das mit OPTION LCDPANEL CONSOLE konfiguriert wurde.

Der Editor ist eine sehr produktive Methode zum Schreiben eines Programms. Mit dem Befehl EDIT kannst du dein Programm eingeben und bearbeiten; durch Drücken der Taste F2 kannst du das Programm dann speichern und ausführen. Wenn dein Programm mit einem Fehler abstürzt, lädt das Drücken der Funktionstaste F4 an der Eingabeaufforderung den Editor und positioniert den Cursor auf der Zeile, die den Fehler verursacht hat. Dieser Zyklus aus Bearbeiten/Ausführen/Bearbeiten ist sehr schnell.

Wenn du zuvor schon einmal einen Editor wie den Windows-Notizblock verwendet hast, wirst du feststellen, dass die Bedienung dieses Editors vertraut ist. Mit den Pfeiltasten bewegst du den Cursor im Text, mit Home und Ende gelangst du an den Anfang bzw. das Ende der Zeile. Die Tasten „Bild auf“ und „Bild ab“ tun genau das, was ihre Bezeichnungen vermuten lassen. Die Entf-Taste löscht das Zeichen an der Cursorposition und die Rücktaste löscht das Zeichen vor dem Cursor. Die Einfügen-Taste wechselt zwischen Einfüge- und Überschreibmodus. Die einzige ungewöhnliche Tastenkombination ist, dass zwei Mal Drücken der Home-Taste dich zum Anfang des Programms und zwei Mal Drücken der End-Taste dich zum Ende des Programms bringt.

Der beste Weg, um den Editor zu erlernen, ist, ihn einfach zu starten und damit zu experimentieren.

## Bearbeitungsfunktionen

Wenn der Editor startet, befindet sich der Cursor an der letzten Bearbeitungsposition oder, falls dein Programm durch einen Fehler angehalten wurde, an der Zeile, die den Fehler verursacht hat. Am unteren Bildschirmrand listet die Statuszeile Details wie die aktuelle Cursorposition und die vom Editor unterstützten Standardfunktionen auf.

Im Einzelnen sind dies folgende Funktionen:

| | |
|---|---|
| ESC | Dadurch verwirft der Editor alle Änderungen und kehrt zur Eingabeaufforderung zurück, ohne den Programmspeicher zu verändern. Wenn du den Text geändert hast, wirst du gefragt, ob du deine Änderungen wirklich verwerfen möchtest. |
| F1: SAVE | Damit wird im Programmspeicher gespeichert und zur Eingabeaufforderung zurückgekehrt. |
| F2: RUN | Dadurch wird die Datei im Programmspeicher gespeichert und sofort ausgeführt. |
| F3: FIND | Hier wirst du aufgefordert, den Text einzugeben, nach dem du suchen möchtest. Wenn du die Eingabetaste drückst, springt der Cursor an den Anfang des ersten gefundenen Eintrags. |
| F4: MARK | Dies wird weiter unten ausführlich beschrieben. |
| F5: PASTE | Hiermit wird (an der aktuellen Cursorposition) der zuvor ausgeschnittene oder kopierte Text eingefügt (siehe unten). |
| F6: FIND AGAIN | Nachdem du die Suchfunktion verwendet hast, kannst du die Suche durch Drücken von F6 wiederholen (Umschalt-F3 funktioniert ebenfalls). |
| F7: REPLACE | Befindet sich der Cursor nach F3 oder F6, öffnet sich ein Dialogfeld, in dem du direkt eine Zeichenfolge in den Einfügepuffer eingeben kannst; durch Drücken der Eingabetaste wird die gesuchte Zeichenfolge ersetzt |
| F8: REPLACE AGAIN | Wenn der Cursor nach F3 oder F6 positioniert ist, ersetzt das System die gesuchte Zeichenfolge durch den Inhalt des Einfügepuffers. |
| F9: INSERT | Du wirst zur Eingabe eines Dateinamens aufgefordert, und wenn du die Eingabetaste drückst, wird diese Datei an der aktuellen Cursorposition eingefügt. |
| F10: SAVE CLIPBOARD | Du wirst zur Eingabe eines Dateinamens aufgefordert, und wenn du die Eingabetaste drückst, wird der Inhalt der Zwischenablage (falls vorhanden) in dieser Datei gespeichert. Siehe den Markierungsmodus weiter unten. |

## Markierungsmodus

Durch Drücken der Markierungstaste (F4) wechselt der Editor in den Markierungsmodus, in dem du Text ausschneiden oder in die Zwischenablage kopieren sowie Text löschen oder speichern kannst.

Im Markierungsmodus dienen die Pfeiltasten dazu, den Text zu markieren, der dann in inverser Darstellung hervorgehoben wird. Wenn du den Anfang oder das Ende des Bildschirms erreichst, markieren die Pfeiltasten den Text weiter, während der Bildschirm scrollt. Dies funktioniert auch mit den Tasten „Bild auf“ und „Bild ab“.

Im Markierungsmodus ändert sich die Statuszeile und zeigt die in diesem Modus verfügbaren Funktionen an. Das sind:

| | |
|---|---|
| ESC | Markierungsmodus verlassen, ohne etwas zu ändern. |
| F4: CUT | Kopiere den markierten Text in die Zwischenablage und entferne ihn aus dem Programm. |
| F5: COPY | Kopiere den markierten Text einfach in die Zwischenablage. |
| F10: SAVE | Frage nach einem Dateinamen und speichere den markierten Text in der Datei, sobald die Eingabetaste gedrückt wird. |
| DELETE | Lösche den markierten Text, ohne die Zwischenablage zu verändern. |

Das Ausschneiden oder Kopieren ist auf maximal 2048 Zeichen begrenzt.

## Alternative Tasten

Du kannst anstelle der oben aufgeführten Funktionstasten auch Tastenkombinationen verwenden. Diese Tastenkombinationen lauten:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| LEFT | Ctrl-S | RIGHT | Ctrl-D | UP | Ctrl-E | DOWN | Ctrl-X |
| HOME | Ctrl-U | END | Ctrl-K | PageUp | Ctrl-P | PageDn | Ctrl-L |
| DEL | Ctrl-] | INSERT | Ctrl-N | F1 | Ctrl-Q | F2 | Ctrl-W |
| F3 | Ctrl-R | F4 | Ctrl-T | F5 | Ctrl-Y | F6 | Ctrl-G |
| F7 | Ctrl-F | F8 | Ctrl-I | | | | |

## Lange Zeilen

Bei langen Zeilen wird nur der erste Teil der Zeile bis zum rechten Rand des Bildschirms angezeigt. Der Rest der Zeile, der über den rechten Rand hinausgeht, ist zwar noch vorhanden, wird aber nicht angezeigt und kann nicht bearbeitet werden. Wenn du eine sehr lange Zeile bearbeiten möchtest, kannst du den Cursor nahe am rechten Rand positionieren und die Eingabetaste drücken. Dadurch wird die lange Zeile in zwei Teile geteilt, die beide separat bearbeitet werden können. Um die Zeile wieder zusammenzufügen, entferne den zuvor eingegebenen Zeilenumbruch mit der Entf- oder Rücktaste.

Alternativ kannst du vor dem Aufrufen des Editors die Fortsetzungszeilen aktivieren (OPTION CONTINUATION LINES ON). Dadurch kannst du am Ende einer Zeile ein Leerzeichen gefolgt von einem Unterstrich verwenden, um anzugeben, dass die nächste Zeile eine Fortsetzung ist und zusammengefügt werden muss, damit das Programm läuft. Die Zusammenführung erfolgt beim Speichern der Datei, und beim erneuten Bearbeiten werden lange Zeilen automatisch aufgeteilt, sobald die Datei in den Editor geladen wird. Die Zeilenumbrüche befinden sich möglicherweise nicht an derselben Stelle, aber der Editor versucht, sie an sinnvollen Positionen zu setzen (am Ende von Wörtern usw.). Es gibt keine Einschränkungen, wo Fortsetzungszeichen platziert werden können. Sie können sich beispielsweise in der Mitte einer in Anführungszeichen gesetzten Zeichenkette befinden. Die Begrenzung auf maximal 255 Zeichen in einer verketteten Zeile gilt weiterhin, und der Editor lässt dich nicht beenden, wenn eine Zeile zu lang ist.

## Verwendung einer Maus

Versionen der PicoMite-Firmware, die VGA/HDMI unterstützen (sowohl in der RP2040- als auch in der RP2350-Version), unterstützen auch die Verwendung einer PS2- oder USB-Maus im Editor. Einzelheiten zum Anschließen einer Maus findest du unter der Überschrift *„Tastatur/Maus/Gamepad“* weiter hinten in diesem Handbuch.

Wenn du den Editor mit angeschlossener Maus startest und dich im Video-MODUS 1 mit aktivierter Farbcodierung befindest, siehst du ein rot hervorgehobenes Zeichen auf weißem Hintergrund. Diese Markierung kann mit der Maus verschoben werden. Ein Linksklick mit der Maus bewegt den Bearbeitungscursor an diese Position (d. h. genau wie bei Verwendung der Cursortasten). Ein Rechtsklick mit der Maus entspricht dem Drücken der Taste F4 auf der Tastatur, und ein Klick auf den Scrollrad entspricht F5.

Das bedeutet, dass du im Normalmodus des Editors den Mauszeiger positionieren kannst und durch einen Rechtsklick den Markierungsmodus (Ausschneiden und Kopieren) aktivierst, wobei der Cursor an der Position des Mauszeigers beginnt. Wenn du dann die Maus bewegst und anschließend mit der linken Maustaste klickst, werden die Zeichen von der Markierungsposition bis zur neuen Mausposition markiert. Ein Rechtsklick (entspricht F4) schneidet den markierten Bereich in die Zwischenablage, während ein Klick auf das Scrollrad (entspricht F5) den markierten Text in die Zwischenablage kopiert, ohne ihn aus dem Text zu löschen. Beide Funktionen versetzen den Editor wieder in den Normalmodus.

Im Normalmodus kannst du den Inhalt der Zwischenablage in den Text einfügen, indem du die Maus an die neue Position bewegst und auf das Scrollrad klickst (entspricht F5).

## Farbcodierte Editoranzeige

Der Editor kann das bearbeitete Programm farblich kennzeichnen, wobei Schlüsselwörter, Zahlen und Kommentare in verschiedenen Farben angezeigt werden. Diese Funktion kann an der Befehlszeile mit den folgenden Befehlen ein- oder ausgeschaltet werden:

```
   OPTION COLOURCODE ON
```

oder

```
   OPTION COLOURCODE OFF
```

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

# Variablen und Ausdrücke

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

## Variablen

Variablen können mit einem Buchstaben oder einem Unterstrich beginnen und beliebige Buchstaben, Ziffern, den Punkt (.) sowie den Unterstrich (_) enthalten. Sie dürfen bis zu 31 Zeichen lang sein.

Ein Variablenname oder eine Beschriftung darf nicht mit einem Befehl, einer Funktion, einer der neun PIO-Anweisungen (IN, OUT, JMP, WAIT, PUSH, PULL, MOV, IRQ, SET) oder einem der folgenden Schlüsselwörter übereinstimmen: THEN, ELSE, GOTO, GOSUB, TO, STEP, FOR, WHILE, UNTIL, LOAD, MOD, NOT, AND, OR, XOR, AS.
Beispiel: step = 5 ist ungültig, da STEP ein Schlüsselwort ist.

MMBasic unterstützt drei Arten von Variablen:

1. Gleitkommazahlen mit doppelter Genauigkeit.
   Diese können eine Zahl mit Dezimalpunkt und Bruchteil speichern (z. B. 45,386), verlieren jedoch an Genauigkeit, wenn mehr als 14 Stellen verwendet werden. Gleitkommavariablen werden durch Hinzufügen des Suffixes „!“ zum Variablennamen angegeben (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 Bruchteile (d. h. den Teil nach dem Dezimalpunkt) speichern. Sie werden durch Hinzufügen des Suffixes „%“ zum Namen einer Variablen angegeben. Zum Beispiel i%, nbr% usw.
3. Eine Zeichenkette.
   Ein String speichert eine Folge von Zeichen (z. B. „Tom“). Jedes Zeichen im String wird als 8-Bit-Zahl gespeichert und kann daher einen Dezimalwert von 0 bis 255 haben. Die Namen von String-Variablen enden mit dem Symbol „$“ (z. B. name$, s$ usw.). Strings können bis zu 255 Zeichen lang sein.

Beachte, dass es nicht zulässig ist, denselben Variablennamen für verschiedene Typen zu verwenden. Die Verwendung von `nbr!` und `nbr%` im selben Programm würde beispielsweise einen Fehler verursachen.

Die meisten Programme verwenden Gleitkommavariablen für arithmetische Operationen, da diese die in typischen Situationen verwendeten Zahlen verarbeiten können und bei Divisionen und Brüchen intuitiver sind als Ganzzahlen. Wenn dich die Details also nicht interessieren, verwende immer Gleitkommazahlen.

## Konstanten

Numerische Konstanten können mit einer Ziffer (0–9) für eine dezimale Konstante, mit &H für eine hexadezimale Konstante, mit &O für eine oktale Konstante oder mit &B für eine binäre Konstante beginnen. Beispielsweise entspricht &B1000 der dezimalen Konstante 8. Konstanten, die mit &H, &O oder &B beginnen, werden immer als 64-Bit-Konstanten vom Typ „unsigned integer“ behandelt.

Dezimalen können ein Minus (-) oder ein Plus (+) vorangestellt werden und mit einem „E“ gefolgt von einer Exponentenzahl enden, um die Exponentialschreibweise zu kennzeichnen. Beispielsweise entspricht 1.6E+4 dem Wert 16000.

Wenn eine konstante Zahl verwendet wird, wird davon ausgegangen, dass es sich um eine Ganzzahl handelt, sofern kein Dezimalpunkt oder Exponent verwendet wird. Beispielsweise wird 1234 als Ganzzahl interpretiert, während 1234.0 als Gleitkommazahl interpretiert wird.

Zeichenfolgenkonstanten müssen in doppelte Anführungszeichen (") gesetzt werden. Z. B. „Hello World“.

## OPTION DEFAULT

Eine Variable kann ohne Suffix (d. h. !, % oder $) verwendet werden; in diesem Fall verwendet MMBasic den Standardtyp Gleitkomma. Beispielsweise wird mit dem folgenden Code eine Gleitkommavariable erstellt:

```
Nbr = 1234
```

Der Standardwert kann jedoch mit dem Befehl OPTION DEFAULT geändert werden. Beispielsweise legt OPTION DEFAULT INTEGER fest, dass alle Variablen ohne spezifischen Typ Ganzzahlen sind. Folgendes erstellt also eine Ganzzahlvariable:

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

Der Standardwert kann auf FLOAT (was der Standard bei Programmstart ist), INTEGER, STRING oder NONE gesetzt werden. Im letzteren Fall müssen alle Variablen explizit typisiert werden, andernfalls tritt ein Fehler auf.

Der Befehl OPTION DEFAULT kann an beliebiger Stelle im Programm platziert und jederzeit geändert werden, doch empfiehlt es sich, ihn am Anfang des Programms zu platzieren und unverändert zu lassen.

## OPTION EXPLICIT

Standardmäßig erstellt MMBasic automatisch eine Variable, wenn sie zum ersten Mal referenziert wird. So erstellt `Nbr = 1234` die Variable und setzt sie gleichzeitig auf den Wert 1234. Das ist praktisch für kurze und schnelle Programme, kann aber in großen Programmen zu subtilen und schwer zu findenden Fehlern führen. Zum Beispiel wurde in der dritten Zeile dieses Ausschnitts die Variable `Nbr` fälschlicherweise als `Nbrs` geschrieben. Infolgedessen würde die Variable `Nbrs` mit dem Wert Null erstellt 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 vor ihrer Verwendung explizit mit den Befehlen DIM, LOCAL oder STATIC (siehe unten) definiert werden. Die Verwendung dieses Befehls wird empfohlen, um gute Programmierpraxis zu unterstützen. Wenn er verwendet wird, sollte er am Anfang des Programms stehen, bevor 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 sichtbar ist und verwendet werden kann, auch innerhalb von Unterprogrammen und Funktionen. Wenn du jedoch möchtest, dass die Definition nur innerhalb eines Unterprogramms oder einer Funktion sichtbar ist, solltest du den Befehl LOCAL am Anfang des Unterprogramms oder der Funktion verwenden. LOCAL hat genau dieselbe Syntax wie DIM.

Wird LOCAL verwendet, um eine Variable mit demselben Namen wie eine globale Variable zu definieren, wird die globale Variable für die Unterroutine oder Funktion ausgeblendet, und alle Verweise auf die Variable beziehen sich nur auf die durch den Befehl LOCAL definierte Variable. Jede durch LOCAL erstellte Variable verschwindet, sobald das Programm die Unterroutine verlässt.

Im einfachsten Fall können DIM und LOCAL verwendet werden, um eine oder mehrere Variablen basierend auf ihrem Typ-Suffix oder der zu diesem Zeitpunkt geltenden OPTION DEFAULT zu definieren. Zum Beispiel:

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

Sie können 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 Ganzzahlen angelegt. Wenn du die Variable innerhalb eines Programms verwendest, musst du das Typ-Suffix nicht angeben. Zum Beispiel funktioniert `MyStr` im folgenden Beispiel perfekt als Zeichenfolgenvariable:

```
DIM STRING MyStr
MyStr = "Hallo"
```

Die Befehle DIM und LOCAL akzeptieren auch die Microsoft-Praxis, den Typ der Variablen nach der Variablen mit dem Schlüsselwort „AS“ anzugeben. Zum Beispiel:

```
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 Liste der Variablen steht).

Die Variablen können auch bei der Definition initialisiert werden. Zum Beispiel:

```
DIM INTEGER a = 5, b = 4, c = 3
DIM s$ = "World", i% = &H8FF8F
```

```
DIM msg AS STRING = "Hallo" + " " + s$
```

Der Wert, mit dem die Variable initialisiert wird, kann ein Ausdruck sein, der benutzerdefinierte Funktionen enthält.

Die Befehle DIM oder LOCAL werden ebenfalls zur Definition eines Arrays verwendet, und alle oben aufgeführten Regeln gelten auch bei der Definition eines Arrays. Du kannst zum Beispiel Folgendes verwenden:

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

Bei der Initialisierung eines Arrays werden die Werte durch Kommas getrennt aufgelistet, wobei die gesamte Liste in Klammern steht. Zum Beispiel:

```
DIM INTEGER nbr(5) = (11, 12, 13, 14, 15, 16)
DIM days(7) AS STRING = ("","So","Mo","Di","Mi","Do","Fr","Sa")
```

## STATIC

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

Dies kannst du mit dem Befehl STATIC erreichen. STATIC kann nur innerhalb einer Subroutine oder Funktion verwendet werden und hat dieselbe Syntax wie LOCAL und DIM. Der Unterschied besteht darin, dass ihr Wert zwischen den Aufrufen der Subroutine oder Funktion beibehalten wird (d. h. sie wird beim zweiten und allen folgenden Aufrufen nicht neu initialisiert).

Wenn du beispielsweise die folgende Subroutine hättest und sie wiederholt aufrufen würdest, 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
```

Beachte, dass die Initialisierung der statischen Variablen auf 5 (wie im obigen Beispiel) nur beim ersten Aufruf der Subroutine wirksam wird. Bei nachfolgenden 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 Floats, Ganzzahlen oder Zeichenketten sowie Arrays davon sein, mit oder ohne Initialisierung. Die Länge des von STATIC erstellten Variablennamens und die Länge des Subroutinen- oder Funktionsnamens dürfen zusammen nicht mehr als 31 Zeichen betragen.

## CONST

Oft ist es sinnvoll, einen Bezeichner zu definieren, der einen Wert repräsentiert, ohne dass die Gefahr besteht, dass dieser Wert versehentlich geändert wird – was passieren kann, wenn Variablen für diesen Zweck verwendet werden (diese Vorgehensweise begünstigt eine weitere Klasse schwer zu findender Fehler).

Mit dem Befehl CONST kannst du einen Bezeichner erstellen, der sich wie eine Variable verhält, aber auf einen Wert gesetzt ist, der nicht geändert werden kann. Zum Beispiel:

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

Die Bezeichner können dann in einem Programm verwendet werden, wo sie für den flüchtigen Leser aussagekräftiger sind als einfache Zahlen. Zum Beispiel:

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

Es können mehrere Konstanten in einer Zeile erstellt werden:

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

Der Wert, der zur Initialisierung der Konstante verwendet wird, wird bei der Erstellung der Konstante ausgewertet und kann ein Ausdruck sein, der benutzerdefinierte Funktionen enthält.

Der Typ der Konstante leitet sich aus dem ihr zugewiesenen Wert ab; so ist beispielsweise die oben genannte `MaxValue` eine Gleitkommakonstante, da 2,4 eine Gleitkommazahl ist. Der Typ einer Konstante kann auch explizit durch Verwendung eines Typ-Suffixes (d. h. !, % oder $) festgelegt werden, muss aber mit dem zugewiesenen Wert übereinstimmen.

## Sonderzeichen in Zeichenketten

Spezielle, nicht druckbare Zeichen können mithilfe des Backslashes (d. h. \) als Escape-Symbol in Zeichenfolgenkonstanten eingefügt werden. Um diese Funktion zu aktivieren, muss der Befehl OPTION ESCAPE am Anfang des Programms stehen.

Dies sind die gültigen Escape-Sequenzen:

| | Hex-Wert | Beschreibung |
|---|---|---|
| \a | 07 | Alarm (Piepton, Glocke) |
| \b | 08 | Rücktaste |
| \e | 1B | Escape-Zeichen |
| \f | 0C | Formfeed-Zeichen Seitenumbruch |
| \n | 0A | Zeilenumbruch (Line Feed) |
| \r | 0D | Wagenrücklauf |
| \q | 22 | Anführungszeichen |
| \t | 09 | Tabulator |
| \v | 0B | Vertikaler Tabulator |
| \\ | 5C | Backslash |
| \nnn | beliebig | Das Byte, dessen Wert durch nnn angegeben wird, interpretiert als Dezimalzahl |
| \&hh | beliebig | Das Byte, dessen Wert durch hh… angegeben wird, interpretiert als Hexadezimalzahl |

Zum Beispiel gibt das Folgende die Wörter „Hello“ und „World“ in getrennten Zeilen aus:

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

## Ausdrücke und Operatoren

MMBasic wertet einen mathematischen Ausdruck nach den üblichen mathematischen Regeln aus. Zum Beispiel werden Multiplikation und Division zuerst durchgeführt, gefolgt von Addition und Subtraktion. Diese Regeln werden als Prioritätsregeln bezeichnet und sind unten näher beschrieben.

Das bedeutet, dass 2 + 3 * 6 das Ergebnis 20 ergibt, ebenso wie 5 * 4 und auch 10 + 4 * 3 – 2.

Wenn du den Interpreter zwingen möchtest, bestimmte Teile des Ausdrucks zuerst auszuwerten, kannst du diesen Teil des Ausdrucks in Klammern setzen. Beispielsweise ergibt (10 + 4) * (3 – 2) das Ergebnis 14 und nicht 20, wie es der Fall gewesen wäre, wenn keine Klammern verwendet worden wären. Die Verwendung von Klammern verlangsamt das Programm nicht nennenswert, daher solltest du sie großzügig einsetzen, wenn die Möglichkeit besteht, dass MMBasic deine Absicht falsch interpretiert.

Die folgenden Operatoren sind in MMBasic implementiert, geordnet nach ihrer Priorität. Operatoren, die auf derselben Ebene stehen (zum Beispiel + und -), werden in der Reihenfolge von links nach rechts verarbeitet, so wie sie in der Programmzeile stehen.

Arithmetische Operatoren:

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

Verschiebungsoperatoren:

| | |
|---|---|
| x << y x >> y | Diese funktionieren auf besondere Weise. << bedeutet, dass der zurückgegebene Wert der Wert von x ist, der um y Bits nach links verschoben wurde, während >> dasselbe bedeutet, nur nach rechts verschoben. Es handelt sich um ganzzahlige Funktionen, und alle verschobenen Bits werden verworfen, während neu hinzugefügte Bits auf Null gesetzt werden. |

Logische Operatoren:

| | |
|---|---|
| NOT INV | invertieren den logischen Wert auf der rechten Seite (z. B. `NOT(a=b)` ist `a<>b`)<br>oder bitweise Inversion des Wertes auf der rechten Seite (z. B. `t a = INV b`) |
| <> < > <= =< >= => | Ungleichheit, kleiner als, größer als, kleiner oder gleich, kleiner oder gleich (alternative Schreibweise), größer oder gleich, größer oder gleich (alternative Schreibweise) |
| = | Gleichheit |
| AND OR XOR | Konjunktion, Disjunktion, Exklusives Oder |

Aus Gründen der Microsoft-Kompatibilität sind die Operatoren AND, OR und XOR bitweise Integer-Operatoren. Zum Beispiel gibt PRINT (3 AND 6) die Zahl 2 aus. Da diese Operatoren sowohl als logische Operatoren (z. B. IF a=5 AND b=8 THEN …) als auch als bitweise Operatoren (z. B. y% = x% AND &B1010) fungieren können, kommt der Interpreter durcheinander, wenn sie im selben Ausdruck gemischt werden. Bewerte logische und bitweise Ausdrücke daher immer in getrennten Ausdrücken.

Die anderen logischen Operationen ergeben die ganze Zahl 0 (Null) für „falsch“ und 1 für „wahr“. Zum Beispiel die Anweisung
PRINT 4 >= 5 gibt die Zahl Null aus, und der Ausdruck A = 3 > 2 speichert +1 in A.

Der NOT-Operator kehrt den logischen Wert zu seiner Rechten um (es handelt sich nicht um eine bitweise Umkehrung), während der INV-Operator eine bitweise Umkehrung durchführt. Beide haben die höchste Priorität, sodass sie fest 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 …
```

Zeichenfolgenoperatoren:

| | |
|---|---|
| + | Zwei Zeichenfolgen verbinden |
| <> < > <= =< >= => | Ungleichheit, kleiner als, größer als, kleiner oder gleich, kleiner oder gleich (alternative Version), größer oder gleich, größer oder gleich (alternative Version) in der Reihenfolge der ASCII-Werte. |
| = | Gleichheit |

Bei Zeichenfolgenvergleichen wird die Groß-/Kleinschreibung beachtet. Zum Beispiel ist „A“ größer als „a“.

## Mischen von Gleitkommazahlen und Ganzzahlen

MMBasic übernimmt automatisch die Umwandlung von Zahlen zwischen Gleitkomma und Ganzzahlen. Wenn eine Operation sowohl Gleitkomma- als auch Ganzzahlen enthält (z. B. `PRINT A% + B!`), wird die Ganzzahl zunächst in eine Gleitkommazahl umgewandelt, dann wird die Operation ausgeführt und eine Gleitkommazahl zurückgegeben. Sind beide Seiten des Operators Ganzzahlen, wird eine Ganzzahloperation durchgeführt und eine Ganzzahl zurückgegeben.

Die einzige Ausnahme bildet die normale Division („/“), bei der immer beide Seiten des Ausdrucks in Gleitkommazahlen umgewandelt werden und anschließend eine Gleitkommazahl zurückgegeben wird. Für die ganzzahlige Division solltest du den Operator für ganzzahlige Division „\“ verwenden.

MMBasic-Funktionen geben je nach ihren Eigenschaften eine Gleitkommazahl oder eine Ganzzahl zurück. Beispielsweise gibt PIN() eine Ganzzahl zurück, wenn der Pin als digitaler Eingang konfiguriert ist, aber eine Gleitkommazahl, wenn er als analoger Eingang konfiguriert ist.

Bei Bedarf kannst du eine Gleitkommazahl mit der Funktion INT() in eine Ganzzahl umwandeln. Es ist nicht notwendig, eine Ganzzahl explizit in eine Gleitkommazahl umzuwandeln, aber falls erforderlich, könnte der ganzzahlige Wert einer Gleitkommavariablen zugewiesen werden, woraufhin er bei der Zuweisung automatisch umgewandelt wird.

## Strukturen

Strukturen (auch als benutzerdefinierte Typen bekannt) ermöglichen es dir, verwandte Variablen unterschiedlicher Typen unter einem einzigen Namen zusammenzufassen. Dies ist nützlich für die Organisation komplexer Daten wie Koordinaten, Datensätze oder beliebiger Sammlungen verwandter Werte. Dieser Abschnitt bietet einen kurzen Überblick, aber Strukturen werden ausführlich in der Datei *MMBasic_Structures_Manual.pdf* beschrieben, die im ZIP-Download der Firmware enthalten ist.

Als Beispiel für eine Struktur nehmen wir an, dass du eine Reihe von Alarmen verfolgen musst. Jeder Alarm hat eine Reihe von Eigenschaften: den Zeitpunkt des Alarms, seine Stufe (1, 2, 3 usw.) und die ergriffene Maßnahme. Durch die Verwendung von Strukturen kannst du die Daten für einen Alarm als ein einziges „Objekt“ behandeln.

Um beispielsweise eine Struktur für einen Alarm zu erstellen, könntest du Folgendes verwenden:

```
TYPE StructAlarm
  time AS INTEGER
  level AS INTEGER
  action AS STRING
END TYPE
```

Die Typen der Elemente können beliebige der normalen Datentypen in MMBasic sein (FLOAT, INTEGER oder STRING). Dieser neue Typ, den du definiert hast, kann dann zur Definition einer Variablen verwendet werden. Du könntest zum Beispiel eine neue Variable namens `alarm` wie folgt definieren:

```
DIM alarm AS StructAlarm
```

Du kannst eine Struktur auch innerhalb einer Subroutine oder Funktion als LOCAL oder STATIC definieren.

Du kannst auf Elemente der Struktur zugreifen, indem du einen Punkt (.) zwischen dem Namen der Variablen und dem Namen des Elements setzt. Zum Beispiel: `alarm.time`.

Du kannst den Elementen der Struktur Werte zuweisen, indem du die normale Zuweisung verwendest. D. h.:

```
alarm.time = EPOCH(NOW)
alarm.level = 3
```

Und du kannst mit derselben Notation auf Elemente der Struktur zugreifen:

```
IF alarm.level > 4 THEN ...
```

Du kannst eine Struktur einer anderen vom gleichen Typ zuweisen:

```
DIM alarm AS StructAlarm
DIM a2 AS StructAlarm
...
alarm = a2
```

Du kannst eine Subroutine oder Funktion aufrufen, indem du eine Struktur als einen oder mehrere ihrer Parameter verwendest:

```
MySub alarm, a2
```

Und eine Funktion kann eine Struktur zurückgeben:

```
FUNCTION MyFun() AS StructAlarm
  MyFun.time = EPOCH(NOW)
  MyFun.level = 1
```

```
END FUNCTION
...
alarm = MyFun()
```

Du kannst ein Array von Strukturen genauso definieren wie jedes normale Array:

```
DIM alarm(nbr) AS StructAlarm
```

Und du kannst ganz einfach auf die Elemente jedes Eintrags im Array zugreifen:

```
IF alarm(i).time < x AND alarm(i).level < y THEN CallSomeSub
```

MMBasic bietet zahlreiche Befehle für die spezielle Bearbeitung von Strukturen, darunter das Speichern und Laden aus einer Datei, das Sortieren eines Struktur-Arrays, das Zurücksetzen der Elemente einer Struktur und vieles mehr. Alle Details findest du in der Datei *MMBasic_Structures_Manual.pdf,* die im ZIP-Download der Firmware enthalten ist.

## 64-Bit-Ganzzahlen ohne Vorzeichen

MMBasic unterstützt 64-Bit-Ganzzahlen mit Vorzeichen. Das bedeutet, dass 63 Bits für die Zahl selbst und ein Bit (das höchstwertige Bit) für das Vorzeichen (positiv oder negativ) verwendet werden. Es ist jedoch möglich, volle 64-Bit-Ganzzahlen ohne Vorzeichen zu verwenden, solange du keine Rechenoperationen mit den Zahlen durchführst.

64-Bit-Zahlen ohne Vorzeichen können durch Voranstellen der Präfixe &H, &O oder &B an eine Zahl erzeugt werden, und diese Zahlen können in einer Integer-Variablen gespeichert werden. Du hast dann eine begrenzte Auswahl an Operationen, die du mit diesen Zahlen durchführen kannst. Das sind << (nach links verschieben), >> (nach rechts verschieben), AND (bitweises Und), OR (bitweises Oder), XOR (bitweises Exklusiv-Oder), INV (bitweise Inversion), = (gleich) und <> (ungleich). Arithmetische Operatoren wie Division oder Addition können durch eine 64-Bit-Zahl ohne Vorzeichen verwirrt werden und unsinnige Ergebnisse liefern.

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

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

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

Zeigt „800F0000FFFF0044“ an

# Unterprogramme und Funktionen

Eine im Programm definierte Subroutine oder Funktion ist einfach ein Block von Programmcode, der in einem Modul enthalten ist und von überall in deinem Programm aufgerufen werden kann. Es ist so, als hättest du der Sprache einen eigenen Befehl oder eine eigene Funktion hinzugefügt.

## Unterprogramme

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

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

Und wenn du die Subroutine aufrufst, kannst du den Argumenten Werte zuweisen. Zum Beispiel:

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

Innerhalb der Subroutine hat `arg1` den Wert `23`, `arg2$` den Wert „`Cat`" und so weiter. Die Argumente verhalten sich wie gewöhnliche Variablen, existieren jedoch nur innerhalb der Subroutine und verschwinden, sobald die Subroutine endet. Du kannst im Hauptprogramm Variablen mit demselben Namen haben; diese werden jedoch von den für die Subroutine definierten Argumenten verdeckt.

Beim Aufruf einer Subroutine kannst du weniger als die erforderliche Anzahl an Werten angeben; in diesem Fall werden die fehlenden Werte entweder als Null oder als leere Zeichenkette angenommen. Du kannst auch einen Wert in der Mitte der Liste weglassen, und es passiert dasselbe. Zum Beispiel:

```
MYSUB  23, , 55
```

Dies führt dazu, dass `arg2$` auf die leere Zeichenkette „ " gesetzt wird .

Anstatt das Typ-Suffix (z. B. das $ in arg2$) zu verwenden, kannst du in der Definition des Unterprogramm-Arguments das Suffix AS <Typ> verwenden; dann wird das Argument als der angegebene Typ erkannt, auch wenn das Suffix nicht verwendet wird. Zum Beispiel:

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

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

## Funktionen

Funktionen ähneln Unterprogrammen, wobei der Hauptunterschied darin besteht, dass eine Funktion dazu dient, einen Wert in einem Ausdruck zurückzugeben. Die Regeln für die Argumentliste in einer Funktion sind ähnlich wie bei Unterprogrammen. Der einzige Unterschied besteht darin, dass beim Aufruf einer Funktion Klammern um die Argumentliste gesetzt werden müssen, auch wenn keine Argumente vorhanden sind (beim Aufruf eines Unterprogramms sind Klammern optional).

Um einen Wert aus der Funktion zurückzugeben, weist du dem Funktionsnamen innerhalb der Funktion einen Wert zu. Wenn der Funktionsname mit einem $, einem % oder einem ! endet, gibt die Funktion diesen Typ zurück, andernfalls gibt sie den Wert zurück, auf den OPTION DEFAULT gesetzt ist. Du kannst den Typ der Funktion auch festlegen, indem du AS <Typ> am Ende der Funktionsdefinition hinzufügst.

Zum Beispiel:

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

## Argumente per Referenz übergeben

Wenn du beim Aufruf einer Subroutine oder einer Funktion eine gewöhnliche Variable (d. h. keinen Ausdruck) als Wert verwendest, verweist das Argument innerhalb der Subroutine/Funktion auf die im Aufruf verwendete Variable zurück, und alle Änderungen am Argument werden auch an der übergebenen Variable vorgenommen. Dies wird als Übergabe von Argumenten per Referenz bezeichnet.

Du könntest beispielsweise eine Subroutine definieren, um zwei Werte wie folgt zu vertauschen:

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

In deinem aufrufenden Programm würdest du für beide Argumente Variablen verwenden:

```
Swap nbr1, nbr2
```

Das Ergebnis ist, dass die Werte von `nbr1` und `nbr2` vertauscht werden.

Damit dies funktioniert, müssen der Typ der übergebenen Variablen (z. B. `nbr1`) und der definierten Argument (z. B. `a`) identisch sein (im obigen Beispiel sind beide standardmäßig vom Typ float).

Wenn du ein Argument als allgemeine Variable innerhalb einer Subroutine oder Funktion verwenden möchtest (d. h. seinen Wert ändern), solltest du seiner Definition das Schlüsselwort BYVAL voranstellen. Dies weist MMBasic an, immer den Wert des Arguments zu verwenden, auch wenn es sich um eine Variable handelt, und niemals auf die im Aufruf verwendete Variable zurückzuverweisen. Der Grund dafür ist, dass ein anderer Nutzer deiner Routine möglicherweise unwissentlich eine Variable in seinem Aufruf verwendet und diese Variable von deiner Routine „auf magische Weise“ verändert werden könnte, wenn du BYVAL nicht verwendest.

Beachte, dass BYVAL nur mit einfachen Variablen funktioniert (nicht mit Arrays oder Strukturen).

## Übergeben von Arrays

Einzelne Elemente eines Arrays können an eine Subroutine oder Funktion übergeben werden und werden dann wie eine normale Variable behandelt. Dies ist zum Beispiel eine gültige Art, die Subroutine „Swap“ (siehe oben) aufzurufen:

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

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

Du kannst auch ein oder mehrere vollständige Arrays an eine Subroutine oder Funktion übergeben, indem du das Array mit leeren Klammern anstelle der normalen Dimensionen angibst. Zum Beispiel `a()`. In der Definition der Subroutine oder Funktion muss der zugehörige Parameter ebenfalls mit leeren Klammern angegeben werden. Der Typ (d. h. float, integer oder string) des übergebenen Arguments und des Parameters in der Definition muss identisch sein.

In der Subroutine oder Funktion übernimmt das Array die Dimensionen des übergebenen Arrays, und diese müssen bei der Indizierung des Arrays beachtet werden. Bei Bedarf können die Dimensionen des Arrays als zusätzliche Argumente an die Subroutine oder Funktion übergeben werden, oder diese können über die Funktion BOUND() ermittelt werden. Das Array wird per Referenz übergeben, was bedeutet, dass alle Änderungen, die innerhalb der Subroutine oder Funktion am Array vorgenommen werden, auch für das übergebene Array gelten.

Wenn du zum Beispiel Folgendes ausführst, wird der Text „Hello World“ ausgegeben:

```
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
```

## Vorzeitiger Abbruch

Es kann nur ein END SUB oder END FUNCTION pro Definition einer Subroutine oder Funktion geben. Um eine Subroutine vorzeitig zu beenden (d. h. bevor der Befehl END SUB erreicht wurde), kannst du den Befehl EXIT SUB verwenden. Dies hat denselben Effekt, als hätte das Programm die Anweisung END SUB erreicht. Ebenso kannst du EXIT FUNCTION verwenden, um eine Funktion vorzeitig zu beenden.

## Rekursion

Rekursion liegt vor, wenn eine Subroutine oder Funktion sich selbst aufruft. Du kannst in MMBasic rekursiv programmieren, aber es gibt dabei einige Einschränkungen (diese sind eine direkte Folge der Beschränkungen von Mikrocontrollern und der BASIC-Sprache):

- Die Rekursionstiefe ist fest begrenzt. In der PicoMite-Firmware sind das 50 Ebenen.
- Wenn du viele Argumente für die Subroutine oder Funktion und viele LOCAL-Variablen (insbesondere Strings) hast, könnte dir leicht der Speicher ausgehen, bevor du die Grenze von 50 Ebenen erreichst.
- Alle FOR…NEXT-Schleifen und DO…LOOPs werden beschädigt, wenn die Subroutine oder Funktion rekursiv aus diesen Schleifen heraus aufgerufen wird.

## Beispiele

Oftmals besteht die Notwendigkeit, einen speziellen Befehl oder eine spezielle Funktion in MMBasic zu implementieren, doch in vielen Fällen lassen sich diese mithilfe einer gewöhnlichen Subroutine oder Funktion erstellen, die dann genau wie ein integrierter Befehl oder eine integrierte Funktion funktioniert.

Beispielsweise besteht manchmal Bedarf an einer TRIM-Funktion, die bestimmte Zeichen am Anfang und Ende einer Zeichenkette entfernt. Im Folgenden findest du ein Beispiel dafür, wie man eine solche einfache Funktion in MMBasic erstellt.

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

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

' Alle Zeichen in c$ am Ende von s$ entfernen
Funktion RTrim$(s$, c$)
  RTrim$ = s$
  Do While Instr(c$, Right$(RTrim$, 1))
    RTrim$ = Mid$(RTrim$, 1, Len(RTrim$) - 1)
  Schleife
End Function

' alle Zeichen in c$ vom Anfang von s$ abschneiden
Function LTrim$(s$, c$)
  LTrim$ = s$
  Do While Instr(c$, Left$(LTrim$, 1))
    LTrim$ = Mid$(LTrim$, 2)
  Schleife
End Function
```

Ein Beispiel für die Verwendung dieser Funktionen:

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

Ergibt „****23.56700“

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

Ergibt „23.567“

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

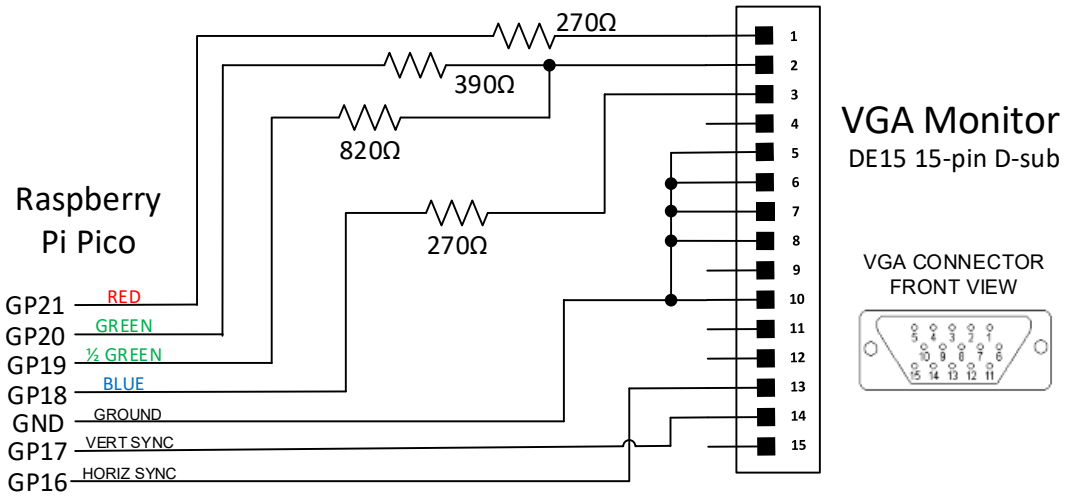
Ergibt „23.56700“

# Videoausgabe

## VGA-Video

Bei Firmware-Versionen, die VGA unterstützen, wird die Videoausgabe beim Start automatisch aktiviert – es müssen keine Optionen eingestellt werden.

Das folgende Diagramm zeigt, wie du den VGA-Monitor anschließt.




Die Ausgabe erfolgt im Standard-VGA-Format mit einer Pixelfrequenz von 25,175 MHz und einer Bildwiederholfrequenz von 60 Hz.

Es gibt zwei oder drei Modi, die mit dem Befehl MODE ausgewählt werden können:

MODE 1 Monochrom mit einer Auflösung von 640 x 480 (Standard beim Start)

MODUS 2 16 Farben mit einer Auflösung von 320 x 240

MODUS 3 16 Farben mit einer Auflösung von 640 x 480 (nur RP2350)

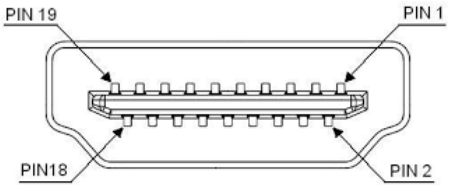
In den Modi 2 und 3 erfolgt die Ausgabe mit 16 Farben im 4-Bit-RGB121-Format (d. h. 1 Bit für Rot, 2 Bits für Grün und 1 Bit für Blau). Im Modus 2 werden die Pixel sowohl entlang der x- als auch der y-Achse dupliziert, was eine Auflösung von 320 x 240 ergibt, während der Monitor weiterhin ein Signal mit 640 x 480 Pixeln empfängt.

Die Ausgabe von MMBasic wird als Bitmap in einen Framebuffer geschrieben. Die Firmware nutzt dann die zweite CPU im Prozessor, um diese Framebuffer-Daten pixelweise über DMA an einen der programmierbaren I/O-Controller (PIO0) des RP2040 zu übertragen und so die Anzeige zu erzeugen. Da dies unabhängig vom Hauptprozessor läuft, hat die Erzeugung des VGA-Ausgangs kaum oder gar keine Auswirkungen auf die Geschwindigkeit von MMBasic.

## HDMI-Video

Für Firmware-Versionen, die HDMI-Video unterstützen, listet die folgende Tabelle die Anschlüsse an die standardmäßige HDMI-Buchse vom Typ A auf. Der HDMI-Ausgang wird beim Start automatisch aktiviert – es müssen keine Optionen eingestellt werden.

| | | | |
|---|---|---|---|
| HDMI Pin 1: | Pin 21 (GP16) via a 220Ω resistor | HDMI Pin 13: | No Connection |
| HDMI Pin 2: | Ground | HDMI Pin 14: | No Connection |
| HDMI Pin 3: | Pin 22 (GP17) via a 220Ω resistor | HDMI Pin 15: | No Connection |
| HDMI Pin 4: | Pin 24 (GP18) via a 220Ω resistor | HDMI Pin 16: | No Connection |
| HDMI Pin 5: | Ground | HDMI Pin 17: | Ground |
| HDMI Pin 6: | Pin 25 (GP19 via a 220Ω resistor | HDMI Pin 18: | +5V via Schottky barrier diode |
| HDMI Pin 7: | Pin 16 (GP12) via a 220Ω resistor | HDMI Pin 19: | No Connection |
| HDMI Pin 8: | Ground | | |
| HDMI Pin 9: | Pin 17 (GP13) via a 220Ω resistor | | |
| HDMI Pin 10: | Pin 19 (GP14) via a 220Ω resistor | | |
| HDMI Pin 11: | Ground | | |
| HDMI Pin 12: | Pin 20 (GP15) via a 220Ω resistor | | |



HDMI Front View

Die HDMI-Signalpins werden mit hoher Frequenz angesteuert, daher solltest du Folgendes beachten:

- Halte die Signalleitungen so kurz wie möglich.
- Stelle sicher, dass alle Signalleitungen gleich lang sind.
- Die 220-Ω-Widerstände sollten vorzugsweise als Oberflächenmontage-Typen verwendet werden.

Um das DVI/HDMI-Signal zu erzeugen, muss die Firmware den RP2350 auf bis zu 372 MHz übertakten, und die meisten Raspberry Pi Pico 2-Module haben bei diesen Geschwindigkeiten keine Probleme. Dies kann jedoch nicht garantiert werden, insbesondere bei Modulen von Drittanbietern. Ein Beispiel ist das Pimoroni Pico Plus 2, das bei den erforderlichen Geschwindigkeiten an der Grenze liegt und daher nicht mit den HDMI-Versionen der PicoMite-Firmware empfohlen werden kann.

Ähnlich wie bei der VGA-Erzeugung wird die Ausgabe von MMBasic in einen Framebuffer geschrieben, der mithilfe der zweiten CPU und DMA an das HSTX-Peripheriegerät weitergeleitet wird, welches wiederum die parallelen Videodaten erzeugt. Das erzeugte Videosignal ist eigentlich DVI (HDMI unterstützt DVI), was bedeutet, dass Audio am HDMI-Ausgang nicht unterstützt wird und auch komplexe HDMI-Funktionen wie High Definition Content Protection (HDCP) und Ethernet nicht unterstützt werden.

HDMI-Video unterstützt eine Reihe von Auflösungen. Um diese einzustellen, verwendest du den folgenden Befehl:

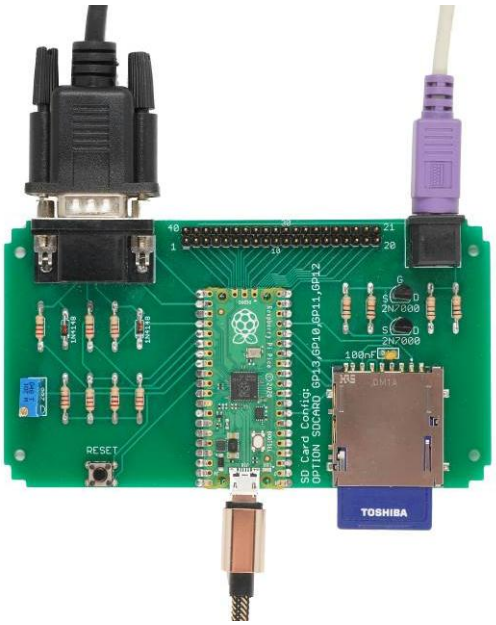
```
OPTION RESOLUTION nn
```

Wobei „nn“ einer der folgenden Werte ist:

640x480 oder 640
1280x720 oder 1280
1024x768 oder 1024
800x600 oder 800
720x400 oder 720
848x480 oder 848

Jede HDMI-Auflösung kann in verschiedenen Modi betrieben werden, die über den Befehl MODE eingestellt werden. Beachte, dass viele Modi die angezeigte Auflösung reduzieren, um Speicherplatz für andere Funktionen zu sparen. Diese Reduzierung erfolgt durch Verdopplung oder Vervierfachung jedes Pixels, der Monitor erkennt jedoch immer die durch den Befehl OPTION RESOLUTION festgelegte Auflösung (d. h. Pixeldichte). Die Standardeinstellung ist RESOLUTION 640x480 und MODE 1

## VGA/PS2-Referenzdesign (Raspberry Pi Pico)




Dies ist ein einfach zu montierendes Design, das den VGA-Ausgang, die PS2-Tastaturschnittstelle und den SD-Kartensteckplatz implementiert (dieses Design wurde im *Silicon Chip* Magazin vorgestellt).

Es verwendet gängige Durchsteckkomponenten und lässt sich in weniger als einer Stunde zusammenbauen.

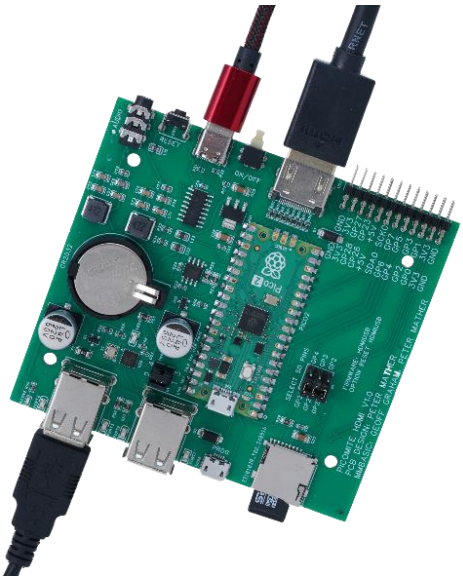
Alle 40 Pins des Raspberry Pi Pico sind auf einen 40-poligen Stecker auf der Rückseite der Leiterplatte geführt, und zwar in derselben Konfiguration wie beim Pico. Das macht den Anschluss externer Geräte einfach, da du das Pinbelegungsdiagramm in diesem Handbuch konsultieren und dann die entsprechenden Pins am 40-poligen Stecker auswählen kannst.

Unter Berücksichtigung der für den VGA-Ausgang, die Tastatur und die SD-Karte reservierten I/O-Pins stehen 14 I/O-Pins für externe Schaltungen zur Verfügung.

Die Platine ist so dimensioniert, dass sie in ein Altronics-Steckgehäuse mit den Maßen 130 x 75 x 28 mm (Teilenummer H0376) passt.

Das Bausatzpaket für dieses Design kann unter https://geoffg.net/picomitevga.html (am Ende der Seite) heruntergeladen werden.

## HDMI/USB-Referenzdesign (Raspberry Pi Pico 2)



Dies ist ein voll ausgestattetes Design auf Basis des Raspberry Pi Pico 2 (mit RP2350-Prozessoren), das Folgendes umfasst:

- HDMI-Videoausgang
- Vier USB-Schnittstellen für Tastatur, Maus, Gamepad usw.
- Hochwertigen Audioausgang für Lautsprecher mit Verstärker.
- USB-Schnittstelle zur seriellen Konsole.
- Batteriegepufferte Echtzeituhr.
- Micro-SD-Kartensteckplatz.
- 14 I/O-Pins an der Rückseite.
- Passend für ein Multicomp MCRM2015S- oder Hammond RM2015S-Gehäuse.

Das Baupaket für diesen Entwurf kannst du hier herunterladen: https://geoffg.net/picomitevga.html (unten auf der Seite).

# Tastatur/Maus/Gamepad

Die PicoMite-Firmware unterstützt Tastatur und Maus über PS2- oder USB-Schnittstellen. Die Wahl zwischen PS2 und USB hängt von der geladenen Firmware-Version ab. Siehe das Kapitel *„Firmware-Versionen und Dateien“* am Anfang dieses Handbuchs.

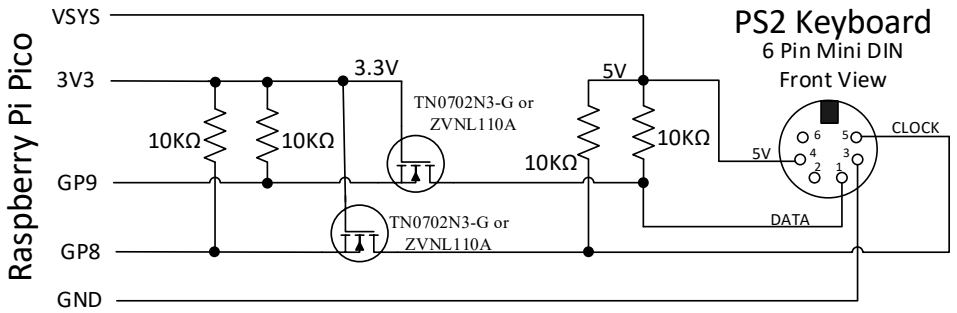
USB-Versionen der Firmware unterstützen auch ein PS3- oder PS4-Gamepad mit USB-Schnittstelle. Bei Versionen ohne USB-Unterstützung können die Befehle WII (CLASSIC) und WII NUNCHUCK verwendet werden, um ein über I²C angeschlossenes Gamepad anzugeben.

Eine Tastatur kann zur Eingabe von Daten in das BASIC-Programm verwendet werden oder, mit einem VGA/HDMI-Videoausgang, zur Erstellung eines eigenständigen Computers mit Tastatur und Bildschirm. Anstelle eines Videoausgangs kannst du bei Versionen, die dies unterstützen, auch ein LCD-Panel anschließen und die MMBasic-Konsolenausgabe auf dem LCD-Panel anzeigen, wodurch eine kompaktere Version eines eigenständigen Computers entsteht. Siehe den Abschnitt *„LCD-Display als Konsolenausgabe“* für Details dazu.

## PS2-Tastatur am Raspberry Pi Pico (RP2040)

Die Takt- und Datensignale der PS2-Tastatur arbeiten mit 5 V, aber die I/O-Pins der RP2040-Prozessoren dürfen nicht mit mehr als 3,6 V beaufschlagt werden. Aus diesem Grund sollte ein Pegelumsetzer verwendet werden, damit der Raspberry Pi Pico Signalspannungen im Bereich von 0 bis 3,3 V sieht, während die Tastatur Spannungen von 0 bis 5 V sieht.

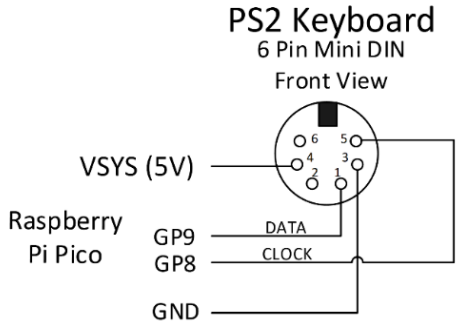
Es gibt viele Möglichkeiten, dies zu bewerkstelligen, aber die folgende Schaltung ist eine einfache und kostengünstige Lösung:




Der empfohlene MOSFET ist ein TN0702N3-G oder ZVNL110A, aber der handelsübliche 2N7000 wurde getestet und funktioniert gut.

Nach dem Anschließen muss die Tastatur mit dem Befehl OPTION KEYBOARD aktiviert werden.

## PS2-Tastatur am Raspberry Pi Pico 2 (RP2350)




Die I/O-Pins der Mikrocontroller der RP2350-Serie halten 5 V stand (im eingeschalteten Zustand), sodass die Tastatur direkt angeschlossen werden kann, wie in der Abbildung gezeigt, mit der 5-V-Spannung, die vom VSYS-Pin des Raspberry Pi Pico 2 geliefert wird.

Die Tastatur wird mit dem Befehl OPTION KEYBOARD aktiviert.

## PS2-Maus

Die für eine PS2-Maus verwendeten I/O-Pins müssen mit den Befehlen OPTION MOUSE (an der Befehlszeile) oder MOUSE OPEN (innerhalb eines Programms) konfiguriert werden.

Eine PS2-Maus wird mit 5 V versorgt, daher ist bei einem Raspberry Pi Pico (RP2040) ein Pegelumsetzer für die Takt- und Datenpins der Maus erforderlich – dieser kann dem oben genannten Schaltkreis für eine PS2-Tastatur entsprechen. Bei einem Raspberry Pi Pico 2 (RP2350) ist kein Pegelumsetzer erforderlich, sodass die Maus direkt angeschlossen werden kann.

## USB-Schnittstelle

Firmware-Versionen (sowohl RP2040 als auch RP2350) mit USB-Unterstützung ermöglichen den Anschluss einer Tastatur, einer Maus und/oder eines Gamepads über die USB-Schnittstelle. Dazu wird der USB-Anschluss des Raspberry Pi Pico in einen USB-Host umgewandelt (im Gegensatz zu seinem normalen Modus als USB-Client). Das ist möglich, weil der Anschluss und die Elektronik des Pico USB-OTG-kompatibel (On The Go) sind, ähnlich wie der Anschluss vieler Mobiltelefone.

Da der USB-Anschluss für andere Aufgaben genutzt wird , muss der Pico über 5 V am VSYS-Pin oder am VbUS-Pin mit Strom versorgt werden, wenn du einen externen Hub/eine externe Tastatur über den Pico mit Strom versorgen möchtest.

Um ein USB-Gerät anzuschließen, benötigst du ein Adapterkabel, das an einem Ende einen Micro-USB-Stecker (für den Pico) und am anderen Ende eine USB-Buchse vom Typ A (für das Gerät) hat. Ein typisches Beispiel ist das Jaycar-Modell WC7725.

Da die USB-Schnittstelle am Pico zu einem USB-Host umgewandelt wurde, hast du keinen Zugriff auf die serielle MMBasic-Konsole. Die Firmware gleicht dies aus, indem sie automatisch Pin 11 (GP8) für den seriellen Konsolen-Tx und Pin 12 (GP9) für den Rx verwendet und die Baudrate auf 115200 Baud einstellt. Um auf diese Konsole zuzugreifen, benötigst du eine USB-zu-Seriell-Brücke, die auf der einen Seite eine serielle TTL-Schnittstelle und auf der anderen Seite eine USB-Schnittstelle bietet (suche nach Modulen mit dem CP2102- oder CH340-Chip). Bei Bedarf kann die OPTION SERIAL CONSOLE verwendet werden, um die für die Konsole verwendeten Pins zu ändern.

## USB-Hub

Die Firmware unterstützt über diese Schnittstelle auch einen USB-Hub, sodass du mehrere Tastaturen oder eine Tastatur plus eine Maus plus ein Gamepad usw. anschließen kannst. Über einen Hub können maximal 4 Geräte angeschlossen werden. Diese werden durch eine Kanalindexnummer (1 bis 4) referenziert. Verwende MM.INFO(USB n), um den Gerätecode für jedes an Kanal n angeschlossene Gerät abzurufen. Standardmäßig wird eine Tastatur Kanal 1 zugewiesen. Eine Maus wird Kanal 2 zugewiesen. Das erste Gamepad wird Kanal 3 zugewiesen und ein zweites Gamepad Kanal 4.

Wenn du einen USB-Hub verwendest, ist es besser, einen nicht mit eigener Stromversorgung ausgestatteten Hub zu nutzen (d. h. einen, der vom Raspberry Pi Pico mit Strom versorgt wird). Der Grund dafür ist, dass der USB-Protokollstack den Hub nicht zurücksetzen kann und es zu Verwirrung kommen kann, wenn der Pico ein- und ausgeschaltet wird, ohne dasselbe für den Hub zu tun. Der Hub kann auch verwirrt werden, wenn Geräte ausgetauscht werden, während der Hub mit Strom versorgt wird. In diesem Fall solltest du den Pico und anschließend den Hub aus- und wieder einschalten und dann die USB-Geräte nacheinander anschließen.

Beachte, dass ein Hub nicht erforderlich ist. Wenn du nur ein Gerät anschließen möchtest (zum Beispiel eine Tastatur), kannst du das Gerät (mit einem Adapterkabel) einfach direkt an den USB-Anschluss des Pico anschließen.

## USB-Tastatur

Wenn eine USB-Tastatur angeschlossen wird, wird sie sofort erkannt (keine Konfiguration erforderlich) und MMBasic ordnet sie standardmäßig Kanal 1 zu – es ist nichts weiter erforderlich.

## USB-Maus

Wenn eine USB-Maus angeschlossen wird, wird sie sofort erkannt (keine Konfiguration erforderlich) und MMBasic weist ihr standardmäßig Kanal 2 zu – es ist nichts weiter erforderlich.

## USB-Gamepad

Ein oder mehrere PS3- oder PS4-Controller oder Gamepads wie beispielsweise ein Super Nintendo SNES-Controller mit USB-Schnittstelle können über USB angeschlossen werden (siehe Abbildung rechts).

Standardmäßig wird das erste Gamepad dem Kanal 3 und ein zweites Gamepad dem Kanal 4 zugewiesen. Innerhalb eines Programms können die Daten vom Gamepad mit der Funktion DEVICE(GAMEPAD) ausgelesen werden.

## Konfigurieren der Tastatur

Standardmäßig wird davon ausgegangen, dass die Tastaturkonfiguration dem US-Standardlayout entspricht. Mit dem Befehl OPTION KEYBOARD kannst du jedoch Layouts für andere Länder konfigurieren.

Die Syntax des Befehls lautet:

```
OPTION KEYBOARD Sprache
```

Dabei ist „language“ ein zweistelliger Code, wie z. B. US für die Standardtastatur, die in den USA, Australien und Neuseeland verwendet wird. Weitere Tastaturlayouts sind Großbritannien (UK), Französisch (FR), Deutsch (GR), Belgien (BE), Italienisch (IT), Brasilianisch (BR) oder Spanisch (ES).

Beachte, dass die Nicht-US-Layouts zwar einige der auf diesen Tastaturen vorhandenen Sondertasten zuordnen, die entsprechenden Sonderzeichen jedoch nicht angezeigt werden, da sie nicht Teil der Standard-PicoMite-Schriftarten sind. Stattdessen wird ein Standard-ASCII-Zeichen verwendet.

## Verwendung der Maus

Die Maus ist besonders nützlich im MMBasic-Programmeditor, wo sie viele Funktionen nachahmen kann, die man aus GUI-Editoren wie Notepad unter Windows kennt (siehe den Abschnitt *„Vollbild-Editor“* weiter oben in diesem Handbuch). Dazu gehören das Setzen der Einfügeposition sowie das Kopieren und Einfügen über die Zwischenablage.

Eine Maus kann auch in einem Programm verwendet werden, in dem ihre Position mithilfe der Funktion DEVICE() abgefragt werden kann. Als Beispiel meldet das folgende Programm jede Mausbewegung.

Beachte, dass die Maus immer Kanal 2 zugewiesen ist

```
' Endlosschleife, um jede Bewegung auf der Konsole auszugeben
Do
  mx=DEVICE(MOUSE 2, x)
  my=DEVICE(MOUSE 2, y)
  Wenn mx ≠ tx oder my ≠ ty, dann drucke mx, my
  tx = mx : ty = my
Schleife
```

# Programm- und Datenspeicher

Das BASIC-Programm wird im Flash-Speicher abgelegt und von dort aus ausgeführt. Wenn ein Programm über EDIT bearbeitet oder über die Konsole geladen wird, wird es dort gespeichert. Der Flash-Speicher ist nichtflüchtig, sodass das Programm nicht verloren geht, wenn die Stromversorgung ausfällt oder der Prozessor zurückgesetzt wird.

Zusätzlich zu diesem Programmspeicher gibt es drei weitere Speicherorte, an denen Programme gespeichert werden können. Diese werden im Detail unter unter „ “ beschrieben und sind die Flash-Slots, das Flash-Dateisystem und eine angeschlossene SD-Karte

## Flash-Slots

Es gibt drei davon, die du nutzen kannst, um völlig unterschiedliche Programme oder frühere Versionen des Programms, an dem du gerade arbeitest, zu speichern (falls du auf eine frühere Version zurückgreifen musst). Außerdem ermöglicht MMBasic es einem BASIC-Programm, ein anderes Programm zu laden und auszuführen, dass an einem nummerierten Flash-Speicherplatz gespeichert ist, wobei alle Variablen und Einstellungen des ursprünglichen Programms beibehalten werden – dies wird als Verkettung bezeichnet und ermöglicht die Ausführung eines viel größeren Programms, als es der Programmspeicher normalerweise zulassen würde.

Um diese nummerierten Speicherplätze im Flash zu verwalten, kannst du die folgenden Befehle verwenden (beachte, dass *n* im Folgenden eine Zahl zwischen 1 und 3 ist):

| | |
|---|---|
| FLASH SAVE *n* | Speichere das Programm im Programmspeicher an der Flash-Adresse *n*. |
| FLASH LOAD *n* | Lade ein Programm vom Flash-Speicherplatz *n* in den Programmspeicher. |
| FLASH RUN *n* | Führe ein Programm von Flash-Speicherplatz *n* aus; löscht alle Variablen, löscht oder ändert jedoch nicht das im Hauptprogrammspeicher befindliche Programm. |
| FLASH LIST | Zeige eine Liste aller Flash-Speicherplätze an, einschließlich der ersten Zeile des Programms. |
| FLASH LIST n [,ALL] | Listet das Programm am Speicherplatz n auf. Verwende FLASH LIST n,ALL, um die Liste ohne Seitenumbrüche anzuzeigen |
| FLASH ERASE *n* | Lösche den Flash-Speicherplatz *n*. |
| FLASH ERASE ALL | Löscht alle Flash-Speicherplätze. |
| FLASH CHAIN *n* | Führe das Programm an Flash-Speicherplatz *n* aus, wobei alle Variablen unverändert bleiben. So können sehr große Programme, die in zwei oder drei Teile aufgeteilt werden können, ausgeführt werden. Das Programm im Hauptprogrammspeicher wird dabei weder gelöscht noch verändert. |
| FLASH OVERWRITE *n* | Lösche Flash-Speicherplatz *n* und speichere dann das Programm im Programmspeicher an diesem Speicherplatz. |
| FLASH DISK LOAD f$ [,O] | Lädt einen Flash-Speicherplatz mit einer Binärdatei, die mit LIBRARY DISK SAVE erstellt wurde. Überschreibt den Speicherplatz, wenn das optionale „O“ angegeben wird. |

Außerdem kann der Befehl OPTION AUTORUN verwendet werden, um einen Speicherort für ein Flash-Programm anzugeben, das beim Einschalten oder Neustart der CPU ausgeführt werden soll. Diese Option kann auch ohne Angabe eines Flash-Speicherorts verwendet werden; in diesem Fall führt MMBasic das Programm automatisch im Programmspeicher aus.

- Es wird empfohlen, als erste Zeile des Programms einen Kommentar einzufügen, der das Programm beschreibt. Dieser wird dann vom Befehl FLASH LIST angezeigt und hilft bei der Identifizierung des Programms.
- Alle im Flash gespeicherten BASIC-Programme können gelöscht werden, wenn du die PicoMite-Firmware aktualisierst (oder auf eine ältere Version zurücksetzt). Stelle also sicher, dass du diese zuvor sicherst.
- Der Befehl LIBRARY nutzt Slot 3 zum Speichern von Bibliotheksdaten, daher stehen nur 2 Slots zur Verfügung, wenn die Bibliotheksfunktion genutzt wird.

## Flash-Dateisystem

Dies ist ein Bereich des Flash-Speichers des Raspberry Pi Pico, der automatisch von der Firmware erstellt wird und für MMBasic wie ein normales Laufwerk aussieht. Er wird als Laufwerk A: bezeichnet, und Daten sowie

Programme können mit den üblichen BASIC-Dateibefahlen (SAVE, RUN, OPEN usw.) gelesen und geschrieben werden. Außerdem können Unterverzeichnisse erstellt und gelöscht sowie lange Dateinamen verwendet werden.

Beispiel für die Ausführung eines Programms:

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

Öffne eine Textdatei für den Direktzugriff:

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

Dieses Laufwerk wird automatisch erstellt, wenn die PicoMite-Firmware geladen wird, sodass es dem BASIC-Programm immer zur Verfügung steht. Es kann zum Speichern von Programmen, Bildern, Musik, Konfigurationsdaten, Protokolldateien und vielem mehr verwendet werden. Seine Größe hängt von der Größe des Flash-Speichers ab – auf einem Raspberry Pi Pico mit 2 MB Flash beträgt sie 200 bis 500 KB, auf einem Raspberry Pi Pico 2 mit 4 MB Flash etwas über 2 MB und auf einem Modul mit 16 MB ist das Flash-Dateisystem bis zu 14 MB groß.

Das System erstellt und verwaltet die Datei „BOOTCOUNT“ im Flash-Dateisystem. Diese zählt, wie oft das Gerät neu gestartet wurde, und kann mit der Funktion MM.INFO(boot count) ausgelesen werden.

## SD-Karten

Ein SD-Kartensteckplatz kann an den Raspberry Pi Pico angeschlossen und als Laufwerk B: genutzt werden. Wie beim Flash-Dateisystem können die üblichen BASIC-Dateibefehle verwendet werden, um Programme zu speichern/zu laden sowie Datendateien zum Lesen/Schreiben zu öffnen.

Es werden Karten mit bis zu 32 GB unterstützt, die mit FAT16 oder FAT32 formatiert sind, und die erstellten Dateien können auch auf PCs mit Windows, Linux oder dem Mac-Betriebssystem gelesen und geschrieben werden. Die PicoMite-Firmware nutzt das SPI-Protokoll für die Kommunikation mit der Karte, was unabhängig vom Kartentyp ist, sodass alle Typen (Klasse 4, 10, UHS-1 usw.) unterstützt werden. Eine Beschreibung von SPI findest du unter: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

Das SPI-Protokoll muss speziell konfiguriert werden, bevor es verwendet werden kann. Dies kann auf zwei Arten erfolgen – entweder über den „System“-SPI-Port oder durch direkte Angabe der zu verwendenden I/O-Pins:

System-SPI-Port

Dies ist ein Port, der für Systemzwecke 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 das Kapitel *„PicoMite-Hardware“*), aber in diesem Beispiel wird SPI an den Pins GP18, GP19 und GP16 für Clock, MOSI bzw. MISO verwendet.

```
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 (CS) verwendet wird:

```
OPTION SDCARD GP22
```

Dedizierte I/O-Pins

Alternativ, wenn keine anderen Geräte den SPI-Bus mit der SD-Karte teilen, kann sie wie folgt eingerichtet werden:

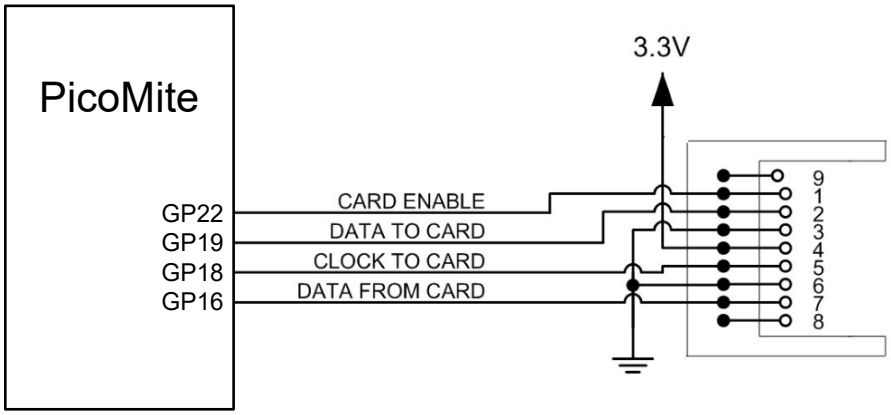
```
OPTION SDCARD  CS_pin, CLK_pin, MOSI_pin, MISO_pin
```

In diesem Fall können die Pins völlig flexibel zugewiesen werden und müssen nicht für den SPI-Betrieb geeignet sein, aber die Leistung der SD-Karte ist besser, wenn gültige SPI-Pins gewählt werden.

Diese Befehle müssen an der Befehlszeile (nicht in einem Programm) eingegeben werden und führen dazu, dass die PicoMite-Firmware neu gestartet wird. Dies hat zur Folge, dass die USB-Konsolenschnittstelle getrennt wird und wieder verbunden werden muss.

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

Der grundlegende Schaltplan für den Anschluss des SD-Kartensteckers unter Verwendung dieser Pinbelegungen ist unten dargestellt.

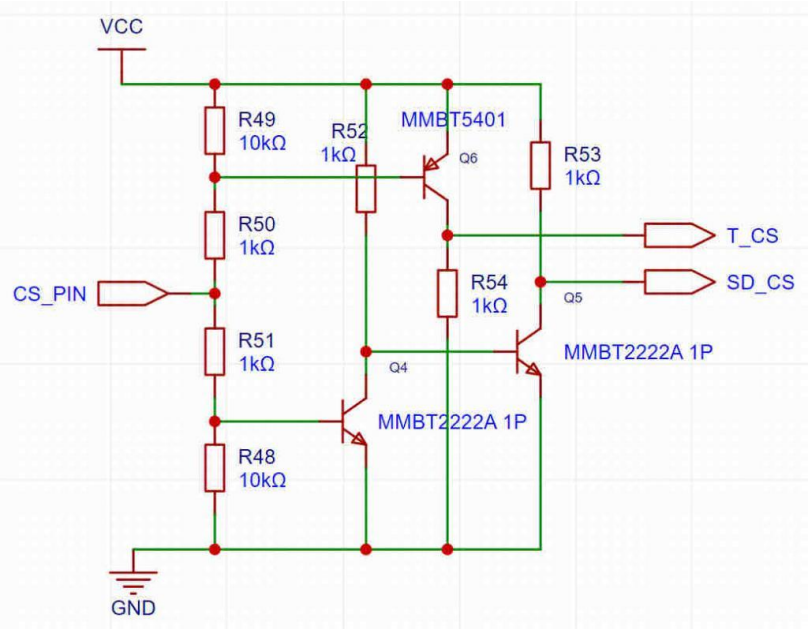



Beachte, dass du viele verschiedene Konfigurationen mit unterschiedlichen Pinbelegungen verwenden kannst – dies ist nur ein Beispiel, das auf den oben aufgeführten Konfigurationsbefehlen basiert.

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

## Kombinierter Chip-Select

Der Chip-Select-Pin, der für die SD-Karte und den Touch-Controller auf einem LCD-Panel verwendet wird, kann mit dem Befehl OPTION SDCARD COMBINED CS kombiniert werden. Wenn dies angegeben wird, ist die folgende Schaltung erforderlich, um den SD-Chip-Select zu implementieren:




Die Firmware nutzt den Touch-Pin wie folgt:

TOUCH_CS low: TOUCH_CS low, SD_CS high

TOUCH CS high: SD_CS low: TOUCH_CS high

TOUCH CS als Eingang (hohe Impedanz) eingestellt: TOUCH_CS und SD_CS high.

## MMBasic-Unterstützung für Flash- und SD-Karten-Dateisysteme

Die MMBasic-Unterstützung für das Flash-Dateisystem und SD-Karten ist nahezu identisch. Dadurch können Programme mit minimalen Änderungen beide Dateisysteme nutzen. Das Flash-Dateisystem wird als Laufwerk A: bezeichnet, während die SD-Karte (sofern angeschlossen) Laufwerk B: ist. Das Standardlaufwerk kann mit dem Befehl DRIVE festgelegt werden; dann ist das Laufwerkspräfix nicht erforderlich.

Beachte dabei Folgendes:

- Groß- und Kleinbuchstaben sowie Leerzeichen sind zulässig. Weder das Dateisystem auf der SD-Karte noch das Flash-Dateisystem unterscheiden zwischen Groß- und Kleinschreibung.
- Lange Datei-/Verzeichnisnamen werden zusätzlich zum alten 8.3-Format unterstützt.

- Die maximale Datei-/Pfadlänge beträgt 63 Zeichen (127 Zeichen bei RP2350-Versionen).
- Jeder Dateipfad, der den Laufwerksbuchstaben verwendet, muss ein vollständiger Pfad ab dem Stammverzeichnis sein (z. B. „A:/mypath/myfile.txt“).
- Verzeichnispfade sind in Datei-/Verzeichnisstrings erlaubt. (z. B. OPEN „A:\dir1\dir2\file.txt“ FOR …).
- In Pfaden können sowohl Schrägstriche als auch Backslashes verwendet werden. Z. B. ist \dir\file.txt dasselbe wie /dir/file.txt.
- Beim Start ist das aktive Laufwerk (d. h. das Standardlaufwerk) A: (das Flash-Dateisystem).
- Die aktuelle PicoMite-Firmware-Zeit wird für die Erstellungs- und letzten Zugriffszeiten der Dateien verwendet.
- Es können bis zu zehn Dateien gleichzeitig geöffnet sein, gemischt zwischen den Laufwerken A: und B:.
- Mit Ausnahme von INPUT, LINE INPUT und PRINT ist das # in #fnbr optional und kann weggelassen werden.

Programme können mit diesen Befehlen aus dem Flash-Dateisystem und von SD-Karten geladen oder dort gespeichert werden.

□ LOAD fname$ [, R]
Lade ein BASIC-Programm. Das optionale Suffix „,R“ bewirkt, dass das Programm nach dem Laden ausgeführt wird (in diesem Fall muss fname$ eine Zeichenfolgenkonstante sein).

□ RUN fname$
Lade ein BASIC-Programm und führe es aus. fname$ kann eine Variable sein.

□ SAVE fname$
Speichere das aktuelle Programm im Flash-Dateisystem oder auf der SD-Karte.

Dies sind die grundlegenden Befehle zum Lesen und Schreiben von Daten.

□ OPEN fname$ FOR mode AS #fnbr
Öffnet eine Datei zum Lesen oder Schreiben. 'fname$' ist der Dateiname. 'mode' kann INPUT, OUTPUT, APPEND oder RANDOM sein. ‘#fnbr’ ist die Dateinummer (1 bis 10).

□ PRINT #fnbr, Ausdruck [[,; ]Ausdruck] … usw.
Gibt Text in die unter #fnbr geöffnete Datei aus.

□ INPUT #fnbr, Liste von Variablen
Liest eine Liste kommagetrennter Daten in die angegebenen Variablen aus der zuvor als #fnbr geöffneten Datei ein.

□ LINE INPUT #fnbr, variable$
Liest eine komplette Zeile in die angegebene Zeichenfolgenvariable aus der zuvor als #fnbr geöffneten Datei ein.

□ FLUSH #fnbr
Erzwingt, dass alle gepufferten Schreibvorgänge auf das Flash-Dateisystem oder die SD-Karte geschrieben werden. Es wird empfohlen, diesen Befehl regelmäßig zu verwenden, wenn es bei einem Stromausfall zu Datenverlusten kommen könnte.

□ CLOSE #fnbr [,#fnbr] …
Schließe die zuvor mit der Dateinummer „#fnbr“ geöffneten Dateien.

Grundlegende Datei- und Verzeichnisbearbeitung. Das meiste davon kannst du an der Befehlszeile oder innerhalb eines BASIC-Programms erledigen.

□ DRIVE drive$
Legt das aktive Laufwerk als „drive$“ fest. „drive$“ kann „A:“ oder „B:“ sein, wobei A das USB-Laufwerk und B die SD-Karte ist (sofern konfiguriert).

□ DRIVE "A:/FORMAT"
Formatiert das Flash-Dateisystem (Laufwerk A:) auf den Ausgangszustand zurück.

- FILES [Platzhalter]
  Durchsucht das aktuelle Verzeichnis und listet die gefundenen Dateien/Verzeichnisse auf.
  Hinweis: Kann nur an der Eingabeaufforderung verwendet werden, nicht innerhalb eines Programms.

- LIST fname$
  Zeige den Inhalt einer Programm- oder Textdatei auf der Konsole an.

- KILL fname$
  Lösche eine Datei im aktuellen Verzeichnis auf dem aktuellen Laufwerk.
  Weitere Informationen zum Löschen mit Platzhaltern findest du in der Befehlsreferenz.

- MKDIR dname$
  Erstellt ein Unterverzeichnis im aktuellen Verzeichnis auf dem aktuellen Laufwerk.

- CHDIR dname$
  Wechsle in das Verzeichnis $dname. $dname kann auch „.“ für das aktuelle Verzeichnis, „..“ (Doppelpunkt) für das übergeordnete Verzeichnis oder „\“ für das Stammverzeichnis sein. Ausgangspunkt ist das aktuelle Verzeichnis auf dem aktuellen Laufwerk.

- RMDIR dir$
  Entferne oder lösche das Verzeichnis „dir$“ im aktuellen Verzeichnis auf dem aktuellen Laufwerk.

- SEEK #fnbr, pos
  Positioniert den Lese-/Schreibzeiger in einer Datei, die für den RANDOM-Zugriff geöffnet wurde, auf das Byte „pos“.

- RENAME fromname$ AS toname$
  Benennt die Datei „fromname$“ im aktuellen Verzeichnis auf dem aktuellen Laufwerk in „toname$“ um

- COPY [Modus] fromname$ TO toname$
  Kopiert die Datei fromname$, sodass die Datei toname$ ~~entsteht~~.
  Weitere Details zu den optionalen Modi und Kopien mit Platzhaltern findest du in der Befehlsreferenz.

Außerdem gibt es eine Reihe von Funktionen, die die oben genannten Befehle unterstützen.

- INPUT$(nbr, #fnbr)
  Gibt eine Zeichenkette zurück, die aus „nbr“ Zeichen besteht, die aus einer zuvor für INPUT oder RANDOM geöffneten Datei mit der Dateinummer „#fnbr“ gelesen wurden. Wenn weniger als „nbr“ Zeichen verfügbar sind, gibt die Funktion das zurück, was vorhanden ist (einschließlich einer leeren Zeichenkette, wenn keine Zeichen verfügbar sind).

- DIR$( fspec, type )
  Sucht 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 mit der Dateinummer „#fnbr“ geöffnete Datei am Ende der Datei positioniert ist.

- LOC( #fnbr )
  Bei einer geöffneten Datei gibt dies die aktuelle Position des Lese-/Schreibzeigers in der Datei zurück.

- LOF( #fnbr )
  Gibt die aktuelle Länge der Datei in Bytes zurück.

- MM.INFO(Laufwerk)
  Gibt das aktuell aktive Laufwerk zurück – z. B. „A:“ oder „B:“

- MM.INFO(freier Speicherplatz)
  Gibt an, wie viel Speicherplatz auf dem aktiven Laufwerk noch frei ist

- MM.INFO(disk size)
  Gibt die Größe des aktiven Laufwerks zurück

- MM.INFO(exists file fname$)
  Gibt „true“ zurück, wenn die Datei existiert

- MM.INFO(exists dir dirname$)
  Gibt „true“ zurück, wenn das Verzeichnis existiert

- MM.INFO(path)
  Gibt den Dateipfad des aktuell ausgeführten Programms zurück. Damit kannst du relative Pfadverweise für alle in einem Programm benötigten Ressourcendateien erstellen.

## XModem-Übertragung

Zusätzlich zur Standardmethode der XModem-Übertragung, bei der Daten in den oder aus dem Programmspeicher kopiert werden, kann die PicoMite-Firmware auch Daten in eine Datei auf dem Flash-Dateisystem oder einer SD-Karte kopieren oder von dort lesen. Die Syntax lautet:

```
XMODEM SEND Dateiname$
XMODEM RECEIVE filename$
```

Dabei ist „filename$“ die Datei, die gespeichert oder gesendet werden soll. „filename$“ kann ein String-Ausdruck, eine Variable oder eine Konstante sein. Handelt es sich um eine Konstante, muss der String in Anführungszeichen gesetzt werden (z. B. XMODEM SEND "prbas.bas"). Beim Empfang einer Datei wird jede Datei mit demselben Namen überschrieben.

## Bild laden und speichern

Ein Bild kann aus dem Flash-Dateisystem oder von der SD-Karte geladen werden, um es auf einem angeschlossenen LCD-Display oder einem VGA-/HDMI-Monitor anzuzeigen. Dies kann verwendet werden, um ein Logo zu zeichnen oder einen Hintergrund auf dem Display hinzuzufügen.

Die Syntax lautet:

```
     LOAD IMAGE filename$ [, StartX, StartY]    (BMP-Bild laden)
oder LOAD JPG filename$ [, StartX, StartY]
oder LOAD PNG filename$ [, StartX, StartY]      (nur Pico2/RP2350)
```

Dabei ist „filename$“ das zu ladende Bild und „StartX“/„StartY“ sind die Koordinaten der oberen linken Ecke des Bildes (diese sind optional und werden standardmäßig auf die obere linke Ecke des Bildschirms gesetzt, wenn sie nicht angegeben werden).

Das Bild muss im entsprechenden Format (BMP, JPG oder PNG) vorliegen, und MMBasic fügt die Dateiendung hinzu, falls sie nicht angegeben ist. Alle Bildtypen werden unterstützt, einschließlich Schwarz-Weiß- und True-Color-Bilder mit 24 Bit.

Das aktuelle Bild auf dem Videoausgang, dem virtuellen LCD oder einem LCD-Panel, das BLIT unterstützt, 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 Bildes als 24-Bit-True-Color-BMP-Datei gespeichert (die Erweiterung .BMP wird hinzugefügt, falls keine Erweiterung angegeben wird).

## Beispiel für sequentielle E/A

Im folgenden Beispiel wird eine Datei erstellt und zwei Zeilen werden in die Datei geschrieben (mit dem Befehl PRINT). Die Datei wird anschließend geschlossen.

```
OPEN "fox.txt" FOR OUTPUT AS #1
PRINT #1, "Der schnelle braune Fuchs"
PRINT #1, "springt über den faulen Hund"
CLOSE #1
```

Du kannst den Inhalt der Datei mit dem Befehl LINE INPUT lesen.  Zum Beispiel:

```
ÖFFNE „fox.txt“ FÜR EINGABE ALS #1
LINE INPUT #1,a$
LINE INPUT #1,b$
CLOSE #1
```

LINE INPUT liest jeweils eine Zeile, sodass die Variable a$ den Text „The quick brown fox“ enthält und b$ den Text „jumps over the lazy dog“.

Eine andere Möglichkeit, aus einer Datei zu lesen, ist die Verwendung der Funktion INPUT$(). Diese liest eine bestimmte Anzahl von Zeichen ein. Zum Beispiel:

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

Der erste INPUT$() liest 12 Zeichen und der zweite drei Zeichen. Die Variable ta$ enthält also „The quick br“ und die Variable tb$ enthält „own“.

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

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

Du kannst den Inhalt dieser Datei mit dem Befehl LINE INPUT lesen, musst den Text dann aber mit VAL() in eine Zahl umwandeln.

Zum Beispiel:

```
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.

## Zufällige Datei E/A

Für den zufälligen Zugriff sollte die Datei mit dem Schlüsselwort RANDOM geöffnet werden. Zum Beispiel:

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

Um zu einem Datensatz innerhalb der Datei zu springen, würdest du den Befehl SEEK verwenden, 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 würdest du 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 Bytes zu lesen, würdest du Folgendes verwenden:

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

Beim Schreiben in die Datei sollte eine feste Datensatzgröße verwendet werden, was sich leicht bewerkstelligen lässt, indem man den zu schreibenden Daten ausreichend Füllzeichen (normalerweise Leerzeichen) hinzufügt. Zum Beispiel:

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

Die Funktion SPACE$() wird verwendet, um genügend Leerzeichen hinzuzufügen, damit die geschriebenen Daten genau die richtige Länge haben (in diesem Beispiel 64 Byte). Das Semikolon am Ende des Druckbefehls unterdrückt das Hinzufügen der Wagenrücklauf- und Zeilenvorschubzeichen, die den Datensatz länger als beabsichtigt machen würden.

Zwei weitere Funktionen können beim zufälligen Dateizugriff hilfreich sein. Die Funktion LOC() gibt die aktuelle Byte-Position des Lese-/Schreibzeigers zurück, und die Funktion LOF() gibt die Gesamtlänge der Datei in Bytes zurück.

Das folgende Programm demonstriert den zufälligen Dateizugriff. Damit kannst du an die Datei anhängen (um zunächst einige Daten hinzuzufügen) und dann Datensätze unter Verwendung zufälliger Datensatznummern lesen/schreiben. Der erste Datensatz in der Datei hat die Nummer 1, der zweite die Nummer 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 Direktzugriff kann auch bei einer normalen Textdatei verwendet werden. Das hier druckt zum Beispiel eine Datei rückwärts aus:

```
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
```

# Tonausgabe

Die PicoMite-Firmware kann Stereo-WAV-, FLAC-, MP3- oder MOD-Dateien abspielen, die sich auf dem Flash-Dateisystem oder der SD-Karte befinden, und mit dem Befehl PLAY präzise Sinuswellen erzeugen.

Beachte, dass der Schaltnetzteilregler des Raspberry Pi Pico Störgeräusche am Audioausgang verursachen kann. Dies lässt sich reduzieren, indem du den Regler deaktivierst und das Modul über einen externen linearen Regler mit Strom versorgst.

## Pulsweitenmoduliertes (PWM) Signal

Der Ton wird über PWM-Ausgänge erzeugt. Bevor du also die PLAY-Befehle verwenden kannst, müssen die PWM-Ausgangspins als Audioausgänge zugewiesen werden:

Dies geschieht mit dem Befehl OPTION AUDIO wie folgt:

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

Dieser Befehl sollte in die Befehlszeile eingegeben werden und wird gespeichert, sodass er nur einmal ausgeführt werden muss. Beide Pins müssen sich auf demselben PWM-Kanal befinden, wobei PWM-A-PIN der linke Audiokanal und PWM-B-PIN der rechte ist.

Zum Beispiel:

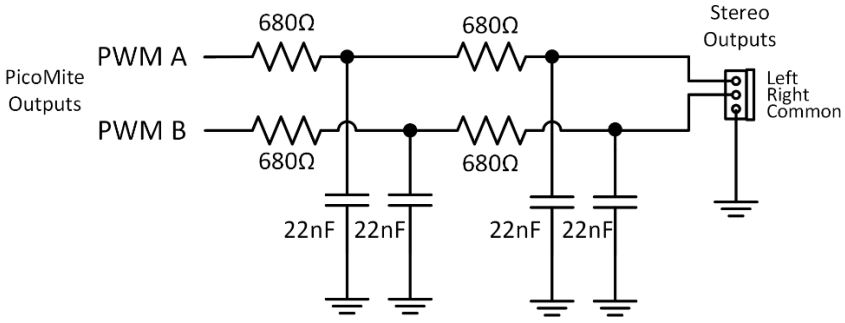
```
OPTION AUDIO GP0, GP1
```

Das Audiosignal wird als pulsweitenmoduliertes (PWM) Signal auf eine 44-kHz-Rechteckwelle (die sogenannte Trägerwelle) aufgelegt. Das bedeutet, dass ein Tiefpassfilter erforderlich ist, um die Trägerwelle zu entfernen und das Audiosignal wiederherzustellen.

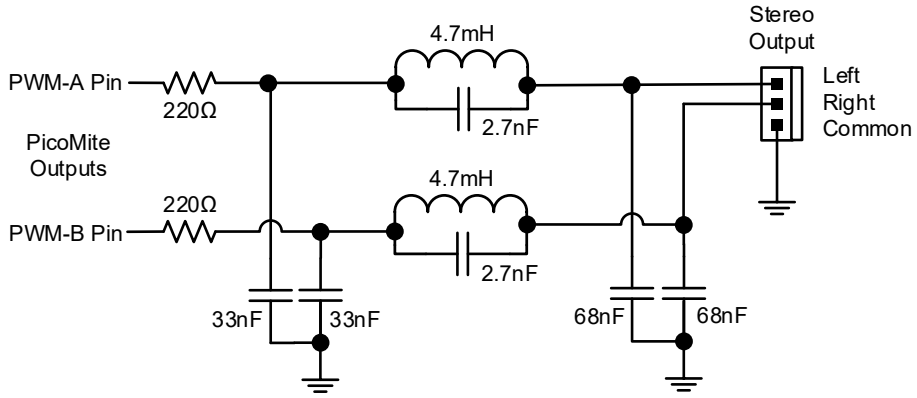
## Filterschaltungen

Die meisten kostengünstigen Aktivlautsprecher (für einen PC) reagieren nicht auf die Trägerfrequenz, sodass sie selbst als Tiefpassfilter wirken. Wenn du es also einfach halten willst, kannst du den PWM-Ausgang direkt an den Eingang eines Aktivlautsprechers anschließen, wodurch eine angemessene Tonausgabe erzielt werden sollte.

Allerdings kann der hohe Pegel des 44-kHz-Trägersignals Probleme für den Verstärker verursachen (z. B. Überhitzung oder Verzerrung), daher wird der folgende Filter empfohlen. Dieser entfernt den größten Teil des Trägersignals und liefert etwa 2 V Spitze-Spitze (0,6 V RMS) mit angemessener Klangtreue bis zu 8 kHz (mehr als genug für die meisten Aktivlautsprecher):




Unten siehst du eine hochwertige Schaltung, die einen erstklassigen Klang liefert und nur einen vernachlässigbaren Restträgeranteil aufweist. Sie eignet sich für eine anspruchsvollere HiFi-Verstärker-/Lautsprecherkonfiguration. Der Ausgang liefert einen Frequenzbereich von 10 Hz bis 15 kHz bei etwa 3 V Spitze-Spitze (1 V RMS) bei 1 kHz.




Beide Schaltungen sind dafür ausgelegt, einen Verstärker zu speisen (und nicht direkt einen Kopfhörer oder Lautsprecher anzusteuern) und nutzen eine Kondensator-Kopplung zum nachfolgenden Verstärker (die meisten verfügen darüber).

## VS1053-Unterstützung

Der Audioausgang kann mit einem VS1053-CODEC-Modul erzeugt werden, das mit dem Befehl

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

Dies erfordert keine Ausgangsfilterung und kann 32-Ω-Kopfhörer direkt ansteuern. Es unterstützt außerdem zusätzliche Wiedergabefunktionen.

Wenn ein VS1053-Codec als Audioausgabegerät verwendet wird, stehen zusätzliche Befehle zur Verfügung:

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

Diese werden im Abschnitt mit der Befehlsliste näher erläutert.

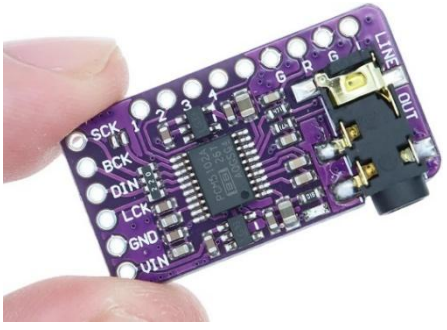
## MCP48n2-DAC-

Der Audioausgang kann auch über einen angeschlossenen MCP48n2-DAC (z. B. MCP4822) erzeugt werden. In diesem Fall wird er mit dem folgenden Befehl konfiguriert:

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

Der DAC benötigt keinen komplexen Tiefpassfilter; ein 120-Ω-Widerstand, der an den DAC-Ausgang angeschlossen ist und dessen anderes Ende über einen 100-nF-Kondensator mit GND verbunden ist, reicht aus. Bei Verwendung eines MCP4822 sollte der LDAC-Pin am DAC mit GND verbunden werden.

## I2S-DAC




Der Audioausgang kann auch über einen I2S-DAC wie den PCM5102A erzeugt werden. Der DAC muss die Erzeugung eines eigenen Master-Taktes unterstützen, da dieser nicht von der Firmware erzeugt wird. Der I2S-DAC auf dem Pico2 (RP2350A/B) nutzt PIO2 zur Erzeugung des Ausgangs, sodass dieser nicht verfügbar ist, wenn der I2S-DAC aktiviert ist. Der I2S-DAC auf dem RP2040 nutzt PIO 0, das bei VGA-Versionen gemeinsam mit dem VGA-PIO genutzt wird.

Der I2S-DAC wird mit dem Befehl konfiguriert:

```
OPTION AUDIO I2S BCLK-PIN, DIN-PIN
```

Der I2S-Word-Clock-Pin (LRCK) befindet sich dann am Pin neben dem BCLK-Pin. Wenn also beispielsweise BCLK auf GP0 gesetzt ist, befindet sich LRCK auf GP1. Sowohl diese beiden Pins als auch der DIN-Pin

müssen frei sein, wenn der Befehl ausgegeben wird. In der Regel benötigt das DAC-Modul außerdem einen GND- und einen Stromversorgungs-Pin (typischerweise 5 V).

Der I2S-DAC liefert Audio in CD-Qualität mit einem vollkommen linearen Frequenzgang von 20 Hz bis 20 kHz. Das funktioniert am besten mit FLAC-Dateien mit bis zu 96.000 Hz; 24-Bit-Formate wurden getestet. Allerdings liefern auch MP3-Dateien eine hohe Ausgabequalität, die nur durch die inhärente MP3-Komprimierung begrenzt ist.

## Wiedergabe von WAV-, FLAC-, MP3- und MOD-Dateien ( )

Mit dem Befehl PLAY kannst du eine WAV-, FLAC-, MP3- (nur RP2350) oder MOD-Datei, die sich auf dem Flash-Dateisystem oder der SD-Karte befindet, über den Audioausgang abspielen. Du kannst damit Hintergrundmusik abspielen, Programmen Soundeffekte hinzufügen und informative Ansagen machen.

Die Befehle lauten:

```
PLAY WAV file$, interrupt
PLAY FLAC file$, interrupt
PLAY MODFILE file$, interrupt
PLAY MP3 file$, interrupt 'RP2350 only
```

„file$“ ist der Name der abzuspielenden Audiodatei. Sie muss sich auf dem Flash-Dateisystem oder der SD-Karte befinden; fehlt die Dateiendung, wird die entsprechende Endung (z. B. .WAV) angehängt. Die Audiodatei wird im Hintergrund abgespielt (d. h., das Programm läuft ohne Unterbrechung weiter). „interrupt“ ist optional und bezeichnet den Namen einer Subroutine, die aufgerufen wird, sobald die Datei vollständig abgespielt wurde.

## Erzeugen von Sinuswellen

Der Befehl PLAY TONE nutzt den Audioausgang, um Sinuswellen mit wählbaren Frequenzen für den linken und rechten Kanal zu erzeugen. Diese Funktion ist dafür gedacht, aufmerksamkeitsstarke Töne zu erzeugen, aber da die Frequenz sehr genau ist, kann sie für viele andere Anwendungen genutzt werden. Zum Beispiel zum Senden von DTMF-Tönen über eine Telefonleitung oder zum Testen des Frequenzgangs von Lautsprechern.

Die Syntax des Befehls lautet:

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

„left“ und „right“ sind die Frequenzen in Hz, die für den linken und rechten Kanal verwendet werden sollen.

Der Ton wird im Hintergrund abgespielt (das Programm läuft nach diesem Befehl weiter) und „Dauer“ gibt die Anzahl der Millisekunden an, die der Ton erklingen soll. „Dauer“ ist optional und wenn sie nicht angegeben wird, ertönt der Ton so lange, bis er explizit gestoppt wird oder das Programm beendet wird. „Unterbrechung“ (falls angegeben) wird ausgelöst, wenn die Dauer abgelaufen ist.

Die angegebene Frequenz kann zwischen 1 Hz und 20 kHz liegen und ist sehr genau (sie basiert auf einem Quarzoszillator). Die Frequenz kann jederzeit durch einen neuen PLAY TONE-Befehl geändert werden.

## Verwendung von PLAY

Es ist wichtig zu wissen, dass der PLAY-Befehl den Ton im Hintergrund erzeugt. Dadurch kann ein Programm (zum Beispiel) den Klang einer Glocke abspielen, während es seine Steuerungsfunktion fortsetzt. Ohne diese Hintergrundfunktion würde das gesamte BASIC-Programm während der Tonwiedergabe einfrieren.

Die Erzeugung des Tons im Hintergrund hat jedoch einige subtile Auswirkungen, die Anfänger verwirren können. Nimm zum Beispiel das folgende Programm:

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

Du erwartest vielleicht, dass der 500-Hz-Ton 2 Sekunden lang ertönt, aber in der Praxis ist überhaupt kein Ton zu hören. Das liegt daran, dass MMBasic den Befehl PLAY TONE ausführt (wodurch die Tonerzeugung im Hintergrund startet) und dann sofort den Befehl END ausführt, der das Programm und den Hintergrundton beendet. Das geschieht so schnell, dass nichts zu hören ist.

Ebenso wird das folgende Programm nicht funktionieren:

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

Das liegt daran, dass der erste Befehl einen Ton mit 500 Hz abspielt, der zweite PLAY-Befehl diesen jedoch sofort durch einen Ton mit 300 Hz ersetzt und das Programm anschließend bis zum Ende läuft und das Programm (sowie den Hintergrundton) beendet, sodass nichts zu hören ist.

Wenn du möchtest, dass MMBasic wartet, während der PLAY-Befehl ausgeführt wird, solltest du entsprechende PAUSE-Befehle verwenden. Zum Beispiel:

```
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.

## Hilfsbefehle

Es gibt eine Reihe von Befehlen, mit denen du die Tonausgabe steuern kannst:

| | |
|---|---|
| `PLAY PAUSE` | Die aktuell abgespielte Datei oder den Ton vorübergehend anhalten (pausieren). |
| `PLAY RESUME` | Setzt die Wiedergabe einer zuvor angehaltenen Datei oder eines Tons fort. |
| `PLAY NEXT` | Die nächste WAV-, MP3- oder FLAC-Datei in einem Verzeichnis abspielen |
| `PLAY PREVIOUS` | Die vorherige WAV-, MP3- oder FLAC-Datei in einem Verzeichnis abspielen |
| `STOP` | Beende die Wiedergabe der Datei oder des Tons, genau wie beim Programmende. |
| `PLAY VOLUME L, R` | Stelle die Lautstärke auf einen Wert zwischen 0 und 100 ein, wobei 100 die maximale Lautstärke ist. Die Lautstärke wird beim Starten eines Programms auf den Maximalwert zurückgesetzt. Es wird eine logarithmische Skala verwendet, sodass PLAY VOLUME 50,50 halb so laut klingen sollte wie 100,100. |

## Spezielle Audioausgabe

Der Befehl PLAY SOUND erzeugt einen Ton, der auf einer Mischung aus Sinus-, Rechteck- und anderen Wellenformen basiert.  Details findest du in der Befehlsliste.

# Verwendung der I/O-Pins

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

Ein I/O-Pin wird durch seine Pin-Nummer bezeichnet, und das kann die Nummer (z. B. 2) oder seine GP-Nummer (z. B. GP1) sein. Zusätzlich kannst du für alle Befehle, die I/O-Pins verwenden, außer dem PORT-Befehl und der PORT-Funktion, auch eine String-Variable oder ein String-Literal in der Form „GPn“ verwenden.

## Digitale Eingänge

Ein digitaler Eingang ist die einfachste Art der Eingangskonfiguration. Liegt die Eingangsspannung über 2,5 V, ist der Logikpegel „wahr“ (numerischer Wert 1), und alles unter 0,65 V ist „falsch“ (numerischer Wert 0). Die Eingänge verwenden einen Schmitt-Trigger, sodass bei Werten zwischen diesen Pegeln der vorherige Logikpegel beibehalten wird.

Beachte, dass die maximale Spannung an den I/O-Pins des RP2040 (d. h. des Raspberry Pi Pico) 3,3 V beträgt. Eine Pegelumsetzung ist erforderlich, wenn ein Gerät 5-V-Pegel für die Signalübertragung verwendet. Der Raspberry Pi Pico 2 mit dem RP2350 verträgt 5 V (im eingeschalteten Zustand), daher ist in diesem Fall keine Pegelumsetzung für Signale bis zu 5 V erforderlich.

In deinem BASIC-Programm würdest du den Eingang als digitalen Eingang festlegen und die Funktion PIN() verwenden, um seinen Pegel abzurufen. Zum Beispiel:

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

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

Der SETPIN-Befehl unterstützt außerdem einige Optionen, die einen internen Widerstand vom Eingang entweder an die Versorgungsspannung oder an Masse anschließen. Dies wird als „Pull-up“- oder „Pull-down“-Widerstand bezeichnet und ist praktisch beim Anschluss an einen Schalter, da dadurch die Installation eines externen Widerstands zur Anlegung einer Spannung an den Kontakten entfällt. Aufgrund eines Hardwareproblems mit dem RP2350-Prozessor wird empfohlen, einen externen Widerstand von 8,2 kΩ oder weniger zu verwenden, wenn bei diesem Prozessor ein Pulldown erforderlich ist.

## Analoge Eingänge

Pins, die mit ADC gekennzeichnet sind, 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. Zum Beispiel:

```
> SETPIN 31, AIN
> PRINT PIN(31)
 2,345
>
```

Du benötigst einen Spannungsteiler, wenn du Spannungen über 3,3 V messen möchtest. Bei niedrigen Spannungen benötigst du möglicherweise einen Verstärker, um die Eingangsspannung in einen für die Messung geeigneten Bereich zu bringen.

Die Messung verwendet die 3,3-V-Versorgungsspannung der CPU als Referenz und geht davon aus, dass diese genau 3,3 V beträgt. Dieser Wert kann mit dem Befehl OPTION VCC geändert werden. Um den bestmöglichen Messwert zu erhalten, wird der analoge Eingang 10 Mal abgetastet. Die Werte werden dann sortiert, die obersten 2 und die untersten 2 verworfen und die verbleibenden 6 gemittelt.

Wenn du den direkten Messwert vom ADC haben möchtest, kannst du den Raw-Modus verwenden, indem du die Option ARAW beim Befehl SETPIN nutzt:

```
SETPIN pinno,ARAW
```

In diesem Fall wird basierend auf einer einzelnen Abtastung ein Wert zwischen 0 und 4095 zurückgegeben.

Die ADC-Befehle bieten eine alternative Methode zur Erfassung analoger Eingänge und sind für die schnelle Erfassung vieler Messwerte in einem Array gedacht.

## Zähl-Eingänge

Beliebige vier Pins können als Zähl-Eingänge 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, sind jedoch standardmäßig auf GP6, GP7, GP8 und GP9 eingestellt, sofern sie nicht geändert werden.

Als Beispiel gibt der folgende Befehl die Frequenz des Signals am Pin GP7 aus:

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

In diesem Fall beträgt die Frequenz 110,374 kHz.

Standardmäßig beträgt die Gate-Zeit eine Sekunde. Das ist die Zeitspanne, die MMBasic benötigt, 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 beim SETPIN-Befehl kannst du eine alternative Gate-Zeit zwischen 10 ms und 100000 ms festlegen. Kürzere Zeiten führen dazu, dass die Messwerte häufiger aktualisiert werden, der zurückgegebene Wert hat jedoch eine geringere Auflösung. Die Funktion PIN() skaliert die zurückgegebene Zahl immer auf die Frequenz in Hz, unabhängig von der verwendeten Gate-Zeit.

Beispielsweise wird mit dem folgenden Befehl die Gate-Zeit auf 10 ms gesetzt, was mit einem entsprechenden Verlust an Auflösung einhergeht:

```
> 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 die PicoMite-Firmware die Anzahl der Millisekunden zwischen aufeinanderfolgenden steigenden Flanken des Eingangssignals. Der Wert wird beim Übergang von Low nach High aktualisiert. Wenn dein Signal also eine Periode von (sagen wir) 100 Sekunden hat, solltest du damit rechnen, diese Zeit abzuwarten, bevor die PIN()-Funktion einen aktualisierten Wert zurückgibt.

Die Zählpins können auch die Anzahl der Impulse an ihrem Eingang zählen. Wenn ein Pin als Zähler konfiguriert ist (zum Beispiel `SETPIN 7,CIN`), wird der Zähler auf Null zurückgesetzt und die PicoMite-Firmware zählt dann jeden Übergang von einer niedrigen zu einer hohen Spannung. Der Zähler kann durch Ausführen `von PIN(7) = 0` wieder auf Null zurückgesetzt werden.

Zähleingänge sind bei der Standard-Prozessorfrequenz bis zu etwa 200 kHz genau. Eine minimale Impulsbreite von etwa 40 ns ist erforderlich, um den Zähler auszulösen. Der RP2350 bietet außerdem die Möglichkeit, GP1 als extrem schnellen Frequenzzähl-Pin zu konfigurieren (siehe den Befehl SETPIN GP1, FFIN).

## Digitale Ausgänge

Alle I/O-Pins können mithilfe des DOUT-Parameters des SETPIN-Befehls als digitaler Ausgang konfiguriert werden. Zum Beispiel:

```
SETPIN GP15, DOUT
```

Das bedeutet: Wenn ein Ausgangspin auf „Logic Low“ gesetzt wird, wird sein Ausgang auf Null gezogen, und wenn er auf „High“ gesetzt wird, wird sein Ausgang auf 3,3 V gezogen. In MMBasic geschieht dies mit dem Befehl PIN. Zum Beispiel setzt `PIN(GP15) = 0` den Pin GP15 auf Low, während `PIN(GP15) = 1` ihn auf High setzt.

## Pulsweitenmodulation

Mit dem Befehl PWM (Pulsweitenmodulation) kann die PicoMite-Firmware Rechteckwellen mit einem programmgesteuerten Tastverhältnis erzeugen.

Durch Variieren des Tastverhältnisses kannst du einen programmgesteuerten Spannungsausgang erzeugen, der zur Steuerung externer Geräte dient, die einen analogen Eingang benötigen (Stromversorgungen,

Motorsteuerungen usw.). Die PWM-Ausgänge eignen sich auch zum Ansteuern von Servos und zur Erzeugung eines Tonausgangs über einen kleinen Wandler.

RP2040 Die PWM-Ausgänge bestehen aus bis zu 8 Kanälen (nummeriert von 0 bis 7), wobei jeder Kanal über zwei Ausgänge (A und B) verfügt. Für jeden Kanal kann die Frequenz ausgewählt und für jeden Ausgang ein anderer Tastgrad eingestellt werden. Mit dem SETPIN-Befehl können bis zu 16 Pins als PWM-Ausgänge konfiguriert werden.

RP2350 Der RP2350 unterstützt bis zu 12 PWM-Kanäle (nummeriert von 0 bis 11) und bis zu 24 Pins können mit dem SETPIN-Befehl als PWM-Ausgänge konfiguriert werden.

## Kommunikationsschnittstellen (Seriell, SPI und I²C)

Diese werden in den Anhängen am Ende dieses Handbuchs beschrieben. Bevor diese Schnittstellen genutzt werden können, müssen die Pins, die für die entsprechenden Signale verwendet werden sollen, mit dem Befehl SETPIN konfiguriert werden.

Einige Geräte wie SD-Karten, LCD-Displays, Touchscreens usw. nutzen ebenfalls SPI- oder I2C-Schnittstellen, und die dafür verwendeten Pins müssen ebenfalls mit dem Befehl OPTION SYSTEM konfiguriert werden, bevor sie genutzt werden können.

### Interrupts

Interrupts sind eine praktische Möglichkeit, mit einem Ereignis umzugehen, das zu einem unvorhersehbaren Zeitpunkt auftreten kann. Ein Beispiel ist, wenn der Benutzer eine Taste drückt. In deinem Programm könntest du nach jeder Anweisung Code einfügen, um zu prüfen, ob die Taste gedrückt wurde, aber ein Interrupt sorgt für ein übersichtlicheres und besser lesbares Programm.

Wenn ein Interrupt auftritt, führt MMBasic eine definierte Subroutine aus und kehrt nach Abschluss zum Hauptprogramm zurück. Das Hauptprogramm nimmt den Interrupt überhaupt nicht wahr und läuft wie gewohnt weiter.

Jeder I/O-Pin, der als digitaler Eingang verwendet werden kann, lässt sich mit dem Befehl SETPIN so konfigurieren, dass er einen Interrupt auslöst, wobei bis zu zehn Interrupts gleichzeitig aktiv sein können. Interrupts können so eingerichtet werden, dass sie bei einem ansteigenden oder abfallenden digitalen Eingangssignal (oder beidem) auftreten, und bewirken einen sofortigen Sprung zu der angegebenen benutzerdefinierten Subroutine. Das Ziel kann für jeden Interrupt gleich oder unterschiedlich sein. Die Rückkehr aus einem Interrupt erfolgt über die Befehle END SUB oder EXIT SUB. Beachte, dass keine Parameter an die Subroutine übergeben werden können; innerhalb des Interrupts sind jedoch Aufrufe anderer Subroutinen und Funktionen erlaubt.

Wenn zwei oder mehr Interrupts gleichzeitig auftreten, werden sie in der unten definierten Reihenfolge verarbeitet. Während der Verarbeitung eines Interrupts werden alle anderen Interrupts deaktiviert, bis die Interrupt-Subroutine zurückkehrt. Während eines Interrupts (und zu jeder Zeit) kann mit der Funktion PIN() auf den Wert des Interrupt-Pins zugegriffen werden.

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

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

- Interrupts werden von MMBasic erst nach Abschluss jedes Befehls geprüft und nicht von der Hardware zwischengespeichert. Das bedeutet, dass ein kurzzeitiger Interrupt übersehen werden kann, insbesondere wenn das Programm Befehle ausführt, deren Ausführung einige Zeit in Anspruch nimmt. Die meisten Befehle werden in weniger als 15 µs ausgeführt, doch einige Befehle wie die Funktion TEMPR() können bis zu 200 ms dauern, sodass ein Interrupt innerhalb dieses Zeitfensters auftreten und wieder verschwinden und somit nicht erkannt werden kann.
- Während eines Interrupts werden alle anderen Interrupts blockiert, daher sollten deine Interrupts kurz sein und so schnell wie möglich beendet werden. Verwende zum Beispiel niemals PAUSE innerhalb eines Interrupts. Wenn du längere Verarbeitungsschritte durchführen musst, solltest du einfach ein Flag setzen und den Interrupt sofort beenden; dann kann deine Hauptprogrammschleife das Flag erkennen und die erforderlichen Maßnahmen ergreifen.

- Die Subroutine, die der Interrupt aufruft (und alle anderen von ihr aufgerufenen Subroutinen oder Funktionen), sollte immer exklusiv für den Interrupt sein. Wenn du eine Subroutine aufrufen musst, die auch von einem Interrupt verwendet wird, musst du den Interrupt zuerst deaktivieren (du kannst ihn wieder aktivieren, nachdem du mit der Subroutine fertig bist).
- Denk daran, einen Interrupt zu deaktivieren, wenn du ihn nicht mehr benötigst – Interrupts im Hintergrund können seltsame und nicht intuitive Fehler verursachen.

Zusätzlich zu Interrupts, die durch den Zustandswechsel eines I/O-Pins ausgelöst werden, können Interrupts auch von anderen Bereichen von MMBasic generiert werden, darunter Timer und Kommunikationsports, und die obigen Hinweise gelten auch für diese.

Die Liste all dieser Interrupts (in der Reihenfolge von hoher bis niedriger Priorität) lautet:

1. PID-Regelkreise
2. ON KEY individuell
3. ON KEY allgemein
4. ON PS2
5. PIO RX FIFO
6. PIO TX FIFO
7. PIO RX DMA-Abschluss
8. PIO TX DMA-Abschluss
9. GUI-Interrupt runter
10. GUI-Interrupt hoch
11. Sprite-Kollision
12. WebMite: TCP-Empfang
13. WebMite: MQTT- compete
14. WebMite: UDP-Empfang
15. USB-Gamecontroller/USB- oder PS2-Maus/Wii-Controller
16. ADC-Abschluss
17. I2C-Slave-Empfang (Rx)
18. I2C-Slave-Senden (Tx)
19. I2C2-Slave-Empfang (Rx)
20. I2C2-Slave-Senden (Tx)
21. WAV fertig
22. COM1: Serieller Anschluss
23. COM2: Serieller Anschluss
24. IR-Empfang
25. Keypad
26. Interrupt-Befehl/CSub-Interrupt
27. I/O-Pin-Interrupts in der Reihenfolge ihrer Definition
28. Tick-Interrupts (1 bis 4 in dieser Reihenfolge)

Ein Beispiel: Wenn ein ON-KEY-Interrupt gleichzeitig mit einem COM1:-Interrupt auftritt, wird zuerst die ON-KEY-Interrupt-Subroutine ausgeführt; erst wenn diese beendet ist, wird die COM1:-Interrupt-Subroutine ausgeführt.

# Unterstützung spezieller Geräte

Um die Interaktion eines Programms mit der Außenwelt zu vereinfachen, enthält die PicoMite-Firmware Treiber für eine Reihe gängiger Peripheriegeräte.

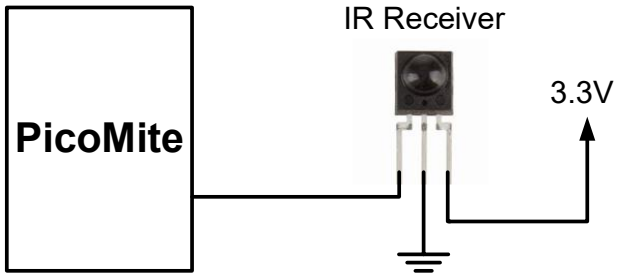
Das sind:

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

## Infrarot-Fernbedienungsdecoder

Du kannst deinem Projekt ganz einfach eine Fernbedienung hinzufügen, indem du den IR-Befehl verwendest. Wenn diese Funktion aktiviert ist, läuft sie im Hintergrund und unterbricht das laufende Programm, sobald eine Taste auf der IR-Fernbedienung gedrückt wird.

Sie funktioniert mit allen NEC- oder Sony-kompatiblen Fernbedienungen, einschließlich solcher, die erweiterte Nachrichten generieren. Die meisten günstigen programmierbaren Fernbedienungen generieren eines dieser Protokolle, und mit einer davon kannst du deinem Pico-basierten Projekt einen raffinierten Touch 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, benötigst du einen IR-Empfänger. NEC-Fernbedienungen verwenden eine 38-kHz-Modulation des IR-Signals; geeignete, auf diese Frequenz abgestimmte Empfänger sind unter anderem der Vishay TSOP4838, der Jaycar ZD1952 und der Altronics Z1611A. Beachte, dass die I/O-Pins des Raspberry Pi Pico nur 3,3 V vertragen und der Empfänger daher mit maximal 3,3 V betrieben werden muss. Der Raspberry Pi Pico 2 ist anders und hält 5 V aus.

Hinweis: Aufgrund eines Hardware-Fehlers im RP2350 vor der Version A4 wird dringend empfohlen, bei Verwendung des RP2350 einen 4K7-Pullup-Widerstand an die Datenleitung des IR-Empfängers anzuschließen.

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

Der IR-Empfänger kann an jeden beliebigen Pin des Raspberry Pi Pico angeschlossen werden. Dieser Pin muss vom Programm mit dem Befehl konfiguriert werden:

```
SETPIN n, IR
```

wobei *n* der für diese Funktion zu verwendende I/O-Pin ist.

Um den Decoder einzurichten, verwendest du den Befehl:

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

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

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

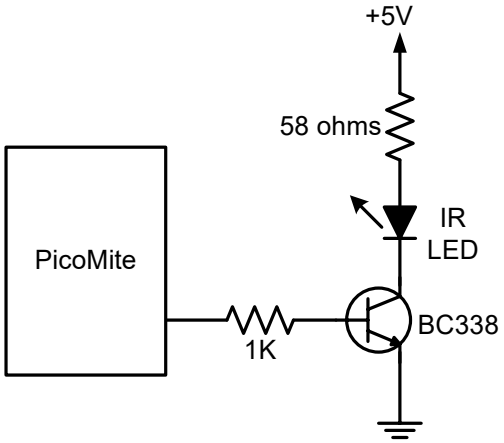
```
SETPIN GP6, IR                  ' den zu verwendenden Pin definieren
DIM INTEGER DevCode, KeyCode    ' vom Decoder verwendete Variablen
IR DevCode, KeyCode, IRInt      ' starte den IR-Decoder
```

```
DO
  ' < Hauptteil des Programms >
LOOP

SUB IRInt                          ' Ein Tastendruck wurde erkannt
  PRINT "Empfangenes Gerät = " DevCode "  Taste = " KeyCode
END SUB
```

IR-Fernbedienungen können viele verschiedene Geräte (Videorekorder, Fernseher usw.) ansteuern, daher würde das Programm normalerweise zuerst den Gerätecode prüfen, um festzustellen, ob das Signal für das Programm bestimmt war, und wenn ja, dann entsprechend der gedrückten Taste reagieren. Da es viele verschiedene Geräte und Tastencodes gibt, ist die beste Methode, um herauszufinden, welche Codes deine Fernbedienung generiert, die Verwendung des obigen Programms, um die Codes zu ermitteln.

## Infrarot-Fernbedienungssender




Mit dem Befehl IR SEND kannst du ein 12-Bit-Infrarot-Fernbedienungssignal von Sony senden. Dies ist für die Kommunikation zwischen Raspberry Pi Pico und Raspberry Pi Pico oder Micromite gedacht, funktioniert aber auch mit Sony-Geräten, die 12-Bit-Codes verwenden. Beachte, dass bei allen Sony-Produkten die Nachricht dreimal mit einer Verzögerung von 26 ms zwischen den einzelnen Nachrichten gesendet werden muss.

Die Schaltung auf der rechten Seite veranschaulicht, was erforderlich ist. Der Transistor wird verwendet, um die Infrarot-LED anzusteuern, da die Ausgangsleistung des Raspberry Pi Pico begrenzt ist. Diese Schaltung liefert etwa 50 mA an die LED.

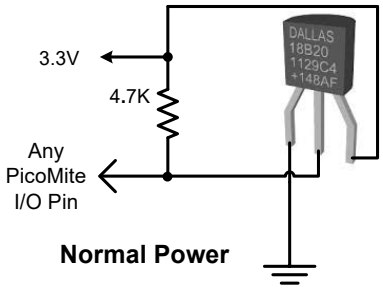
Um ein Signal zu senden, verwendest du den Befehl:

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

Dabei ist pin der verwendete I/O-Pin, dev der zu sendende Gerätecode und key der Schlüsselcode. Jeder I/O-Pin des Raspberry Pi Pico kann verwendet werden, und du musst ihn nicht vorher einrichten (IR SEND erledigt das automatisch).

Die verwendete Modulationsfrequenz beträgt 38 kHz und entspricht den gängigen IR-Empfängern (wie auf der vorherigen Seite beschrieben), um maximale Empfindlichkeit bei der Kommunikation zwischen zwei Raspberry Pi Picos oder mit einem Micromite zu gewährleisten.

## Temperaturmessung



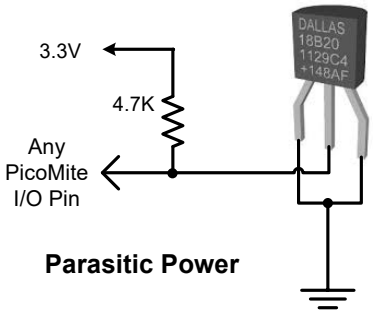

Die Funktion TEMPR() ruft die Temperatur von einem DS18B20-Temperatursensor ab. Dieses Gerät ist bei eBay für etwa 5 US-Dollar in verschiedenen Ausführungen erhältlich, darunter auch eine wasserdichte Sondenversion.

Der DS18B20 kann separat über eine 3,3-V-Stromversorgung betrieben werden oder, wie rechts dargestellt, über parasitäre Energie vom Raspberry Pi Pico. Es können mehrere Sensoren verwendet werden, allerdings sind für jeden Sensor ein separater I/O-Pin und ein 4,7-kΩ-Pullup-Widerstand erforderlich.

Um die aktuelle Temperatur abzurufen, verwendest du einfach die Funktion TEMPR() in einem Ausdruck. Zum Beispiel:

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

Dabei ist „pin“ der I/O-Pin, an den der Sensor angeschlossen ist. Du musst den I/O-Pin nicht konfigurieren, das übernimmt MMBasic.




Der Rückgabewert liegt in Grad Celsius mit einer Auflösung von 0,25 °C und einer Genauigkeit von ±0,5 °C. Tritt während der Messung ein Fehler auf, beträgt der Rückgabewert 1000.

Die für die gesamte Messung benötigte Zeit beträgt 200 ms, und das laufende Programm wird für diesen Zeitraum angehalten, während die Messung durchgeführt wird. Das bedeutet auch, dass Interrupts für diesen Zeitraum deaktiviert sind. Wenn du das nicht möchtest, kannst du die Messung separat mit dem Befehl TEMPR START auslösen und später die Funktion TEMPR() verwenden, um den Temperaturwert abzurufen. Die Funktion TEMPR() wartet immer, wenn der Sensor noch mit der Messung beschäftigt ist.

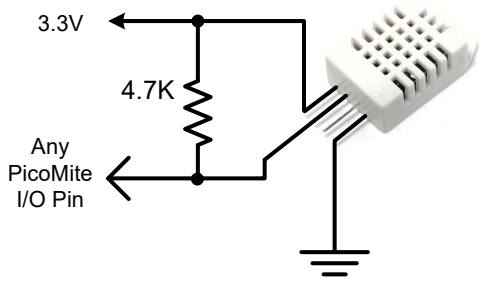
Zum Beispiel:

```
TEMPR START GP15
< andere Aufgaben ausführen >
PRINT "Temperatur: " TEMPR(GP15)
```

Der Befehl TEMPR START kann auch verwendet werden, um die Auflösung der Messung (von der Standardeinstellung 0,25 °C) und die damit verbundene Messzeit zu ändern.

## Luftfeuchtigkeit und Temperatur messen

Der Befehl HUMID liest die Luftfeuchtigkeit und Temperatur von einem DHT22-Feuchte-/Temperatursensor aus. Dieses Gerät wird auch als RHT03 oder AM2302 verkauft, aber alle sind kompatibel und können bei eBay für unter 5 $ gekauft werden. Der DHT11-Sensor wird ebenfalls unterstützt.




Der DHT22 muss mit 3,3 V (oder bis zu 5 V beim Raspberry Pi Pico 2) versorgt werden und sollte, wie abgebildet, einen Pull-up-Widerstand an der Datenleitung haben. Dies eignet sich für lange Kabelstrecken (bis zu 20 Meter), aber bei kurzen Strecken kann der Widerstand weggelassen werden, da die PicoMite-Firmware auch einen internen schwachen Pull-up bereitstellt.

Um die Temperatur oder Luftfeuchtigkeit abzurufen, verwendest du den Befehl HUMID mit drei Argumenten wie folgt:

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

Dabei ist „pin“ der I/O-Pin, an den der Sensor angeschlossen ist. Der I/O-Pin wird automatisch von MMBasic konfiguriert.

„tVar“ ist eine Gleitkommavariable, in der die Temperatur zurückgegeben wird, und „hVar“ ist eine zweite Variable für die Luftfeuchtigkeit. Die Temperatur wird in Grad Celsius mit einer Genauigkeit von einer Dezimalstelle (z. B. 23,4) zurückgegeben, und die Luftfeuchtigkeit wird als relative Luftfeuchtigkeit in Prozent (z. B. 54,3) zurückgegeben.

Wenn der optionale Parameter „DHT11“ auf 1 gesetzt ist, verwendet der Befehl die für dieses Gerät geeigneten Zeitvorgaben. In diesem Fall werden die Ergebnisse mit einer Auflösung von 1 Grad und 1 % Luftfeuchtigkeit zurückgegeben

Dieses Beispiel zeigt, wie man mit dem DHT22 die aktuelle Temperatur und Luftfeuchtigkeit jede Sekunde anzeigt:

```
DIM FLOAT temp, humidity
DO
  HUMID GP15, temp, humidity
  PRINT "Die Temperatur beträgt" temp _
        "und die Luftfeuchtigkeit beträgt" humidity
  PAUSE 1000
LOOP
```

## Echtzeituhr-Schnittstelle

Mit dem RTC-Befehl GETTIME lässt sich die aktuelle Uhrzeit ganz einfach von einer Echtzeituhr vom Typ PCF8563, DS1307, DS3231 oder DS3232 sowie von kompatiblen Geräten wie dem M41T11 abrufen. Diese integrierten Schaltkreise sind beliebt und günstig, halten die Zeit auch bei unterbrochener Stromversorgung genau und sind bei eBay für 2 bis 8 US-Dollar erhältlich. Komplette Module inklusive Batterie sind bei eBay für etwas mehr Geld erhältlich.

Der PCF8563 und der DS1307 weichen über einen Monat hinweg um höchstens ein bis zwei Minuten ab, während der DS3231 und der DS3232 besonders präzise sind und über ein Jahr hinweg auf eine Minute genau bleiben.

Die Firmware V6.01.01 oder höher unterstützt den RV3028, der eine Genauigkeit von ±1 ppm (30 Sekunden im Jahr) aufweist.

Diese Chips sind I²C-Geräte und sollten an die I²C-I/O-Pins des Raspberry Pi Pico angeschlossen werden.

An den I²C-I/O-Pins sind interne Pull-up-Widerstände (100 kΩ) vorhanden, sodass in vielen Fällen keine externen Widerstände benötigt werden.

Um die RTC zu aktivieren, musst du zunächst die zu verwendenden I²C-Pins mit dem folgenden Befehl zuweisen:

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

Die vom RTC verwendete Zeit muss ebenfalls eingestellt werden. Dies geschieht mit dem Befehl RTC SETTIME, der das folgende Format verwendet:

```
RTC SETTIME Jahr, Monat, Tag, Stunde, Minute, Sekunde
```

Beachte, dass die Uhrzeit im 24-Stunden-Format angegeben werden muss.

Beispielsweise wird mit dem folgenden Befehl die Echtzeituhr auf 16:00 Uhr am 10. November 2025 eingestellt:

```
RTC SETTIME 2025, 11, 10, 16, 0, 0
```

Um die Uhrzeit abzurufen, verwendest du den Befehl RTC GETTIME, der die Uhrzeit vom Echtzeituhr-Chip ausliest und die Uhr im Raspberry Pi Pico einstellt. Normalerweise wird dieser Befehl am Anfang des Programms oder in der Subroutine MM.STARTUP platziert, damit die Uhrzeit beim Hochfahren eingestellt wird.

Der Befehl OPTION RTC AUTO ENABLE kann auch verwendet werden, um eine automatische Aktualisierung der schreibgeschützten Variablen TIME$ und DATE$ vom Echtzeituhr-Chip beim Booten und stündlich einzustellen.

## Entfernung messen


Mit einem HC-SR04-Ultraschallsensor und der Funktion DISTANCE() kannst du 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 funktioniert, indem es einen Ultraschallimpuls aussendet und die Zeit misst, die das Echo benötigt, um zurückzukommen.

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). Andere, von denen berichtet wird, dass sie gut funktionieren, verwenden den CS100-Chip – wie beispielsweise der HC-SR04 und der US-025.

In der PicoMite-Firmware verwendest du die DISTANCE-Funktion wie folgt:

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

Der zurückgegebene Wert ist die Entfernung zum Ziel in Zentimetern.

Dabei ist trig der I/O-Pin, der mit dem „trig"-Eingang des Sensors verbunden ist, und echo der Pin, der mit dem „echo"-Ausgang des Sensors verbunden ist. Du kannst auch 3-Pin-Geräte verwenden; in diesem Fall wird nur eine Pin-Nummer angegeben.

Beachte, dass die maximale Spannung an allen I/O-Pins des Raspberry Pi Pico 3,3 V beträgt. Für diesen Sensor ist eine Pegelumsetzung erforderlich, da er für seinen Echo-Ausgang 5-V-Pegel verwendet. Der Raspberry Pi Pico 2 verträgt 5 V (im eingeschalteten Zustand), daher ist in diesem Fall keine Pegelumsetzung erforderlich.

## LCD-Anzeige

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

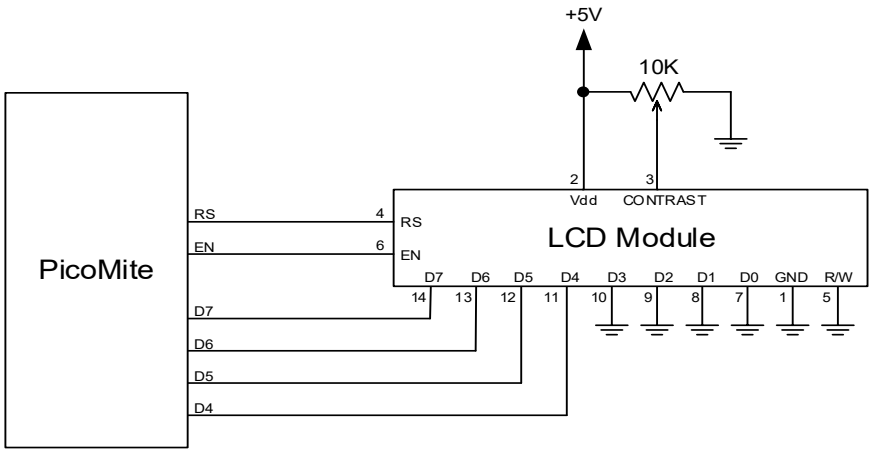
Dieser Befehl funktioniert mit LCD-Modulen, die den Controller-Chip KS0066, HD44780 oder SPLC780 verwenden und über 1, 2 oder 4 Zeilen verfügen. Typische Displays sind das LCD16X2 (futurlec.com), das Z7001 (altronics.com.au) und das QP5512 (jaycar.com.au). eBay ist eine weitere gute Bezugsquelle, wo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

Um das Display einzurichten, verwendest du den Befehl DEVICE LCD INIT:

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

d4, d5, d6 und d7 sind die Nummern der I/O-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 werden). „rs" ist der Pin, der mit dem Registerauswahl-Eingang am Modul verbunden ist (manchmal auch als CMD oder DAT bezeichnet). „en" ist der Pin, der mit dem Freigabe- oder Chipauswahl-Eingang am Modul verbunden ist.

Es können beliebige I/O-Pins des Raspberry Pi Pico verwendet werden, und du musst sie nicht vorher einrichten (der LCD-Befehl erledigt das automatisch für dich). Im Folgenden wird eine typische Konfiguration gezeigt.




Um Zeichen auf dem Modul anzuzeigen, verwendest du den LCD-Befehl:

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

Dabei ist „line“ die Zeile auf dem Display (1 bis 4) und „pos“ die Position in der Zeile, an der die Daten geschrieben werden sollen (die erste Position in der Zeile ist 1). „data$“ ist eine Zeichenkette, die die Daten enthält, die auf das LCD-Display geschrieben werden sollen. Die Zeichen in „data$“ überschreiben alles, was sich zuvor an dieser Stelle auf dem LCD befand.

Im Folgenden wird eine typische Anwendung gezeigt, bei der d4 bis d7 an die Pins GP2 bis GP5 angeschlossen sind, rs an Pin GP6 und en an Pin GP7.

```
LCD INIT GP2, GP3, GP4, GP5, GP6, GP7
LCD 1, 2, „Temperatur“
LCD 2, 6, STR$(TEMPR(GP15))    ' DS18B20 an Pin GP15 angeschlossen
```

Beachte, dass dieses Beispiel ebenfalls die Funktion TEMPR() verwendet, um die Temperatur abzurufen (wie oben beschrieben).

Außerdem unterstützt die Firmware die weit verbreiteten LCD-Displays mit I2C-Backpack, die die I2C-I/O-Expander-Chips der PCF8574-Serie verwenden. Details findest du im Befehl I2CLCD.

## Tastatur-Schnittstelle

Eine Tastatur ist eine einfache, aber effektive Methode zur Eingabe numerischer Daten. Die PicoMite-Firmware 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 ausgelöst, den das Programm verarbeiten kann.

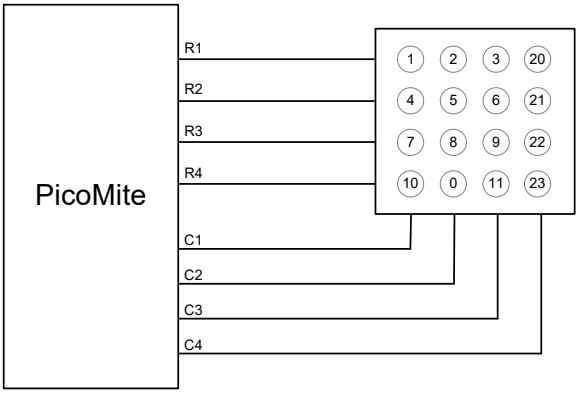
Beispiele für eine 4x3-Tastatur und eine 4x4-Tastatur sind die Altronics S5381 und S5383 (siehe www.altronics.com ).

Um die Tastaturfunktion zu aktivieren, verwendest du den Befehl:

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

, der aufgerufen wird, wenn ein neuer Tastendruck erkannt wurde. „r1“, „r2“, „r3“ und „r4“ sind die Pin-Nummern für die vier Zeilenanschlüsse des Tastenfelds (siehe Abbildung unten) und „c1“, „c2“, „c3“ und „c4“ sind die Spaltenanschlüsse. „c4“ wird nur bei 4x4-Tastaturen verwendet und sollte weggelassen werden, wenn du eine 4x3-Tastatur verwendest.

Es können beliebige I/O-Pins des Raspberry Pi Pico verwendet werden, und du musst sie nicht vorher einrichten – der KEYPAD-Befehl erledigt das automatisch für dich.




Die Erkennung und Dekodierung von Tastendrücken erfolgt im Hintergrund, und das Programm läuft nach diesem Befehl ohne Unterbrechung weiter. Wenn ein Tastendruck erkannt wird, wird der Wert der Variablen var auf die Zahl gesetzt, die die Taste repräsentiert (das ist die Zahl in den Kreisen in der obigen Abbildung). Dann wird der Interrupt aufgerufen.

Zum Beispiel:

```
Keypad KeyCode, KP_Int,GP2,GP3,GP4,GP5,GP6,GP7,GP8    ' 4x3-Tastatur
DO
  < Programmkörper >
LOOP

SUB KP_Int                              ' Ein Tastendruck wurde erkannt
  PRINT "Tastendruck = " KeyCode
END SUB
```

**Siehe auch den erweiterten KEYPAD-Befehl in der Befehlsbeschreibung, der eine beliebige Anzahl von Zeilen und Spalten mit benutzerdefinierten Rückgabecodes ermöglicht.**

## WS2812-Unterstützung

Die PicoMite-Firmware bietet integrierte Unterstützung für den WS2812-Mehrfarben-LED-Chip. Dieser Chip benötigt ein sehr spezifisches Timing, um ordnungsgemäß zu funktionieren, und mit dem Befehl DEVICE WS2812 lassen sich diese Geräte mit minimalem Aufwand leicht steuern.

Dieser Befehl gibt die erforderlichen Signale aus, um eine Kette von WS2812-LED-Chips anzusteuern, die an den angegebenen Pin angeschlossen sind, und legt die Farben jeder LED in der Kette fest. Die Syntax des Befehls lautet:

```
WS2812 Typ, Pin, nbr%, Farben%[()]
```

Beachte, dass der Pin auf einen digitalen Ausgang gesetzt sein muss, bevor dieser Befehl verwendet wird. Das Array colours%() sollte so dimensioniert sein, dass es mindestens genauso viele Elemente enthält wie die Anzahl der anzusteuernden LEDs (nbr%). Jedes Element im Array sollte die Farbe im normalen RGB888-Format (0 – HFFFFFF) enthalten. Wenn eine einzelne LED angesteuert werden soll, sollte colours% eine einfache Variable sein.

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

'type' ist ein einzelnes Zeichen, das den Typ des angesteuerten Chips wie folgt angibt:

O = Original WS2812

B = WS2812B

S = SK6812

W = SK6812W (RGBW)

Zum Beispiel:

```
DIM b%(4)=(RGB(red), Rgb(green), RGB(blue), RGB(Yellow), rgb(cyan))
SETPIN GP5, DOUT
WS2812 O, GP5, 5, b%()
```

gibt die angegebenen Farben an eine Reihe von fünf WS2812-LEDs aus, die über den Pin GP5 in Reihe geschaltet sind.

Es kann sein, dass ein WS2812 mit dem 3,3-V-Ausgang des Raspberry Pi Pico nicht zuverlässig funktioniert. In diesem Fall gibt es mehrere Lösungen:

- Verwende den WS2812B, der mit einer 3,3-V-Versorgung und -Eingängen funktioniert.
- Verwende den Raspberry Pi Pico 2, der 5 V verträgt (im eingeschalteten Zustand), sodass in diesem Fall keine Pegelumsetzung erforderlich ist.
- Verwende einen einzelnen WS2812, der mit 3,3 V betrieben wird, als erste Stufe, um den Eingang der ersten „echten“ LED in der Kette zu puffern. Die Mindestversorgungsspannung für den WS2812 beträgt 4 V, aber in vielen Fällen funktioniert er auch bei 3,3 V.

## OV7670-Kameramodul

Die PicoMite-Firmware unterstützt ein OV7670-Kameramodul. Details findest du unter dem Befehl CAMERA

# Display Panels

**NICHT VERFÜGBAR BEI HDMI- ODER VGA-VERSIONEN**

Die PicoMite-Firmware unterstützt viele LCD-Displays, die eine SPI-, I²C- oder parallele Schnittstelle verwenden.

Diese Befehle müssen an der Befehlszeile (nicht in einem Programm) eingegeben werden und führen zu einem Neustart der PicoMite-Firmware. Dies hat zur Folge, dass die USB-Konsolenverbindung unterbrochen wird und neu hergestellt werden muss.

Beachte, dass die maximale Spannung an allen I/O-Pins des Raspberry Pi Pico 3,3 V beträgt. Für Displays, die 5-V-Pegel zur Signalübertragung verwenden, ist eine Pegelumsetzung erforderlich. Der Raspberry Pi Pico 2 verträgt 5 V (im eingeschalteten Zustand), daher ist in diesem Fall keine Pegelumsetzung erforderlich.

## System-SPI-Bus

Der System-SPI-Bus des ist ein dedizierter SPI-Kanal, der von vielen LCD-Panels, allen Touch-Controllern und auch für die Kommunikation mit einer SD-Karte genutzt wird. Wenn eines dieser Geräte angeschlossen ist, müssen die für den System-SPI-Bus verwendeten I/O-Pins zuerst definiert werden.

Dies geschieht mit folgendem Befehl:

```
OPTION SYSTEM SPI CLK-Pin, MOSI-Pin, MISO-Pin
```

Dies muss an der Befehlszeile eingegeben werden und führt dazu, dass die Firmware neu startet und die USB-Konsolenverbindung getrennt wird, die anschließend wiederhergestellt werden muss. Diese Option wird beim Start erneut angewendet, und die Pins werden reserviert und stehen für andere Zwecke nicht zur Verfügung.

Ein typisches Beispiel ist:

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

Beachte, dass die Geschwindigkeit beim Zeichnen auf SPI-basierten Displays und beim Zugriff auf SD-Karten nicht von der CPU-Geschwindigkeit beeinflusst wird.

## SPI-basierte Display-Panels

Diese Display-Panels werden mit den folgenden Befehlen konfiguriert. Für alle muss zuvor der System-SPI-Bus (siehe oben) definiert worden sein.

In allen Befehlen lauten die Parameter:

| | |
|---|---|
| OR | 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 Präfix R kennzeichnet die umgekehrte oder „auf dem Kopf stehende“ Ausrichtung. |
| DC | Steuerpin für Anzeigedaten/Befehle. |
| RESET | Display-Reset-Pin (wenn auf Low gesetzt). |
| CS | Display-Chip-Select-Pin (aktiv bei Low). |
| BL | Optionaler Pin zur Steuerung der Helligkeit der Hintergrundbeleuchtung mittels Pulsweitenmodulation (PWM). |
| INVERT | Diese Option bewirkt eine Invertierung der Farben, um ein nicht standardmäßiges Panel auszugleichen. |

```
OPTION LCDPANEL ILI9341, OR, DC, RESET, CS [,BL] [,INVERT]
```

Initialisiert ein TFT-Display mit dem Controller „ “ ILI9341. Dieser unterstützt eine Auflösung von 320 × 240 . Displays mit diesem Controller können transparenten Text anzeigen und funktionieren mit den Befehlen BLIT und BLIT READ.

```
OPTION LCDPANEL ILI9163, OR, DC, RESET, CS [,BL] [,INVERT]
```

Initialisiert ein TFT-Display mit dem ILI9163-Controller. Dieser unterstützt eine Auflösung von 128 × 128.

```
OPTION LCDPANEL ILI9481, OR, DC, RESET, CS [,BL] [,INVERT]
```

Initialisiert ein TFT-Display mit dem ILI9481-Controller. Dieser unterstützt die Auflösungen 480 * 320 .

`OPTION LCDPANEL ILI9481IPS, OR, DC, RESET, CS [,BL] [,INVERT]`
Initialisiert ein IPS-Display mit dem ILI9481-Controller. Dies unterstützt eine Auflösung von 480 * 320.

`OPTION LCDPANEL ILI9488, OR, DC, RESET, CS [,BL] [,INVERT]`
Initialisiert ein TFT-Display mit dem ILI9488-Controller von . Dieser unterstützt eine Auflösung von 480 × 320 . Beachte, dass dieser Controller ein Problem mit dem LCD_SDO-Pin (MISO) hat, der den Touch-Controller stört. Damit dieses Display funktioniert, darf der LCD_SDO-Pin nicht direkt mit dem System-SPI-MISO verbunden werden.

`OPTION LCDPANEL ILI9488P, OR, DC, RESET, CS [,BL] [,INVERT]`
Initialisiert ein TFT-Display mit dem ILI9488-Controller. Dies unterstützt eine Auflösung von 320 × 320. Beachte, dass dieser Controller ein Problem mit dem LCD_SDO-Pin (MISO) hat, der den Touch-Controller stört. Damit dieses Display funktioniert, darf der LCD_SDO-Pin nicht direkt mit dem System-SPI-MISO verbunden werden. Diese Konfiguration unterstützt den PicoCalc mit dem Display im Hochformat.

`OPTION LCDPANEL ILI9488W, OR, DC, RESET, CS [,BL] [,INVERT]`
Initialisiert ein TFT-Display mit dem ILI9488-Controller. Dies unterstützt das Waveshare 3,5"-Display, wie es auf dem Pico Eval-Board verwendet wird, sowie den normalen 3,5"-Displayadapter.

`OPTION LCDPANEL N5110, OR, DC, RESET, CS [,contrast] [,INVERT]`
Initialisiert ein LCD-Display mit dem Nokia 5110-Controller. Dies unterstützt eine Auflösung von 84 × 48. Ein zusätzlicher Parameter „contrast“ kann angegeben werden, um den Kontrast des Displays zu steuern. Probiere Kontrastwerte zwischen &HA8 und &HD0 aus, um sie an dein Display anzupassen; der Standardwert bei Auslassung ist &HB1

`OPTION LCDPANEL SSD1306SPI, OR, DC, RESET, CS [,offset] [,INVERT]`
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 des Displays zu steuern. 0,96-Zoll-Displays benötigen typischerweise den Wert 0. 1,3-Zoll-Displays benötigen typischerweise den Wert 2. Der Standardwert bei Auslassung ist 0.

`OPTION LCDPANEL SSD1331, OR, DC, RESET, CS [,BL] [,INVERT]`
Initialisiert ein Farb-OLED-Display mit dem SSD1331-Controller. Dies unterstützt eine Auflösung von 96 × 64.

`OPTION LCDPANEL ST7735, OR, DC, RESET, CS [,BL] [,INVERT]`
Initialisiert ein TFT-Display mit dem ST7735-Controller. Unterstützt eine Auflösung von 160 × 128.

`OPTION LCDPANEL ST7735S, OR, DC, RESET, CS [,BL] [,INVERT]`
Initialisiert ein IPS-Display mit dem ST7735S-Controller. Dies unterstützt eine Auflösung von 160 × 80.

`OPTION LCDPANEL ST7735S_W, OR, DC, RESET, CS [,BL][,INVERT]`
Initialisiert ein Waveshare 128x128 ST7735S-Display. Dies unterstützt eine Auflösung von 128 × 128.

`OPTION LCDPANEL ST7789, OR, DC, RESET, CS [,BL] [,INVERT]`
Initialisiert ein IPS-Display mit dem 7789-Controller. Dies unterstützt eine Auflösung von 240 × 240.
HINWEIS: Display-Boards ohne CS-Pin werden derzeit in der PicoMite-Firmware nicht unterstützt, es sei denn, sie wurden modifiziert.

`OPTION LCDPANEL ST7789_135, OR, DC, RESET, CS [,BL][,INVERT]`

Initialisiert ein IPS-Display mit dem 7789-Controller. Dies unterstützt eine Auflösung von 240 × 135.
HINWEIS: Displayplatinen ohne CS-Pin werden derzeit in der PicoMite-Firmware nicht unterstützt, es sei denn, sie wurden modifiziert.

```
OPTION LCDPANEL ST7789_320, OR, DC, RESET,CS [,BL][,INVERT]
```

Initialisiert ein IPS-Display mit dem 7789-Controller. Dieser Typ unterstützt die 320 × 240-Auflösung von Waveshare ( https://www.waveshare.com/wiki/Pico-ResTouch-LCD-2.8 ).
Diese Displays unterstützen transparenten Text und funktionieren mit den Befehlen BLIT und BLIT READ.
HINWEIS: Display-Boards ohne CS-Pin werden derzeit in der PicoMite-Firmware nicht unterstützt, es sei denn, sie wurden modifiziert.

```
OPTION LCDPANEL ST7796S, OR, DC, RESET,CS [,BL][,INVERT]
```

Initialisiert ein IPS-Display mit dem ST7796S-Controller. Dies unterstützt eine Auflösung von 480 × 320.
HINWEIS: Damit transparenter Text und BLIT ordnungsgemäß funktionieren, sollte die Diode D1 auf der Rückseite des Displays überbrückt werden, und es wird empfohlen, auch J1 zu überbrücken, um mit 3,3 V zu arbeiten

```
OPTION LCDPANEL ST7796SP, OR, DC, RESET,CS [,BL][,INVERT]
```

Initialisiert ein TFT-Display mit dem ST7796S-Controller. Dieser unterstützt eine Auflösung von 320 × 320.
Diese Konfiguration unterstützt den PicoCalc mit dem Display im Hochformat.

```
OPTION LCDPANEL GC9A01, OR, DC, RESET, CS [,BL] [,INVERT]
```

Initialisiert ein IPS-Display mit dem GC9A01-Controller. Dies unterstützt eine Auflösung von 240 × 240.

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

Initialisiert ein LCD-Display mit dem ST7920-Controller. Dies unterstützt eine Auflösung von 128 × 64.
Beachte, dass dieses Display keinen Chip-Select-Pin unterstützt, sodass der SPI-Bus nicht gemeinsam genutzt werden kann, wenn dieses Display verwendet wird.

## I²C-basierte LCD-Panels

Alle I²C-basierten Display-Controller nutzen die I²C-Pins des Systems gemäß der Pinbelegung des jeweiligen Geräts. Andere I²C-Geräte können den Bus gemeinsam nutzen, sofern ihre Adressen eindeutig sind.

Um den System-I²C-Bus einzurichten, verwende den Befehl:

```
   OPTION SYSTEM I2C sdapin, sclpin
```

Wenn ein I²C-Display konfiguriert ist, ist es nicht notwendig, den I²C-Port für ein zusätzliches Gerät zu „öffnen“ (I2C OPEN), I2C CLOSE ist gesperrt, und alle I²C-Geräte müssen für den Betrieb mit 100 kHz ausgelegt sein. Die Geschwindigkeit des I²C-Busses wird durch Änderungen der CPU-Taktfrequenz nicht beeinflusst

Diese Panels werden mit den folgenden Befehlen konfiguriert.  In allen Befehlen ist der Parameter OR die Ausrichtung des Displays und kann LANDSCAPE, PORTRAIT, RLANDSCAPE oder RPORTRAIT lauten. Diese können mit L, P, RL oder RP abgekürzt werden. Das Präfix R kennzeichnet die umgekehrte oder „auf dem Kopf stehende“ Ausrichtung.

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

Initialisiert ein OLED-Display unter Verwendung des SSD1306- -Controllers mit I²C-Schnittstelle. Dieser unterstützt eine Auflösung von 128 × 64. Ein zusätzlicher Parameter „offset“ kann angegeben werden, um die Position des Displays zu steuern. 0,96-Zoll-Displays benötigen typischerweise den Wert 0. 1,3-Zoll-Displays benötigen typischerweise den Wert 2. Der Standardwert bei Auslassung ist 0.

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

Der SSD1306I2C-Treiber funktioniert auch mit SSD1315- und SH1106-Controllern.

```
OPTION LCDPANEL SSD1306I2C32, ODER
```

Initialisiert ein OLED-Display mithilfe des SSD1306-Controllers mit einer I²C-Schnittstelle. Es unterstützt eine Auflösung von 128 × 32.

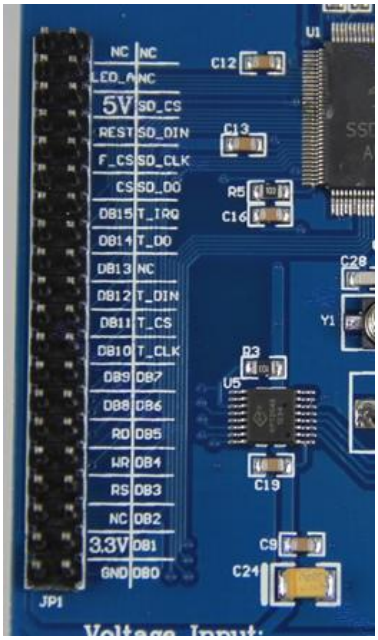
## 8-Bit-Parallel-LCD-Panels

Zusätzlich zu den SPI- und I²C-basierten Controllern unterstützt die PicoMite-Firmware LCD-Displays mit dem SSD1963-Controller (wie abgebildet) und dem ILI9341-Controller.

Diese nutzen eine parallele Schnittstelle, sind in Größen von 2,8" bis 9" erhältlich und haben bessere Spezifikationen als die kleineren Displays. Alle diese Displays verfügen über einen SD-Kartensteckplatz, der von MMBasic vollständig unterstützt wird. Auf eBay findest du passende Displays, indem du nach dem Namen des Controllers suchst (z. B. SSD1963).

Da sie eine parallele Schnittstelle nutzen, können Daten viel schneller übertragen werden als über eine SPI-Schnittstelle, was zu einer sehr schnellen Bildschirmaktualisierung führt.

Insbesondere die SSD1963-Displays sind zudem viel größer, haben mehr Pixel und sind heller. MMBasic kann einige davon mit 24-Bit-True-Color für eine vollständige Farbwiedergabe (16 Millionen Farben) ansteuern.




Die Eigenschaften dieser Displays sind:

- Ein 2,8-, 3,2-, 4,3-, 5-, 7-, 8- oder 9-Zoll-Display
- Auflösung von 320 x 240, 480 x 272 Pixel (4,3-Zoll-Version) oder 800 x 480 Pixel (5-, 7-, 8- oder 9-Zoll-Versionen).
- Ein SSD1963-Display-Controller oder ILI9341-Display-Controller mit paralleler Schnittstelle (8080-Format)
- Ein Touch-Controller (SPI-Schnittstelle).
- Ein SD-Kartensteckplatz in voller Größe.

Es gibt eine Reihe verschiedener Designs, die den SSD1963-Controller verwenden, aber glücklicherweise haben sich die meisten Anbieter auf einen einzigen Stecker standardisiert, wie rechts abgebildet.

Es wird dringend empfohlen, dass jedes gekaufte Display über einen passenden Stecker verfügt – dies gibt dir die Gewissheit, dass der Hersteller den Standard eingehalten hat, für den die PicoMite-Firmware ausgelegt ist.

## Anschluss eines 8-Bit-Parallel-LCD-Panels

Der Controller nutzt eine parallele Schnittstelle, während der Touch-Controller und die SD-Karte eine SPI-Schnittstelle verwenden. Die Touch- und SD-Karten-Funktionen sind optional, aber wenn sie genutzt werden, verwenden sie den zweiten SPI-Port (SPI2).

Die folgende Tabelle listet die erforderlichen Verbindungen zwischen der Displayplatine und dem Raspberry Pi Pico auf, um die 8-Bit-Parallelschnittstelle und das LCD-Display zu unterstützen. Die Schnittstellen für den Touch-Controller und die SD-Karte sind unten aufgeführt.

| 8-bit parallel Display | Description | Raspberry Pi Pico |
|---|---|---|
| DB0 | Parallel Data Bus bit 0 | Pin 1/GP0 |
| DB1 | Parallel Data Bus bit 1 | Pin 2/GP1 |
| DB2 | Parallel Data Bus bit 2 | Pin 4/GP2 |
| DB3 | Parallel Data Bus bit 3 | Pin 5/GP3 |
| DB4 | Parallel Data Bus bit 4 | Pin 6/GP4 |
| DB5 | Parallel Data Bus bit 5 | Pin 7/GP5 |

| | | |
|---|---|---|
| DB6 | Parallel Data Bus bit 6 | Pin 9/GP6 |
| DB7 | Parallel Data Bus bit 7 | Pin 10/GP7 |
| CS | Chip Select (active low) | Ground (ie, always selected) |
| WR | Write (active low) | Pin 19/GP14 |
| RD | Read (active low) | Pin 20/GP15 |
| DC | Command/Data | Pin 17/GP13 |
| RESET | Reset the SSD1963 | Pin 21/GP16 |
| LED_A | Backlight control for an unmodified display panel | Configurable see OPTION LCDPANEL |
| 5V | 5V power for the backlight on some displays (most displays use the 3.3V supply for this). | |
| 3.3V | Power supply. | |
| GND | Ground | |

Die Pins DC, WR, RD und RESET können mithilfe des optionalen Parameters *DCpin* als 4er-Block anderen Pins zugewiesen werden.
Beim RP2350B kann der optionale Parameter *DB0pin* verwendet werden, um den Start-Pin für die 8 oder 16 aufeinanderfolgenden Daten-Pins anzugeben, die vom Display genutzt werden.

Die folgende Tabelle listet die erforderlichen Anschlüsse zur Unterstützung der Touch-Controller-Schnittstelle auf:

| **8-bit parallel Display** | **Description** | **Raspberry Pi Pico** |
|---|---|---|
| T_CS | Touch Chip Select | Recommended Pin 24/GP18 |
| T_IRQ | Touch Interrupt | Recommended Pin 25/GP19 |
| T_DIN | Touch Data In (MOSI) | Recommended Pin 15/GP11 |
| T_CLK | Touch SPI Clock | Recommended Pin 14/GP10 |
| T_DO | Touch Data Out (MISO) | Recommended Pin 16/GP12 |

Die folgende Tabelle listet die erforderlichen Anschlüsse für den SD-Kartenstecker auf:

| **8-bit parallel Display** | **Description** | **Raspberry Pi Pico** |
|---|---|---|
| SD_CS | SD Card Chip Select | Recommended Pin 29/GP22 |
| SD_DIN | SD Card Data In (MOSI) | Recommended Pin 15/GP11 |
| SD_CLK | SD Card SPI Clock | Recommended Pin 14/GP10 |
| SD_DO | SD Card Data Out (MISO) | Recommended Pin 16/GP12 |

Wenn eine Verbindung als „empfohlen“ (Recommended) aufgeführt ist, handelt es sich lediglich um einen Vorschlag, und je nach Hardwarekonfiguration können auch andere Pins verwendet werden. Unabhängig davon sollte der spezifische Pin im entsprechenden OPTION-Befehl angegeben werden (siehe unten).

Im Allgemeinen verfügen Displays ab 7 Zoll über einen separaten Pin am Stecker (mit der Bezeichnung 5V) zur Versorgung der Hintergrundbeleuchtung über eine 5-V-Spannung. Wenn dieser Pin nicht vorhanden ist, wird die Stromversorgung für die Hintergrundbeleuchtung über den 3,3-V-Pin bezogen. Beachte, dass der Stromverbrauch der Hintergrundbeleuchtung beträchtlich sein kann. Ein 7-Zoll-Display zieht beispielsweise typischerweise 330 mA vom 5-V-Pin.

Wenn der 3,3-V-Ausgang des Pico zur Stromversorgung eines Displays plus dessen Hintergrundbeleuchtung verwendet wird, kann dies leicht mehr Strom erfordern, als der Pico liefern kann. Zu den Symptomen einer unzureichenden Stromversorgung können Fehler bei der TOUCH-Kalibrierung oder beim SD-Zugriff gehören. In diesem Fall sollte eine externe 3,3-V-Stromversorgung verwendet werden.

Der von der Hintergrundbeleuchtung aufgenommene Strom kann zudem einen Spannungsabfall am Masse-Pin des LCD-Displays verursachen, was wiederum die vom Display-Controller wahrgenommenen Logikpegel verschieben kann, was zu verfälschten Farben oder Text führt. Eine einfache Möglichkeit, diesen Effekt zu diagnostizieren, besteht darin, die CPU-Geschwindigkeit auf (sagen wir) 48 MHz zu reduzieren. Wenn das Problem dadurch behoben wird, ist das ein starker Hinweis darauf, dass dies die Ursache ist. Eine Möglichkeit zur Umgehung besteht darin, die Strom- und Massekabel direkt an die Leiterplatte des LCD-Displays zu löten.

Bei Displays, die sich den SPI-Port mit mehreren Geräten teilen (SD-Karte, Touchscreen usw.), ist Vorsicht geboten. In diesem Fall müssen alle Chip-Select-Signale in MMBasic konfiguriert oder durch eine permanente Verbindung mit 3,3 V deaktiviert werden. Wird dies nicht getan, schwebt der Pin, was dazu führt, dass der falsche Controller auf Befehle am SPI-Bus reagiert.

In der PicoMite-Firmware kann jeder der beiden SPI-Kanäle für die Kommunikation mit dem Touch-Controller und der SD-Kartenschnittstelle verwendet werden, je nach Einstellung unter „OPTION SYSTEM SPI“. Wenn diese Option aktiviert ist, steht dieser SPI-Kanal für BASIC-Programme nicht zur Verfügung (diese können den anderen SPI-Kanal nutzen).

## Konfiguration eines 8-Bit-Parallel-LCD-Panels

Um das Display zu nutzen, muss MMBasic mit dem Befehl OPTION LCDPANEL konfiguriert werden, der normalerweise an der Befehlszeile eingegeben wird. Bei jedem Neustart der PicoMite-Firmware initialisiert MMBasic das Display automatisch.

Die Syntax lautet:

OPTION LCDPANEL controller, orientation [,backlightpin] [,DCpin] [,NORESET] [,INVERT] [,DB0pin]

Wobei „controller“ entweder sein kann:

- SSD1963_4 Für ein 4,3-Zoll-Display
- SSD1963_5 Für ein 5-Zoll-Display
- SSD1963_5A Für eine alternative Version des 5-Zoll-Displays, falls SSD1963_5 nicht funktioniert
- SSD1963_7 Für ein 7-Zoll-Display
- SSD1963_7A Für eine alternative Version des 7-Zoll-Displays, falls SSD1963_7 nicht funktioniert.
- SSD1963_8 Für 8-Zoll- oder 9-Zoll-Displays.
- ILI9341_8 Für ein 2,8-Zoll- oder 3,2-Zoll-Display

„orientation“ kann LANDSCAPE, PORTRAIT, RLANDSCAPE oder RPORTRAIT sein. Diese können mit L, P, RL oder RP abgekürzt werden. Das Präfix R kennzeichnet die umgekehrte oder „auf dem Kopf stehende“ Ausrichtung.

„DCpin“ ist optional und bezeichnet den Daten-/Befehlspin (früher als RS-Pin bezeichnet). Wird dieser Parameter weggelassen, erfolgt die Pin-Zuweisung wie oben in der Tabelle angegeben. Wird er angegeben, werden die Pins DC, WR, RD und RESET nacheinander vom DC-Pin aus zugewiesen.

Der optionale Parameter „NORESET“ kann verwendet werden, um einen Pin zu sparen. In diesem Fall sollte der Reset-Pin auf High gelegt werden.

Der optionale Parameter „INVERT“ legt fest, dass die Farbpalette invertiert werden soll (erforderlich für bestimmte EsatRisng-Displays)

Auf dem RP2350B-Prozessor kann der optionale Parameter „DB0pin“ verwendet werden, um den Start-Pin für die 8 oder 16 aufeinanderfolgenden Daten-Pins anzugeben, die vom Display verwendet werden.

Dieser Befehl muss nur einmal ausgeführt werden. Von da an initialisiert MMBasic das Display automatisch beim Start oder Reset. Unter bestimmten Umständen kann es notwendig sein, die Stromversorgung des LCD-Panels zu unterbrechen, während die PicoMite-Firmware läuft (z. B. um Batteriestrom zu sparen), und in diesem Fall kann der GUI-Befehl RESET LCDPANEL verwendet werden, um das Display neu zu initialisieren.

Wenn das LCD-Panel nicht mehr benötigt wird, kann der Befehl OPTION LCDPANEL DISABLE verwendet werden, wodurch die I/O-Pins wieder für den allgemeinen Gebrauch freigegeben werden.

Um die Konfiguration zu überprüfen, kannst du den Befehl „OPTION LIST“ verwenden, um alle eingestellten Optionen aufzulisten, einschließlich der Konfiguration des LCD-Panels.

Um das Display zu testen, kannst du den Befehl GUI TEST LCDPANEL eingeben. Du solltest eine animierte Darstellung von farbigen Kreisen sehen, die schnell übereinander gezeichnet werden. Drücke die Leertaste auf der Tastatur der Konsole, um den Test zu beenden.

## 8- und 9-Zoll-Displays

Die Controller-Konfiguration SSD1963_8 wurde nur mit den 8- und 9-Zoll-Displays von EastRising getestet (erhältlich unter www.buydisplay.com). Diese müssen als TFT-LCD-Panel mit 8080-Schnittstelle, 800x480-Pixel-LCD, SSD1963-Display-Controller und XPT2046-Touch-Controller erworben werden. Beachte, dass die EastRising-Panels eine nicht standardmäßige Pinbelegung des Schnittstellensteckers verwenden, sodass du beim Anschließen an den Raspberry Pi Pico die Datenblätter zu Rate ziehen musst. Einen geeigneten Adapter zur Umwandlung auf den standardmäßigen 40-Pin-Anschluss kannst du hier kaufen:
https://www.rictech.nz/micromite-products

## 16-Bit-Parallel-LCD-Panels

SSD1963-Panels können auch für den 16-Bit-Parallelbetrieb aktiviert werden. In diesem Fall werden standardmäßig die Pins GP0–GP15 für die Datenverbindungen und die Pins GP16 bis GP19 für die Steuersignale DC, WR, RD und RESET verwendet.

Bei Systemen, die den RP2350B verwenden, können die verwendeten Datenpins über den optionalen Parameter DB0pin im Konfigurationsbefehl ausgewählt werden. Um den 16-Bit-Betrieb zu aktivieren, füge „_16“ an den Controller an. Z. B. SSD1963_4_16. Die Firmware unterstützt außerdem ILI9341-, ILI9486-, NT35510- und OTM8009A-Panels im 16-Bit-Modus mit den Controllertypen ILI9341_16, ILI9486_16 und IPS_4_16 (unterstützt sowohl NT35510 als auch OTM8009A).

Gültige 16-Bit-„Controller“ können sein:

- SSD1963_4_16 Für ein 4,3-Zoll-Display
- SSD1963_5_16 Für ein 5-Zoll-Display
- SSD1963_5A_16 Für eine alternative Version des 5-Zoll-Displays, falls SSD1963_5 nicht funktioniert
- SSD1963_7_16 Für ein 7-Zoll-Display
- SSD1963_7A_16 Für eine alternative Version des 7-Zoll-Displays, falls SSD1963_7 nicht funktioniert.
- SSD1963_8_16 Für 8-Zoll- oder 9-Zoll-Displays.
- ILI9341_16 Für 2,8-Zoll- oder 3,2-Zoll-Displays
- ILI9486_16 Unterstützt auch NXP R61529
- IPS_4_16

## Erweiterte Display-Unterstützung für RP2350

Der RP2350 stellt dem MMBasic-Programmierer 320 KB Speicher zur Verfügung. Dadurch lassen sich mithilfe gepufferter Display-Treiber verschiedene zusätzliche Funktionen implementieren.

Die PicoMite-Firmware für den RP2350 unterstützt die folgenden neuen Display-Treiber:

SPI

| | |
|---|---|
| ILI9341BUFF: | 320 x 240 |
| ST7796SPBUFF: | 320 x 320 |
| ST7796SBUFF: | 480 x 320 |
| ILI9488PBUFF: | 320 x 320 |
| ILI9488BUFF: | 480 x 320 |
| ILI9488WBUFF: | 480 x 320 'Waveshare Pico-Restouch-lcd-3.5 |
| ST7789_320BUFF: | 320 x 240 'Waveshare Pico-Restouch-lcd-2.8 |

Parallel

| | |
|---|---|
| SSD1963_5_BUFF: | 400 x 240 (8-Bit-Datenbus) |
| SSD 1963_7_ BUFF: | 400 x 240 (8-Bit-Datenbus) |

SSD 1963_5_ 12BUFF: 400x240
SSD 1963_7_ 12BUFF: 400x240
SSD 1963_5_16 BUFF: 400x240
SSD 1963_7_16 BUFF: 400x240

In allen Fällen wird durch die Aktivierung eines dieser Treiber ein RGB332-Framebuffer aus dem allgemeinen Speicher belegt, sodass im schlimmsten Fall bei einem ILI9488BUFF der allgemeine Speicher um 150 KB (von 320 KB) reduziert wird

Um einen der SPI-Treiber zu verwenden, musst du zunächst die zu verwendenden SPI-Pins zuweisen:
```
OPTION LCD SPI clkpin, mosipin, misopin
```
Diese Pins sind für das Display reserviert und müssen von den Pins getrennt sein, die für das SYSTEM-SPI verwendet werden, das möglicherweise noch für Touch und die SD-Karte benötigt wird (Hinweis: Bei den Waveshare-Boards führt die Verwendung dieses Treibers zu Konflikten mit der gleichzeitigen Nutzung von Touch und SD-Karte, da diese sich den SPI-Kanal teilen). Der Grund für die dedizierten Pins ist, dass die Bildschirmaktualisierungen alle auf dem zweiten Prozessor stattfinden und dieser für maximale Leistung den SPI-Kanal nicht teilen darf.

Das Format für den Befehl zur Aktivierung eines SPI-gepufferten Treibers ist dasselbe wie bei jedem SPI-Treiber
```
OPTION LCDPANEL controller, orientation, DCpin, RESETpin, CSpin [,BLpin] [,INVERT]
```
Die SSD1963-Treiber verwenden 8, 12 oder 16 Datenpins, wie im Treibernamen angegeben. In allen Fällen belegt die Aktivierung eines dieser Treiber einen 400x240-RGB332-Framebuffer aus dem Heap-Speicher. Bei Verwendung dieser Treiber kannst du mit dem Befehl MODE 800/400 zwischen dem gepufferten Treiber und einem herkömmlichen Treiber wechseln, der ohne Neustart zwischen den Modi 800x480 und 400x240 umschaltet. Mit dem Befehl OPTION LCD320 ON/OFF kannst du ein 320x240-Bild auf dem 400x240-Display zentrieren.

Das Format für den Befehl zum Aktivieren eines SDD1963-Puffertreibers ist dasselbe wie bei jedem SSD1963-Treiber
```
OPTION LCDPANEL controller, orientation [,backlightpin]  [,DCpin]
[,NORESET] [,INVERT] [,DB0pin]
```
Der Vorteil der gepufferten Treiber besteht darin, dass die Grafikbefehle in MMBasic lediglich in einen Speicherpuffer schreiben und die Aktualisierung des physischen Displays dann im Hintergrund auf dem zweiten Prozessor erfolgt. Dadurch kann das Basic-Programm die Verarbeitung fortsetzen, ohne auf das Display warten zu müssen. Natürlich verändert die Verwendung eines gepufferten Treibers den Gesamtdurchsatz zur physischen Anzeige nicht wesentlich. Kontinuierliche Schreibvorgänge müssen also wie bei einem Standardtreiber auf die Anzeige warten. Der große Vorteil zeigt sich dort, wo ein Programm eine rechenintensive Schleife hat, in der Bildschirm-Schreibvorgänge die Verarbeitung verzögern. In diesem Fall wird der Overhead der Verarbeitungsschleife drastisch reduziert, solange die Gesamtbildschirm-Aktualisierungsrate überschaubar ist.

## VGA222-Treiber

Der PICOMITE RP2350 und der PICOMITEUSB RP2350 unterstützen außerdem einen erweiterten VGA-Treiber, der im RGB222-Modus mit 64 Farben arbeitet. Dieser wird mit dem folgenden Befehl konfiguriert:
```
OPTION LCDPANEL driver, hsyncpin, bluelpin
```
Gültige Treiber sind: VGA222_640, VGA222_320, VGA_720, VGA_360 mit den VGA222-Auflösungen 640x480, 320x240, 720x400 und 360x200.

Der Befehl OPTION prüft die Verfügbarkeit von hysyncpin und hysyncpin+1 (vsync) sowie bluelpin – bluelpin+5 (blue_l, blue_h, green_l, green_h, red_l, red_h) und richtet, sofern alle frei sind, den VGA222-Ausgang auf den angegebenen Pins ein.

Um die Datenpins mit dem VGA-Anschluss zu verbinden, verwende 390- und 820-Ohm-Widerstände von den H- und L-Ausgängen, wie im Handbuch (Seite 30) für den grünen Kanal beschrieben. RGB222 bietet eine viel schönere Farbwiedergabe als das bei der Standard-VGA-Version verfügbare RGB121. Beachte, dass es keine Modi gibt. Die Auflösungen 640x480, 320x240, 720x400 und 360x200 sind unterschiedliche Treiber und benötigen unterschiedliche Mengen an Speicher (bei 640x480 und 720x400 deutlich mehr). Um den VGA-Ausgang zu erzeugen, nutzt die Firmware PIO0 und PIO1, sodass PIO2 verfügbar bleibt (es sei denn, du verwendest auch den I2S-Audiotreiber.

## Hintergrundbeleuchtungssteuerung

Für die Modelle ILI9163, ILI9341, ST7735, ST7735S, SSD1331, ST7789, ILI9481, ILI9488, ILI9488W, ST7789_135, ILI9341_8 und ST7789_320 kann am Ende der Konfigurationsparameter ein optionaler Parameter „, backlight“ hinzugefügt werden, der einen Pin angibt, der zur Steuerung der Helligkeit der Hintergrundbeleuchtung (LED_A) verwendet werden soll. Dadurch wird an diesem Pin ein PWM-Ausgang mit einer Frequenz von 50 kHz und einem anfänglichen Tastverhältnis von 99 % eingerichtet.

Du kannst dann 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 PWM-Kanal in Konflikt stehen, der möglicherweise für Audio eingerichtet wurde.

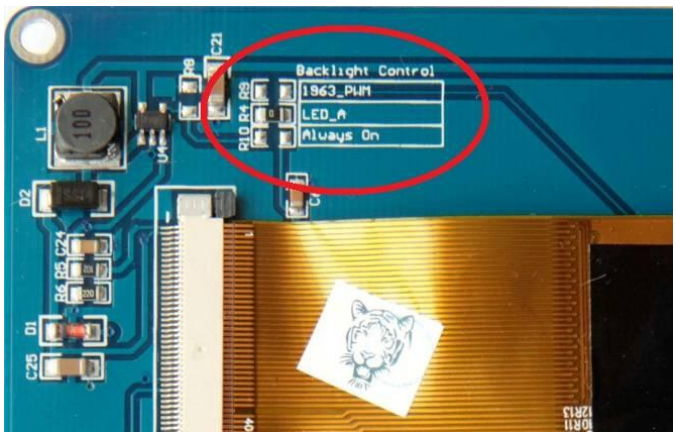
Zum Beispiel:

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

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

```
BACKLIGHT 40
```

Die meisten SSD1963-basierten LCD-Panels verfügen über drei Paare von Lötpads auf der Leiterplatte, die unter der Überschrift „Backlight Control“ zusammengefasst sind, wie rechts abgebildet.




Normalerweise werden die mit „LED-A“ gekennzeichneten Pads mit einem Null-Ohm-Widerstand kurzgeschlossen, wodurch die Helligkeit der Hintergrundbeleuchtung über ein PWM-Signal (Pulsweitenmodulation) am LED-A-Pin des Hauptsteckers des Displays gesteuert werden kann.

Es ist jedoch besser, den SSD1963-Controller zur Erzeugung dieses Signals zu verwenden, da dadurch ein I/O-Pin frei wird und eine feinere Steuerung möglich ist.

Um die SSD1963-Steuerung zu nutzen, sollte der Null-Ohm-Widerstand von dem mit „LED-A“ gekennzeichneten Paar entfernt und zum Kurzschließen des nahegelegenen Paares von Lötpads mit der Bezeichnung „1963-PWM“ verwendet werden. Die Helligkeit kann dann über den SSD1963-Controller mit dem Befehl BACKLIGHT eingestellt werden (dies geschieht automatisch, es muss nichts konfiguriert werden).

## Touch-Unterstützung

Viele LCD-Panels werden mit einem resistiven Touchscreen und einem zugehörigen Controller-Chip geliefert. MMBasic unterstützt diese Schnittstelle vollständig, wodurch viele der in einem Projekt verwendeten physischen Drehregler und Schalter als durch Berührung aktivierte Bildschirmsteuerelemente implementiert werden können. Als Alternative zum resistiven Touchscreen unterstützt MMBasic auch kapazitiven Touchscreen auf Basis des FT6336-Controllers.

In allen Fällen sollte das LCD-Display selbst zuerst konfiguriert und getestet werden, erst dann kann die Touch-Funktion konfiguriert werden. Der Touch-Controller nutzt das SPI-Protokoll für die Kommunikation, und bei LCD-Panels, die das SPI-Protokoll verwenden, wurde dies normalerweise zuvor mit dem Befehl OPTION SYSTEM SPI erledigt.

Bei Displays mit I²C- oder paralleler Schnittstelle muss der Befehl OPTION SYSTEM SPI separat verwendet werden, um den System-SPI-Bus für die Nutzung durch den Touch-Controller zu definieren. Dieser Befehl wurde zu Beginn dieses Kapitels behandelt und im Kapitel *„Optionen“* dieses Handbuchs ausführlich beschrieben.

Wenn du beispielsweise die empfohlenen Pins für ein 8-Bit-Parallel-Display (wie oben beschrieben) verwendest, würdest du Folgendes verwenden:

```
OPTION SYSTEM SPI GP10, GP11, GP12
```

Um die Touch-Funktion nutzen zu können, muss MMBasic mit dem Befehl OPTION TOUCH mitgeteilt werden, dass der Touch-Controller auf dem System-SPI-Bus verfügbar ist. Dadurch erfährt MMBasic, welche Pins für die Chip-Select- und Interrupt-Signale verwendet werden.

Bei einem typischen ILI9341-Display wird dadurch beispielsweise „Chip Select“ auf den GP12-Pin und „Interrupt“ auf GP11 gesetzt:

```
OPTION TOUCH GP12, GP11
```

Dies muss an der Befehlszeile eingegeben werden und führt dazu, dass die Firmware neu gestartet und die USB-Konsolenschnittstelle getrennt wird, die anschließend wieder angeschlossen werden muss.

Wenn die PicoMite-Firmware neu gestartet wird, initialisiert MMBasic automatisch den Touch-Controller. Um die Konfiguration zu überprüfen, kannst du den Befehl OPTION LIST verwenden, um alle eingestellten Optionen aufzulisten, einschließlich der Konfiguration des Displays und des Touchscreens.

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

## Kalibrieren des Touchscreens


Bevor die Touch-Funktion genutzt werden kann, muss sie über den Befehl CALIBRATE in der GUI von kalibriert werden.

Dieser Befehl zeigt ein Ziel in der oberen linken Ecke des Bildschirms an (wie abgebildet). Drücke mit einem spitzen, aber stumpfen Gegenstand (z. B. einem Zahnstocher) genau auf die Mitte des Ziels und halte ihn mindestens eine Sekunde lang gedrückt. MMBasic speichert diese Position und setzt die Kalibrierung fort, indem es das Ziel nacheinander in den anderen drei Ecken des Bildschirms zur Berührung und Kalibrierung anzeigt.

Die Kalibrierungsroutine warnt möglicherweise, dass die Kalibrierung nicht genau war. Dies ist nur eine Warnung, und du kannst die Touch-Funktion trotzdem nutzen, wenn du möchtest, aber es wäre besser, die Kalibrierung mit größerer Sorgfalt zu wiederholen.

Nach der Kalibrierung kannst du die Touch-Funktion mit dem GUI-Befehl TEST TOUCH testen. Dieser Befehl macht den Bildschirm schwarz und wartet auf eine Berührung. Wenn der Bildschirm berührt wird, erscheint ein weißer Punkt auf dem Display, der die Position der Berührung 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 beenden, kannst du die Leertaste auf der Tastatur der Konsole drücken.

## Touch-Funktionen

Um festzustellen, ob und wo der Bildschirm berührt wird, kannst du die folgenden Funktionen in einem BASIC-Programm verwenden:

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

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

## Touch-Interrupts

Ein Interrupt kann auf die IRQ-Pin-Nummer gesetzt werden, die bei der Konfiguration der Touch-Funktion angegeben wurde. Um eine Berührung zu erkennen, sollte der Interrupt als INTL (d. h. von High auf Low) konfiguriert werden. Der Interrupt kann mit dem Befehl SETPIN pin, OFF deaktiviert werden.

Das folgende Programm veranschaulicht, wie der Touch-Interrupt verwendet werden kann. Jedes Mal, wenn der Bildschirm berührt wird, werden die Koordinaten dieser Berührung auf der Konsole ausgegeben. Es wird davon ausgegangen, dass der Befehl OPTION TOUCH 7, 15 zur anfänglichen Konfiguration der Touch-Funktion verwendet wurde:

```
SETPIN 15, INTL, MyInt   'setzt voraus, dass OPTION TOUCH 7, 15 verwendet wurde
DO : LOOP

SUB MyInt                ' Unterprogramm, das bei einem Touch-Interrupt aufgerufen wird
  PRINT TOUCH(X) TOUCH(Y)
END SUB
```

## LCD-Display als Konsolenausgabe

Eine PS2- oder USB-Tastatur kann zusammen mit einem LCD-Display verwendet werden, um einen vollständig eigenständigen Computer zu erstellen, der direkt an der MMBasic-Eingabeaufforderung startet. Diese Funktion funktioniert besonders gut mit 5- und 7-Zoll-LCD-Displays, die den SSD1963-Controller verwenden. Diese können viel Text anzeigen und schnell aktualisieren, was zu einer Benutzererfahrung führt, die mit einem PicoMite-basierten Computer mit VGA- oder HDMI-Videoausgang vergleichbar ist.

Das LCD muss zunächst mit OPTION LCDPANEL konfiguriert werden. Um dann die Konsolenausgabe auf dem LCD-Panel zu aktivieren, solltest du den folgenden Befehl verwenden:

```
OPTION LCDPANEL CONSOLE [font [, fc [, bc [, blight]]] [,NOSCROLL]
```

„font“ ist die Standardschriftart, „fc“ ist die Standard-Vordergrundfarbe, „bc“ ist die Standard-Hintergrundfarbe und „blight“ ist die Standardhelligkeit der Hintergrundbeleuchtung (2 bis 100). Diese Einstellungen werden im Flash-Speicher gespeichert und dienen zur Konfiguration von MMBasic beim Hochfahren. Sie sind alle optional und standardmäßig auf Schriftart 2, Weiß, Schwarz und 100 % eingestellt.

Bei Displays, bei denen der Framebuffer nicht gelesen werden kann, setzt die Firmware automatisch die Option NOSCROLL. Wenn diese Option aktiviert ist und die Ausgabe den unteren Bildschirmrand erreicht, wird der Bildschirm gelöscht und die Ausgabe setzt oben wieder fort. Dies gilt auch für den integrierten Editor. Bei SPI-Displays, die den Framebuffer lesen können (z. B. ILI9341), ist das Scrollen sehr langsam, sodass der Parameter NOSCROLL gesetzt werden kann, um die Ausgabe- und Bearbeitungsleistung zu verbessern.

Beachte, dass dies nicht notwendig ist, wenn das Display im Hochformat verwendet wird, da dann H/W-Scrolling verwendet wird.

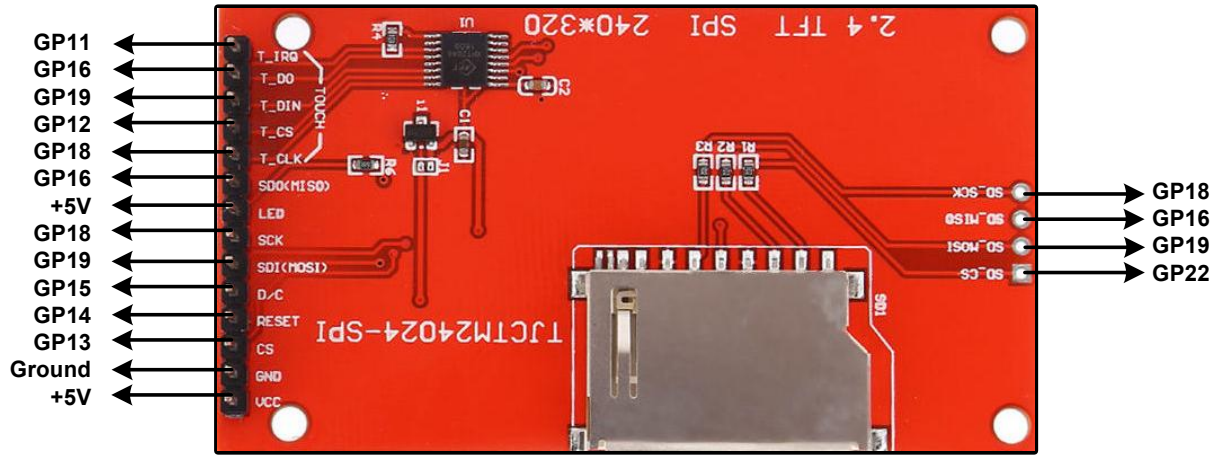
Die Farbcodierung im Editor wird durch diesen Befehl ebenfalls aktiviert (siehe das Kapitel *„Vollbild-Editor“*). Um die Verwendung des LCD-Panels als Konsole zu deaktivieren, lautet der Befehl OPTION LCDPANEL NOCONSOLE.

## Beispiel für die Konfiguration eines SPI-LCD-Panels

Im Folgenden findest du eine Übersicht darüber, 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 findest du auf ebay.com und ähnlichen Seiten, indem du nach dem Stichwort „ILI9341“ suchst. Stelle sicher, dass die Anschlüsse auf der Rückseite des Panels denen unten abgebildeten entsprechen:

Das Panel sollte wie abgebildet an den Raspberry Pi Pico angeschlossen werden:




Um die oben gezeigten Anschlüsse nachzubilden, solltest du die folgenden Konfigurationsbefehle nacheinander in die Befehlszeile eingeben:

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

Diese Befehle werden gespeichert und beim Hochfahren automatisch angewendet. Beachte, dass die Firmware nach der Eingabe jedes Befehls neu startet und die USB-Verbindung unterbrochen wird und neu hergestellt werden muss.

Als Nächstes sollte der Touchscreen mit folgendem Befehl kalibriert werden:

```
GUI CALIBRATE
```

Anschließend kannst du die verschiedenen Komponenten testen.

Der folgende Befehl zeichnet mehrere bunte, sich überlappende Kreise auf dem LCD-Bildschirm, wodurch bestätigt wird, dass das LCD korrekt angeschlossen ist:

```
GUI TEST LCDPANEL
```

Der folgende Befehl testet die Touch-Oberfläche. Wenn du den LCD-Bildschirm berührst, sollte genau an der Berührungsstelle ein Punkt auf dem Bildschirm erscheinen.

```
GUI TEST TOUCH
```

Wenn dies nicht genau funktioniert, musst du den Befehl GUI CALIBRATE möglicherweise ein zweites Mal ausführen und dabei genauer vorgehen.

Schließlich listet der folgende Befehl die Dateien auf der SD-Karte auf. Wenn er ohne Fehler ausgeführt wird, kannst du sicher sein, dass die SD-Karten-Schnittstelle in Ordnung ist.

```
FILES
```

Sollten beim Einrichten des Displays Probleme auftreten, lohnt es sich, alle Verbindungen zu trennen und die Optionen mit dem Befehl OPTION RESET zurückzusetzen, damit du bei Null anfangen kannst. Schließe dann die Komponenten Schritt für Schritt wieder an und konfiguriere sowie teste jede neue Stufe, während du voranschreitest. Denk dabei daran, dass jedes an den SPI-Bus angeschlossene Gerät in MMBasic konfiguriert werden muss oder seine Chip-Select-Leitung auf Logik-High gehalten werden muss. Das ist wichtig, da die Spannung an einer nicht angeschlossenen Chip-Select-Leitung schwanken kann, was dazu führen kann, dass das falsche Gerät auf Signale reagiert, die für ein anderes Gerät bestimmt sind.

Beachte außerdem, dass der ILI9341-Controller sehr empfindlich gegenüber elektrostatischer Entladung ist. Wenn das Panel also nicht reagiert, könnte es leicht beschädigt sein, und es lohnt sich, es mit einem anderen Panel zu testen.

# Grafikfunktionen

Diese Befehle und Funktionen wirken auf angeschlossene LCD-Panels und VGA-/HDMI-Videoausgänge. Ein Tutorial zur Verwendung dieser Funktionen ist in der Firmware-Distributionsdatei enthalten. Siehe die Datei: *Graphics in the PicoMite.pdf*

## Unterstützte Hardware

LCD-Panels

Die Auflösung und die Anzahl der Farben, die von einem LCD-Panel unterstützt werden, hängen vom Panel selbst und vom Treiber ab – siehe das Kapitel *„Display Panels“* für weitere Details.

VGA-Video

Es gibt eine Reihe von Modi, die mit dem Befehl MODE ausgewählt werden können:

OPTION RESOLUTION 640x480

- MODE 1 640x480 monochrom mit RGB121-Kacheln, optionaler Layer-Puffer
- MODE 2 320x240 4-Bit-Farbe, optionaler Layer-Puffer (nur RP2350) zweiter optionaler Layer-Puffer
- MODE 3 640x480 4-Bit-Farbe, optionaler Layer-Puffer (nur RP2350)

OPTION RESOLUTION 720x400

- MODE 1 720x400 monochrom mit RGB121-Kacheln, optionaler Layer-Puffer
- MODE 2 360x200 4-Bit-Farbe, optionaler Layer-Puffer (nur RP2350) zweiter optionaler Layer-Puffer
- MODE 3 720x400 4-Bit-Farbe, optionaler Layer-Puffer (nur RP2350)

OPTION RESOLUTION 800x600 (nur RP2350)

- MODE 1 800x600 monochrom mit RGB121-Kacheln, optionaler Layer-Puffer
- MODE 2 400x300 4-Bit-Farbe, zwei optionale Ebenen
- MODE 3 800x600 4-Bit-Farbe, optionaler Layer-Puffer

OPTION RESOLUTION 848x480 (nur RP2350)

- MODE 1 848x480 monochrom mit RGB121-Kacheln, optionaler Ebenenpuffer
- MODE 2 424x240 4-Bit-Farbe, zwei optionale Ebenen
- MODE 3 848x480 4-Bit-Farbe, optionaler Layer-Puffer

HDMI-Video (nur RP2350)

Jede HDMI-Auflösung kann in verschiedenen Modi betrieben werden, die mit dem Befehl MODE eingestellt werden:

OPTION RESOLUTION 640x480

- MODE 1 640x480x2-Farben mit RGB555, optionaler Layer-Puffer
- MODE 2 320x240x16 Farben und Farbabbildung auf die RGB555-Palette, zwei optionale Ebenen
- MODE 3 640x480x16 Farben und Farbabbildung auf die RGB555-Palette, optionaler Layer-Puffer
- MODE 4 320x240x32768 Farben, optionaler Ebenenpuffer
- MODE 5 320x240x256 Farben und Farbabbildung auf die RGB555-Palette, optionaler Layer-Puffer

OPTION RESOLUTION 720x400

- MODE 1 720x400 monochrom mit RGB555-Kacheln, optionaler Layer-Puffer
- MODE 2 360x200 4-Bit-Farbe und Farbabbildung auf RGB555-Palette, zwei optionale Ebenen
- MODE 3 720x400 4-Bit-Farbe und Farbabbildung auf RGB555-Palette, optionaler Layer-Puffer
- MODE 4 360x200x32768 Farben, optionaler Ebenenpuffer
- MODE 5 360x200x256 Farben und Farbabbildung auf RGB555-Palette, optionaler Layer-Puffer

OPTION RESOLUTION 800x600 (nur RP2350)

- MODE 1 800x600 monochrom mit RGB332-Kacheln, optionaler Layer-Puffer
- MODE 2 400x300 4-Bit-Farbe und Farbabbildung auf RGB332-Palette, optionaler Layer-Puffer

MODE 3 800x600 4-Bit-Farbe und Farbabbildung auf RGB332-Palette, optionaler Layer-Puffer
MODE 5 400x300x256 Farben, optionaler Layer-Puffer

OPTION RESOLUTION 848x480 (nur RP2350)
MODE 1 848x480 monochrom mit RGB332-Kacheln, optionaler Layer-Puffer
MODE 2 424x240 4-Bit-Farbe und Farbabbildung auf RGB332-Palette, optionaler Layer-Puffer
MODE 3 848x480 4-Bit-Farbe und Farbabbildung auf RGB332-Palette, optionaler Layer-Puffer
MODE 5 424x240x256 Farben, optionaler Layer-Puffer

OPTION RESOLUTION 1280x720
MODE 1 1280x720x2 Farben mit RGB332, optionaler Layer-Puffer
MODE 2 320x180x16 Farben und Farbabbildung auf RGB332-Palette, optionaler Layer-Puffer
MODE 3 640x360x16 Farben und Farbabbildung auf die RGB332-Palette, optionaler Layer-Puffer
MODE 5 320x180x256 Farben, optionaler Layer-Puffer

OPTION RESOLUTION 1024x768
MODE 1 1024x768x2 Farben mit RGB332-Kacheln, optionaler Layer-Puffer
MODE 2 256x192x16 Farben und Farbabbildung auf RGB332-Palette, optionaler Layer-Puffer
MODE 3 512x384x16 Farben und Farbabbildung auf die RGB332-Palette, optionaler Layer-Puffer
MODE 5 256x192x256 Farben, optionaler Layer-Puffer

OPTION RESOLUTION 1024x600
MODE 1 1024x600 Mono mit Kacheln
MODE 2 256x150 RGB121
MODE 3 512x300 RGB121
MODE 5 256x150 RGB332

OPTION RESOLUTION 800x480 (Hinweis: reduziert den verfügbaren Heap-Speicher)
MODE 1 800x480 Mono mit Kacheln
MODE 2 400x240 RGB121
MODE 3 800x480 RGB121
MODE 5 400x240 RGB332

## Farben

Die Farbe wird als 24-Bit-True-Color-Zahl angegeben, wobei die oberen acht Bits die Intensität der roten Farbe, die mittleren acht Bits die Intensität der grünen Farbe und die unteren acht Bits die Intensität der blauen Farbe darstellen. Am einfachsten lässt sich diese Zahl mit der Funktion RGB() erzeugen, die folgende Form hat:

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

Die Funktion RGB() unterstützt auch eine Kurzform, mit der du gängige Farben durch Benennung angeben kannst. Zum Beispiel RGB(red) oder RGB(cyan). Die Farben, die mit der Kurzform benannt werden können, sind white, black, blue, green, cyan, red, magenta, yellow, brown, white, orange, pink, gold, salmon, beige, lightgrey und grey (oder in der US-Schreibweise gray/lightgray).

MMBasic wandelt alle Farben automatisch in das vom jeweiligen Display-Controller geforderte Format um. Beispielsweise sind es beim ILI9341-LCD-Controller 64K Farben im 565-Format.

Die Standardeinstellung für Befehle, die einen Farbparameter erfordern, kann mit dem Befehl COLOUR (kann auch als COLOR geschrieben werden) festgelegt werden. Das ist praktisch, wenn dein Programm ein einheitliches Farbschema verwendet; du kannst dann die Standardeinstellungen festlegen und in deinem gesamten Programm die Kurzform der Zeichenbefehle verwenden.

Der Befehl COLOUR hat das folgende Format: `COLOUR Vordergrundfarbe, Hintergrundfarbe`

## Bildschirmkoordinaten

Alle Koordinaten und Maße auf dem Bildschirm werden in Pixeln angegeben, wobei die X-Koordinate die horizontale Position und die Y-Koordinate die vertikale Position bezeichnet. Die obere linke Ecke des Bildschirms hat die Koordinaten X = 0 und Y = 0, und die Werte steigen an, je weiter du dich auf dem Bildschirm nach unten und nach rechts bewegst.

Es gibt vier schreibgeschützte Variablen, die nützliche Informationen über das aktuell angeschlossene Display liefern:

- MM. HRES
  Gibt die Breite des Displays (die X-Achse) in Pixeln zurück.
- MM.VRES
  Gibt die Höhe des Bildschirms (die Y-Achse) in Pixeln zurück.
- MM.INFO(FONTHEIGHT)
  Gibt die Höhe der aktuellen Standardschriftart (in Pixeln) zurück. Alle Zeichen einer Schriftart haben dieselbe Höhe.
- MM.INFO(FONTWIDTH)
  Gibt die Breite eines Zeichens in der aktuellen Schriftart zurück (in Pixeln). Alle Zeichen haben dieselbe Breite.

## Schriftarten

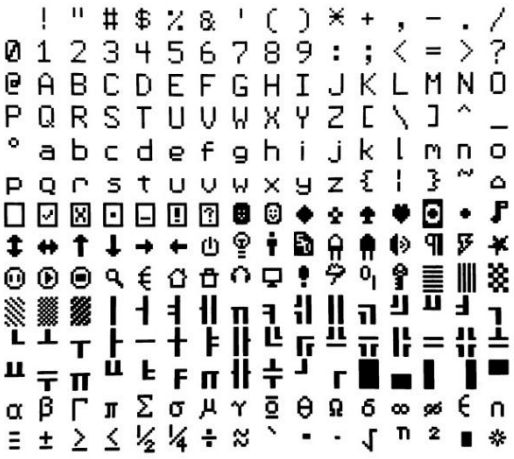
Es gibt acht integrierte Schriftarten. Das sind:

| Schriftart Nummer | Größe (Breite x Höhe) | Zeichen Zeichensatz | Beschreibung |
|---|---|---|---|
| 1 | 8 x 12 | Alle 95 ASCII-Zeichen plus 7F bis FF (hex) | Standardschriftart (Standard beim Start). |
| 2 | 12 x 20 | Alle 95 ASCII-Zeichen | Mittlere Schriftgröße. |
| 3 | 16 x 16 | Alle 95 ASCII-Zeichen | Eine große Schriftart für VGA-Versionen. |
| 3 | 16 x 24 | Alle 95 ASCII-Zeichen | Eine große Schriftart für HDMI-Versionen und LCD-Bildschirme. |
| 4 | 10x16 | Alle 95 ASCII-Zeichen plus 7F bis FF (hex) | Eine Schriftart mit erweiterten grafischen Zeichen. Geeignet für hochauflösende Displays. |
| 5 | 24 x 32 | Alle 95 ASCII-Zeichen | Extra große Schrift, sehr gut lesbar. |
| 6 | 32 x 50 | 0 bis 9 plus einige Symbole | Zahlen sowie Dezimalpunkt, Plus-, Minus-, Gleichheits-, Grad- und Doppelpunktzeichen. Sehr gut lesbar. |
| 7 | 6 x 8 | Alle 95 ASCII-Zeichen | Eine kleine Schriftart, die bei niedrigen Auflösungen nützlich ist. |
| 8 | 4 x 6 | Alle 95 ASCII-Zeichen | Eine noch kleinere Schriftart. |

Beachte, dass Schriftart 3 bei Verwendung mit VGA-Videoausgang eine Größe von 16 x 16 Pixel hat, bei HDMI und LCD-Bildschirmen jedoch eine Größe von 16 x 24.

In allen Schriftarten (einschließlich Schriftart Nr. 6) wurde das Backquote-Zeichen (60 hex oder 96 dezimal) durch das Gradzeichen (º) ersetzt.

Schriftart Nr. 1 (die Standardschriftart) und Schriftart Nr. 4 verfügen über einen erweiterten Zeichensatz, der alle Zeichen von CHR$(32) bis CHR$(255) bzw. 20 bis FF (hex) abdeckt, wie rechts dargestellt.



## Eingebettete Schriftarten

Bei Bedarf können zusätzliche Schriftarten in ein BASIC-Programm eingebettet werden. Diese Schriftarten funktionieren genau wie die integrierte Schriftart (d. h. sie werden mit dem Befehl FONT ausgewählt oder im Befehl TEXT angegeben).

Das Format einer eingebetteten Schriftart lautet:

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

Es muss mit dem Schlüsselwort „DefineFont“ beginnen, gefolgt von der Schriftartnummer (der optional ein #-Zeichen vorangestellt werden kann). Es kann jede Schriftartnummer im Bereich von 2 bis 5 und 8 bis 16 angegeben werden; wenn sie mit einer integrierten Schriftart übereinstimmt, ersetzt sie diese.

Der Hauptteil der Schriftart besteht aus einer Folge von 8-stelligen Hex-Wörtern, wobei jedes Wort durch ein oder mehrere Leerzeichen oder eine neue Zeile getrennt ist. Die Schriftartdefinition wird durch das Schlüsselwort „End DefineFont“ beendet. Diese können an beliebiger Stelle im Programm platziert werden, und MMBasic überspringt sie. Dieses Format entspricht dem vom Micromite verwendeten.

Weitere Schriftarten und Informationen findest du im Ordner „Embedded Fonts“ im PicoMite-Firmware-Download. Diese Schriftarten decken eine breite Palette von Zeichensätzen ab, darunter eine Symbolschriftart (Dingbats), die sich gut zum Erstellen von Bildschirmsymbolen usw. eignet.

## Zeichenbefehle

Es gibt zehn grundlegende Zeichenbefehle, die du in MMBasic-Programmen zum Zeichnen von Grafiken verwenden kannst. Die meisten davon haben optionale Parameter. Du kannst diese am Ende eines Befehls komplett weglassen oder zwei aufeinanderfolgende Kommas verwenden, um einen fehlenden Parameter anzugeben. Der fünfte Parameter des LINE-Befehls ist beispielsweise optional, sodass du dieses Format verwenden kannst:

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

Optionale Parameter sind unten kursiv dargestellt, zum Beispiel: *font*.

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

Die grundlegenden Zeichenbefehle sind:

- `CLS` *`C`*
  Löscht den Bildschirm und setzt ihn auf die Farbe C. Wenn C nicht angegeben wird, wird die aktuelle Standard-Hintergrundfarbe verwendet.
- `PIXEL X, Y,` *`C`*
  Leuchtet an der Position ( ) ein Pixel auf. Wenn C nicht angegeben wird, 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 Linienbreite und gilt nur für horizontale oder vertikale Linien. Der Standardwert ist 1, wenn nichts angegeben wird oder wenn die Linie diagonal verläuft. Es gibt eine erweiterte Version für diagonale Linien (siehe LINE AAA).
- `BOX X, Y, W, H,` *`LW, C, FILL`*
  Zeichnet ein Rechteck, das bei X und Y beginnt und W Pixel breit sowie H Pixel hoch ist.
  LW ist die Breite der Seiten des Kastens und kann Null sein. Der Standardwert ist 1.
- `RBOX X, Y, W, H,` *`R, C, FILL`*
  Zeichnet ein Kästchen mit abgerundeten Ecken, beginnend bei X und Y, das W Pixel breit und H Pixel hoch ist.
  R ist der Radius der Ecken des Rechtecks. 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 ist 1). A ist das Seitenverhältnis, das eine Gleitkommazahl ist und standardmäßig 1 beträgt. Ein Seitenverhältnis von 0,5 zeichnet beispielsweise ein Oval, bei dem die Breite halb so groß ist wie die Höhe.

- `TEXT X, Y, STRING, ALIGNMENT, FONT, SCALE, C, BC`
  Zeigt eine Zeichenkette an, beginnend bei X und Y. ALIGNMENT ist 0, 1 oder 2 Zeichen (ein Zeichenkettenausdruck oder eine Variable ist ebenfalls zulässig), wobei der erste Buchstabe die horizontale Ausrichtung um X herum angibt und L, C oder R für links-, zentriert- oder rechtsbündigen Text sein kann, und der zweite Buchstabe die vertikale Ausrichtung um Y herum angibt und T, M oder B für oben-, mittig- oder untenbündigen Text sein kann. Die Standardausrichtung ist links/oben. Ein zusätzlicher Code-Buchstabe kann verwendet werden, um den Text zu drehen (Details siehe unten). FONT und SCALE sind optional und entsprechen standardmäßig den Einstellungen des FONT-Befehls. C ist die Zeichenfarbe und BC ist die Hintergrundfarbe. Sie sind optional und entsprechen standardmäßig den Einstellungen des COLOUR-Befehls.
- `GUI BITMAP X, Y, BITS, WIDTH, HEIGHT, SCALE, C, BC`
  Zeigt die Bits einer Bitmap an, beginnend bei X und Y. HEIGHT und WIDTH sind die Abmessungen der Bitmap, wie sie auf dem LCD-Display angezeigt werden, und sind standardmäßig auf 8x8 eingestellt. SCALE, C und BC entsprechen denen des TEXT-Befehls. Die Bitmap kann eine Ganzzahl, eine Zeichenkette oder eine Konstante sein und wird so gezeichnet, dass das erste Byte als erste Bits der obersten Zeile verwendet wird (zuerst Bit 7, dann Bit 6 usw.), gefolgt vom nächsten Byte usw. Wenn die oberste Zeile gefüllt ist, beginnt die nächste Zeile der angezeigten Bitmap mit dem nächsten Bit der Ganzzahl oder Zeichenkette.
- `POLYGON n, xarray%(), yarray%() [, bordercolour] [, fillcolour]`
  Zeichnet ein gefülltes oder umrandetes Polygon mit n xy-Koordinatenpaaren in xarray%() und yarray%(). Wenn „fillcolour“ weggelassen wird, wird nur die Polygonumrandung gezeichnet. Wenn „bordercolour“ weggelassen wird, wird standardmäßig die aktuelle Standard-Vordergrundfarbe verwendet.
- `ARC x, y, r1, [r2], a1, a2 [, c]`
  Zeichnet einen Kreisbogen mit einer bestimmten Farbe und Breite zwischen zwei Radien (in Grad angegeben). Parameter für den Befehl ARC sind die x- und y-Koordinaten des Bogenmittelpunkts, der innere und äußere Radius, der Start- und Endwinkel des Bogens sowie die Farbe des Bogens. Der Nullgrad-Bezugspunkt liegt bei 12 Uhr

## Gedrehter Text

Wie oben beschrieben, kann die Ausrichtung des Textes im TEXT-Befehl durch die Verwendung von einem oder zwei Zeichen in einem Zeichenfolgenausdruck für den dritten Parameter des Befehls festgelegt werden. In dieser Zeichenfolge kannst du auch ein drittes Zeichen angeben, um die Drehung des Textes anzugeben.

Dieses Zeichen kann eines der folgenden sein:

- N für normale Ausrichtung
- V für vertikalen Text, bei dem jedes Zeichen unter dem vorherigen steht und von oben nach unten verläuft.
- I Der Text wird gespiegelt (d. h. auf den Kopf gestellt)
- U der Text wird um 90° gegen den Uhrzeigersinn gedreht
- D der Text wird um 90° im Uhrzeigersinn gedreht

Als Beispiel wird mit dem folgenden Code der Text „LCD Display“ vertikal am linken Rand des Displays angezeigt und dabei vertikal zentriert:

```
TEXT 0, 250, „LCD Display“, „LMV“, 5
```

Die Positionierung bezieht sich auf die obere linke Ecke des Zeichens bei normaler Ansicht. Bei invertierten Werten von 100,100 befindet sich also der obere linke Pixel des ersten Zeichens bei 100,100, und der Text liegt dann oberhalb von y=101 und links von x=101. Ebenso wird das „R“ in der Ausrichtungszeichenfolge aus der Perspektive des Zeichens betrachtet, unabhängig von dessen Ausrichtung (nicht vom Bildschirm).

## Transparenter Text

Der VGA- oder HDMI-Videoausgang oder LCD-Displays, die den SSD1963, ST7796S, ILI9341, ST7789_320 oder ILI9488 mit angeschlossenem MISO verwenden, unterstützen transparenten Text.

In diesem Fall 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.

## Framebuffer und Ebenen

Alle Varianten der Firmware können einen oder zwei Framebuffer im Speicher und einen oder zwei Layer-Puffer erstellen (dies ist speicherabhängig). Dabei handelt es sich um Speicherbereiche mit derselben Breite

und Höhe wie das Hauptdisplay. Bei HDMI- und VGA-Displays haben sie dieselbe Farbtiefe wie der aktuelle Modus. Bei LCD-Displays haben sie 4 Bit pro Pixel (16 Farben).

Je nach Firmware-Version und aktuellem Anzeigemodus wird für die Erstellung von Framebuffern oder Layer-Puffern entweder vorab zugewiesener Speicher verwendet oder Speicher aus dem Benutzerspeicher zugewiesen.

Framebuffer können verwendet werden, um Bilddaten zu erstellen, die auf das physische Display kopiert werden können. Layer-Buffer werden typischerweise verwendet, um Teilbilder zu erstellen, die über einem Hintergrundbild liegen können, und bieten eine effiziente Methode, um Anzeigeelemente über einen statischen Hintergrund zu bewegen.

Alle Standard-Grafikbefehle können auf einem Framebuffer oder Layer-Buffer genauso verwendet werden, als würdest du auf das physische Display schreiben. Der Befehl FRAMEBUFFER WRITE wird verwendet, um das Ziel der Grafikausgabe mithilfe eines *Codes* festzulegen.

Der *Code* ist ein einzelnes Zeichen, das Folgendes sein kann:

- N Das physische Ausgabegerät.
- F Der Framebuffer.
- 2 Ein zweiter Framebuffer (nur RP2350)
- L Der Layerbuffer
- T Ein zweiter Layerbuffer (nur RP2350)

Die grundlegenden Framebuffer-Befehle lauten:

```
FRAMEBUFFER CREATE  ‘ Code F
FRAMEBUFFER LAYER   ‘ Code L
FRAMEBUFFER CLOSE
FRAMEBUFFER WRITE code
FRAMEBUFFER COPY code1, code2 [,B]
```

Weitere Framebuffer-Befehle findest du in den detaillierten Befehlsbeschreibungen.

Bei den VGA- und HDMI-Versionen der Firmware werden die Ebenen je nach Anzeigemodus und CPU-Geschwindigkeit (>=252 MHz) automatisch auf das Hauptbild der Anzeige gelegt, während dieses auf den Bildschirm ausgegeben wird. Bei LCD-Displays wird der Befehl FRAMEBUFFER MERGE verwendet, um das endgültige Bild aus einem Framebuffer und einem Layer-Buffer zu erstellen. Das Spiel PETSCII Robots zeigt, wie diese Technik sehr effektiv eingesetzt werden kann.

Die automatische Anwendung eines Layer-Puffers ist in den VGA-Modi 2 und 3 (nur RP2350) sowie in den HDMI-Modi 2, 3, 4 und 5 implementiert. Zwei Layer-Puffer sind nur auf dem RP2350 und in den folgenden Modi verfügbar: VGA-Modus 2, HDMI-Modi 2 und 5.

## BLIT- und Sprite-Befehle

In früheren Versionen der Firmware waren die Befehle „blit“ und „sprite“ Synonyme für dieselbe Funktionalität. Ab Version 6.00.00 sind sie separate Befehle. Der Unterschied besteht darin, dass BLIT ein einfacher Speichervorgang ist, bei dem Daten von einem Display oder Speicher auf ein anderes Display oder einen anderen Speicher kopiert werden. Sprites sind komplexer und ermöglichen es dem Programmierer, Elemente über einem Hintergrund anzuzeigen und sie dann über den Hintergrund zu verschieben, ohne das Hintergrundbild zu verfälschen. Außerdem kann der Programmierer die Sprite-Funktionalität nutzen, um Kollisionen zwischen Sprites sowie zwischen einem Sprite und den Rändern des Displays zu erkennen.

Sprites können nur verwendet werden, wenn das Display das Auslesen aus seinem Framebuffer unterstützt; auch die BLIT-Funktionalität ist eingeschränkt, wenn dies nicht der Fall ist. Sprites sind für alle Firmware-Versionen aktiviert, wenn sie auf einem Framebuffer im Arbeitsspeicher verwendet werden, sowie bei den VGA- und HDMI-Versionen der Firmware direkt mit dem Bildschirm.

Sprites werden immer als RGB121-Nibbles gespeichert, wobei 2 Pixel auf ein Byte kommen. Im Gegensatz dazu werden BLIT-Puffer als RGB888-Werte gespeichert und können daher mit Vollfarb-LCD-Displays verwendet werden. Das geht natürlich auf Kosten eines deutlich höheren Speicherbedarfs.

Weitere Informationen zur Verwendung von Sprites findest du unter dem SPRITE-Befehl und der Funktion sowie in Anhang G.

Wenn das Display transparenten Text anzeigen kann, ermöglicht der BLIT-Befehl, einen Teil des aktuell auf dem Display angezeigten Bildes in einen Speicherpuffer zu kopieren und später wieder auf das Display zurückzukopieren. Das ist nützlich, wenn etwas über den Hintergrund gezeichnet und später wieder entfernt

werden soll, ohne das Bild im Hintergrund zu beschädigen. Beispiele hierfür sind ein Spiel, in dem sich eine Figur vor einer Landschaft bewegt, oder die sich bewegende Nadel einer fotorealistischen Anzeige.

Die verfügbaren Standard-Blit-Befehle sind:

```
BLIT READ #b, x, y, w, h
BLIT WRITE #b, x, y [,mode]
BLIT LOAD #b, f$, x, y, w, h
BLIT CLOSE #b
```

#b ist die Puffernummer im Bereich von 1 bis 64. x und y sind die Koordinaten der oberen linken Ecke und w und h sind die Breite und Höhe des Bildes. READ kopiert das Bild vom Bildschirm in den Puffer, WRITE kopiert den Puffer auf den Bildschirm und CLOSE gibt den Puffer frei und gewinnt den belegten 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 unterstützen (d. h. bei Verwendung von SSD1963, ILI9341, ST7789_320 oder ILI9488 mit angeschlossenem MISO) sowie auf VGA- und HDMI-Displays und allen Framebuffern im Speicher

Diese Befehle können verwendet werden, um einen Teil des Bildschirms an einen anderen Ort zu kopieren (indem man ihn in einen Puffer kopiert und dann an einer anderen Stelle schreibt), aber eine einfachere Methode ist die Verwendung einer alternativen Version des BLIT-Befehls wie folgt:

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

BLIT RESIZE src, dst, sx, sy, sw, sh, dx, dy, dw, dh

Dieser Befehl skaliert einen Quellbereich auf einen Zielbereich unter Verwendung von Nearest-Neighbor-Sampling (schnelle ganzzahlige Skalierung).

Quellkoordinaten und -größe werden streng überprüft und müssen vollständig innerhalb von 0..MM.HRES-1 und 0..MM.VRES-1 liegen (ansonsten Fehler 21).

Die Zielkoordinaten werden auf den sichtbaren Anzeigebereich beschnitten. Wenn nach dem Beschneiden keine Zielpixel übrig bleiben, findet keine Zeichnung statt.

Wenn entweder src oder dst N (Anzeigepuffer) ist, muss der aktuelle Modus RGB121 sein (Modus 2 oder Modus 3).

Um die Korrektheit bei Überlappungen innerhalb desselben Puffers zu gewährleisten, wird eine temporäre Quellkopie nur verwendet, wenn sich Quelle und Ziel überlappen; bei Nicht-Überlappung erfolgt direktes Lesen/Schreiben.

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 führt die Kopie korrekt aus.

Diese Form des BLIT-Befehls eignet sich besonders gut zum Erstellen von Grafiken, die horizontal oder vertikal scrollen können, wenn neue Daten hinzugefügt werden.

Darüber hinaus bietet die Firmware die Befehle BLIT MEMORY, BLIT COMPRESSED, BLIT FRAMEBUFFER und BLIT MERGE. Diese erweiterten Befehle können verwendet werden, um Spiele mit Hunderten von Anzeigeelementen zu programmieren, wie beispielsweise die Portierung der PETSCII-Roboter auf MMBasic.

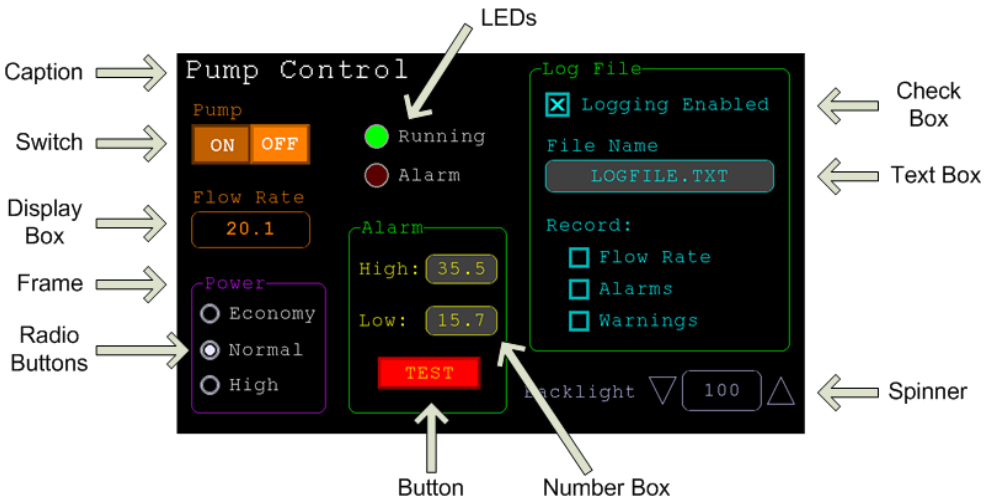
## Bild laden

Mit den Befehlen LOAD IMAGE und LOAD JPG kannst du ein Bild aus dem Flash-Dateisystem oder von der SD-Karte laden und auf dem LCD-Display anzeigen. Damit kannst du ein Logo zeichnen oder den auf dem Display angezeigten Grafiken einen verzierten Hintergrund hinzufügen.

## Erweiterte Grafik

NICHT VERFÜGBAR IN DEN VERSIONEN VGA/HDMI UND WEBMITE RP2040

Die PicoMite-Firmware enthält eine Reihe erweiterter Grafikfunktionen, die es Programmierern erleichtern, berührungsempfindliche Steuerelemente auf einem LCD-Panel zu erstellen, wie abgebildet. Dazu gehören Bildschirmschalter, Tasten, Anzeigeleuchten, Tastaturen usw.




MMBasic zeichnet das Bedienelement und animiert es (d. h., ein Schalter sieht aus, als würde er bei Berührung gedrückt). Der BASIC-Programmierer muss lediglich einen einzigen Befehl aufrufen, um die grundlegenden Details des Bedienelements festzulegen.

Diese Funktionen machen es einfach, ein Bedienfeld zu erstellen, um beliebige Steuerungsfunktionen wie eine Drehmaschine, eine Motorsteuerung, eine Heizungsanlage, kleine industrielle Prozesse und so weiter zu verwalten.

Die erweiterten Grafikfunktionen werden ausführlich im Dokument *„Advanced Graphics Functions.pdf“* beschrieben, dass im Firmware-Download-Paket enthalten ist.

## 3D-Engine

NICHT IN DEN WEBMITE-VERSIONEN VERFÜGBAR

Die 3D-Engine von „ “ enthält zehn Befehle zur Bearbeitung von 3D-Bildern, darunter das Einstellen der Kamera, das Erstellen, Ausblenden, Drehen usw. Eine vollständige Beschreibung dieser Befehle und ihrer Verwendung findest du im Dokument *„3D_Graphics_User_Manual.pdf“* im PicoMite-Firmware-Download.

## Beispiel für LCD-Grafiken

Als Beispiel für die Verwendung der einfachen Grafikbefehle zeichnet das folgende Programm eine einfache Digitaluhr auf einem LCD-Display auf Basis des ILI9341. Das Programm wird beendet und kehrt zur Befehlszeile zurück, wenn der Bildschirm berührt wird.

Zunächst müssen die Anzeige- und Touch-Optionen konfiguriert werden, indem du die am Anfang dieses Kapitels aufgeführten Befehle eingibst. Das genaue Format hängt davon ab, wie du das Display angeschlossen hast.

Gib dann das Programm ein und führe es aus:

```
CONST DBlue = RGB(0, 0, 128)                ' Eine dunkelblaue Farbe
COLOUR RGB(GREEN), RGB(BLACK)               ' Standardfarben festlegen
FONT 1, 3                                   ' Die Standardschriftart
festlegen

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 beginnt damit, eine Konstante mit einem Wert zu definieren, der einer dunkelblauen Farbe entspricht, und legt dann die Standardeinstellungen für die Farben und die Schriftart fest. Anschließend zeichnet es ein Rechteck mit roten Seiten und einem dunkelblauen Inneren.

Danach geht das Programm in eine Endlosschleife über, in der es drei Funktionen ausführt:

1. Zeigt die aktuelle Uhrzeit innerhalb des zuvor gezeichneten Kastens an. Die Zeichenfolge wird sowohl horizontal als auch vertikal mittig im Kasten gezeichnet. Beachte, dass der TEXT-Befehl sowohl die Standardschriftart als auch die Standardfarben überschreibt, um seine eigenen Parameter festzulegen.
2. Zeichnet das Datum zentriert in der unteren Hälfte des Bildschirms. In diesem Fall verwendet der TEXT-Befehl die zuvor festgelegten Standardschriftart und -farben.
3. Prüft auf eine Berührung des Bildschirms.  Dies wird angezeigt, wenn die Funktion TOUCH(X) einen anderen Wert als -1 zurückgibt.  In diesem Fall wird das Programm beendet.

Die Bildschirmanzeige sollte so aussehen (die in dieser Abbildung verwendete Schriftart ist eine andere):

# WLAN- und Internetfunktionen

**NUR WEBMITE-VERSION**

Dieses Kapitel bietet einen Überblick über die WLAN- und Internetfunktionen, die in der WebMite-Version für den Raspberry Pi Pico W und den Raspberry Pi Pico 2 W implementiert sind.

Diese Funktionen sind komplex, daher solltest du einige Punkte beachten:

- Die Implementierung der Internetprotokolle beansprucht einen Großteil der Prozessorressourcen (insbesondere des Arbeitsspeichers), daher sollte die WebMite-Firmware nur dort eingesetzt werden, wo WLAN- und Internetverbindung wichtig sind und gewisse Leistungseinbußen im Vergleich zum Standard-Raspberry Pi Pico in Kauf genommen werden können.
- Dieses Handbuch beschreibt, wie WebMite mit den WLAN- und Internetprotokollen interagiert, und enthält einige einfache Beispiele, geht jedoch nicht auf HTML, TCP und die vielen anderen beteiligten Protokolle und Konventionen ein. Diese können für Einsteiger verwirrend sein, die daher einige der zahlreichen im Internet verfügbaren Einführungen zu Rate ziehen sollten.
- Bei der Verarbeitung komplexer Internetprotokolle kann die WebMite-Firmware verwirrt werden und hängen bleiben oder einen Fehler ausgeben und zur Befehlszeile zurückkehren. Um dies zu berücksichtigen, sollten Programme die WATCHDOG-Funktion nutzen, um solche Situationen zu beheben. Es wird außerdem empfohlen, dass das Programm von Zeit zu Zeit (z. B. einmal pro Woche) einen Neustart mit CPU RESTART erzwingt, um sicherzustellen, dass die Protokollstacks zurückgesetzt werden.

## Verbindung zu einem WLAN-Netzwerk

Bevor du irgendetwas tust, musst du die WLAN-Funktion auf dem WebMite aktivieren. Dies geschieht mit dem folgenden Befehl, den du an der Befehlszeile eingibst:

```
OPTION WIFI ssid, password
```

Dabei ist „ssid“ der Name des WLAN-Netzwerks und „password“ das Sicherheitspasswort, das für den Zugriff auf das Netzwerk verwendet wird. Beide sind Zeichenketten, und wenn Zeichenkettenkonstanten verwendet werden, sollten sie wie in diesem Beispiel in Anführungszeichen gesetzt werden:

```
OPTION WIFI "MyNetwork", "secret"
```

Wenn du das eingibst, startet der WebMite neu, was bedeutet, dass du die USB-Verbindung und den Zugriff auf die Konsole verlierst. Wenn du dich schnell wieder verbindest, siehst du die folgende Meldung:

```
Connecting to WiFi...
Connected 192.168.1.52
```

Die in der letzten Zeile angegebene Adresse ist die IP-Adresse, die der Router dem WebMite zugewiesen hat, und variiert je nach Netzwerk. Wenn der WebMite keine Verbindung herstellen kann, siehst du diese Meldung:

```
Connecting to WiFi...
failed to connect.
```

Wenn die Verbindung fehlschlägt, solltest du die Konfiguration deines WLAN-Routers überprüfen:

- Die WLAN-Sicherheit muss WPA-PSK mit TKIP- oder AES-Verschlüsselung (oder beidem) sein.
- Für den WLAN-Zugang muss ein Passwort festgelegt sein.
- DHCP muss konfiguriert sein.

Dieser Befehl muss nur einmal eingegeben werden und wird bei jedem Neustart deines WebMite gespeichert. Du kannst die Einstellung mit dem Befehl `OPTION LIST` überprüfen. Wenn die Verbindung hergestellt ist, kannst du die zugewiesene IP-Adresse wie folgt überprüfen:

```
> PRINT MM.Info(IP-Adresse)
192.168.1.52
```

Bei manchen Routern kann es etwas dauern oder mehrere Versuche erfordern, bis die Verbindung hergestellt ist. Wenn du also OPTION AUTORUN ON verwendest, lohnt es sich, gleich am Anfang deines Programms etwa Folgendes einzufügen:

```
DO WHILE MM.INFO(IP ADDRESS) = "0.0.0.0"
  IF TIMER > 5000 THEN CPU RESTART
LOOP
```

Dadurch wird 5 Sekunden lang auf eine Verbindung gewartet und ein Neustart durchgeführt, falls noch keine Verbindung besteht.

## Fernzugriff auf die Konsole

Du kannst dich über WLAN per Telnet aus der Ferne mit der Konsole des WebMite verbinden. Das ist praktisch, wenn sich das Gerät an einem schwer zugänglichen Ort befindet. Sobald die Verbindung über Telnet hergestellt ist, kannst du alles tun, was du normalerweise über die USB-Konsole tun würdest, einschließlich des Startens des Editors.

Um diese Funktion zu aktivieren, gib Folgendes in die Befehlszeile ein: `OPTION TELNET CONSOLE ON`

Dies muss nur einmal eingegeben werden und wird bei jedem Neustart deines WebMite gespeichert. Außerdem wird der WebMite dadurch neu gestartet, sodass du die Verbindung zur USB-Konsole erneut herstellen musst.

Tera Term ( http://tera-term.en.lo4d.com ) unterstützt Telnet und wird für diese Aufgabe empfohlen, da es über eine gute VT100-Emulation verfügt und das XModem-Dateiübertragungsprotokoll unterstützt.

## Dateiübertragung

Dateien können über XModem oder TFTP zum und vom WebMite übertragen werden. XModem wird von Tera Term unterstützt und funktioniert über Telnet genauso wie über eine direkte serielle Verbindung. TFTP ist jedoch viel schneller und zuverlässiger als XModem, daher ist es die empfohlene Methode für die Dateiübertragung zum und vom WebMite.

Ein TFTP-Server auf dem WebMite wird automatisch aktiviert, sobald du mit einem WLAN-Netzwerk verbunden bist, sodass dort keine weiteren Schritte erforderlich sind. Du benötigst jedoch einen TFTP-Client für deinen PC, und für Windows, Mac OS und Linux stehen viele verschiedene Implementierungen zur Verfügung. Beachte, dass viele Tutorials zu TFTP die Einrichtung eines TFTP-Servers behandeln – dies ist nicht erforderlich, du benötigst lediglich einen Client.

Unter Windows ist ein TFTP-Client im Betriebssystem enthalten, muss aber über die Systemsteuerung aktiviert werden. Wähle dazu:

```
Systemsteuerung -> Programme und Funktionen -> Windows-Funktionen
aktivieren oder deaktivieren
```

Scrolle dann in der Liste nach unten und setze ein Häkchen bei „`TFTP-Client`“.

Um eine Datei von deinem PC an den WebMite zu senden, führe Folgendes in einem Befehlszeilen- oder PowerShell-Fenster auf deinem Windows-PC aus („IP-Addr“ ist die IP-Adresse des WebMite):

```
TFTP -i IP-Addr PUT Dateiname
```

Und um eine Datei vom WebMite auf deinen PC zu kopieren:

```
TFTP -i IP-Addr GET Dateiname
```

Um die Funktionen des TFTP-Clients aufzulisten, verwende Folgendes:

```
TFTP -h
```

Eine alternative und einfache grafische Windows-TFTP-Client-Anwendung ist: http://www.3iii.dk/linux/dd-wrt/tftp2.exe

## Die Uhrzeit abrufen

Ein üblicher erster Schritt in einem Programm ist es, die Uhrzeit/das Datum abzurufen und die Uhr im WebMite einzustellen. Diese Aktion bestätigt auch, dass der WebMite das Internet erreichen kann.

Das Abrufen der Uhrzeit erfolgt mit dem Befehl `WEB NTP` wie folgt:

```
WEB NTP [timeoffset [, NTPserver$]]
```

Dabei ist „timeoffset“ die Zeitzone, in der du dich befindest, und „NTPserver$“ ist der Name oder die IP-Adresse des zu verwendenden Zeitservers. Dieser letzte Parameter ist optional; wird er weggelassen, verwendet die Firmware einen öffentlichen Zeitserver-Pool. Wird auch der Parameter „timeoffset“ weggelassen, wird die Uhr des WebMite auf UTC eingestellt.

Hier ein typisches Beispiel für ein Gerät in der Zeitzone von Los Angeles:

```
> WEB NTP -10
ntp address 27.124.125.251
got ntp response: 08/03/2023 05:34:57
```

Wenn der WebMite keinen Zugang zum Internet hat, erhältst du eine Fehlermeldung. Dies kann mit dem Befehl `ON ERROR SKIP` abgefangen und eine geeignete Maßnahme ergriffen werden (z. B. Neustart oder Anzeige einer Meldung für den Bediener).

Zum Beispiel:

```
ON ERROR SKIP 3
WEB NTP -10
```

```
IF MM.ERRNO THEN WEB NTP -10
IF MM.ERRNO THEN PRINT "Verbindung zum Internet fehlgeschlagen" : CPU RESTART
```

Der Grund, warum wir den Befehl WEB NTP zweimal ausführen, ist, dass der erste Versuch möglicherweise aufgrund eines Timing-Fehlers fehlgeschlagen ist (das kann passieren) – beim zweiten Versuch sollte es jedoch klappen.

## Ein Webserver einrichten

Eine häufige Anforderung ist die Einrichtung eines Webservers, der die vom WebMite gesammelten Daten auf einer benutzerfreundlichen Webseite anzeigt.

Der erste Schritt besteht darin, die Serverfunktion mit diesem Befehl an der Eingabeaufforderung zu konfigurieren:

```
OPTION TCP SERVER PORT nn
```

Dabei ist „nn“ die zu verwendende Portnummer (normalerweise 80 für eine Webseite). Typischerweise lautet der Befehl:

```
OPTION TCP SERVER PORT 80
```

Wie bei den anderen oben aufgeführten OPTION-Befehlen muss dies nur einmal eingegeben werden und wird bei jedem Neustart des WebMite gespeichert. Dadurch wird der WebMite neu gestartet, und wenn du dich schnell wieder verbindest, siehst du Folgendes (mit einer anderen IP-Adresse):

```
Starting server at 192.168.1.52 on port 80
```

Mit dem obigen Schritt wurde die WebMite-Firmware so konfiguriert, dass sie einen TCP-Server unterstützt. In deinem Programm musst du den Server mit dem folgenden Befehl starten:

```
WEB TCP INTERRUPT InterruptSub
```

Dabei ist „InterruptSub“ der Name deiner Subroutine, die immer dann aufgerufen wird, wenn eine Anfrage beim TCP-Server eingeht. Diese Subroutine kann den Befehl `WEB TCP READ` verwenden, um die eingehende Anfrage vom Remote-Client zu lesen, und den Befehl `WEB TRANSMIT PAGE`, um die angeforderte Webseite zu senden.

Hier ist zum Beispiel das vollständige Programm zur Implementierung einer einfachen Webseite (vergiss nicht, zuerst den Befehl OPTION TCP SERVER zu verwenden):

```
1  DIM buff%(4096/8)
2  WEB TCP INTERRUPT WebInterrupt
3  DO
4     '<do some processing here>'
5  LOOP
6
7  SUB WebInterrupt
8     LOCAL a%, p%, t%, s$
9     FOR a% = 1 To MM.INFO(MAX CONNECTIONS)
10       WEB TCP READ a%, buff%()
11       p% = LINSTR(buff%(),"GET")
12       t% = LINSTR(buff%(),"HTTP")
13       If (p% <> 0) And (t% > p%) Then
14             WEB TRANSMIT PAGE a%,"index.html"
15       ENDIF
16    NEXT a%
17 END SUB
```

Im Folgenden wird dieses Programm im Detail beschrieben:

Zeile 1 Zunächst wird ein 4-KB-Puffer für die eingehende Anfrage vom Client erstellt. In diesem Beispiel werden die Befehle für lange Zeichenfolgen in MMBasic zur Verarbeitung der Daten verwendet, und dies ist der Puffer dafür (eine Beschreibung langer Zeichenfolgen findest du im nächsten Kapitel mit dem Titel *„Lange Zeichenfolgen*“). Die Größe dieses Puffers begrenzt die Menge der vom Client empfangenen Daten.

Zeile 2 Hiermit wird der Server gestartet und die Interrupt-Subroutine zur Bearbeitung eingehender Anfragen als „`WebInterrupt`“ festgelegt.

Zeile 4 Dies ist deine Hauptprogrammschleife, die normalerweise Daten sammelt, Ausgänge umschaltet usw.

Zeile 7 Dies ist die Subroutine, die jedes Mal aufgerufen wird, wenn eine Anfrage vom Browser des Remote-Clients gestellt wird.

Zeile 9 Durchlaufe alle eingehenden Verbindungen (es kann mehrere gleichzeitige Verbindungen geben).

Zeile 10 Lies die eingehende Nachricht in den langen Zeichenfolgenpuffer ein.

Zeilen 11 & 12 Ermittle die Positionen der Schlüsselwörter in der Nachricht.

Zeile 13 Prüfe, ob die Schlüsselwörter vorhanden sind und in der richtigen Reihenfolge stehen.

Zeile 14 Sende die Seite. Es handelt sich um eine Datei namens index.html, die sich im Standardverzeichnis auf dem internen Flash-Dateisystem oder der SD-Karte befindet. Sie ist im HTML-Format, was bedeutet, dass sie Tags wie `<h1>This is a heading</h1>`enthalten kann. Weitere Informationen findest du unten unter *„Eine typische Webseite*“.

## Daten in die Webseite einfügen

Normalerweise musst du Daten, die vom BASIC-Programm generiert wurden, in die zu übertragende Webseite einfügen. Dies geschieht ganz einfach, indem du den Namen der BASIC-Variablen, umgeben von geschweiften Klammern (d. h. { und }), in den Text der Webseite einfügst.

Wenn dein Programm beispielsweise eine Variable namens `CurrentTemp` enthält, die den Wert 24 hat und die aktuelle Temperatur angibt, würde die folgende Webseite: `Die Temperatur beträgt {CurrentTemp}` im Browser des Clients wie folgt angezeigt werden: `Die Temperatur beträgt 24`.

Der Bezeichner zwischen den geschweiften Klammern kann ein Float, eine Ganzzahl oder eine Zeichenkette, ein Array-Element und sogar ein Ausdruck (z. B. A + B) sein. Du kannst auch Funktionen verwenden. Wenn du also eine Gleitkommazahl mit der richtigen Anzahl an Dezimalstellen formatieren möchtest, kannst du die Formatierungsfunktion Str$() verwenden. Beachte, dass ein Fehler im Ausdruck einen entsprechenden Fehler verursacht, wenn der Befehl `WEB TRANSMIT PAGE` ausgeführt wird.

Wenn du eine öffnende geschweifte Klammer in deiner Webseite verwenden musst, kannst du zwei als Paar verwenden (d. h. {{}), und diese wird in eine einzelne öffnende Klammer umgewandelt. Eine schließende geschweifte Klammer ohne öffnendes Gegenstück bleibt unverändert.

## Mehrere Seiten senden

Wenn ein Remote-Client Daten anfordert, ohne eine Seite anzugeben, sendet er die Anfrage als `GET / HTTP`, wobei der Schrägstrich die Standardseite des Servers darstellt (normalerweise index.html). Das obige Beispiel hat dies nicht überprüft, sondern einfach bei allen Anfragen dieselbe Seite gesendet.

Wenn deine Seite jedoch anklickbare Links wie `<a href="page2.html">Nächste Seite</a>` enthält und der Benutzer auf diesen Link klickt, sendet der Remote-Client eine weitere Anfrage mit `GET /page2.html HTTP`.

Dies lässt sich leicht berücksichtigen, indem du die angeforderten Daten zwischen den Schlüsselwörtern GET und HTML überprüfst. Ersetze zum Beispiel Zeile 15 im obigen Beispiel durch Folgendes (s$ ist die angeforderte Datei):

```
s$ = LGetStr$(buff%(), p% + 4, t% - p% - 5)
IF s$ = "/" THEN
   WEB TRANSMIT PAGE a%,"index.html"
ELSE IF s$ = "/page2.html" THEN
   WEB TRANSMIT PAGE a%,"page2.html"
ENDIF
```

Das lässt sich auf beliebig viele Seiten ausweiten.

## Ein Bild senden

Du kannst ein Bild in deine Webseite einfügen, indem du den folgenden HTML-Code verwendest: `<img src="pix.jpg"/>`. Wenn der Remote-Browser dies liest, sendet er die folgende `GET`-Anfrage: `GET /pix.jpg HTTP`. Du kannst dann das angeforderte Bild mit dem fettgedruckten Code senden:

```
s$ = LGetStr$(buff%(), p% + 4, t% - p% - 5)
IF s$ = "/" THEN
   WEB TRANSMIT PAGE a%,"index.html"
ELSE IF s$ = "/page2.html" THEN
   WEB TRANSMIT PAGE a%,"page2.html"
ELSE IF s$ = "/pix.jpg" THEN
   WEB TRANSMIT FILE a%,"pix.jpg","image/jpeg"
ENDIF
```

Beachte, dass `pix. jpg` ein JPEG-Bild sein muss, das sich im Standardverzeichnis des internen Flash-Dateisystems oder auf der SD-Karte befindet. Der WebMite ist kein schneller Server, daher sind kleine und einfache Bilder vorzuziehen.

Der Parameter „image/jpeg“ wird als MIME-Typ bezeichnet, und es gibt viele verschiedene Typen; andere gängige Bildtypen sind image/bmp, image/png und image/gif.

### Antwort „Seite nicht gefunden“ (404)

Wenn ein Remote-Client eine Seite oder eine Datei anfordert, die von deinem Programm nicht unterstützt wird, kannst du den Befehl WEB `TRANSMIT CODE` verwenden, um wie folgt einen 404-Fehler zu senden:

```
s$ = LGetStr$(buff%(), p% + 4, t% - p% - 5)
IF s$ = "/" THEN
   WEB TRANSMIT PAGE a%,"index.html"
ELSE IF s$ = "/page2.html" THEN
   WEB TRANSMIT PAGE a%,"page2.html"
ELSE IF s$ = "/pix.jpg" THEN
   WEB TRANSMIT FILE a%,"pix.jpg","image/jpeg"
ELSE
   WEB TRANSMIT CODE a%, 404
ENDIF
```

## Live-Grafikdaten auf einer Webseite

Zahlen und Text auf einer Webseite darzustellen ist nützlich, aber oft möchtest du auch grafische Elemente wie Kreisdiagramme, Liniendiagramme, historische Trends usw. einbinden, die aus den vom Programm gesammelten Daten abgeleitet wurden. Das lässt sich auf Umwegen bewerkstelligen, indem du WebMite mit einem virtuellen Anzeigefeld konfigurierst, dann auf diesem -Display mit den Standard-Zeichenbefehlen (Pixel, Linie, Kreis usw.) zeichnest und das Ergebnis als BMP-Bild speicherst. Diese Datei kann dann als Bild in die Webseite eingebunden werden. Im Einzelnen läuft dieser Vorgang wie folgt ab:

Zunächst muss ein virtuelles Display konfiguriert werden. Es stehen zwei zur Auswahl: VIRTUAL_C ist ein 320x240-Pixel-Bild mit 16 Farben und VIRTUAL_M ist ein 640x480-Pixel-Monochrom-Bild. Zum Beispiel:

```
OPTION LCDPANEL VIRTUAL_C
```

Wie bei den anderen OPTION-Befehlen muss dies an der Befehlszeile eingegeben werden, führt zu einem Neustart und muss nur einmal eingegeben werden.

Anschließend kannst du in deinem Programm mit den im Kapitel *„Grafikbefehle und -funktionen“* beschriebenen Befehlen Bilder und Text auf diesem „Display“ zeichnen. Zum Beispiel:

```
CIRCLE 100, 100, 50, 1, 1, RGB(red), RGB(blue)
LINE 10, 10, 200, 200, 1, RGB(yellow)
```

Wenn du fertig bist, speichere dieses Bild:

```
SAVE COMPRESSED IMAGE "graph.bmp"
```

Innerhalb der Webseite, die du bereitstellst, kannst du dieses Bild mit dem folgenden HTML-Code einfügen:

```
<img src="graph.bmp"/>
```

Schließlich musst du in deinem BASIC-Programm dafür sorgen, dass diese Datei gesendet wird, wenn die Webseite vom Remote-Browser geladen wird, wie oben beschrieben (siehe *„Ein Bild senden“*). Füge zum Beispiel Folgendes in die oben beschriebene ELSE IF-Kette ein:

```
ELSE IF s$ = "/graph.bmp " THEN
  WEB TRANSMIT FILE a%, "graph.bmp", "image/bmp"
```

Dein BASIC-Programm muss diese Datei aktualisieren, sobald neue Daten aufgezeichnet werden. Da dies jedoch nur dann erforderlich ist, wenn der Remote-Browser das Bild anfordert, sollte dies keine übermäßige Beanspruchung des Flash-Speichers verursachen.

## Ein vollständiger Allzweckserver

Auf den vorangegangenen Seiten wurden die einzelnen Komponenten beschrieben, aus denen ein Webserver besteht. Im Folgenden findest du ein Beispiel für einen vollständigen und integrierten Allzweckserver, der die meisten Anfragen eines Browsers bearbeiten kann. Er prüft zunächst die Dateiendung der angeforderten Datei und verwendet dann den entsprechenden WEB TRANSMIT-Befehl, um die angeforderten Daten zu senden. Er kann als Drop-in-Modul für jedes Projekt verwendet werden, bei dem der WebMite als Webserver fungieren soll.

```
WEB TCP INTERRUPT WebInterrupt
DO
   '<do some processing here>'
LOOP

' sub to handle all web server requests
SUB WebInterrupt
  LOCAL a%, p1%, p2%, file$, buff%(4096/8)
  FOR a% = 1 To MM.INFO(MAX CONNECTIONS)
    WEB TCP READ a%, buff%()
    P1% = LINSTR(buff%(),"GET")
    P2% = LINSTR(buff%(),"HTTP")
    If (p1% <> 0) And (p2% <> 0) And (p2% > p1%) Then
      file$ = LCASE$(LGetStr$(buff%(), p1% + 4, p2% - p1% - 5))
      IF file$ = "/" THEN file$ = "/index.html"
      ON ERROR SKIP
      OPEN file$ FOR INPUT AS #1       ' check that the file exists
      IF MM.ERRNO THEN WEB TRANSMIT CODE a%, 404 : CONTINUE FOR
      CLOSE #1
      SELECT CASE RIGHT$(file$, 4)
        CASE "html", ".htm", ".txt"
          WEB TRANSMIT PAGE a%, file$
        CASE ".bmp", ".png", ".gif"
          WEB TRANSMIT FILE a%, file$, "image/" + RIGHT$(file$, 3)
        CASE ".jpg", "jpeg"
          WEB TRANSMIT FILE a%, file$, "image/jpeg"
        CASE ".ico"
          WEB TRANSMIT FILE a%, file$, "image/vnd.microsoft.icon"
      END SELECT
    ENDIF
  NEXT a%
END SUB
```

Beachte, dass alle Dateien im Stammverzeichnis des Flash-Dateisystems oder auf der SD-Karte gespeichert werden müssen und ihre Namen ausschließlich Kleinbuchstaben enthalten dürfen (das Flash-Dateisystem unterscheidet zwischen Groß- und Kleinschreibung).

## Eine typische Webseite

Die WEB-Seite, die über den Befehl WEB TRANSMIT PAGE übertragen werden soll, muss gemäß dem HTML-Standard aufgebaut sein. Sie kann so einfach wie eine einzelne Textzeile ohne Formatierung sein, z. B.:

```
Die Temperatur beträgt {CurrentTemp}
```

Oder du kannst eine einfache Formatierung einfügen:

```
<title>WebMite</title>
<h2>Temperaturmonitor</h2>
Die Temperatur beträgt {CurrentTemp}
```

Oder du kannst eine komplexe Seite senden. Typischerweise haben diese einen Kopf- und einen Hauptteil, gliedern den Text in Absätze und verwenden das Break-Tag für Abstände. Dies ist das Grundgerüst einer solchen Seite:

```
<html>
<head>
<title>WebMite</title>
</head>
<body>
<h1>Das ist eine Überschrift</h1>
<br>
<p>Die Temperatur beträgt {CurrentTemp}</p>
</body>
</html>
```

Im Internet gibt es viele Ressourcen, die HTML-Tutorials für Anfänger anbieten. Ein typisches Beispiel ist http://www.simplehtmlguide.com/. Außerdem gibt es zahlreiche WYSIWYG-HTML-Editoren. Zum Beispiel: https://onlinehtmleditor.dev/

## Eingabefelder und Steuerung

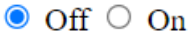
Mit HTML-Eingabefeldern kannst du Schaltflächen, Kontrollkästchen, Optionsfelder usw. auf deiner Webseite platzieren, über die der Benutzer Anfragen an den WebMite senden kann. Auf diese Weise kann der Benutzer Dinge aus der Ferne ein- und ausschalten, Steuerparameter einstellen und vieles mehr – alles über die vom WebMite bereitgestellte Webseite.

Dazu musst du ein *HTML-Formular* in die Webseite einfügen, das ein oder mehrere Eingabefelder enthält. Es stehen viele Arten von Eingabefeldern zur Auswahl (siehe https://www.w3schools.com/tags/att_input_type.asp), aber in unserem einfachen Beispiel verwenden wir zwei Optionsfelder, um ein fiktives Gerät ein- und auszuschalten.

Der folgende Code muss in die Standard-Webseite (d. h. index.html) eingefügt werden:

```
<form method='get'>
    <input name='RB' type='radio' value='OFF' {offb$} onClick='this.form.submit()'> Off
    <input name='RB' type='radio' value='ON' {onb$} onClick='this.form.submit()'> On
</form>
```

Dadurch werden zwei Optionsfelder erstellt, die im Browser wie folgt aussehen:




Beachte, dass der obige HTML-Code zwei BASIC-Variablen enthält (`offb$` und `onb$`), die durch den Befehl WEB TRANSMIT PAGE ersetzt werden. Diese Variablen steuern, wie die Schaltfläche angezeigt wird. Wenn sie auf eine leere Zeichenkette gesetzt sind, wird die Schaltfläche als nicht markiert angezeigt; wenn sie hingegen auf die Zeichenkette „`checked='checked'`“ gesetzt sind, wird die Schaltfläche als markiert angezeigt. Diese Variablen sollten initialisiert werden, sobald das Programm läuft.

Wenn der Benutzer auf die erste Schaltfläche klickt, sendet der Browser eine Anfrage mit folgendem Inhalt: GET /?RB=OFF HTML
und wenn auf die zweite geklickt wird, lautet die Anfrage: GET /?RB=ON HTML

In der ELSE IF-Kette in der TCP-Interrupt-Subroutine können wir auf diese Anfragen reagieren:

```
ELSE IF s$ = "/?RB=OFF"
    ' <insert code here to turn off the device>
    offb$ = "checked='checked'" : onb$ = ""
    WEB TRANSMIT PAGE a%, "index.html"
ELSE IF s$ = "/?RB=ON"
    ' <insert code here to turn on the device>
    offb$ = "" : onb$ = "checked='checked'"
    WEB TRANSMIT PAGE a%, "index.html"
```

Im Wesentlichen bewirkt dieser Code nur, dass das Gerät wie gewünscht ein- oder ausgeschaltet wird, die Variablen `onb$` und `offb$` so gesetzt werden, dass sie den neuen Status der Schaltflächen widerspiegeln, und dann die gesamte Webseite zurück an den Browser gesendet wird.

Dieses Beispiel hat viele Details ausgelassen, und du kannst es extrem kompliziert gestalten, wenn du möchtest – mit mehreren Eingaben über Schaltflächen, Texteingaben, Dropdown-Listen, Passwortfeldern, Anfragen für Datei-Uploads und vielem mehr. Dazu musst du dich jedoch tiefer in die HTML-Programmierung einarbeiten. Ein Beispiel dafür findest du im Projekt *„Garten-Bewässerungssteuerung“* unter: https://geoffg.net/retic.html

## Implementierung eines TCP-Clients

Der WebMite kann auch als TCP-Client fungieren, um Daten von einem Remote-Server anzufordern. Dies wird mit drei Befehlen gesteuert. Der erste lautet:

```
WEB OPEN TCP CLIENT Domain$, PortNumber
```

Dies öffnet eine TCP-Verbindung zu `Domain$` (zum Beispiel „openweathermap.org“) unter Verwendung der angegebenen `PortNumber` (normalerweise 80 für eine Webseite).

Sobald die Verbindung besteht, kannst du mit diesem Befehl eine oder mehrere Anfragen senden:

```
WEB TCP CLIENT REQUEST query$, inbuf [, timeout]
```

Die zu sendende Anfrage lautet „query$“ und die Antwort wird in „inbuf“ gespeichert, was normalerweise eine lange Zeichenfolgenvariable ist, wie z. B. `buff%( 4096/8 )`. Die Größe dieses Puffers (in Byte) begrenzt die vom Server empfangene Datenmenge und sollte erhöht werden, wenn mehr Daten erwartet werden.

„timeout“ ist optional und gibt die Zeitüberschreitung in Millisekunden an.

Wenn du auf eine Website zugreifst, kann „query$“ so einfach sein wie „`GET / HTTP`“, wodurch die Standardseite dieser Website abgerufen wird. Dein Programm ist dann dafür zuständig, die gewünschten Daten aus der Antwort herauszufiltern.

Zum Schluss schließt du die Verbindung mit:

```
WEB CLOSE TCP CLIENT
```

Das folgende Beispiel ist ein vollständiges Programm (unter Verwendung dieser Befehle), um die aktuelle Temperatur für die Stadt Paris von openweathermap.com abzurufen. Damit dies funktioniert, benötigst du ein (kostenloses) Konto bei Open Weather Map (https://openweathermap.org), das einen API-Schlüssel enthält. Dies ist eine 32-stellige Hexadezimalzahl, mit der du die Abfrage durchführen kannst. Diese Zahl sollte anstelle des Platzhalters in der ersten Zeile eingesetzt werden.

```
CONST Key = "nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn"
CONST Query = "GET /data/2.5/weather?q=Paris,fra&APPID="+Key+Chr$(13)+Chr$(10)
DIM buff%(4096/8)

WEB OPEN TCP CLIENT " api.openweathermap.org ", 80
WEB TCP CLIENT REQUEST Query, buff%()
WEB CLOSE TCP CLIENT
temp = VAL(JSON$(buff%(), "main.temp"))
PRINT "Die aktuelle Temperatur in Paris beträgt:" temp - 273
```

Die Funktion JSON() wird verwendet, um den gewünschten Wert aus der im JSON-Format (JavaScript Object Notation) zurückgegebenen Antwort zu extrahieren.

Wenn dieses Programm ausgeführt wird, solltest du etwa Folgendes sehen:

```
Connected
Die aktuelle Temperatur in Paris beträgt: 13,54
```

## Verwendung von UDP

Das User Datagram Protocol (UDP) ist ein schnelles und schlankes Protokoll, das zum Austausch von Nachrichten zwischen Geräten verwendet wird. Es kann unzuverlässig sein, da es nicht garantiert, dass Datenpakete zugestellt werden oder in der richtigen Reihenfolge ankommen; dennoch wird es häufig für zeitkritische Anwendungen genutzt, bei denen Geschwindigkeit Vorrang vor Zuverlässigkeit hat.

Das Einrichten eines UDP-Servers ähnelt dem Einrichten eines TCP-Servers. Zunächst wird UDP durch Eingabe der folgenden Option an der Eingabeaufforderung aktiviert (dies führt auch zu einem Neustart):

```
OPTION UDP SERVER PORT port_nbr
```

In deinem Programm musst du den Server mit dem folgenden Befehl starten:

```
WEB UDP INTERRUPT InterruptSub
```

Dabei ist „InterruptSub“ der Name deiner Subroutine, die immer dann aufgerufen wird, wenn eine Nachricht vom UDP-Server empfangen wird. In diesem Fall wird die IP-Adresse des sendenden Geräts in der schreibgeschützten Variablen MM.ADDRESS$ gespeichert und die empfangene Nachricht in MM.MESSAGE$.

Du kannst eine UDP-Nachricht wie folgt senden:

```
WEB UDP send ip_address$, port_nbr, message$
```

Die folgende Demonstration verwendet zwei WebMites, die mit OPTION WIFI im selben Netzwerk konfiguriert sind.

Vor dem Start müssen beide WebMites mit OPTION UDP SERVER PORT 77 konfiguriert werden.

Notiere dir die IP-Adresse des ersten WebMites und gib das folgende Programm ein:

```
WEB UDP interrupt myint
Do
   '<do some processing here>'
Loop

' interrupt subroutine
Sub myint
   Print "Received " + mm.message$ + " from " + mm.address
   pause 100
   WEB UDP send mm.address$, 77, " WebMite #1 message"
End Sub
```

Führe dann das Programm aus, das zunächst in einer Schleife hängt und nichts tut.

Gib auf dem zweiten WebMite das folgende Programm ein und ersetze dabei die IP-Adresse durch die des ersten WebMite. Führe dann das Programm aus:

```
WEB UDP interrupt myint
WEB UDP send "192.168.1.127", 77, "Starting UDP echo"
Do
```

```
   '<do some processing here>'
Loop

' interrupt subroutine
Sub myint
   Print "Received " + mm.message$ + " from " + mm.address$
   pause 100
   WEB UDP send mm.address$, 77, "WebMite #2 message"
End Sub
```

Das Ergebnis ist ein sehr schneller Austausch von UDP-Nachrichten zwischen den beiden WebMites.

## E-Mails senden

Wenn du ein Remote-Gerät wie den WebMite hast, ist es nützlich, wenn es E-Mails versenden kann, um bei Störungen Alarm zu schlagen, über seinen Status zu berichten und so weiter. Der WebMite kann dies tun, indem er sich über das SMTP-Protokoll mit einem Server verbindet, der die E-Mail dann an ihren Bestimmungsort weiterleitet.

Das folgende Beispiel nutzt den kostenlosen SMTP-Relay-Dienst, der unter von SMTP2GO angeboten wird und 1000 E-Mails pro Monat zulässt (mehr als genug für den WebMite). Beachte, dass Registrierungsanfragen von Personen mit einer generischen kostenlosen E-Mail-Adresse (z. B. xxx@gmail.com) nicht akzeptiert werden.

Um loszulegen, musst du dich kostenlos unter SMTP2GO ( https://www.smtp2go.com/ ) anmelden, einen verifizierten Absender registrieren und einen zugehörigen Benutzernamen und ein Passwort erstellen. Beide müssen dann in Base64-kodierte Zeichenfolgen umgewandelt werden (die folgende Website erledigt das für dich: https://www.base64encode.org ).

Der Base64-kodierte Benutzername sollte verwendet werden, um die Zeichenfolge „ `nnnnnnnnn` “ in der ersten Zeile des folgenden Programms zu ersetzen, während das Base64-kodierte Passwort verwendet werden sollte, um das „xxxxxxxxxxxx“ in der zweiten Zeile zu ersetzen. Die anderen vier Zeilen am Anfang des Programms sollten ebenfalls durch deine Daten ersetzt werden:

```
CONST userBase64$ =  "nnnnnnnnnnn"
CONST paswdBase64$ = "xxxxxxxxxxx"
CONST mailfrom$ =    "from@server.com"
CONST mailto$ =      "to@server.com"
CONST subject$ =     "Test EMail"
CONST message$ =     "Test of SMTP2GO"

CONST cr = Chr$(13)+Chr$(10)
DIM buff%(4096/8), body$

body$ = "From: " + mailfrom$ + cr + "To: " + mailto$ + cr
body$ = body$ + "Subject: " + subject$ + cr + cr
body$ = body$ + message$ + cr + "." + cr

WEB OPEN TCP CLIENT "mail.smtp2go.com", 2525
WEB TCP CLIENT REQUEST "EHLO" + cr, buff%()
WEB TCP CLIENT REQUEST "AUTH LOGIN" + cr, buff%()
WEB TCP CLIENT REQUEST userBase64$ + cr, buff%()
WEB TCP CLIENT REQUEST paswdBase64$ + cr, buff%()
WEB TCP CLIENT REQUEST "MAIL FROM: " + mailfrom$ + cr, buff%()
WEB TCP CLIENT REQUEST "RCPT TO: " + mailto$ + cr, buff%()
WEB TCP CLIENT REQUEST "DATA" + cr, buff%()
PAUSE 300
WEB TCP CLIENT REQUEST body$, buff%()
PAUSE 300
WEB CLOSE TCP CLIENT
IF LINSTR(buff%(), "250 OK") = 0 THEN
  PRINT "Email send failed"
Else
  Print "Email sent OK"
EndIf
```

Beachte, dass die im obigen Programm verwendete E-Mail-Adresse „mailfrom$“ genau mit der übereinstimmen MUSS, die du bei der Registrierung des verifizierten Absenders bei SMTP2GO angegeben hast. Wenn sie nicht identisch sind, lehnt SMTP2GO die E-Mail ab (dies ist eine Maßnahme gegen Spam).

Heutzutage wird die Nutzung eines SMTP-Relay-Dienstes durch die unterschiedlichen SMTP-Protokolle der einzelnen Anbieter und die verschiedenen Schutzmaßnahmen zur Reduzierung von Spam erschwert. Dieses Beispiel bezieht sich speziell auf das von SMTP2GO verwendete SMTP-Protokoll; es können jedoch auch andere Dienste genutzt werden, wobei das Programm an deren jeweilige SMTP-Protokollversion angepasst werden muss.

## Base64-Kodierung

Base64 ist ein System zur Umwandlung von Binärdaten in eine Textzeichenfolge, die ausschließlich ASCII-Zeichen verwendet (d. h., es gibt keine Steuerzeichen). Es wurde entwickelt, um das Versenden von Binärdaten über das Internet mit Protokollen zu vereinfachen, die keine Binärdaten akzeptieren, und viele Protokolle verlangen dessen Verwendung.

Die MATH-Funktion BASE64-encode/decode übernimmt die Kodierung und Dekodierung für dich (die vollständige Syntax findest du in der detaillierten Funktionsliste). Eine typische Anwendung ist das oben genannte Programm zum Versenden von E-Mails. Anstatt einen externen Dienst zu nutzen, um Benutzername und Passwort in Base64 zu konvertieren, kannst du dies mit dieser Funktion direkt im Programm erledigen.

Zum Beispiel:

```
DIM n, userBase64$, paswdBase64$
CONST user$ = "MyUserName"        ' user name in plain text
CONST paswd$ = "MyPassword"       ' password in plain text
...
' encode the user name and password
n = MATH(BASE64 ENCODE, user$, userBase64$)
n = MATH(BASE64 ENCODE, paswd$, paswdBase64$)
...
```

## MQTT-Client

MQTT ist ein Protokoll, das es Clients wie dem WebMite ermöglicht, Nachrichten auf einem Server (auch MQTT-Broker genannt) zu veröffentlichen oder abzurufen. Das funktioniert ähnlich wie ein Schwarzes Brett oder ein webbasiertes Forum, wo Leute Nachrichten posten, die andere später nach Belieben lesen können – der Hauptunterschied ist, dass MQTT für die Kommunikation zwischen Maschinen konzipiert ist.

Ein typisches Beispiel wäre ein batteriebetriebener WebMite, der den Wasserstand in einem Stausee überwacht. Zweimal täglich würde er sich einschalten, die Messung vornehmen, das Ergebnis an einen MQTT-Broker senden und sich wieder ausschalten. Ein Client-Programm (vielleicht auf einem PC) könnte diese Nachrichten später lesen, die Ergebnisse anzeigen und grafisch darstellen.

Es gibt viele kostenlose Broker; suche bei Google nach „kostenloser MQTT-Broker“.

Der WebMite verwendet fünf Befehle zum Senden oder Abrufen von Nachrichten:

```
WEB MQTT CONNECT        Connect to an MQTT broker.
WEB MQTT PUBLISH        Publish content to an MQTT topic (ie, post a messages).
WEB MQTT SUBSCRIBE      Subscribe to an MQTT topic (ie, retrieve messages).
WEB MQTT UNSUBSCRIBE    Unsubscribe from an MQTT topic.
WEB MQTT CLOSE          Close the MQTT connection.
```

## Ping

Der WebMite reagiert auf eine Ping-Nachricht, sodass du überprüfen kannst, ob er aktiv und erreichbar ist. Wenn er mit dem öffentlichen Internet verbunden ist, kannst du einen kostenlosen Dienst wie https://uptimerobot.com/ nutzen, der dich benachrichtigt, falls er nicht mehr läuft.

## Audio-Streaming

Die Befehle WEB OPEN TCP STREAM und WEB TCP CLIENT STREAM können zusammen mit dem Befehl PLAY STREAM ganz einfach eine grundlegende Internetradio-Funktion implementieren. Dies wird im folgenden Code demonstriert, der das britische Programm ClassicFM empfängt. Hinweis: Für dieses Programm ist ein VS1053-Audio-Codec erforderlich.

```
Option escape
Option default none
' create the request for the radio site (ClassicFM)
Dim a$="ice-the.musicradio.com"
Dim q$="GET "
Inc q$,"/ClassicFMMP3"
Inc q$," HTTP/1.1\r\n"
Inc q$,"Host: "
Inc q$,a$
Inc q$,"\r\nConnection: close\r\n\r\n"

'create a circular buffer for reading the internet stream and
'read and write pointers
Dim buff%(4095),w%,r%

' Configure the VS1053 and tell it to play from the circular buffer
Play stream buff%(), r%, w%

' Open the internet radio site
WEB open tcp stream a$,80

' Send the request to start the stream using the circular buffer specified
WEB TCP CLIENT STREAM q$, buff%(), r%, w%

'sit back and listen
Do : Pause 500: Loop
```

# Lange Zeichenfolgen

Lange Zeichenfolgen sind eine Reihe von Befehlen und Funktionen, mit denen MMBasic Zeichenfolgen beliebiger Länge bearbeiten kann. Sie sind besonders nützlich beim Umgang mit Daten, die über WLAN und das Internet gesendet werden. Standard-Zeichenfolgen in MMBasic sind auf eine maximale Länge von 255 Zeichen begrenzt. Lange Zeichenfolgen duplizieren diese Funktionen, funktionieren aber mit Zeichenfolgen beliebiger Länge, die nur durch die Menge des verfügbaren RAM begrenzt sind.

## Variablen für lange Zeichenfolgen

Variablen zur Speicherung langer Zeichenfolgen müssen als Integer-Arrays definiert werden. Die Routinen für lange Zeichenfolgen speichern keine Zahlen in diesen Arrays, sondern nutzen sie lediglich als Speicherblöcke zur Speicherung langer Zeichenfolgen.

Beim Erstellen dieser Arrays sollten sie als eindimensionale Integer-Arrays definiert werden, wobei die Anzahl der Elemente auf die für die maximale Zeichenfolgenlänge erforderliche Zeichenanzahl geteilt durch acht gesetzt wird. Der Grund für die Division durch acht ist, dass jede Ganzzahl in einem MMBasic-Array acht Bytes belegt.

Hier ist ein Beispiel für die Deklaration von drei Variablen für lange Zeichenfolgen, die jeweils bis zu 2048 Zeichen aufnehmen können:

```
CONST MaxLen = 2048
DIM INTEGER Str1(MaxLen/8), Str2(MaxLen/8), Str3(MaxLen/8)
```

Diese enthalten bei ihrer Erstellung leere Zeichenfolgen (d. h., ihre Länge beträgt Null). Wenn diese Variablen an die Long-String-Funktionen übergeben werden, sollten sie als Variablenname gefolgt von leeren Klammern eingegeben werden. Zum Beispiel:

```
LONGSTRING COPY Str1(), Str2()
```

Lange Zeichenfolgenvariablen können als Argumente an benutzerdefinierte Unterprogramme und Funktionen übergeben werden. Zum Beispiel:

```
Sub MySub longarg() AS INTEGER
  PRINT  "Die Länge der langen Zeichenkette beträgt" LLEN(longarg())
END SUB
```

Und so könnte man sie aufrufen:

```
MySub str1()
```

## Befehle für lange Zeichenfolgen

Diese sind ausführlich in dem Abschnitt *„Befehle* und *Funktionen“* dieses Handbuchs dokumentiert. Die Befehle lauten:

| | |
|---|---|
| LONGSTRING AES128 ENCRYPT/DECRYPT | Verschlüsselt oder entschlüsselt eine lange Zeichenkette |
| LONGSTRING APPEND array%(), string$ | Hängt eine normale Zeichenkette an eine lange Zeichenkette an |
| LONGSTRING BASE64 ENCODE/DECODE | Kodiert oder dekodiert eine lange Zeichenkette mit Base64 |
| LONGSTRING CLEAR array%() | Löscht (d. h. setzt auf leer) eine lange Zeichenkette |
| LONGSTRING COPY dest%(), src%() | Kopiert eine lange Zeichenkette |
| LONGSTRING CONCAT dest%(), src%() | Zwei lange Zeichenfolgen verketten |
| LONGSTRING LCASE array%() | Eine lange Zeichenkette in Kleinbuchstaben umwandeln |
| LONGSTRING LEFT dest%(), src%(), nbr | Die ersten nbr Zeichen einer langen Zeichenkette abrufen |
| LONGSTRING LOAD array%(), nbr, string$ | Zeichen in eine lange Zeichenkette kopieren |
| LONGSTRING MID dest%(), src%(), start, nbr | Zeichen aus der Mitte einer langen Zeichenkette abrufen |
| LONGSTRING PRINT [#n,] src%() [;] | Eine lange Zeichenkette ausgeben |
| LONGSTRING REPLACE array%() , string$, start | Zeichen in einer langen Zeichenkette ersetzen |
| LONGSTRING RESIZE addr%(), nbr | Länge einer langen Zeichenkette festlegen |
| LONGSTRING RIGHT dest%(), src%(), nbr | Die letzten nbr Zeichen aus einer langen Zeichenkette abrufen |
| LONGSTRING SETBYTE addr%(), nbr, data | Ein Byte in einer langen Zeichenkette setzen |
| LONGSTRING TRIM array%(), nbr | Zeichen am Anfang einer langen Zeichenkette entfernen |
| LONGSTRING UCASE array%() | Eine lange Zeichenkette in Großbuchstaben umwandeln |
| LMID(array%(),start [,num])=string$ | Text in einen langen String einfügen/ersetzen |

## Funktionen für lange Zeichenfolgen

| | |
|---|---|
| r = LGETBYTE(array%(), n) | Gibt den Wert eines Bytes in einer langen Zeichenkette zurück |
| r$ = LGETSTR$(array%(), start, length) | Gibt einen Teil einer langen Zeichenkette als normale Zeichenkette zurück. |
| r = LINSTR(array%(), search$ [,start] [,size]) | Gibt die Position einer Zeichenfolge in einer langen Zeichenfolge zurück |
| r = LLEN(array%()) | Gibt die Länge einer langen Zeichenkette zurück |
| r = LINPUT(array%(), fnbr, nbr) | Liest nbr Bytes aus einer Datei und gibt die gelesene Anzahl zurück |
| MATH(BASE64 ENCODE/DECODE) | Kodiert oder dekodiert Daten unter Verwendung der Basis 64 |

# MMBasic-Eigenschaften

## Namenskonventionen

Bei Befehlsnamen, Funktionsnamen, Labels, Variablennamen usw. wird die Groß-/Kleinschreibung nicht berücksichtigt, sodass „Run“ und „RUN“ gleichbedeutend sind und „dOO“ und „Doo“ sich auf dieselbe Variable beziehen.

Der Typ einer Variablen kann im DIM-Befehl oder durch Hinzufügen eines Suffixes am Ende des Variablennamens angegeben werden. Das Suffix für eine Ganzzahl lautet beispielsweise „%“; wenn also eine Variable namens nbr% automatisch erstellt wird, handelt es sich um eine Ganzzahl. Es gibt drei Arten von Variablen:

1. Gleitkomma. Diese können Zahlen mit Dezimalpunkt und Bruchteil (z. B. 45,386) sowie sehr große Zahlen speichern. Allerdings verlieren sie an Genauigkeit, wenn mehr als 14 signifikante Stellen gespeichert oder bearbeitet werden. Das Suffix lautet „!“ und Gleitkomma ist die Standardeinstellung, 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 Bruchteile (d. h. den Teil nach dem Dezimalpunkt) speichern. Das Suffix für eine Ganzzahl ist „%“
3. Zeichenketten. Diese speichern eine Zeichenfolge (z. B. „Tom“). Das Suffix für eine Zeichenkette ist das Symbol „$“ (z. B. name$, s$ usw.). Zeichenketten können bis zu 255 Zeichen lang sein.

Variablennamen und Labels können mit einem Buchstaben oder einem Unterstrich beginnen und beliebige Buchstaben, Ziffern, den Punkt (.) sowie den Unterstrich (_) enthalten. Sie dürfen bis zu 31 Zeichen lang sein. Ein Variablenname oder eine Beschriftung darf nicht mit einem Befehl, einer Funktion oder einem der folgenden Schlüsselwörter übereinstimmen: THEN, ELSE, TO, STEP, FOR, WHILE, UNTIL, MOD, NOT, AND, OR, XOR, AS. Z. B. ist „step = 5“ unzulässig.

Informationen zu Dateinamen findest du im Abschnitt *„MMBasic-Unterstützung für Flash- und SD-Karten-Dateisysteme*“.

## Konstanten

Numerische Konstanten können mit einer Ziffer (0–9) für eine dezimale Konstante, mit &H für eine hexadezimale Konstante, mit &O für eine oktale Konstante oder mit &B für eine binäre Konstante beginnen. Zum Beispiel entspricht &B1000 der dezimalen Konstante 8. Konstanten, die mit &H, &O oder &B beginnen, werden immer als 64-Bit-Ganzzahlkonstanten behandelt.

Dezimalen können ein Minus (-) oder ein Plus (+) vorangestellt werden und mit einem „E“ gefolgt von einer Exponentenzahl enden, um die Exponentialschreibweise zu kennzeichnen. Zum Beispiel entspricht 1.6E+4 dem Wert 16000. Wenn die Dezimalkonstante einen Dezimalpunkt oder einen Exponenten enthält, wird sie als Gleitkommakonstante behandelt; andernfalls wird sie als 64-Bit-Ganzzahlkonstante behandelt.

Zeichenfolgenkonstanten werden in doppelte Anführungszeichen (") gesetzt. Z. B. "Hello World".

## Implementierungsmerkmale

Maximale Programmgröße – siehe Tabelle. Beachte, dass MMBasic das Programm beim Speichern im Flash-Speicher tokenisiert, sodass die endgültige Größe im Flash-Speicher von der Größe des Klartextes abweichen kann.

Die maximale Länge einer Befehlszeile beträgt 255 Zeichen.

Die maximale Länge eines Variablennamens oder einer Beschriftung beträgt 31 Zeichen.

Maximale Anzahl von Dimensionen – siehe Tabelle.

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 beträgt 50.

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

Maximale Anzahl benutzerdefinierter Labels, Unterprogramme und Funktionen (insgesamt) – siehe Tabelle.

Maximale Anzahl konfigurierbarer Interrupt-Pins: 10

Zahlen werden als Gleitkommazahlen mit doppelter Genauigkeit oder als vorzeichenbehaftete 64-Bit-Ganzzahlen gespeichert und verarbeitet. Der Bereich der Gleitkommazahlen reicht von 1,797693134862316e+308 bis 2,225073858507201e-308.

Der Bereich der 64-Bit-Ganzzahlen (ganze Zahlen), die bearbeitet werden können, beträgt ± 9223372036854775807.

Die maximale Zeichenfolgenlänge beträgt 255 Zeichen.
Die maximale Zeilenzahl beträgt 65000.
Die maximale Anzahl von Hintergrundimpulsen, die mit dem Befehl PULSE ausgelöst werden können, beträgt 5.
Maximale Anzahl globaler Variablen und Konstanten – siehe Tabelle.
Maximale Anzahl lokaler Variablen – siehe Tabelle
Die maximale Anzahl an Dateien, die mit dem Befehl FILES aufgelistet werden können, beträgt 1000.
Die maximale Länge eines Dateinamens/Pfads beträgt 63 Zeichen (RP2040) oder 127 Zeichen (RP2350).
Eigenschaften vs. Firmware-Funktionen:

| | Maximum Program size | Maximum frequency | Maximum numbers of subroutines or functions | Maximum Array Dimensions | Default number of global variables | Default number of local variables* |
|---|---|---|---|---|---|---|
| PicoMite RP2040 | 128Kb | 420MHz | 256 | 6 | 256 | 240 |
| PicoMiteUSB RP2040 | 120Kb | 420MHz | 256 | 6 | 256 | 240 |
| PicoMiteVGA RP2040 | 108Kb | 378MHz | 256 | 6 | 240 | 240 |
| PicoMiteVGAUSB RP2040 | 100Kb | 378MHz | 256 | 6 | 240 | 240 |
| WebMite RP2040 | 88Kb | 252MHz | 256 | 6 | 240 | 240 |
| PicoMite RP2350 | 328Kb | 420MHz | 512 | 5 | 512 | 240 |
| PicoMiteUSB RP2350 | 328Kb | 420MHz | 512 | 5 | 512 | 240 |
| PicoMiteVGA RP2350 | 192Kb | 378MHz | 512 | 5 | 480 | 256 |
| PicoMiteVGAUSB RP2350 | 192Kb | 378MHz | 512 | 5 | 480 | 256 |
| PicoMiteHDMI RP2350 | 184Kb | 378MHz | 512 | 5 | 480 | 256 |
| PicoMiteHDMIUSB RP2350 | 184Kb | 378MHz | 512 | 5 | 480 | 256 |
| WebMite RP2350 | 208Kb | 252MHz | 512 | 5 | 512 | 256 |

* Das Verhältnis zwischen globalen und lokalen Variablen kann geändert werden.
Siehe OPTION LOCAL VARIABLES und MM.INFO(MAX VARS)

## Kompatibilität

MMBasic implementiert einen großen Teil von Microsofts GW-BASIC. Aufgrund physikalischer und praktischer Erwägungen gibt es zahlreiche Unterschiede, aber die meisten Standard-BASIC-Befehle und -Funktionen sind im Wesentlichen identisch. Ein Online-Handbuch für GW-BASIC ist unter http://www.antonis.de/qbebooks/gwbasman/index.html verfügbar und bietet eine detailliertere Beschreibung der Befehle und Funktionen.

MMBasic implementiert außerdem 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-Anweisung, die SELECT…CASE-Anweisungen und strukturierte IF . THEN … ELSE … ENDIF-Anweisungen.

# Vordefinierte schreibgeschützte Variablen

Diese Variablen werden von MMBasic gesetzt und können vom laufenden Programm nicht geändert werden. Beachte, dass sie von sich aus nichts bewirken; sie müssen ausgegeben oder einer Variablen zugewiesen werden.

Zum Beispiel:

```
> PRINT MM.VER
6.0002
>
```

oder in einem Programm:

```
If MM.VER < 6.0002 Then Error "Benötigt Version 6.00.02 oder höher"
```

| | |
|---|---|
| MM.VER | Gibt die Versionsnummer der Firmware als Gleitkommazahl im Format aa.bbcc zurück, wobei aa die Hauptversionsnummer, bb die Nebenversionsnummer und cc die Revisionsnummer ist. Beispielsweise würde die Version 6.03.00 den Wert 6.03 zurückgeben, während die Version 6.03.01 den Wert 6.0301 zurückgibt. |
| MM.ADDRESS$ | NUR WEBMITE-VERSION<br>Diese Variable gibt die IP-Adresse des Absenders des zuletzt empfangenen UDP-Datagramms zurück |
| MM.CMDLINE$ | Diese Funktion gibt alle Befehlszeilenargumente zurück, die an das aktuelle Programm übergeben wurden, wenn ein MMBasic-Programm läuft.<br>Siehe RUN und *-Befehle für Details.<br>Die Funktion gibt eine leere Zeichenkette zurück, wenn Programme über den Editor oder mit OPTION AUTORUN ausgeführt werden. |
| MM.DEVICE$ | Eine Zeichenkette, die das Gerät oder die Plattform angibt, auf der MMBasic läuft. |
| MM.DISPLAY | Gibt 1 zurück, wenn ein physischer Bildschirm konfiguriert ist, andernfalls 0 |
| MM.ERRNO<br>MM.ERRMSG$ | Wenn eine Anweisung einen Fehler verursacht hat, der ignoriert wurde, werden diese Variablen entsprechend gesetzt. MM.ERRNO ist eine Zahl, wobei ein Wert ungleich Null bedeutet, dass ein Fehler aufgetreten ist, und MM.ERRMSG$ ist eine Zeichenkette, 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 eine leere Zeichenkette zurückgesetzt. |
| MM.FLAGS | Gibt den Wert des Systemflag-Registers zurück |
| MM.FONTHEIGHT<br>MM.FONTWIDTH | Gibt die Höhe oder Breite der aktuellen Schriftart in Pixeln zurück |
| MM.HRES<br>MM.VRES | Ganzzahlen, die die aktuelle horizontale und vertikale Auflösung des VGA-/HDMI-Videoausgangs oder des LCD-Displays (sofern konfiguriert) in Pixeln angeben. |
| MM.HEIGHT<br>MM.WIDTH | Gibt die Anzahl der Zeichen in der Breite des physischen Displays mit der aktuellen Schriftart oder die Anzahl der Zeichen in der Höhe des Displays mit der aktuellen Schriftart zurück |

| | |
|---|---|
| MM.HPOS<br>MM.VPOS | Gibt die aktuelle horizontale und vertikale Position (in Pixeln) nach dem letzten Grafik- oder Druckbefehl zurück. |
| MM.SUPPLY | Bei Modulen mit einem Pin zur Messung der eingehenden Versorgungsspannung wie dem Pico und Pico2 gibt MM.SUPPLY die Spannung zurück. |
| MM.INFO()<br>MM.INFO$() | Diese beiden Versionen können austauschbar verwendet werden, aber gute Programmierpraxis verlangt, dass du diejenige verwendest, die dem zurückgegebenen Datentyp entspricht. |
| MM.INFO$(AUTORUN) | Gibt die Einstellung des Befehls OPTION AUTORUN zurück |
| MM.INFO(ADC) | Gibt die Nummer des Puffers zurück, der bei Verwendung von ADC RUN gerade zum Lesen bereit ist (1 oder 2). Gibt 0 zurück, wenn nichts bereit ist. |
| MM.INFO(ADC DMA) | Gibt „true“ (1) zurück, wenn der ADC-DMA aktiv ist. |
| MM.INFO(BATTERY) | Nur PicoCalc: Gibt den Batteriestand zurück |
| MM.INFO(BIOS) | Nur PicoCalc: Gibt die BIOS-Version zurück |
| MM.INFO(BOOT) | Zeigt dir den Grund für den letzten Neustart des Pico an<br>Gibt zurück:<br>Restart - *das Gerät wurde mit CPU RESTART oder einem OPTION-Befehl neu gestartet*<br>S/W Watchdog - *das Gerät wurde durch einen Software-Watchdog-Timeout neu gestartet*<br>H/W Watchdog - *das Gerät wurde durch einen Hardware-Watchdog-Timeout neu gestartet*<br>Firmware update - *das Gerät wurde nach einem Firmware-Update neu gestartet*<br>Power On - *das Gerät wurde eingeschaltet*<br>Reset Switch - *das Gerät wurde über den Reset-Knopf neu gestartet*<br>Unknown code &Hn - *unbekannter Grund – bitte gib den Code und die Version RP2040/2350 an* |
| MM.INFO (BOOT COUNT) | Gibt die Anzahl der Neustarts des Pico seit der letzten Formatierung des Flash-Laufwerks zurück. |
| MM.INFO(CHARGING) | Nur PicoCalc: Gibt den Ladezustand zurück |
| MM.INFO$(CALLTABLE) | Gibt die Basisadresse der MMBasic-Aufruftabelle zurück, die Zeiger auf jede MMBasic-Funktion enthält. |
| MM.INFO$(CPUSPEED) | Gibt die CPU-Geschwindigkeit als Zeichenkette zurück. |
| MM.INFO$(LCDPANEL) | Gibt den Namen des konfigurierten LCD-Bildschirms oder eine leere Zeichenkette zurück. Bei VGA- oder HDMI-Versionen gibt es den aktuellen Modus zurück. |
| MM.INFO(LCD320) | Gibt „true“ zurück, wenn das Display mit dem Befehl OPTION LCD320 im 320x240-Modus betrieben werden kann |
| MM.INFO$(SDCARD) | Gibt den Status der SD-Karte zurück. Gültige Rückgabewerte sind: DISABLED, NOT PRESENT, READY und UNUSED |
| MM.INFO$(CURRENT) | Gibt den Namen des aktuellen Programms zurück, wenn es aus einer Datei geladen wurde, oder NONE, wenn der Befehl nach einem NEW-, AUTOSAVE-, XMODEM- oder EDIT-Befehl aufgerufen wird. |

| | |
|---|---|
| MM.INFO$(PATH) | Gibt den Pfad des aktuellen Programms zurück oder NONE, wenn nach einem NEW- oder EDIT-Befehl aufgerufen. |
| MM.INFO(DISK SIZE) | Gibt die Kapazität des Flash-Dateisystems oder der SD-Karte in Byte zurück, je nachdem, welches Laufwerk gerade aktiv ist |
| MM.INFO$( DRIVE) | Gibt das aktive Laufwerk „A:“ oder „B:“ zurück |
| MM.INFO(EXISTS FILE fname$) | Gibt 1 zurück, wenn die angegebene Datei existiert, gibt -1 zurück, wenn fname$ ein Verzeichnis ist, andernfalls gibt es 0 zurück. |
| MM.INFO(EXISTS DIR dirname$) | Gibt einen booleschen Wert zurück, der angibt, ob das angegebene Verzeichnis existiert. |
| MM.INFO(FREE SPACE) | Gibt den freien Speicherplatz auf dem Flash-Dateisystem oder der SD-Karte zurück, je nachdem, welches Laufwerk aktiv ist. |
| MM.INFO(FILESIZE file$) | Gibt die Größe von file$ in Byte zurück oder 0, wenn die Datei nicht gefunden wird. |
| MM.INFO$(MODIFIED file$) | Gibt das Datum und die Uhrzeit zurück, zu der file$ geändert wurde; eine leere Zeichenkette, wenn die Datei nicht gefunden wird. |
| MM.INFO$(SYSTEM I2C) | Gibt je nach Status von OPTION SYSTEM I2C „I2C“, „I2C2“ oder „Not set“ zurück |
| MM.INFO(FCOLOUR) | Gibt die aktuelle Vordergrundfarbe zurück. |
| MM.INFO(BCOLOUR) | Gibt die aktuelle Hintergrundfarbe zurück. |
| MM.INFO(FONT) | Gibt die Nummer der aktuell aktiven Schriftart zurück. |
| MM.INFO(FONT ADDRESS n) | Gibt die Adresse des Speicherplatzes mit der Adresse von FONT n zurück. |
| MM.INFO(FONT POINTER n) | Gibt einen Zeiger auf den Anfang von FONT n im Speicher zurück. |
| MM.INFO(FONTHEIGHT)<br>MM.INFO(FONTWIDTH) | Ganzzahlen, die die Höhe und Breite der aktuellen Schriftart (in Pixeln) angeben. |
| MM.INFO(FLASH) | Gibt an, aus welchem Flash-Slot das Programm geladen wurde, falls zutreffend. |
| MM.INFO(FLASH-ADRESSE n) | Gibt die Adresse des Flash-Steckplatzes n zurück. |
| MM.INFO(HEAP) | Gibt die Menge an freiem MMbasic-Heap-Speicher zurück. Der MMbasic-Heap wird für Strings, Arrays und verschiedene andere temporäre und permanente Puffer (z. B. Audio) verwendet |
| MM.INFO(HPOS)<br>MM.INFO(VPOS) | Die aktuelle horizontale und vertikale Position (in Pixeln) nach dem letzten Grafik- oder Druckbefehl. |
| MM.INFO(ID) | Gibt die eindeutige ID des Pico zurück. |
| MM.INFO$(IP-ADRESSE) | Gibt die IP-Adresse des WebMite zurück |
| MM.INFO(MAX GP) | Gibt die höchste gültige GPno auf dem Chip zurück |

| | |
|---|---|
| MM.INFO(MAX VARS) | Gibt die Gesamtzahl der in der aktuellen Version verfügbaren Variablen zurück |
| MM.INFO$(MODBUFF-ADDRESS) | Gibt die Speicheradresse des Puffers zurück, der zum Speichern von MOD-Dateien verwendet wird |
| MM.INFO$(OPTION option) | Gibt den aktuellen Wert einer Reihe von Optionen zurück, die beeinflussen, wie ein Programm<br>ausgeführt wird. „option“ kann einer der folgenden Werte sein: AUTORUN, AUDIO, BASE, BREAK,<br>CONSOLE, DEFAULT, EXPLICIT, KEYBOARD, ANGLE, HEIGHT, WIDTH, FLASH SIZE |
| MM.INFO$(PIN pinno) | Gibt den Status des E/A-Pins „pinno“ zurück. Gültige Rückgabewerte sind: OFF, DIN, DOUT, AIN usw. |
| MM.INFO(PINNO GPnn) | Gibt die physikalische Pin-Nummer für eine bestimmte GP-Nummer zurück. GPnn kann eine Zeichenkette ohne Anführungszeichen (GP01), ein Zeichenkettenliteral („GP01“) oder eine Zeichenkettenvariable sein. D. h., A$ = „GP01“: MM.INFO(PINNO A$) . |
| MM.INFO(PIO RX DMA) | Zeigt an, ob der PIO-RX-DMA-Kanal belegt ist |
| MM.INFO(PIO TX DMA) | Zeigt an, ob der PIO-TX-DMA-Kanal belegt ist |
| MM.INFO$(PLATFORM) | Gibt die zuvor mit OPTION PLATFORM festgelegte Zeichenkette zurück. |
| MM.INFO(PROGRAM) | Gibt die Speicheradresse des aktuell laufenden Programms zurück |
| MM.INFO(PSRAM SIZE) | Gibt die Größe des PSRAM zurück, falls verfügbar. Der RP2040 gibt immer 0 zurück. |
| MM.INFO(PS2) | Gibt die letzte auf der PS2-Schnittstelle empfangene Rohnachricht zurück, sofern diese aktiviert ist. |
| MM.INFO(PWM COUNT) | Gibt die Anzahl der vom Chip unterstützten PWM-Kanäle zurück |
| MM.INFO(PWM DUTY C%, n%) | Gibt den aktuellen Tastgrad in Taktzyklen des PWM-Kanals C%,N% zurück Wobei N%=0 für A und 1 für B |
| MM.INFO(PWM SLICE pinno) | Dies gibt den PWM-Slice für einen bestimmten Pin zurück (0–7 RP2040, 0–11 RP2350) |
| MM.INFO$(SOUND) | Gibt den aktuellen Status des Audioausgangs zurück (OFF, PAUSED, TONE, WAV, FLAC, MP3, SOUND) |
| MM.INFO(SPI SPEED) | Gibt die tatsächliche Geschwindigkeit des SYSTEM-SPI zurück oder einen Fehler, falls nicht gesetzt |
| MM.INFO(STACK) | Gibt den C-Stack-Zeiger zurück. Komplexer oder rekursiver Basic-Code kann den Fehler „Stack overflow, expression too complex at depth %“ verursachen. Dies tritt auf, wenn der Stack unter &H 2003f800 liegt. Durch Überwachen des Stacks kann der Programmierer Möglichkeiten zur Vereinfachung des Basic-Codes erkennen, um den Fehler zu vermeiden. |
| MM.INFO$(SYSTEM I2C) | Gibt je nach Fall I2C, I2C2 oder NOT SET zurück. |
| MM.INFO(SYSTEM HEAP) | Gibt den freien Speicherplatz auf dem System-Heap zurück. |

| | |
|---|---|
| MM.INFO(SYSTICK) | Gibt den aktuellen Wert des 24-Bit-Systick-Timers des Systems zurück, der mit der CPU-Taktfrequenz läuft |
| MM.INFO(TILE HEIGHT) | NUR VGA- UND HDMI-VERSIONEN<br>Gibt die aktuelle Einstellung der Kachelhöhe zurück. |
| MM.INFO(TRACK) | Gibt den Namen der FLAC-, MP3-, WAV- oder MIDI-Datei zurück, die gerade über den Audioausgang abgespielt wird. |
| MM.INFO$(TOUCH) | Gibt den Status des Touch-Controllers zurück. Gültige Rückgabewerte sind: „Deaktiviert“, „Nicht kalibriert“ und „Bereit“. |
| MM.INFO(USB n) | Gibt den Gerätecode für jedes an Kanal „n“ angeschlossene Gerät zurück, wobei „n“ eine Zahl zwischen 1 und 4 ist. Der zurückgegebene Gerätecode kann lauten:<br>0 = nicht in Gebrauch, 1 = Tastatur, 2 = Maus, 128 = PS4, 129 = PS3, 130 = SNES/Generisch<br>Standardmäßig wird eine angeschlossene Tastatur dem Kanal 1 zugewiesen, eine Maus dem Kanal 2 und Gamepads dem Kanal 3 und dann dem Kanal 4. Wenn 2 oder mehr Tastaturen oder Mäuse oder 3 oder mehr Gamepads angeschlossen sind, werden die zusätzlichen Geräte dem höchsten verfügbaren Kanal zugewiesen. |
| MM.INFO(USB VID n) | Gibt die VID des USB-Geräts auf Kanal n zurück |
| MM.INFO(USB-PID n) | Gibt die PID des USB-Geräts auf Kanal n zurück |
| MM.INFO(VARCNT) | Gibt die Anzahl der im MMBasic-Programm verwendeten Variablen zurück. |
| MM.INFO$(LINE) | Gibt die aktuelle Zeilennummer als Zeichenkette zurück. LIBRARY wird zurückgegeben, wenn du dich in der Bibliothek befindest, und UNKNOWN, wenn du dich nicht in einem Programm befindest. Hilft bei der Diagnose während der Unit-Tests. |
| MM.INFO(UPTIME) | Gibt die Zeit in Sekunden seit dem Booten als Fließkommazahl zurück. |
| MM.INFO(VALID CPUSPEED speed%) | Gibt 1 zurück, wenn „speed%“ für OPTION CPUSPEED „speed%“ gültig ist |
| MM.INFO(VERSION) | Die Versionsnummer der Firmware (MM.VER konvertiert zu dieser) |
| MM.INFO(WRITEBUFF) | Gibt die Speicheradresse des aktuellen Puffers zurück, der für Zeichenbefehle verwendet wird. |
| MM.INFO(TCP PORT) | NUR WEBMITE<br>Gibt den als Server eingestellten TCP-Port zurück oder 0, falls nicht eingestellt |
| MM.INFO(UDP PORT) | Gibt den als Server festgelegten UDP-Port zurück oder 0, falls nicht festgelegt |

| | |
|---|---|
| MM.INFO(TCPIP-STATUS)<br>MM.INFO(WIFI-STATUS) | NUR WEBMITE<br>Gibt den TCPIP-Status der Verbindung zurück<br>Gibt den WLAN-Status der Verbindung zurück.<br>Gültige Rückgabewerte sind:<br>0 WLAN ist ausgefallen<br>1 Mit WLAN verbunden<br>2 Mit WLAN verbunden, aber keine IP-Adresse (nur TCPIP-STATUS)<br>3 Mit WLAN verbunden und mit IP-Adresse (nur TCPIP STATUS)<br>-1 Verbindung fehlgeschlagen<br>-2 Keine passende SSID gefunden (möglicherweise außerhalb der Reichweite oder ausgefallen)<br>-3 Authentifizierung fehlgeschlagen |
| MM.MESSAGE$ | NUR WEBMITE<br>Gibt den Inhalt des zuletzt empfangenen UDP-Datagramms oder des zuletzt empfangenen MQTT-Pakets zurück |
| MM.TOPIC$ | NUR WEBMITE<br>Gibt das Thema des zuletzt empfangenen MQTT-Pakets zurück |
| MM.ADDRESS$ | NUR WEBMITE<br>Gibt die Adresse des Absenders des zuletzt empfangenen UDP-Datagramms oder des zuletzt empfangenen MQTT-Pakets zurück |
| MM.ONEWIRE | Nach einer 1-Wire-Reset-Funktion wird diese Integer-Variable gesetzt, um das Ergebnis des Vorgangs anzuzeigen: 0 = Gerät nicht gefunden, 1 = Gerät gefunden, 2 = Zeitüberschreitung des Geräts. |
| MM.I2C | Nach einem I2C-Schreib- oder Lesebefehl wird diese Integer-Variable gesetzt, um das Ergebnis des Vorgangs wie folgt anzuzeigen:<br>0 = Der Befehl wurde fehlerfrei ausgeführt.<br>1 = NACK-Antwort empfangen<br>2 = Zeitüberschreitung beim Befehl |
| MM.PERSISTENT | Gibt einen Wert zurück, der mit dem Befehl SAVE PERSISTENT gespeichert wurde |
| MM.PS2 | Gibt den zuletzt auf der PS2-Schnittstelle empfangenen Code zurück, sofern diese aktiviert ist. |
| MM.WATCHDOG | Eine Ganzzahl, die wahr ist, wenn MMBasic infolge eines Watchdog-Timeouts neu gestartet wurde (siehe den Befehl WATCHDOG), andernfalls falsch. |

# Optionen

Diese Tabelle listet die verschiedenen Optionsbefehle auf, mit denen du MMBasic konfigurieren und dessen Funktionsweise ändern kannst. Optionen, die als dauerhaft markiert sind, werden im nichtflüchtigen Speicher gespeichert und beim Neustart der PicoMite-Firmware automatisch wiederhergestellt. Optionen, die nicht dauerhaft sind, werden beim Start, beim Zurücksetzen und in vielen Fällen beim Ausführen und/oder Beenden eines Programms zurückgesetzt.

Viele OPTION-Befehle erzwingen einen Neustart der PicoMite-Firmware, was dazu führt, dass die USB-Konsolen-Schnittstelle zurückgesetzt wird. Das Programm im Speicher geht nicht verloren, da es in einem nichtflüchtigen Flash-Speicher gespeichert ist.

| | **Permanent?** | |
|---|---|---|
| OPTION ANGLE RADIANS \| DEGREES | | Dieser Befehl schaltet die Trigonometriefunktionen zwischen Grad und Radiant um. Wirkt auf SIN, COS, TAN, ATN, ATAN2, MATH ATAN3, ACOS, ASIN |
| OPTION AUDIO PWMnApin, PWMnBpin<br>oder<br>OPTION AUDIO DISABLE | ✓ | Konfiguriert einen der PWM-Kanäle als Audioausgang.<br>„PWMnApin“ ist der linke Audiokanal, „PWMnBpin“ der rechte. Beide Pins müssen zum selben Audiokanal gehören.<br>Beispiel: OPTION AUDIO GP18, GP19 würde PWM1A und PWM1B auf den Pins 24 bzw. 25 verwenden.<br>Diese Option verhindert die Verwendung dieser Pins im BASIC-Programm.<br>Der Audioausgang wird mittels PWM erzeugt, daher ist ein Tiefpassfilter am Ausgang erforderlich. Der Audioausgang des Raspberry Pi Pico ist sehr verrauscht. Die Verwendung von OPTION POWER und/oder die Stromversorgung über einen separaten 3,3-V-Linearregler kann dies reduzieren.<br>Dieser Befehl muss an der Eingabeaufforderung ausgeführt werden (nicht in einem Programm). |
| OPTION AUDIO SPI CSpin, CLKpin, MOSIpin<br>oder<br>OPTION AUDIO DISABLE | ✓ | Konfiguriert den Audioausgang so, dass er an einen MCP48n2-DAC weitergeleitet wird, der an die angegebenen Pins angeschlossen ist. Der LDAC-Pin am DAC sollte mit GND verbunden werden. |
| OPTION AUDIO VS1053 CLK-Pin, MOSI-Pin, MISO-Pin, XCS-Pin, XDC-Pin, DREQ-Pin, XRST-Pin<br>oder<br>OPTION AUDIO DISABLE | ✓ | Konfiguriert den Audioausgang so, dass er an einen VS1053-CODEC geleitet wird. Dies ermöglicht neben den anderen unterstützten Formaten auch die Wiedergabe von MP3 und MIDI sowie die Echtzeit-MIDI-Ausgabe. Weitere Details findest du unter dem Befehl PLAY |
| OPTION AUDIO I2S BCLK-Pin, DIN-Pin<br>oder<br>OPTION AUDIO DISABLE | ✓ | Konfiguriert die Audioausgabe so, dass sie an einen I2S-DAC geleitet wird, der an die angegebenen Pins angeschlossen ist. Der LRCK-Pin am DAC sollte mit dem nächsten aufeinanderfolgenden GPIO-Pin an BCLKpin verbunden werden. |
| OPTION AUTOREFRESH OFF \| ON | | Schwarz-Weiß-Displays können jeweils nur einen ganzen Bildschirm auf einmal aktualisieren. Mit OPTION AUTOREFRESH OFF/ON kannst du steuern, ob ein Schreibbefehl das Display sofort aktualisiert oder nicht. Wenn AUTOREFRESH auf OFF steht, kann der REFRESH- |

| | | |
|---|---|---|
| | | Befehl verwendet werden, um das Schreiben auszulösen. Dies gilt für folgende Displays: N5110, SSD1306I2C, SSD1306I2C32, SSD1306SPI und ST7920 |
| OPTION AUTORUN ON [,NORESET]<br>oder<br>OPTION AUTORUN n [,NORESET]<br>oder<br>OPTION AUTORUN OFF | ✓ | Weist MMBasic an, ein Programm beim Hochfahren oder Neustart automatisch auszuführen.<br>ON bewirkt, dass das aktuelle Programm im Programmspeicher ausgeführt wird.<br>Wenn du „n“ angibst, wird der entsprechende Speicherplatz im Flash-Speicher ausgeführt. „n“ muss im Bereich von 1 bis 3 liegen.<br>Wenn du den optionalen Parameter „NORESET“ angibst, bleibt AUTORUN auch dann erhalten, wenn das Programm einen Systemfehler verursacht (standardmäßig führt dies dazu, dass die Firmware jede OPTION AUTORUN-Einstellung aufhebt).<br>OFF deaktiviert die Autorun-Option und ist die Standardeinstellung für ein neues Programm.<br>Durch Drücken der Unterbrechungstaste (standardmäßig STRG-C) an der Konsole wird das laufende Programm unterbrochen und du kehrst zur Eingabeaufforderung zurück. |
| OPTION BASE 0 \| 1 | | Lege den niedrigsten Wert für Array-Indizes auf entweder 0 oder 1 fest.<br>Dies muss vor der Deklaration von Arrays erfolgen und wird beim Einschalten auf den Standardwert 0 zurückgesetzt. |
| OPTION BACKLIGHT KEYBOARD n<br>OPTION LCD-HINTERGRUNDBELEUCHTUNG n | ✓ | <u>Nur PicoCalc</u><br>Stellt die Helligkeit der Tastaturbeleuchtung auf 0–100 % ein<br><br>Stellt die Helligkeit der Display-Hintergrundbeleuchtung auf 0–100 % ein |
| OPTION BAUDRATE nn | | Stellt die Baudrate der seriellen Konsole ein (sofern konfiguriert). |
| OPTION BREAK nn | | Stellt den Wert der Break-Taste auf den ASCII-Wert 'nn' ein. Diese Taste dient dazu, ein laufendes Programm zu unterbrechen.<br>Der Wert der Break-Taste ist beim Hochfahren und beim Ausführen eines Programms auf die STRG-C-Taste gesetzt, kann aber mit diesem Befehl in einem Programm auf eine beliebige Tastaturtaste geändert werden (zum Beispiel setzt OPTION BREAK 4 die Break-Taste auf die STRG-D-Taste). Wenn du diese Option auf Null setzt, wird die Break-Funktion komplett deaktiviert. |
| OPTION CASE LOWER \| UPPER \| TITLE | ✓ | Ändere die Groß-/Kleinschreibung, die bei der Verwendung des LIST-Befehls für die Auflistung von Befehls- und Funktionsnamen verwendet wird. Die Standardeinstellung ist TITLE, aber der alte Standard von MMBasic kann mit OPTION CASE UPPER wiederhergestellt werden. |
| OPTION COLOURCODE ON<br>oder<br>OPTION COLOURCODE OFF | ✓ | Schalte die Farbcodierung für die Ausgabe des Editors ein oder aus. Schlüsselwörter werden in Cyan, Kommentare in Gelb usw. angezeigt. Die Standardeinstellung ist OFF.<br>Das Schlüsselwort COLORCODE (US-Schreibweise) kann ebenfalls verwendet werden.<br>Dies funktioniert bei VGA/HDMI-Video und der seriellen Konsole unter Verwendung eines Terminalemulators mit VT100-Emulation (z. B. Tera |

| | | |
|---|---|---|
| | | Term). Dieser Befehl muss an der Eingabeaufforderung ausgeführt werden (nicht in einem Programm). |
| OPTION CONSOLE output | | Legt fest, wo Ausgabeanweisungen angezeigt werden. Gültige Einstellungen sind BOTH (d. h. SCREEN und SERIAL), SERIAL, SCREEN, NONE. Dies ist eine temporäre Option, die beim Beenden eines Programms zurückgesetzt wird. |
| OPTION CONTINUATION LINES ON/OFF | ✓ | Aktiviert oder deaktiviert die Verwendung von Zeilenfortsetzungen im Editor und beim Befehl LIST. Eine Zeilenfortsetzung wird durch ein Leerzeichen gefolgt von einem Unterstrich am Zeilenende angezeigt. Wenn diese Option aktiviert ist, teilt der Editor Zeilen beim Einlesen aus der Datei automatisch auf und fügt die erforderlichen Fortsetzungszeichen hinzu. Beim Verlassen des Editors werden die Fortsetzungszeichen vor dem Speichern entfernt. Im Editor kannst du lange Zeilen erstellen, indem du deine eigenen Fortsetzungszeichen hinzufügst. Das erleichtert die Verwendung eines kleinen Bildschirms als Konsole erheblich. |
| OPTION CPUSPEED speed | ✓ | <u>NICHT BEI HDMI- ODER VGA-VERSIONEN</u><br>Ändere die CPU-Taktfrequenz.<br>„speed“ ist die CPU-Taktfrequenz in kHz im Bereich von 48000 bis 420000. Geschwindigkeiten über 200 MHz (150 MHz für den RP2350) gelten als Übertaktung, da dies die angegebene Höchstgeschwindigkeit des Standard-Raspberry Pi Pico ist.<br>Die Standardgeschwindigkeit beträgt 200000 für den RP2040 und 150000 für den RP2350.<br>Dieser Befehl muss an der Eingabeaufforderung ausgeführt werden (nicht in einem Programm). |
| OPTION COUNT pin1, pin2, pin3, pin4 | ✓ | Legt fest, welche Pins als Zählereingänge verwendet werden sollen. Standardmäßig sind dies GP6, GP7, GP8 und GP9. Der Befehl SETPIN definiert den Zählmodus.<br>Dieser Befehl muss an der Eingabeaufforderung ausgeführt werden (nicht in einem Programm). |
| OPTION DEFAULT FLOAT \| INTEGER \| STRING \| NONE | | Wird verwendet, um den Standardtyp für eine Variable festzulegen, die nicht explizit definiert ist.<br>Wenn OPTION DEFAULT NONE verwendet wird, muss der Typ aller Variablen explizit definiert werden, sonst tritt der Fehler „Variablentyp nicht angegeben“ auf.<br>Wenn ein Programm ausgeführt wird, ist die Standardeinstellung FLOAT, um die Kompatibilität mit Microsoft BASIC und früheren Versionen von MMBasic zu gewährleisten. |
| OPTION DEFAULT COLOURS foreground [,background] | ✓ | Legt die Standardfarben für Vordergrund und Hintergrund sowohl für den Monochrom- als auch für den Farbmodus fest. Die Farbe muss eine der folgenden sein: white, yellow, lilac, brown, fuchsia, rust, magenta, red, cyan, green, cerulean, midgreen, cobalt, myrtle, blue und black. Ein numerischer Wert kann nicht verwendet werden. Die Standardeinstellung ist white, black.<br>Wenn der Hintergrund weggelassen wird, ist die Standardeinstellung black. |
| OPTION DEFAULT MODE n | ✓ | Dies legt den Standard-Anzeigemodus beim Booten fest. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden. |

| | | |
|---|---|---|
| OPTION DISK SAVE fname$<br>OPTION DISK LOAD fname$ | ✓ | Mit diesen Befehlen kannst du den gesamten Satz definierter Optionen in einer Datei auf einer Diskette speichern und von dort wiederherstellen. Die Datei kann dann per XMODEM auf einen Host-Computer übertragen werden, sodass zusätzliche Geräte einfach konfiguriert oder Optionen nach einem Firmware-Upgrade wiederhergestellt werden können |
| OPTION DISPLAY lines [,chars] | ✓ | Lege die Eigenschaften des für die Konsole verwendeten Bildschirmterminals fest. Sowohl der Befehl LIST als auch der Befehl EDIT benötigen diese Informationen, um den Text korrekt für die Anzeige zu formatieren.<br>„lines“ ist die Anzahl der Zeilen auf dem Bildschirm und „chars“ ist die Breite des Bildschirms in Zeichen. Der Standardwert ist 24 Zeilen x 80 Zeichen, und wenn diese Option geändert wird, bleibt sie auch nach dem Ausschalten gespeichert. Die Maximalwerte sind 100 Zeilen und 240 Zeichen.<br>Dadurch wird eine ESC-Sequenz gesendet, um das VT100-Terminal auf die entsprechende Größe einzustellen. TerraTerm, Putty und MMCC reagieren auf diese Sequenz und stellen die Terminalbreite ein (sofern die Option in den Terminaleinstellungen aktiviert ist).<br>**Diese Option ist nicht verfügbar, wenn ein LCD-Display als Konsole verwendet wird.** |
| OPTION ESCAPE | | Aktiviert die Möglichkeit, Escape-Sequenzen in Zeichenfolgenkonstanten einzufügen. Siehe den Abschnitt *„Sonderzeichen in Zeichenfolgen*“. |
| OPTION EXPLICIT | | Wenn du diesen Befehl an den Anfang eines Programms setzt, muss jede Variable explizit mit den Befehlen DIM, LOCAL oder STATIC deklariert werden, bevor sie im Programm verwendet werden kann.<br>Diese Option ist standardmäßig deaktiviert, wenn ein Programm ausgeführt wird. Wenn sie verwendet wird, muss sie vor der Verwendung von Variablen angegeben werden. |
| OPTION FAST AUDIO ON\|OFF | | Bei Verwendung des Befehls PLAY SOUND können Änderungen an Sounds, Lautstärke oder Frequenzen zu hörbaren Klicks in der Ausgabe führen. Die Firmware versucht, dies zu mildern, indem sie die Lautstärke der vorherigen Ausgabe des Kanals vor der Änderung der Ausgabe langsam herunterfährt und anschließend wieder hochfährt. Dies verbessert die Audioausgabe erheblich, geht jedoch mit einer kurzen Verzögerung beim Befehl PLAY SOUND einher (im schlimmsten Fall 3 ms). Diese Verzögerung lässt sich vermeiden, indem du in einem Programm OPTION FAST AUDIO ON verwendest. Die hörbaren Klicks können dann wieder auftreten, aber das liegt im Ermessen des Programmierers.<br>Dies ist eine temporäre Option, die bei jedem Programmstart auf OFF zurückgesetzt wird. |
| OPTION FNKey string$ | ✓ | Definiere die Zeichenfolge, die generiert wird, wenn eine Funktionstaste an der Befehlszeile gedrückt wird. „FNKey“ kann F1 sowie F5 bis F9 sein. Wenn du string$ auf "" setzt, wird die gespeicherte Funktion gelöscht und die ursprüngliche Funktion der Taste wiederhergestellt.<br>Beispiel:<br>OPTION F8 „RUN “+chr$(34)+”myprog” +chr$(34)+chr$(13)+chr$(10).<br>Dieser Befehl muss in der Eingabeaufforderung ausgeführt werden (nicht in einem Programm). |

| | | |
|---|---|---|
| OPTION GPS | ✓ | Konfiguriert den PicoMite für die Anbindung an ein GPS-Modul. Diese Integration ermöglicht es dem PicoMite, NMEA-Datenströme automatisch im Hintergrund zu analysieren, wodurch Standort-, Zeit- und Geschwindigkeitsdaten über die Funktion `GPS()` für das Anwenderprogramm verfügbar gemacht werden.<br>Dies wird ausführlich in der Datei *Option_GPS_User_Manual.pdf* beschrieben, die im ZIP-Download der Firmware enthalten ist. |
| OPTION HEARTBEAT ON/OFF [HEARTBEATpin] | ✓ | Aktiviert oder deaktiviert die Ausgabe der Heartbeat-LED.<br>Beim Pico-W befindet sich der Heartbeat auf einem Pin, der vom CWY43-Chip gesteuert wird.<br>NICHT WEBSITE-VERSION<br>Standardmäßig ist der Heartbeat bei RP2350A-Chips auf GP25 aktiviert.<br>Wenn er deaktiviert ist, kann das Programm die LED über GP25 steuern.<br>Bei RP2350B-Chips ist der Heartbeat nicht aktiviert.<br>Wenn der Heartbeat deaktiviert ist, kann der Befehl sowohl zum Aktivieren als auch optional zum Festlegen des zu verwendenden Pins (Standard GP25) verwendet werden |
| OPTION HDMI-PINS clockpositivepin, d0positivepin, d1positivepin, d2positivepin | ✓ | NUR HDMI-VERSION<br>Lege die I/O-Pins fest, die für den HDMI-Videoausgang verwendet werden. Dies ist nur bei nicht standardmäßigen PCB-Layouts erforderlich.<br>Die positiven HDMI-Signalpins werden gemäß dem unten stehenden „nbr“ festgelegt. Gültige Werte sind 0–7, und die Pins dürfen sich für jeden Kanal nicht überschneiden. Ist „nbr“ eine gerade Zahl, liegt der negative Ausgang am physikalischen Pin +1, ist „nbr“ ungerade, liegt er am physikalischen Pin –1.<br>nbr — HSTX Nbr — Physikalischer Pin<br>0 — HSTX0 — GP12<br>1 — HSTX1 — GP13<br>2 — HSTX2 — GP14<br>3 — HSTX3 — GP15<br>4 — HSTX4 — GP16<br>5 — HSTX5 — GP17<br>6 — HSTX6 — GP18<br>7 — HSTX7 — GP19<br>Die Standardeinstellung lautet: OPTION HDMI PINS 2, 0, 6, 4<br>Das bedeutet:<br>CK+ und CK- sind GP14 und GP15 zugewiesen<br>D0+ und D0- sind GP12 und GP13 zugewiesen<br>D1+ und D1- sind GP18 und GP19 zugewiesen<br>D2+ und D2- sind GP16 und GP17 zugewiesen |
| OPTION KEYBOARD nn [,capslock] [,numlock] [,repeatstart] [,repeatrate]<br>oder<br>OPTION KEYBOARD DISABLE | ✓ | Konfiguriere eine Tastatur. Dies kann für die Konsoleneingabe verwendet werden, und alle eingegebenen Zeichen stehen über alle Befehle zur Verfügung, die von der Konsole lesen (seriell über USB).<br>„nn“ ist ein zweistelliger Code, der das Tastaturlayout definiert. Zur Auswahl stehen „US“ für das Standard-Tastaturlayout in den USA, Australien und Neuseeland, „UK“ für Großbritannien, „GR“ für Deutschland, „FR“ für Frankreich, „BR“ für Brasilien und „ES“ für Spanien. |

| | | |
|---|---|---|
| | | Dieser Befehl muss in der Befehlszeile eingegeben werden und führt zu einem Neustart. Diese Einstellung kann mit folgendem Befehl zurückgesetzt werden: OPTION KEYBOARD DISABLE<br>Die optionalen Parameter „capslock“ und „numlock“ sind boolesche Ganzzahlen, die den Anfangszustand der Tastatur festlegen (Standardwerte sind 0 und 1).<br>Die optionalen Parameter „repeatstart“ und „repeatrate“ legen die Zeit für die erste Wiederholung einer gedrückten Taste und die nachfolgenden Wiederholungen fest. Bei einer USB-Tastatur liegen sie zwischen 100 und 2000 ms bzw. zwischen 25 und 2000 ms. Bei einer PS2-Tastatur liegen sie zwischen 0 und 3, was 250 ms, 500 ms, 750 ms und 1000 ms entspricht (Standard ist 1), sowie zwischen 0 und 31, was 33 ms bis 500 ms gemäß der PS2-Tastaturspezifikation entspricht (Standard ist 12 oder 100 ms). |
| OPTION KEYBOARD I2C | ✓ | Konfiguriert die Unterstützung für die Solderparty bbq20 mini I2C-Tastatur.<br>Hinweis: OPTION SYSTEM I2C muss vor der Ausführung dieses Befehls gesetzt werden |
| OPTION KEYBOARD PINS clockpin, datapin | ✓ | Ermöglicht es dem Benutzer, die Pins auszuwählen, die für den Anschluss einer PS2-Tastatur verwendet werden sollen. Die Standardeinstellung ist Pin 11 (GP8) und Pin 12 (GP9).<br>Die PS2-Tastatur muss deaktiviert sein (OPTION KEYBOARD DISABLE) |
| OPTION KEYBOARD REPEAT repeatstart , repeatrate | ✓ | <u>NUR USB-TASTATUR</u><br>Legt die Zeit für die erste Wiederholung einer gedrückten Taste (100–2000) und die nachfolgenden Wiederholungen (25–2000) in Millisekunden fest. |
| OPTION LCD-PINS clkpin, mosipin, misopin | ✓ | Die Firmware unterstützt die Trennung der SPI-Pins, die zur Ansteuerung eines LCD-Displays verwendet werden, von denen, die für Touch und/oder die SD-Karte genutzt werden. Durch diese Trennung kann die Firmware den zweiten Prozessor für die Arbeit mit dem Display nutzen, ohne dass andere Funktionen beeinträchtigt werden. Die Einstellung der LCD-SPI-Pins ist eine Voraussetzung für die Verwendung gepufferter Treiber mit dem RP2350 PicoMite. Die angegebenen Pins müssen gültige SPI-Pins für die Funktionen clk, mosi (tx) und miso (rx) sein. Wenn auch SYSTEM-SPI-Pins angegeben werden, müssen diese auf einem anderen SPI-Kanal liegen. |
| OPTION LCDPANEL<br><br>OPTION LCDPANEL VIRTUAL_C<br>oder<br>OPTION LCDPANEL VIRTUAL_M | ✓ | <u>NICHT BEI VGA- ODER HDMI-VERSIONEN</u><br>Konfiguriert ein LCD-Panel bei Versionen, die einen angeschlossenen LCD-Bildschirm unterstützen.<br><br>Konfiguriert ein virtuelles LCD-Panel ohne physisch angeschlossenes Panel.<br>VIRTUAL _C = Farbe, 4 Bit, 320 x 240<br>VIRTUAL _M = Monochrom, 640 x 480<br>Mit dieser Funktion kann ein Programm grafische Bilder auf diesem virtuellen Panel zeichnen und sie dann als BMP-Datei speichern. Nützlich, um ein grafisches Bild für den Export ohne angeschlossenes Display zu erstellen |

| | | |
|---|---|---|
| OPTION LCDPANEL-Optionen<br>oder<br>OPTION LCDPANEL DISABLE | | Konfiguriert die PicoMite-Firmware für die Verwendung mit einem angeschlossenen LCD-Panel.<br>Siehe das Kapitel *„LCD-Displays“* für weitere Details.<br>Dieser Befehl muss an der Eingabeaufforderung ausgeführt werden (nicht in einem Programm). |
| OPTION LCDPANEL CONSOLE [font [, fc [, bc [, blight]]] [,NOSCROLL]<br>oder<br>OPTION LCDPANEL NOCONSOLE | | Konfiguriert das LCD-Display zur Verwendung als Konsolenausgabe. Das LCD muss transparenten Text unterstützen (z. B. die Controller SSD1963_x, ILI9341 oder ST7789_320).<br>„font“ ist die Standardschriftart, „fc“ ist die Standard-Vordergrundfarbe, „bc“ ist die Standard-Hintergrundfarbe. Diese Parameter sind optional und standardmäßig auf Schriftart 1, Weiß, Schwarz und 100 % eingestellt. Diese Einstellungen werden beim Einschalten angewendet.<br>Der optionale Befehl NOSCROLL ändert die Firmware so, dass bei der Ausgabe an die letzte Zeile des Displays das Display nicht scrollt, sondern gelöscht wird und die Ausgabe am oberen Rand des Displays fortgesetzt wird. Dadurch können Displays, die kein Lesen unterstützen, als Konsolengerät verwendet werden und:<br>Beachte, dass bei anderen Displays als dem SSD1963 der Bildlauf bei der Konsolenausgabe sehr langsam ist; daher wird empfohlen, für diese Displays die Option NOSCROLL zu verwenden. Diese Einstellung wird im Flash-Speicher gespeichert und beim Start automatisch angewendet. Um sie zu deaktivieren, verwende den Befehl OPTION LCDPANEL NOCONSOLE.<br>Dieser Befehl muss an der Eingabeaufforderung ausgeführt werden. |
| OPTION LCDPANEL USER hres, vres | | Konfiguriert einen vom Benutzer geschriebenen Bildschirmtreiber in MMBasic. Eine Beschreibung, wie du den Treiber schreibst, findest du in der Datei „User Display Driver.txt“ in der PicoMite-Firmware-Distribution. |
| OPTION LCDPANEL CONSOLE [font [, fc [,bc]]<br>oder<br>OPTION LCDPANEL NOCONSOLE | ✓ | <u>NUR VGA- UND HDMI-VERSIONEN</u><br>Ändert die Standardschriftart, die auf dem VGA- oder HDMI-Bildschirm verwendet wird.<br>„fc“ ist die Vordergrundfarbe und „bc“ die Hintergrundfarbe.<br>Deaktiviert die Konsolenausgabe auf dem VGA-/HDMI-Display. Diese Option ist dauerhaft; sowohl die Druckausgabe als auch die Konsolenausgabe werden deaktiviert, und nur Grafikbefehle werden auf dem VGA-Bildschirm ausgegeben. Wenn die Ausgabe in einem Programm vorübergehend deaktiviert werden soll, verwende den Befehl OPTION CONSOLE<br>Bei SSD1963-basierten Displays im Querformat und SPI-Displays im Hochformat nutzt die Firmware H/W-Scrolling, um die Leistung der Display-Konsole zu verbessern. |
| OPTION LCD320 ON/OFF | | <u>NICHT BEI VGA- ODER HDMI-VERSIONEN</u><br>Dies aktiviert oder deaktiviert 16-Bit-LCD-Displays im 320x240-Modus, wodurch beispielsweise Spiele auf diesen größeren LCD-Displays möglich werden. Bei 800x480-Displays wird das 320x240-Bild um den Faktor 2 skaliert und nimmt den Bildschirmbereich 80,0 bis 719,479 ein<br>Bei 480x272-Displays wird das 320x240-Bild in einem Fenster angezeigt und nimmt den Bildschirmbereich 80,16 bis 399,255 ein |
| OPTION LEGACY ON<br>oder | | Damit wird der Kompatibilitätsmodus mit den im ursprünglichen Colour Maximite verwendeten Grafikbefehlen ein- oder ausgeschaltet. Die |

| | | |
|---|---|---|
| OPTION LEGACY OFF | | Befehle COLOUR, LINE, CIRCLE und PIXEL verwenden die alte Syntax, und alle Zeichenbefehle akzeptieren Farben im Bereich von 0 bis 7. Hinweise:<br>• Schlüsselwörter wie RED, BLUE usw. sind nicht implementiert, daher sollten sie bei Bedarf als Konstanten definiert werden.<br>• Die Syntax der Legacy-Befehle findest du im Colour Maximite MMBasic Language Manual. Dieses kannst du unter https://geoffg.net/OriginalColourMaximite.html herunterladen. |
| OPTION LIST | | Hier werden die Einstellungen aller Optionen aufgelistet, die von ihrer Standardeinstellung abweichen und dauerhaft gespeichert sind. OPTION LIST zeigt außerdem die Versionsnummer und die geladene Firmware an. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden. |
| OPTION LOCAL VARIABLES n | | Dieser Befehl legt die Anzahl der Variablen fest (MM.INFO(MAX VARS)), die als lokale Variablen zugewiesen werden sollen. Im Allgemeinen sollte dies nicht erforderlich sein, aber wenn du beispielsweise viele globale Konstanten zuweisen möchtest und nicht viele lokale Variablen benötigst, könnte dies genutzt werden, um das Verhältnis anzupassen. Dies ist eine temporäre Option, die so lange besteht, bis der Prozessor zurückgesetzt wird. Sie kann nicht aufgerufen werden, wenn bereits Variablen definiert wurden. |
| OPTION MILLISECONDS ON\|OFF | | Dies aktiviert oder deaktiviert die Millisekunden-Ausgabe in der TIME$-Funktion.<br>D. h., HH:MM:SS.mmm<br>Der Millisekundenzähler wird auf Null gesetzt, sobald die Zeit mit dem Befehl TIME, dem Befehl WEB NTP oder dem Befehl RTC GETTIME aktualisiert wird. Die Standardeinstellung ist OFF. |
| OPTION MODBUFF ENABLE/DISABLE [sizeinK] | ✓ | Erstellt oder entfernt einen Bereich des Flash-Speichers, der zum Laden und Abspielen von .MOD-Dateien verwendet wird. Wenn aktiviert, wird ein Mod-Puffer mit einer Größe von 128 KB erstellt. Dies kann mit „sizeinK“ überschrieben werden.<br>Beachte, dass diese Option einen Teil des Flash-Dateisystems reserviert (d. h., sie verkleinert das Flash-Dateisystem). Die Standardeinstellung ist deaktiviert.<br>Hinweis: Diese Option ist auf einem RP2350 mit aktiviertem PSRAM nicht erforderlich. In diesem Fall wird die MOD-Datei in den PSRAM geladen. |
| OPTION MOUSE CLKpin, DATApin<br><br>OPTION MOUSE DISABLE | ✓<br><br>✓ | NUR FÜR NICHT-USB-FIRMWARE<br>Lege die Pins fest, die zum Anschluss einer PS2-Maus verwendet werden sollen. Mit diesem Befehl wird die Maus beim Booten automatisch konfiguriert und du kannst ohne zusätzliche Befehle Interrupts einrichten und Werte auslesen. Dies unterscheidet sich von MOUSE OPEN, das nur während der Programmausführung eine Verbindung zur Maus herstellt. Die PS2-Maus MUSS deaktiviert sein.<br><br>Deaktiviert die automatische Verbindung zu einer PS2-Maus und gibt die Pins für den normalen Gebrauch frei. |
| OPTION MOUSE SENSITIVITY f! | ✓ | NUR USB-FIRMWARE |

| | | |
|---|---|---|
| | | Standardmäßig betreibt die Firmware eine USB-Maus im Boot-Modus. Das bedeutet, dass sie 8-Bit-x- und y-Positionen sowie den Status der drei Standardtasten zurückgibt. Wenn du OPTION MOUSE SENSITIVITY einstellst, weist die Firmware die Maus an, mit ihrem vollen Funktionsumfang zu arbeiten. Das bedeutet, dass sie versucht, den vollständigen USB-Bericht der Maus zu interpretieren, einschließlich der Radposition, aller Tasten und der 8-, 12- oder 16-Bit-x/y-Positionsdaten.<br>Die Einstellung „f$“ aktiviert den vollständigen Mausbericht und skaliert die x/y-Positionen um „f!“.<br>Die Firmware unterstützt nicht alle Mausmodelle. Sollte dies bei einer bestimmten Maus zu Problemen führen, setze den Wert unter OPTION MOUSE SENSITIVITY auf 0 zurück, um in den Boot-Modus zu gelangen.<br>Die Aktivierung dieser Option mit dem Wert 1,0 bei Verwendung einer standardmäßigen Microsoft Basic Optical Mouse wurde vollständig getestet und ermöglicht die Verwendung des Rads in einem Programm |
| OPTION NOCHECK ON/OFF | | Wenn dieser Befehl auf ON gesetzt ist, wird die standardmäßige Überprüfung auf Interrupts und Strg-C am Ende jedes Befehls deaktiviert. Die Einstellung auf ON ermöglicht zeitkritische Verarbeitungsprozesse ohne Unterbrechungsrisiko. Der Befehl sollte jedoch mit Bedacht eingesetzt werden, da das Programm sonst möglicherweise nur durch einen Hardware-Reset gestoppt werden kann. |
| OPTION PICO ON/OFF | ✓ | <u>ALLE VERSIONEN AUSSER WEBMITE</u><br>Wenn auf OFF gesetzt, sind die Pins GP23, GP24 und GP29 nicht für den normalen Pico-Betrieb konfiguriert und stehen sofort zur Verfügung. Standardmäßig ON für RP2350A und RP2040, OFF für RP2350B |
| OPTION PIN nbr | | Lege „nbr“ als PIN (Persönliche Identifikationsnummer) für den Zugriff auf die Konsolenaufforderung fest. „nbr“ kann eine beliebige Zahl zwischen 0 und 8 sein.<br>Wann immer ein laufendes Programm aus irgendeinem Grund versucht, zur Befehlszeile zu wechseln, fragt MMBasic diese Nummer ab, bevor die Eingabeaufforderung angezeigt wird. Dies ist eine Sicherheitsfunktion, da ein Eindringling ohne Zugriff auf die Befehlszeile weder das Programm im Speicher auflisten oder ändern noch den Betrieb von MMBasic in irgendeiner Weise beeinflussen kann. Um diese Funktion zu deaktivieren, gib für die PIN-Nummer eine Null ein (d. h. OPTION PIN 0).<br>Eine dauerhafte Sperre kann durch die Verwendung von 99999999 als PIN-Nummer aktiviert werden. Wenn eine dauerhafte Sperre aktiviert wurde oder die PIN-Nummer verloren geht, ist die einzige Möglichkeit zur Wiederherstellung das Neuladen der PicoMite-Firmware.<br>Dieser Befehl muss an der Eingabeaufforderung ausgeführt werden (nicht in einem Programm). |
| OPTION PLATFORM name$ | ✓ | Ermöglicht es einem Benutzer, eine bestimmte Hardwarekonfiguration zu identifizieren, die dann in Programmen verwendet werden kann, um den Programmablauf zu steuern.<br>„name$“ kann bis zu 31 Zeichen lang sein. Dies ist eine permanente Option.<br>MM.INFO$(PLATFORM) gibt diese Zeichenfolge zurück.<br>Das kannst du zum Beispiel bei einer bestimmten Hardwarekonfiguration so anwenden: |

| | | |
|---|---|---|
| | | `OPTION PLATFORM "GameMite"`<br>Dann können Programme, die auf dieser oder anderen Plattformen laufen, Folgendes verwenden:<br>`IF MM.INFO$(PLATFORM) = "GameMite" THEN …` |
| OPTION POWER PFM \| PWM | ✓ | Ändert den Betrieb des 3,3-V-Schaltnetzteils.<br>Standardmäßig läuft dies im PFM-Modus. PWM bietet eine bessere Rauschunterdrückung, ist aber weniger energieeffizient. Beachte, dass das System unter hoher Last unabhängig von dieser Einstellung im PWM-Modus läuft. |
| OPTION PSRAM PIN n<br>oder<br>OPTION PSRAM PIN DISABLE | ✓ | Aktiviert/deaktiviert die PSRAM-Unterstützung.<br>„n“ ist der PSRAM-Chip-Select-Pin (CS) und kann GP0, GP8, GP19 oder GP47 sein.<br>Typischerweise wird bei Pimoroni-Boards GP47 verwendet. Standardmäßig ist die Funktion deaktiviert.<br>Nach dem Einschalten ist der Inhalt des PSRAM unbestimmt; verwende RAM ERASE, um ihn vor der Verwendung zu löschen. |
| OPTION RESET | ✓ | Setzt alle gespeicherten Optionen auf ihre Standardwerte zurück.<br>Dieser Befehl muss an der Befehlszeile ausgeführt werden (nicht in einem Programm). |
| OPTION RESET cfg<br>oder<br>OPTION RESET LIST | ✓ | Setzt alle Optionen für die Konfiguration „cfg“ auf die Standardwerte zurück.<br>OPTION RESET LIST listet alle verfügbaren Konfigurationen auf.<br>Dieser Befehl muss in der Eingabeaufforderung ausgeführt werden (nicht in einem Programm). |
| OPTION RESOLUTION nn [,cpuspeedinKhz] | | <u>NUR FÜR HDMI- UND VGA-VERSIONEN</u><br>Bei Firmware mit HDMI-Video stellst du die Videoauflösung auf „nn“ ein.<br>Wobei „nn“ steht für:<br>640x480 oder 640<br>720x400 oder 720<br>800x600 oder 800 (nur RP2350)<br>848x480 oder 848 (nur RP2350)<br>1280x720 oder 1280 (nur HDMI)<br>1024x768 oder 1024 (nur HDMI)<br>Für 640x480 kann die Bildwiederholfrequenz auf 60 Hz (252 MHz oder 378 MHz) oder 75 Hz (315 MHz) eingestellt werden, indem du „cpuspeedinKHz“ an den Befehl anhängst (z. B. 252000, 378000 oder 315000).<br>Jede VGA- und HDMI-Auflösung kann in verschiedenen Modi betrieben werden, die mit dem Befehl MODE eingestellt werden.<br>Beachte, dass die Auflösungen 800x600 und 848x480 sowohl die maximale Programmgröße als auch den für Basic-Programme verfügbaren variablen Speicherplatz reduzieren |
| OPTION RTC AUTO ENABLE \| DISABLE | ✓ | Aktiviert das automatische Laden von time$ & date$ aus der RTC beim Booten und stündlich. Wenn diese Option aktiviert ist und die RTC nicht reagiert, wird jedes laufende Programm mit einer Fehlermeldung |

| | | |
|---|---|---|
| | | abgebrochen. An der Befehlszeile wird eine Informationsmeldung ausgegeben.<br>Dieser Befehl muss an der Befehlszeile ausgeführt werden (nicht in einem Programm). |
| OPTION SDCARD CSpin [,CLKpin, MOSIpin, MISOpin]<br>oder<br>OPTION SDCARD DISABLE | ✓ | Lege die für die SD-Kartenschnittstelle zu verwendenden I/O-Pins fest oder deaktiviere sie.<br>Wenn die optionalen Pins nicht angegeben werden, verwendet die SD-Karte die durch OPTION SYSTEM SPI angegebenen Pins.<br>Hinweis: Die mit OPTION SYSTEM SPI angegebenen Pins müssen ein gültiger Satz von Hardware-SPI-Pins (SPI oder SPI2) sein, die mit OPTION SDCARD angegebenen Pins können jedoch beliebige Pins sein. Die mit OPTION SYSTEM SPI und OPTION SDCARD angegebenen Pins dürfen nicht identisch sein.<br>Dieser Befehl muss an der Eingabeaufforderung ausgeführt werden (nicht in einem Programm). |
| OPTION SDCARD COMBINED CS | ✓ | Dies legt fest, dass der Touch-Chip-Select-Pin auch für die SD-Karte verwendet wird. In diesem Fall ist eine externe Schaltung erforderlich, um den SD-Chip-Select zu implementieren. Siehe „SD-Karten“ im Kapitel „Programm- und Datenspeicherung“. |
| OPTION SERIAL CONSOLE uartapin, uartbpin [,B]<br><br><br><br>OPTION SERIAL CONSOLE DISABLE | ✓ | Gib an, dass der Zugriff auf die Konsole über einen seriellen Hardware-Port erfolgen soll (anstelle einer virtuellen seriellen Schnittstelle über USB).<br>„uartapin“ und „uartbpin“ können ein beliebiges gültiges Paar von rx- und tx-Pins für COM1 oder COM2 sein. Die Reihenfolge, in der sie angegeben werden, spielt keine Rolle. Die Geschwindigkeit ist standardmäßig auf 115200 Baud eingestellt, kann aber mit OPTION BAUDRATE geändert werden. Durch Hinzufügen des Parameters „B“ wird die Ausgabe sowohl an den seriellen Port als auch an den USB-Anschluss gesendet.<br><br>Zurück zur normalen USB-Konsole.<br><br>Diese Befehle müssen an der Eingabeaufforderung ausgeführt werden (nicht in einem Programm). |
| OPTION SYSTEM I2C sdapin, sclpin [,SLOW/FAST] | ✓ | Lege den I²C-Port und die Pins fest, die von Systemgeräten (LCD-Panel und RTC) verwendet werden sollen.<br>Die PicoMite-Firmware verwendet einen bestimmten I²C-Port für Systemgeräte und lässt den anderen für den Programmierer frei. Dieser Befehl legt fest, welche Pins verwendet werden sollen und somit, welcher der I²C-Ports verwendet wird.<br>Die dem SYSTEM-I2C zugewiesenen Pins stehen nicht für andere MMBasic-SETPIN-Einstellungen zur Verfügung, können aber mit dem Standard-I2C-Befehl für weitere I2C-Geräte genutzt werden. Hinweis: I2C(2) OPEN und I2C(2) CLOSE sind in diesem Fall nicht verfügbar.<br>Standardmäßig wird der I²C-Port mit einer Geschwindigkeit von 400 kHz und einer Zeitüberschreitung von 100 ms geöffnet. Die I²C-Frequenz kann mit dem optionalen dritten Parameter eingestellt werden, der die Werte FAST = 400 kHz oder SLOW = 100 kHz annehmen kann.<br>Dieser Befehl muss an der Eingabeaufforderung ausgeführt werden (nicht in einem Programm). |

| | | |
|---|---|---|
| OPTION SYSTEM SPI CLKpin, MOSIpin, MISOpin<br>oder<br>OPTION SYSTEM SPI DISABLE | ✓ | Lege den SPI-Port und die Pins fest, die von Systemgeräten (SD-Karte, LCD-Panel usw.) verwendet werden sollen, oder deaktiviere sie.<br>Die PicoMite-Firmware nutzt einen bestimmten Hardware-SPI-Port für Systemgeräte, während der andere für den Benutzer reserviert bleibt. Dieser Befehl legt fest, welche Pins verwendet werden sollen und somit, welcher der SPI-Ports genutzt wird. Die dem SYSTEM-SPI zugewiesenen Pins stehen für andere MMBasic-Befehle nicht zur Verfügung.<br>Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden. |
| OPTION TAB 2 \| 3 \| 4 \| 8 | ✓ | Lege den Abstand für die Tabulatortaste fest. Der Standardwert ist 2. |
| OPTION TCP-SERVER-PORT n | ✓ | NUR WEBMITE<br>Startet bei jedem Neustart von WebMite einen TCP-Server auf Port 'n'. Normalerweise verwenden HTTP-Server Port 80.<br>Verwende „OPTION TCP SERVER PORT 0“, um die Funktion zu deaktivieren<br>Wenn der Server läuft, kann er auf bis zu MM.INFO(MAX CONNECTIONS) Verbindungen reagieren |
| OPTION TELNET CONSOLE OFF\|ONLY\|ON | ✓ | NUR WEBMITE<br>Konfiguriert die Handhabung der Konsole über Telnet.<br>ON = Konsolenausgabe wird an USB und Telnet gesendet<br>ONLY = Konsolenausgabe wird nur an Telnet gesendet<br>OFF = Konsolenausgabe wird nur an USB gesendet |
| OPTION TFTP OFF\|ON | ✓ | WEBMITE ONLY<br>Aktiviert oder deaktiviert den TFTP-Server. Standardmäßig ist er aktiviert. |
| OPTION TOUCH T_CS-Pin, T_IRQ-Pin [, Beep]<br>oder<br>OPTION TOUCH DISABLE | ✓ | NICHT BEI VGA- ODER HDMI-VERSIONEN<br>Konfiguriert MMBasic für die Touch-Funktion eines angeschlossenen LCD-Bildschirms.<br>„T_CS-Pin“ und „T_IRQ-Pin“ sind die I/O-Pins, die jeweils für Chip-Select und Touch-Interrupt verwendet werden (es können beliebige freie Pins verwendet werden). Die übrigen Pins werden mit denen verbunden, die mit dem Befehl OPTION SYSTEM SPI festgelegt wurden.<br>„Beep“ ist ein optionaler Pin, der an einen kleinen Summer/Piepser angeschlossen werden kann, um ein „Klick“- oder Piepton zu erzeugen, wenn ein Advanced-Graphics-Steuerelement berührt wird (z. B. Optionsfeld, Schalter usw.). Dies wird in *Advanced Graphics Functions.pdf* beschrieben.<br>Dieser Befehl muss an der Eingabeaufforderung ausgeführt werden (nicht in einem Programm). |
| OPTION TOUCH FT6336 IRQpin, RESETpin [,BEEPpin] [,sensitivity] | ✓ | NICHT BEI VGA- ODER HDMI-VERSIONEN<br>Aktiviert die Touch-Unterstützung für den kapazitiven Touch-Chip FT6336. Die Empfindlichkeit ist eine Zahl zwischen 0 und 255 – der Standardwert ist 50, niedrigere Werte bedeuten eine höhere Empfindlichkeit.<br>SDA und SCK sollten an gültige I2C-Pins angeschlossen und mit OPTION SYSTEM I2C eingerichtet werden. Siehe auch die TOUCH- |

| | | |
|---|---|---|
| | | Funktion. |
| OPTION VCC voltage | | Legt die Spannung (voltage /Vcc) fest, die dem Raspberry Pi Pico zugeführt wird.<br>Wenn du die ADC-Pins zur Spannungsmessung verwendest, nutzt die PicoMite-Firmware die Spannung am Pin mit der Bezeichnung VREF als Referenz. Diese Spannung kann mit einem Multimeter genau gemessen und mit diesem Befehl eingestellt werden, um genauere Messergebnisse zu erzielen.<br>Der Parameter wird nicht gespeichert und sollte im Programm initialisiert werden. Der Standardwert, falls nicht festgelegt, ist 3,3. |
| OPTION UDP SERVER PORT n | ✓ | NUR WEBMITE-VERSION<br>Richtet einen Listening-Socket auf dem angegebenen Port ein. Alle an diesem Port empfangenen UDP-Datagramme werden verarbeitet und der Inhalt in MM.MESSAGE$ gespeichert. Die IP-Adresse des Absenders wird in MM.ADDRESS$ gespeichert. Hinweis: Wenn das UDP-Datagramm länger als 255 Zeichen ist, werden alle überschüssigen Zeichen verworfen.<br>Verwende „OPTION UDP SERVER PORT 0“, um die Funktion zu deaktivieren |
| OPTION VGA PINS HSYNCpin, BLUEpin | ✓ | NUR VGA-VERSION<br>Ändert die für die VGA-Bildschirmausgabe verwendeten Pins und ermöglicht so mehr Flexibilität beim PCB-Design oder bei der Verdrahtung. „HSYNCpin“ definiert den Anfang eines Paares aufeinanderfolgender GP-nummerierter Pins, die mit HSYNC und VSYNC verbunden sind<br>„BLUEpin“ definiert den Anfang von vier aufeinanderfolgenden GP-nummerierten Pins, die mit BLUE, GREEN_LSB, GREEN_MSB und RED verbunden sind. |
| OPTION WEB MESSAGES ON/OFF | | NUR WEBMITE-VERSION<br>Deaktiviert informative Web-Meldungen, wenn auf OFF gesetzt. Standard ist ON |
| OPTION WIFI ssid$, passwd$, [name$] [,ipaddress$, mask$, gateway$] | ✓ | NUR WEBMITE-VERSION<br>Konfiguriert die Firmware so, dass sie sich beim Neustart automatisch mit einem WLAN-Netzwerk verbindet.<br>„ssid$“ ist der Name des Netzwerks und „password$“ ist das Passwort für den Zugriff. Beide sind Zeichenfolgen, und wenn Zeichenfolgenkonstanten verwendet werden, sollten sie in Anführungszeichen gesetzt werden.<br>Optional kann ein Name für das Gerät unter „name$“ angegeben werden, andernfalls wird ein Name aus der eindeutigen Geräte-ID erstellt.<br>Optional können eine statische IP-Adresse, eine IP-Maske und eine Gateway-Adresse als „ipaddress$“, „mask$“ und „gateway$“ angegeben werden<br>z. B. OPTION WIFI “mysid”,”mypassword”, ”myPico”, “192.168.1.111”, ”255.255.255.0”, ”192.168.1.1” |

# Befehle

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

| | |
|---|---|
| ‘ (einfaches Anführungszeichen) | Leitet einen Kommentar ein, und der darauf folgende Text wird ignoriert. Kommentare können an beliebiger Stelle in einer Zeile stehen. |
| *file | Der Sternchenbefehl ist eine Abkürzung für RUN, die nur an der MMBasic-Eingabeaufforderung verwendet werden darf. Z. B.<br>* RUN<br>*foo RUN „foo“<br>*"foo bar" RUN „foo bar“<br>*foo –wombat RUN „foo“, „--wombat“<br>*foo "wom" RUN "foo", CHR$(34) + "wom" + CHR$(34)<br>*foo --wom="bat" RUN „foo“, „--wom=“ + CHR$(34) + „bat“ + CHR$(34)<br>Zeichenfolgenausdrücke werden von diesem Befehl nicht unterstützt/ausgewertet; alle angegebenen Argumente werden als literale Zeichenfolge an den RUN-Befehl übergeben. |
| ? (Fragezeichen) | Abkürzung für den PRINT-Befehl. |
| /*<br>*/ | Anfang und Ende von mehrzeiligen Kommentaren. /* und */ müssen die ersten Nicht-Leerzeichen am Anfang einer Zeile sein und ein Leerzeichen oder ein Zeilenende nach sich ziehen (d. h. sie sind MMBasic-Befehle). Mehrzeilige Kommentare können nicht innerhalb von Unterprogrammen und Funktionen verwendet werden. Alle Zeichen nach */ in einer Zeile werden ebenfalls als Kommentar behandelt. |
| A: oder B: | Abkürzung für DRIVE „A:“ und DRIVE „B:“ an der Eingabeaufforderung |
| ADC | Die ADC-Befehle bieten eine alternative Methode zur Erfassung analoger Eingänge und sind für die schnelle Erfassung vieler Messwerte in ein Array gedacht. |
| ADC OPEN freq, n_channels [,interrupt] | Dadurch werden bis zu 4 ADC-Kanäle aus der Gruppe ADC0–ADC3 zur Verwendung zugewiesen und so eingestellt, dass sie mit der angegebenen Frequenz umgewandelt werden.<br>Der Pin-Bereich umfasst GP26, GP27, GP28 und GP29 für den RP2040 und RP2350A bzw. GP40, GP41, GP42 und GP43 beim RP2350B. Wenn die Anzahl der Kanäle eins ist, wird immer GP26 (ADC0) verwendet, bei zwei werden GP26 und GP27 (ADC0 und ADC1) verwendet usw. Die Abtastung mehrerer Kanäle erfolgt sequenziell (es gibt nur einen ADC). Die Pins sind an die Funktion gebunden, wenn ADC OPEN aktiv ist<br>Die maximale Gesamtfrequenz beträgt CPUspeed/96 (z. B. 520 kHz, wenn alle vier Kanäle bei einer CPU-Taktfrequenz von 200 MHz abgetastet werden sollen). Beachte, dass eine Gesamtabtastfrequenz von über 500 kHz eine Übertaktung des ADC darstellt.<br>Der optionale Interrupt-Parameter gibt einen Interrupt an, der aufgerufen werden soll, wenn die Umwandlung abgeschlossen ist. Wenn nichts angegeben wird, erfolgt die Umwandlung blockierend |
| ADC FREQUENCY freq | Dies ändert die Abtastfrequenz der ADC-Wandlung, ohne dass geschlossen und erneut geöffnet werden muss |
| ADC CLOSE | Gibt die Pins für den normalen Gebrauch frei |

| | |
|---|---|
| ADC START array1!() [,array2!()] [,array3!()] [,array4!()] [,C1min] [,C1max] [,C2min] [,C2max] [,C3min] [,C3max] [,C4min] [,C4max] | Dies startet die Umwandlung in die angegebenen Arrays. Die Arrays müssen Gleitkommatypen sein und dieselbe Größe haben. Die Größe der Arrays bestimmt die Anzahl der Umwandlungen. Start kann wiederholt aufgerufen werden, sobald der ADC OPEN ist<br>„Cxmin“ und „Cxmax“ skalieren die Messwerte. Beispielsweise erzeugen C1min=200 und C1max=100 Werte im Bereich von 200 bis 100 für Äquivalenzspannungen von 0 bis 3,3. Wird die Skalierung nicht verwendet, werden die Ergebnisse als Spannung zwischen 0 und OPTION VCC (Standardwert 3,3 V) zurückgegeben. |
| ADC RUN array1%(),array2%) | Lässt den ADC kontinuierlich im doppelt gepufferten Modus laufen. Der ADC füllt zuerst array1%, dann array2% und anschließend wieder array1% usw. Wenn im Befehl ADC OPEN mehr als ein ADC-Kanal angegeben ist, werden die Daten verschachtelt. Die Daten werden als gepackte 8-Bit-Werte zurückgegeben (verwende MEMORY UNPACK, um sie in ein normales Array zu konvertieren). MM.INFO(ADC) gibt die Nummer des derzeit zum Lesen verfügbaren Puffers zurück (1 oder 2). |
| ARRAY ADD in(), value, out() | Dies fügt den Wert 'value' zu jedem Element der Matrix in() hinzu (oder hängt ihn bei Zeichenketten an) und speichert das Ergebnis in out(). Funktioniert für Arrays beliebiger Dimension mit Zeichenketten sowie mit Ganzzahlen und Gleitkommazahlen (kann zwischen Ganzzahlen und Gleitkommazahlen konvertieren). Die Einstellung von num auf 0 oder „“ ist optimiert und stellt eine schnelle Methode zum Kopieren eines gesamten Arrays dar. in() und out() können dasselbe Array sein. |
| ARRAY INSERT targetarray(), [d1] [,d2] [,d3] [,d4] [,d5] , sourcearray() | Dies ist das Gegenteil von ARRAY SLICE, hat eine sehr ähnliche Syntax und ermöglicht es dir beispielsweise, mit einem einzigen Befehl einen einzelnen Vektor in ein Vektorarray oder ein eindimensionales String-Array in ein zweidimensionales String-Array zu ersetzen. Die Arrays können numerisch oder aus Strings bestehen, und „sourcearray“ und „destinationarray“ müssen identisch sein (Hinweis: Bei numerischen Arrays ist die Konvertierung zwischen Ganzzahlen und Gleitkommazahlen möglich).<br><br>z. B.<br>`OPTION BASE 1`<br>`DIM targetarray(3,4,5)`<br>`DIM sourcearray(4)=(1,2,3,4)`<br>`ARRAY INSERT targetarray(), 2, , 3, sourcearray()`<br>Setzt die Elemente 2,1,3 auf 1, 2,2,3 auf 2, 2,3,3 auf 3 und 2,4,3 auf 4 |
| ARRAY SET value, array() | Setzt alle Elemente in array() auf den Wert „value“. „value“ kann eine Zahl oder eine Zeichenkette sein, und „array“ muss dasselbe sein (Hinweis: Es kann zwischen Ganzzahlen und Gleitkommazahlen konvertiert werden). Beachte, dass dies der schnellste Weg ist, ein Array zu löschen, indem man es auf Null oder eine leere Zeichenkette setzt. |

| | |
|---|---|
| ARRAY SLICE sourcearray(), [d1] [,d2] [,d3] [,d4] [,d5] destinationarray() | Dieser Befehl kopiert einen bestimmten Satz von Werten aus einem mehrdimensionalen Array in ein eindimensionales Array. Das geht viel schneller als die Verwendung einer FOR-Schleife. Der Ausschnitt wird festgelegt, indem für alle bis auf einen der Indizes des Quellarrays ein Wert angegeben wird, und es sollten im Befehl genauso viele Indizes vorhanden sein – einschließlich des leeren – wie das Quellarray Dimensionen hat. Die Arrays können numerisch oder Zeichenfolgen sein, und „sourcearray“ und „destinationarray“ müssen identisch sein (Hinweis: Bei numerischen Arrays ist eine Konvertierung zwischen Ganzzahlen und Gleitkommazahlen möglich).<br>z. B.<br>`OPTION BASE 1`<br>`DIM a(3,4,5)`<br>`DIM b(4)`<br>`ARRAY SLICE a(), 2, , 3, b()`<br>Kopiert die Elemente 2,1,3 und 2,2,3 und 2,3,3 und 2,4,3 in das Array b() |
| ARC x, y, r1, [r2], a1, a2 [, c] | Zeichnet einen Kreisbogen mit einer bestimmten Farbe und Breite zwischen zwei Radialen (in Grad angegeben). Die Parameter für den Befehl ARC sind:<br>x: X-Koordinate des Bogenmittelpunkts<br>y: Y-Koordinate des Bogenmittelpunkts<br>r1: Innenradius des Bogens<br>r2: Außenradius des Bogens – kann weggelassen werden, wenn er 1 Pixel breit ist<br>a1: Startwinkel des Bogens in Grad<br>a2: Endwinkel des Bogens in Grad<br>c: Farbe des Bogens (wenn nicht angegeben, wird standardmäßig die Vordergrundfarbe verwendet)<br>Null Grad entspricht der 12-Uhr-Position. |
| ASTRO | Berechnet die Position eines Himmelsobjekts anhand der manuellen Zeit- und Ortsangabe, die mit dem Befehl LOCATION festgelegt wurde.<br>Dies ist einer aus einer Reihe von Befehlen, die hochpräzise astronomische Berechnungen durchführen, die sich für die Ausrichtung von Teleskopen oder zur Navigation eignen, und wird ausführlich in der Datei *GPS_Astro_Reference.pdf* beschrieben, die im ZIP-Download der Firmware enthalten ist. |

| | |
|---|---|
| AUTOSAVE<br>oder<br>AUTOSAVE CRUNCH<br>oder<br>AUTOSAVE APPEND<br>oder<br>AUTOSAVE N | Wechsle in den automatischen Programmeingabemodus. Dieser Befehl nimmt Textzeilen aus der seriellen Konsole auf und speichert sie im Programmspeicher.<br>Dies ist eine Möglichkeit, ein BASIC-Programm auf den Raspberry Pi Pico zu übertragen. Das zu übertragende Programm kann in einen Terminalemulator eingefügt werden, woraufhin dieser Befehl den Textstrom erfasst und im Programmspeicher ablegt. Er kann auch verwendet werden, um ein kleines Programm direkt über die Konsoleneingabe einzugeben.<br>Dieser Modus wird durch Eingabe von Strg-Z oder F1 beendet, wodurch die empfangenen Daten in den Programmspeicher übertragen werden und das vorherige Programm überschreiben. Verwende F2, um den Modus zu verlassen und das Programm sofort auszuführen.<br>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 genutzt werden, damit sie in den begrenzten Speicher passen. CRUNCH kann mit dem einzelnen Buchstaben C abgekürzt werden.<br>Die Option APPEND lässt das bestehende Programm unverändert und hängt die neuen Daten aus der seriellen Eingabe an dessen Ende an.<br>Die Option N führt die automatische Speicherung wie gewohnt durch, gibt die Daten jedoch nicht an die Konsole zurück. Sie wartet auf das Senden des ersten Zeichens und verwendet dann einen Timer, der<br>prüft, ob zwischen den Zeichen mehr als 100 ms liegen. Wenn diese Verzögerung festgestellt wird, wechselt es zurück in den normalen Eingabemodus und gibt die Meldung<br>„Drücke Strg-Z, F1 oder F2, um zu beenden“<br>Du kannst dann weitere Zeichen eingeben, die gespeichert werden sollen, oder einen der normalen Beendigungsbefehle verwenden.<br>Dieser Befehl kann jederzeit mit Strg-C abgebrochen werden, wodurch der Programmspeicher unverändert bleibt. |
| BACKLIGHT n [,DEFAULT]<br>BACKLIGHT n [,FreqInHz] | <u>NICHT-VGA- ODER HDMI-VERSIONEN</u><br>Stellt die Hintergrundbeleuchtung des Displays ein, gültige Werte sind 0 bis 100. Wenn DEFAULT angegeben wird, stellt die Firmware die Hintergrundbeleuchtung beim Hochfahren automatisch auf diesen Wert ein. Dies ist besonders nützlich im Batteriebetrieb, wo eine Reduzierung der Hintergrundbeleuchtung die Batterielaufzeit deutlich verlängern kann.<br>Einige Schaltungen sind zu langsam, um die Standard-PWM-Frequenz für die Hintergrundbeleuchtung zu verwenden, die gewählt wurde, um Interferenzen mit dem Audio zu vermeiden. In diesem Fall kann eine benutzerdefinierte Frequenz angegeben werden. Dies ist eine temporäre Option und muss beim Neustart neu eingestellt werden. |
| BEZIER x%(),y%() [,n] [,colour] | Zeichnet eine Bézier-Kurve mit einer unbegrenzten Anzahl von Kontrollpunkten. „x%()“ und „y%()“ sind eindimensionale Integer-Arrays, die die Koordinaten der Kontrollpunkte enthalten. Die Bézier-Kurve beginnt immer am ersten Kontrollpunkt. Die Arrays müssen dieselbe Dimension haben.<br>Der optionale Parameter „n“ legt fest, wie viele Kontrollpunkte für die Darstellung verwendet werden sollen. Wird er weggelassen, bestimmt die Größe von „x%()“ und „y%()“ die Anzahl. N muss >=2 und <= der Array-Größe sein.<br>Der optionale Parameter „colour“ legt die Farbe fest, in der die Kurve gezeichnet wird (Standard ist Weiß). |

| | |
|---|---|
| BIT(var%, bitno) = value | Setzt ein bestimmtes Bit (0–63) in einer ganzzahligen Variablen. „value“ kann 0 oder 1 sein.<br>Siehe auch die Funktion BIT |
| BITBANG | Wurde durch den Befehl DEVICE ersetzt. Aus Kompatibilitätsgründen kann BITBANG weiterhin in Programmen verwendet werden und wird automatisch in DEVICE umgewandelt |
| BITSTREAM<br><br>BITSTREAM pin1, count1, array1() [, mode1] [,pin2, count2, array2()] [,mode2] [,logic] | Der Befehl `BITSTREAM` erzeugt extrem genaue Bitfolgen an einem oder zwei GPIO-Pins. Jeder Zeitwert im Array gibt die Dauer (in Mikrosekunden) an, bevor der Pin-Zustand wechselt. Dies ermöglicht die präzise Erzeugung von Protokollen wie IR-Fernbedienungssignalen, PS/2-Tastatur-/Maus-Kommunikation und anderen zeitkritischen Wellenformen.<br><br>`pin1`, `pin2` sind die GPIO-Pin-Nummer(n) für den Ausgang. Du kannst GP-Notation (z. B. GP15) oder physikalische Pin-Nummern verwenden.<br>`count1`, `count2` geben die Anzahl der aus den Arrays zu verwendenden Zeitwerte an (1 bis 10000).<br>`array1()`, `array2()` sind numerische Arrays, die Zeitwerte in Mikrosekunden enthalten. Jeder Wert gibt die Zeit bis zum nächsten Pin-Wechsel an.<br>`mode1`, `mode2` geben den Ausgabemodus an (optional, Standardwert ist 0) 0 = Push-Pull (auf High und Low getrieben), 1 = Open-Collector (auf Low getrieben, hohe Impedanz bei High)<br>`logic` ist eine spezielle Funktion für den Same-Pin-Modus (optional, Standardwert ist 0, nur gültig, wenn pin1 = pin2) 0 = XOR (Umschaltung bei jedem Ereignis, ursprüngliches Verhalten), 1 = AND (Ausgang HIGH nur, wenn beide logischen Zustände HIGH sind), 2 = OR (Ausgang HIGH, wenn einer der logischen Zustände HIGH ist)<br><br>Weitere Informationen und Beispiele zur Verwendung des Befehls „bitstream“ zum Erstellen eines PS2-Ausgangs und zum Ansteuern eines NEC-IR-Senders findest du im Handbuch „BITSTREAM_User_Manual“ |
| BLIT<br><br>BLIT READ [#]b, x, y, w, h<br><br>BLIT WRITE [#]b, x, y [,mode] | BLIT ist ein einfacher Speicherbefehl zum Kopieren zwischen einem Display und einem Speicher oder von einem Speicher zu einem Display.<br>Hinweise:<br>• Es stehen 32 Puffer zur Verfügung, nummeriert von #1 bis #32.<br>• Bei der Angabe der Puffernummer ist das Symbol # optional.<br>• Alle anderen Argumente sind in Pixeln angegeben.<br>BLIT READ kopiert einen Teil des Bildschirms in den Speicherpuffer '#b'. Die Quellkoordinaten sind 'x' und 'y', die Breite des zu kopierenden Bildbereichs ist 'w' und die Höhe ist 'h'. Bei Verwendung dieses Befehls wird der Speicherpuffer automatisch angelegt und ausreichend Speicher zugewiesen. Dieser Puffer kann mit dem Befehl BLIT CLOSE freigegeben und der Speicher zurückgewonnen werden.<br>BLIT WRITE kopiert den Speicherpuffer '#b' auf den Bildschirm. Die Zielkoordinaten sind 'x' und 'y'.<br>Der optionale Parameter „mode“ ist standardmäßig auf 0 gesetzt und legt fest, wie die gespeicherten Bilddaten beim Auslesen verändert werden. Es handelt sich um das bitweise UND der folgenden Werte:<br>&B001 = von links nach rechts gespiegelt<br>&B010 = von oben nach unten gespiegelt<br>&B100 = transparente Pixel nicht kopieren |

| | |
|---|---|
| BLIT LOAD[BMP] [#]b, fname$ [,x] [,y] [,w] [,h] | BLIT LOAD lädt einen Blit-Puffer aus einer 24-Bit-BMP-Bilddatei. x und y legen die Startposition im Bild fest, an der das Laden beginnen soll, und w und h geben die Breite und Höhe des zu ladenden Bereichs an. Dieser Befehl funktioniert auf den meisten Bildschirmen (nicht nur auf solchen mit dem ILI9341-Controller).<br>Beispiel:<br>BLIT LOAD #1,"image1", 50,50,100,100<br>lädt einen Bereich von 100 x 100 Pixeln mit der oberen linken Ecke bei 50,50 aus dem Bild image1.bmp |
| BLIT CLOSE [#]b | BLIT CLOSE schließt den Speicherpuffer „#b“, damit er für einen weiteren BLIT READ-Befehl verwendet werden kann und der belegte Speicher freigegeben wird. |
| | NICHT VGA- ODER HDMI-VERSIONEN |
| BLIT MERGE colour, x, y, w, h | Kopiert einen Bereich des Framebuffers, der durch die Pixelkoordinaten „x“ und „y“ der oberen linken Ecke definiert ist und eine Breite von „w“ sowie eine Höhe von „h“ hat, auf das LCD-Display. Im Rahmen des Kopiervorgangs überlagert es das LCD-Display mit Pixeln aus dem Layer-Puffer, die nicht auf die angegebene „Farbe“ gesetzt sind. Die Farbe wird als Zahl zwischen 0 und 15 angegeben und steht für:<br>Black, Blue, Myrtle, Cobalt, Midgreen, Cerulean, green, cyan, red, magenta, rust, fuschia, brown, lilac, yellow und white<br>Erfordert sowohl einen Framebuffer als auch einen Layer-Buffer, um zu funktionieren. Wartet automatisch auf die Bildausblendung, bevor der Kopiervorgang auf ILI9341-, ST7789_320- und ILI9488-Displays gestartet wird |
| BLIT FRAMEBUFFER from, to, xin, yin, xout, yout, width, height [,colour] | Kopiert einen Bereich eines bestimmten „from“-Framebuffers N, F oder L in einen anderen „to“-Framebuffer N, F oder L. „xin“ und „yin“ definieren die linke obere Ecke des Bereichs mit den Maßen „width“ und „height“ auf dem Quell-Framebuffer, der kopiert werden soll. „xout“ und „yout“ definieren die linke obere Ecke des Bereichs auf dem Ziel-Framebuffer, der die Kopie aufnehmen soll. Der optionale Parameter „colour“ definiert eine Pixelfarbe auf der Quelle, die nicht kopiert wird. Wird er weggelassen, werden alle Pixel kopiert. Die Farbe wird als Zahl zwischen 0 und 15 angegeben, die Folgendes darstellt:<br>Black, Blue, Myrtle, Cobalt, Midgreen, Cerulean, green, cyan, red, magenta, rust, fuschia, brown, lilac, yellow und white<br>Erfordert, dass sowohl ein Framebuffer als auch ein Layer-Buffer erstellt wurden, um zu funktionieren. Wartet bei ILI9341-, ST7789_320- und ILI9488-Displays automatisch auf die Bildausblendung, bevor der Kopiervorgang gestartet wird |
| BLIT MEMORY Adresse, x, y [,col] | Kopiert einen Speicherbereich, der als gepacktes Array von Farb-Nibbles behandelt wird, in die aktuelle grafische Ausgabe, wie durch FRAMEBUFFER WRITE festgelegt. Die Farbe wird als Zahl zwischen 0 und 15 angegeben, die Folgendes darstellt:<br>Black, Blue, Myrtle, Cobalt, Midgreen, Cerulean, green, cyan, red, magenta, rust, fuschia, brown, lilac, yellow und white<br>Das erste Wort des Speicherbereichs, der bei „address%“ beginnt, muss die Breite und Höhe des zu kopierenden Bereichs als 16-Bit-Ganzzahlen enthalten, wobei die Breite die unteren 16 Bits bildet. Die Adresse muss an einer Wortgrenze ausgerichtet sein (durch 4 teilbar).<br>Wenn der optionale Parameter „col“ angegeben wird, wird diese bestimmte Farbe nicht kopiert. |

| | |
|---|---|
| | Wenn das oberste Bit entweder der Breite oder der Höhe auf 1 gesetzt ist, werden die Farbdaten als komprimiert behandelt (die verbleibenden 15 Bits werden als Breite und/oder Höhe verwendet). Der Komprimierungsalgorithmus ist einfach: Jedes Byte enthält im unteren Nibble eine Zählung (1–15) und im oberen Nibble eine Farbe (0–15). Falls mehr als 15 Pixel dieselbe Farbe haben, werden für diese Farbe zusätzliche Bytes verwendet. |
| BLIT COMPRESSED address%, x, y [,col] | Funktioniert genauso wie BLIT MEMORY, geht jedoch davon aus, dass die Daten komprimiert sind, und ignoriert das oberste Bit in Breite und Höhe |
| BLIT FLASH from, to, xin, yin, xout, yout, width, height [,colour] | Kopiert einen Bereich eines bestimmten „from“-Flash-Slots in einen „to“-Framebuffer N, F oder L. „xin“ und „yin“ definieren die linke obere Ecke des Bereichs mit der Breite „width“ und Höhe „height“ auf dem zu kopierenden Flash-Slot.<br>„xout“ und „yout“ definieren die linke obere Ecke des Bereichs auf dem Ziel-Framebuffer, der die Kopie empfangen soll.<br>Der optionale Parameter „colour“ definiert eine Pixelfarbe im Flash-Slot, die nicht kopiert wird. Wird er weggelassen, werden alle Pixel kopiert. Die Farbe wird als Zahl zwischen 0 und 15 angegeben, die Folgendes darstellt:<br>Schwarz, Blau, Myrte, Kobalt, Mittelgrün, Cerulean, Grün, Cyan, Rot, Magenta,<br>Rost, Fuchsia, Braun, Flieder, Gelb und Weiß |
| BLIT RESIZE src, dst, sx, sy, sw, sh, dx, dy, dw, dh [,transparent] | Skaliert einen rechteckigen Quellbereich mithilfe von Fast-Nearest-Neighbor-Sampling auf ein Zielrechteck.<br>`src` Auswahl des Quellpuffers. Unterstützt: `L`, `F`, `N` (VGA/HDMI-Builds), `2` (zweiter Framebuffer, VGA/HDMI-Builds), `T` (oberste Ebene, RP2350 VGA/HDMI-Builds).<br>`dst` \| Zielpuffer-Selektor. Unterstützt: `L`, `F`, `N` (VGA/HDMI-Builds), `2` (zweiter Framebuffer, VGA/HDMI-Builds), `T` (oberste Ebene, RP2350 VGA/HDMI-Builds).<br>`sx`, `sy` Koordinaten der Quelle oben links<br>`sw`, `sh` Breite und Höhe der Quelle (müssen > 0 sein<br>‘dx`, `dy` Koordinate der linken oberen Ecke des Ziels<br>‘dw`, `dh` Zielbreite und -höhe (muss > 0 sein)<br>`transparent` Optionaler RGB121-Pixelwert, der beim Schreiben des Ziels übersprungen werden soll. Bereich `-1` bis `15`, Standardwert `-1` (deaktiviert). |
| BLIT x1, y1, x2, y2, w, h | Kopiere einen Ausschnitt des Bildschirms in einen anderen Bereich des Bildschirms.<br>Die Quellkoordinaten sind 'x1' und 'y1'. Die Zielkoordinaten sind 'x2' und 'y2'. Die Breite des zu kopierenden Bildschirmbereichs ist 'w' und die Höhe ist 'h'.<br>Alle Argumente sind in Pixeln angegeben.<br>Wenn die Ausgabe auf ein LCD-Panel erfolgt, muss es sich entweder um die Controller SSD19863, ILI9341_8, ILI9341, ILI9488 (wenn MISO angeschlossen) oder ST7789_320 handeln. |

| | |
|---|---|
| BOX x, y, w, h [, lw] [,c] [,fill] | Zeichnet ein Rechteck auf dem Display, dessen linke obere Ecke bei „x“ und „y“ liegt, mit einer Breite von „w“ Pixeln und einer Höhe von „h“ Pixeln.<br>'lw' ist die Breite der Seiten des Kastens und kann Null sein. Der Standardwert ist 1.<br>'c' gibt die Farbe an und ist standardmäßig die Standard-Vordergrundfarbe, wenn nichts anderes angegeben wird. 'fill' ist die Füllfarbe. Sie kann weggelassen oder auf -1 gesetzt werden; in diesem Fall wird der Rahmen nicht gefüllt.<br>Alle Parameter können als Arrays angegeben werden, und die Software zeichnet die Anzahl der Kästchen, die durch die Abmessungen des kleinsten Arrays bestimmt wird. „x“ und „y“ müssen beide Arrays oder einzelne Variablen/Konstanten sein, andernfalls wird ein Fehler ausgegeben. „w“, „h“, „lw“, „c“ und „fill“ können entweder Arrays oder einzelne Variablen/Konstanten sein.<br>Eine Definition der Farben und Grafikkoordinaten findest du im Kapitel *„Grafikbefehle und -funktionen“* |
| BYTE(var$, byteno)=value | Setzt ein bestimmtes Byte in einer Zeichenkette auf einen ganzzahligen Wert. „Wert“ kann im Bereich von 0 bis 255 liegen. Die Bytenummer kann zwischen 1 und der aktuellen Länge der Zeichenkette liegen. Dies entspricht MID$(var$,byteno,1)=CHR$(value), wird aber viel schneller ausgeführt.<br>Siehe auch die Funktion BYTE. |
| CALL usersubname$ [,usersubparameters,..] | Dies ist eine effiziente Methode, um benutzerdefinierte Unterprogramme programmgesteuert aufzurufen (siehe auch die Funktion CALL()). In vielen Fällen kannst du damit komplexe SELECT- und IF THEN ELSEIF ENDIF-Klauseln vermeiden, und die Verarbeitung erfolgt wesentlich effizienter.<br>„usersubname$“ kann eine beliebige Zeichenkette, Variable oder Funktion sein, die sich auf den Namen einer normalen benutzerdefinierten Subroutine auflöst (kein integrierter Befehl). Die „usersubparameters“ sind dieselben Parameter, die auch beim direkten Aufruf der Subroutine verwendet würden. Eine typische Anwendung wäre das Schreiben eines beliebigen Emulators, bei dem je nach einer bestimmten Variablen eine von vielen Subroutinen aufgerufen werden soll. Außerdem ermöglicht es, einen Subroutinennamen als Variable an eine andere Subroutine oder Funktion zu übergeben. |
| CAMERA<br><br>CAMERA OPEN XLKpin, PLKpin, HSpin, VSCpin, RETpin, D0pin | <u>KEINE VGA- ODER HDMI-VERSIONEN</u><br>Befehl zur Unterstützung des OV7670-Kameramoduls.<br><br>Dies initialisiert die Kamera. Es gibt einen 12-MHz-Takt an XLK (PWM) aus und überprüft, ob Signale an PLK, VS und HS korrekt empfangen werden. Die Kamera wird auf eine Auflösung von 160x120 (QQVGA) eingestellt, was das Maximum ist, das innerhalb der Grenzen des verfügbaren Speichers erreichbar ist.<br>Aktiviere OPTION SYSTEM I2C in der PicoMite-Firmware und verbinde SCL und SDA mit den entsprechenden Pins (können auf dem Kameramodul mit SIOC und SIOD beschriftet sein). Diese Verbindungen müssen einen Pull-up auf 3,3 V haben – 2K7 empfohlen)<br>Die anderen Pins werden gemäß dem OPEN-Befehl verdrahtet. (Hinweis: VS kann auf deinem Modul als VSYNC, HS als HREF, PLK als PCLK, RET als RESET und XLK als XCLK beschriftet sein)<br>Der D0-Pin definiert den Anfang eines Bereichs von 8 zusammenhängenden Pins (z. B. GP0 – GP7). |

| | |
|---|---|
| CAMERA CAPTURE [scale, [x , y]] | Dies nimmt ein Bild von der Kamera (RGB565) auf und zeigt es auf einem LCD-Bildschirm an. Ein SPI-LCD muss angeschlossen und aktiviert sein, damit der Befehl funktioniert. (ILI9341 und ST7789_320 empfohlen).<br>Der Skalierungsfaktor ist standardmäßig auf 1 gesetzt, x und y jeweils auf 0<br>Standardmäßig wird ein Bild mit einer Auflösung von 160x120 auf dem LCD ausgegeben, wobei die linke obere Ecke bei 0,0 auf dem LCD liegt. Wenn du den Skalierungsfaktor auf 2 setzt, füllt das Bild ein 320x240-Display aus. Durch Einstellen der x- und y-Parameter wird die linke obere Ecke des Bildes auf dem LCD verschoben.<br>Die Aktualisierungsrate in einer Endlosschleife beträgt 7 FPS auf dem Display im Maßstab 1:1 und 5 FPS bei einer Skalierung auf 320x240.<br>Vorausgesetzt, das Display ist über MISO verkabelt, ist es möglich, das Bild mit dem Befehl SAVE IMAGE auf der Festplatte zu speichern. |
| CAMERA CLOSE | Schließt das Kamerasubsystem und gibt alle im Befehl OPEN zugewiesenen Pins frei. |
| CAMERA CHANGE image%(),change! [,scale [,x ,y]] | Die Kamera-Firmware kann mit diesem Befehl auch Bewegungen im Sichtfeld der Kamera erkennen. Dazu wird die Kamera im YUV-Modus statt im RGB-Modus betrieben. Das hat den Vorteil, dass die Helligkeits- und Farbinformationen getrennt sind und für ein Graustufenbild mit 256 Stufen nur ein Byte benötigt wird, was ideal für die Bewegungserkennung ist.<br>image% ist ein Array der Größe 160x120 Bytes (DIM image%(160,120/8-1)<br>Beim Aufruf des Befehls enthält es ein gepacktes 8-Bit-Graustufenbild.<br>Die Variable „change!“ gibt den Prozentsatz zurück, um den sich das Bild seit dem letzten Aufruf des Befehls verändert hat.<br>Optional wird, wenn „scale“ gesetzt ist, das Bild-Delta auf dem Bildschirm ausgegeben, d. h. die Differenz zwischen dem vorherigen Bild und diesem. Wie beim Befehl CAPTURE kann das Delta-Bild nach Bedarf skaliert und positioniert werden. Wenn der Parameter „scale“ weggelassen wird, wird das LCD durch diesen Befehl nicht aktualisiert. |
| CAMERA TEST tnum | Aktiviert oder deaktiviert ein Testsignal von der Kamera. tnum=2 erzeugt Farbbalken und tnum=0 schaltet zurück auf den Videoeingang. |
| CAMERA REGISTER reg%, data% | Setzt das Register „reg%“ in der Kamera auf den Wert „data%“. Bei Verwendung meldet der Befehl den vorherigen Wert an die Konsole und bestätigt automatisch, dass der neue Wert wie gewünscht gesetzt wurde. Die Farbwiedergabe der Kamera ist nach der Initialisierung angemessen, könnte aber wahrscheinlich durch die Anpassung der verschiedenen Kameraregister weiter verbessert werden. |
| CAT S$, N$ | Verkettet die Zeichenfolgen, indem N$ an S$ angehängt wird. Dies entspricht funktional `S$ = S$ + N$`, läuft aber etwas schneller. |
| CHAIN fname$ [cmdline$] | Ermöglicht es einem Programm, ein anderes Programm auszuführen, wobei der Variablenbereich erhalten bleibt – es wird empfohlen, den Befehl im Hauptprogramm und nicht innerhalb einer Subroutine zu verwenden. Wenn der optionale Parameter „cmdline$“ angegeben wird, wird dieser in MM.CMDLINE$ an das verkettete Programm übergeben. |
| CHDIR dir$ | Ändert das aktuelle Arbeitsverzeichnis auf dem Standardlaufwerk in „dir$“<br>Der spezielle Eintrag „..“ steht für das übergeordnete Verzeichnis des aktuellen Verzeichnisses und „.“ für das aktuelle Verzeichnis. „/“ ist das Stammverzeichnis. |

| | |
|---|---|
| CIRCLE x, y, r [,lw] [, a] [, c] [, fill] | Zeichnet einen Kreis mit dem Mittelpunkt bei 'x' und 'y' und einem Radius von 'r' auf dem Bildschirm. ‘lw’ ist optional und gibt die Linienbreite an (Standardwert ist 1).<br>'c' ist die optionale Farbe und wird standardmäßig auf die aktuelle Vordergrundfarbe gesetzt, wenn sie nicht angegeben wird. Das optionale 'a' ist eine Gleitkommazahl, die das Seitenverhältnis festlegt. Wenn das Seitenverhältnis nicht angegeben wird, ist der Standardwert 1,0, was einen Standardkreis ergibt. 'fill' ist die Füllfarbe und kann weggelassen oder auf -1 gesetzt werden; in diesem Fall wird das Feld nicht gefüllt.<br>Alle Parameter können als Arrays angegeben werden, und die Software zeichnet die Anzahl der Kreise, die durch die Abmessungen des kleinsten Arrays bestimmt wird. 'x',<br>'y' und 'r' müssen alle Arrays oder alle einzelne Variablen/Konstanten sein, andernfalls wird ein Fehler ausgegeben. 'lw', 'a', 'c' und 'fill' können entweder Arrays oder einzelne Variablen/Konstanten sein.<br>Eine Definition der Farben und Grafikkoordinaten findest du im Kapitel *„Grafikbefehle und -funktionen*“. |
| CLEAR | Löscht alle Variablen und gibt den von ihnen belegten Speicher frei.<br>Siehe ERASE zum Löschen bestimmter Array-Variablen. |
| CLOSE [#]fnbr [,[#]fnbr] … | Schließe die zuvor mit der Dateinummer „#fnbr“ geöffneten Dateien. Das # ist optional. Siehe auch den Befehl OPEN. |
| CLS [colour] | Löscht den Bildschirm des LCD-Displays. Optional kann „Farbe“ angegeben werden, die beim Löschen des Bildschirms als Hintergrundfarbe verwendet wird. |
| CMM2 LOAD<br>oder<br>CMM2 RUN | Lädt und/oder führt ein Programm von der Festplatte mithilfe des CMM2-Programmlade-Mechanismus aus. Dies beinhaltet eine aggressive Komprimierung des Programms und unterstützt #INCLUDE-Dateien sowie #DEFINE-Textersetzungen. Dies kann zur Kompatibilität mit CMM2-Programmen oder zur Strukturierung von Programmen in separate Module verwendet werden. Es ist wichtig zu beachten, dass bei Verwendung dieser Befehle jegliche Bearbeitung von Programmen offline oder direkt von und auf die Festplatte erfolgen muss, da die Quelldateien nicht aus der durch diese Befehle geladenen Version rekonstruiert werden können. |
| COLOUR fore [, back]<br>oder<br>COLOR fore [, back] | Legt die Standardfarbe für Befehle (PRINT usw.) fest, die auf dem angeschlossenen LCD-Bildschirm angezeigt werden. „fore“ ist die Vordergrundfarbe, „back“ die Hintergrundfarbe. Der Hintergrund ist optional und wird, falls nicht angegeben, standardmäßig auf die zuvor festgelegte Hintergrundfarbe gesetzt oder auf Schwarz, falls diese zuvor nicht geändert wurde. |
| COLOUR MAP inarray%(), outarray%() [,colourmap%()] | Dieser Befehl generiert RGB888-Farben in outarray% aus den Farbcodes (0–15) in inarray%. Wenn der optionale Parameter colourmap% verwendet wird, muss dieser 16 Elemente lang sein. In diesem Fall werden die Werte in inarray% den Farben für den entsprechenden Indexwert in colourmap% zugeordnet |
| CONFIGURE cfg<br><br>CONFIGURE LIST | Konfiguriert eine Platine gemäß dem in „cfg“ angegebenen Äquivalent von OPTION RESET.<br><br>Listet alle für die Firmware-Version verfügbaren Konfigurationen auf. |

| | |
|---|---|
| CONST id = expression<br>[, id = expression] … etc | Erstellt einen konstanten Bezeichner, der nach der Erstellung nicht mehr geändert werden kann.<br>'id' ist der Bezeichner, der denselben Regeln folgt wie Variablen. Der Bezeichner kann ein Typ-Suffix (!, %, oder $) haben, dies ist jedoch nicht erforderlich. Wenn es angegeben wird, muss es mit dem Typ von 'expression' übereinstimmen. 'expression' ist der Wert des Bezeichners und kann ein normaler Ausdruck sein (einschließlich benutzerdefinierter Funktionen), der bei der Erstellung der Konstante ausgewertet wird.<br>Eine außerhalb einer Subroutine oder Funktion definierte Konstante ist global und im gesamten Programm sichtbar. Eine innerhalb einer Subroutine oder Funktion definierte Konstante ist lokal für diese Routine und verdeckt eine globale Konstante mit demselben Namen. |
| CONTINUE | Setzen die Ausführung eines Programms fort, das durch eine END-Anweisung, einen Fehler oder STRG-C angehalten wurde. Das Programm wird mit der nächsten Anweisung nach dem vorherigen Haltepunkt neu gestartet.<br>Beachte, dass es nicht immer möglich ist, das Programm korrekt fortzusetzen – dies gilt insbesondere für komplexe Programme mit Grafiken, verschachtelten Schleifen und/oder verschachtelten Unterprogrammen und Funktionen. |
| CONTINUE DO<br>oder<br>CONTINUE FOR | Springe zum Ende einer DO/LOOP- oder einer FOR/NEXT-Schleife. Die Schleifenbedingung wird dann geprüft, und wenn sie noch gültig ist, wird die Schleife mit der nächsten Iteration fortgesetzt. |
| COPY fname1$ TO fname2$<br><br><br><br>COPY fname$ TO dirname$ | Kopiere eine Datei von „fname1$“ nach „fname2$“. Beides sind Zeichenfolgen.<br>Sowohl in „fname$“ als auch in „fname2$“ kann ein Verzeichnispfad verwendet werden. Wenn sich die Pfade unterscheiden, wird die in „fname$“ angegebene Datei unter dem angegebenen Dateinamen in den in „fname2$“ angegebenen Pfad kopiert. Die Dateinamen können die Laufwerksangabe enthalten, falls du auf ein nicht aktives Laufwerk kopierst oder von einem solchen (siehe den Befehl DRIVE)<br>Kopieren mit Platzhaltern. Das Massenkopieren wird ausgelöst, wenn fname$ ein '*'- oder ein '?'-Zeichen enthält. dirname$ muss ein gültiger Verzeichnisname sein und darf NICHT mit einem Schrägstrich enden. Wenn fname$ keinen Platzhalter enthält, wird nur die einzelne Datei kopiert.<br>Die Firmware verhindert das Kopieren einer Datei auf sich selbst. |
| CPU RESTART | Erzwingt einen Neustart der Prozessoren.<br>Dadurch werden alle Variablen gelöscht und alles zurückgesetzt (z. B. Timer, COM-Ports, I2C usw.), ähnlich wie beim Einschalten, jedoch ohne den Startbildschirm.<br>Wenn die Option AUTORUN gesetzt ist, wird das Programm am angegebenen Flash-Speicherort oder im Programmspeicher neu gestartet. |
| CPU SLEEP n | Versetzt die Prozessoren für „n“ Sekunden in den Ruhezustand. Beachte, dass die CPU keinen echten Energiesparmodus hat, sodass die Energieeinsparung begrenzt ist. |
| CSUB name [type [, type] …]<br>hex [[ hex[…]<br>hex [[ hex[…]<br>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 wie ein integrierter Befehl verwendet werden.<br>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 den Offset in Bytes vom Anfang des CSUB bis zum Einstiegspunkt der eingebetteten Routine (normalerweise die Funktion main()) angibt. Die folgenden „hex“-Wörter sind der kompilierte Binärcode für das Modul. Diese werden automatisch in MMBasic programmiert, wenn das Programm gespeichert wird. Jedes „Hex“-Wort muss genau acht Hexadezimalziffern umfassen, die die Bits eines 32-Bit-Worts darstellen, und durch ein oder mehrere Leerzeichen oder Zeilenumbrüche getrennt sein. Der Befehl muss mit einem entsprechenden END CSUB abgeschlossen werden. Fehler im Datenformat werden bei der Ausführung des Programms gemeldet. Während der Ausführung überspringt MMBasic alle CSUB-Befehle, sodass sie an beliebiger Stelle im Programm platziert werden können.<br>Der Typ jedes Parameters kann in der Definition angegeben werden. Zum Beispiel:<br>`CSUB MySub integer, integer, string`<br>Dies gibt an, dass es drei Parameter gibt, wobei die ersten beiden Ganzzahlen und der dritte eine Zeichenkette sind.<br>Hinweis:<br>• Es können bis zu zehn Argumente angegeben werden („arg1“, „arg2“ usw.).<br>• Wenn eine Variable oder ein Array als Argument angegeben wird, erhält die C-Routine einen Zeiger auf den der Variable oder dem Array zugewiesenen Speicher, und die C-Routine kann diesen Speicher ändern, um einen Wert an den Aufrufer zurückzugeben. Im Falle von Arrays sollten diese mit leeren Klammern übergeben werden, z. B. arg(). In der CSUB wird das Argument als Zeiger auf das erste Element des Arrays übergeben.<br>• Konstanten und Ausdrücke werden an die eingebettete C-Routine als Zeiger auf einen temporären Speicherbereich übergeben, der den Wert enthält. |
| DATA constant[,constant].. | Speichert numerische und Zeichenfolgenkonstanten, auf die mit READ zugegriffen werden kann.<br>Im Allgemeinen sollten Zeichenfolgenkonstanten in doppelte Anführungszeichen (") gesetzt werden. Eine Ausnahme bildet der Fall, dass die Zeichenfolge nur aus alphanumerischen Zeichen besteht, die keine MMBasic-Schlüsselwörter darstellen (wie THEN, WHILE usw.). In diesem Fall sind keine Anführungszeichen erforderlich.<br>Numerische Konstanten können auch Ausdrücke sein, wie z. B. 5 * 60. |
| DATE$ = "DD-MM-YY[YY]"<br>oder<br>DATE$ = "DD/MM/YY[YY]"<br>oder<br>DATE$ = „YYYY-MM-DD“<br>oder<br>DATE$ = „YYYY/MM/DD“ | Stelle das Datum der internen Uhr/des Kalenders ein.<br>DD, MM und YY sind Zahlen, zum Beispiel: DATE$ = "28-7-2014"<br>Mit OPTION RTC AUTO ENABLE startet die PicoMite-Firmware mit dem in der RTC programmierten DATE$.<br>Ohne OPTION RTC AUTO ENABLE startet die PicoMite-Firmware beim Einschalten mit dem Datum „01-01-2024“. |
| DEFINEFONT #Nbr<br>hex [[ hex[…]<br>hex [[ hex[…]<br>END DEFINEFONT | Damit definierst du eine eingebettete Schriftart, die neben den auf einem angeschlossenen LCD-Panel verwendeten integrierten Schriftarten genutzt oder diese ersetzen kann. Diese funktionieren genau wie die integrierten Schriftarten (d. h. sie werden mit dem FONT-Befehl ausgewählt oder im TEXT-Befehl angegeben). Im Ordner *„Embedded Fonts“* in der ZIP-Datei der PicoMite-Firmware-Distribution findest du eine Auswahl eingebetteter Schriftarten sowie eine vollständige Beschreibung, wie man sie erstellt. |

| | |
|---|---|
| | „#Nbr“ ist die Referenznummer der Schriftart (von 1 bis 16). Sie kann mit einer integrierten Schriftart übereinstimmen; in diesem Fall ersetzt sie die integrierte Schriftart. Jeder „Hex“-Wert muss genau acht Hexadezimalziffern umfassen und durch Leerzeichen oder Zeilenumbrüche vom nächsten getrennt sein.<br>• Es können mehrere Zeilen mit „hex“-Wörtern verwendet werden, wobei der Befehl durch ein entsprechendes END DEFINEFONT beendet wird.<br>• In einem Programm können mehrere eingebettete Schriftarten verwendet werden, wobei jede eine andere Schriftart mit einer anderen Schriftartnummer definiert.<br>• Während der Ausführung überspringt MMBasic alle DEFINEFONT-Befehle, sodass sie an beliebiger Stelle im Programm platziert werden können.<br>• Etwaige Fehler im Datenformat werden beim Speichern des Programms gemeldet. |
| DEVICE BITSTREAM | Siehe Befehl BITSTREAM |
| DEVICE CAMERA | Siehe Befehl CAMERA |
| DEVICE GAMEPAD | Siehe Befehl GAMEPAD |
| DEVICE HUMID | Siehe Befehl HUMID |
| DEVICE KEYPAD | Siehe Befehl KEYPAD |
| DEVICE MOUSE | Siehe Befehl MOUSE |
| DEVICE LCD | Siehe Befehl LCD |
| DEVICE SERIALTX pinno, baudrate, ostring$ | Gibt „ostring$“ als seriellen Datenstrom über „pinno“ aus. „baudrate“ kann zwischen 110 und 230400 liegen (bei 230400 muss die CPU möglicherweise übertaktet werden).<br>Beachte, dass das Programm während der Übertragung anhält und Interrupts ignoriert. |
| DEVICE SERIALRX pinno, baudrate, istring$, timeout_in_ms, status% [,nbr] [,terminators$] | Liest serielle Daten über „pinno“ ein. „baudrate“ kann zwischen 110 und 230400 liegen (bei 230400 muss die CPU möglicherweise übertaktet werden).<br>„status%“ gibt Folgendes zurück:<br>-1 = Timeout (Hinweis: Verwende LEN(istring$), um die Anzahl der empfangenen Zeichen zu sehen)<br>2 = Anzahl der angeforderten Zeichen erfüllt<br>3 = Beendigungszeichen erreicht<br>„nbr“ gibt die Anzahl der Zeichen an, die empfangen werden müssen, bevor der Befehl zurückkehrt. „terminators$“ gibt ein oder mehrere einzelne Zeichen an, die zum Beenden des Empfangs verwendet werden können.<br>Das Programm wird angehalten und Interrupts werden ignoriert, während dieser Befehl ausgeführt wird. |
| DEVICE WII | Siehe WII Befehl |
| DEVICE WS2812 | Siehe WS2812 Befehl |
| DIM [type] decl [,decl]..<br>where 'decl' is:<br>var [length] [type] [init] | Deklariert eine oder mehrere Variablen (d. h. teilt dem Interpreter den Variablennamen und dessen Eigenschaften mit). |

| | |
|---|---|
| 'var' is a variable name with optional dimensions<br>'length' is used to set the maximum size of the string to 'n' as in LENGTH n<br>'type' is one of FLOAT or INTEGER or STRING (the type can be prefixed by the keyword AS - as in AS FLOAT)<br>'init' is the value to initialise the variable and consists of:<br>= <expression><br>For a simple variable one expression is used, for an array a list of comma separated expressions surrounded by brackets is used.<br><br>Examples:<br>DIM nbr(50)<br>DIM INTEGER nbr(50)<br>DIM name AS STRING<br>DIM a, b$, nbr(100), strn$(20)<br>DIM a(5,5,5), b(1000)<br>DIM strn$(200) LENGTH 20<br>DIM STRING strn(200) LENGTH 20<br>DIM a = 1234, b = 345<br>DIM STRING strn = "text"<br>DIM x%(3) = (11, 22, 33, 44) | Wenn OPTION EXPLICIT verwendet wird (wie empfohlen), sind die Befehle DIM, LOCAL oder STATIC die einzige Möglichkeit, eine Variable zu erstellen. Wird diese Option nicht verwendet, ist die Verwendung des Befehls DIM optional, und wenn er nicht verwendet wird, wird die Variable automatisch erstellt, sobald sie zum ersten Mal referenziert wird.<br>Der Typ der Variablen (d. h. String, Float oder Integer) kann auf drei Arten angegeben werden:<br>Durch Verwendung eines Typ-Suffixes (d. h. !, % oder $ für Float, Integer oder String). Zum Beispiel:<br>`DIM nbr%, amount!, name$`<br>Durch Verwendung eines der Schlüsselwörter FLOAT, INTEGER oder STRING unmittelbar nach dem Befehl DIM und vor der Auflistung der Variablen. Der angegebene Typ gilt dann für alle aufgeführten Variablen (d. h. er muss nicht wiederholt werden). Zum Beispiel:<br>`DIM STRING Vorname, Nachname, Stadt`<br>Durch die Verwendung der Microsoft-Konvention, bei der das Schlüsselwort „AS“ und das Typ-Schlüsselwort (d. h. FLOAT, INTEGER oder STRING) nach jeder Variablen steht. Wenn du diese Methode verwendest, muss der Typ für jede Variable angegeben werden und kann von Variable zu Variable geändert werden.<br>Zum Beispiel:<br>`DIM amount AS FLOAT, name AS STRING`<br>Gleitkomma- oder Ganzzahlvariablen werden bei der Erstellung auf Null gesetzt, und Zeichenfolgen werden auf eine leere Zeichenfolge (d. h. „“) gesetzt. Du kannst den Wert der Variablen initialisieren, indem du ein Gleichheitszeichen (=) und einen Ausdruck nach der Variablendefinition verwendest. Zum Beispiel:<br>`DIM STRING city = "Perth", house = "Brick"`<br>Der Initialisierungswert kann ein Ausdruck sein (einschließlich anderer Variablen) und wird bei der Ausführung des DIM-Befehls ausgewertet. Weitere Beispiele zur Syntax findest du im Kapitel *„Variablen definieren und verwenden*“.<br>Der Befehl „DIM“ dient nicht nur zur Deklaration einfacher Variablen, sondern auch zur Deklaration von Array-Variablen (d. h. einer indizierten Variablen mit mehreren Dimensionen). Nach dem Variablennamen werden die Dimensionen durch eine Liste von Zahlen angegeben, die durch Kommas getrennt und in Klammern gesetzt sind. Zum Beispiel:<br>`DIM array(10, 20)`<br>Jede Zahl gibt den Indexbereich in jeder Dimension an. Normalerweise beginnt die Indizierung jeder Dimension bei 0, aber mit dem Befehl OPTION BASE kannst du dies auf 1 ändern.<br>Das obige Beispiel definiert 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 8 Byte benötigt, werden insgesamt 1848 Byte Speicher zugewiesen.<br>Zeichenfolgen reservieren standardmäßig 255 Byte (d. h. Zeichen) Speicherplatz pro Element, was bei der Definition von Zeichenfolgen-Arrays schnell zu Speicherengpässen führen kann. In diesem Fall kannst du das Schlüsselwort LENGTH verwenden, um die Speichermenge festzulegen, die jedem Element zugewiesen werden soll, und damit die maximale Länge der Zeichenfolge, die gespeichert werden kann. Diese Zuweisung („n“) kann zwischen 1 und 255 Zeichen liegen.<br>Beispiel: `DIM STRING s(5, 10)` deklariert ein String-Array mit 66 Elementen, das 16.896 Byte Speicherplatz beansprucht, während: |

| | |
|---|---|
| | `DIM STRING s(5, 10) LENGTH 20`<br>nur 1.386 Byte Speicherplatz beansprucht. Beachte, dass die für jedes Element zugewiesene Speichermenge n + 1 beträgt, da das zusätzliche Byte dazu dient, die tatsächliche Länge der in jedem Element gespeicherten Zeichenkette zu verfolgen.<br>Wenn einem Element des Arrays eine Zeichenkette zugewiesen wird, die länger als 'n' ist, wird ein Fehler ausgegeben. Ansonsten verhalten sich mit dem Schlüsselwort LENGTH erstellte Zeichenfolgen-Arrays genau wie andere Zeichenfolgen-Arrays. Dieses Schlüsselwort kann auch mit Zeichenfolgenvariablen verwendet werden, die keine Arrays sind, spart jedoch keinen Speicherplatz.<br>Im obigen Beispiel kannst du auch die Microsoft-Syntax verwenden, bei der der Typ nach dem Längenqualifizierer angegeben wird. Zum Beispiel:<br>`DIM s(5, 10) LENGTH 20 AS STRING`<br>Der Längenparameter kann für einfache (nicht als Array angelegte) Zeichenfolgen mit den folgenden Einschränkungen verwendet werden: Beim RP2040 wird die Länge ignoriert, wenn sie größer als 9 ist, und es werden standardmäßig 255 Byte zugewiesen. Beim RP2350 wird die Länge ignoriert und standardmäßig 255 Byte zugewiesen, wenn sie größer als 15 ist. Liegt die Länge unterhalb oder gleich der Grenze, ist die Zeichenkette im Variablenheader enthalten und es werden 256 Byte eingespart. Dies ist wahrscheinlich besonders nützlich für kurze konstante Zeichenketten<br>Arrays können auch bei ihrer Deklaration initialisiert werden, indem am Ende der Deklaration ein Gleichheitszeichen (=) gefolgt von einer Liste von Werten in Klammern hinzugefügt wird. Zum Beispiel:<br>`DIM INTEGER nbr(4) = (22, 44, 55, 66, 88)`<br>oder `DIM s$(3) = ("foo", "boo", "doo", "zoo")`<br>Beachte, dass die Anzahl der Initialisierungswerte mit der Anzahl der Elemente im Array übereinstimmen muss, einschließlich des durch OPTION BASE festgelegten Basiswerts. Wenn ein mehrdimensionales Array initialisiert wird, wird zuerst die erste Dimension initialisiert, gefolgt von der zweiten usw.<br>Beachte außerdem, dass die Initialisierungswerte nach dem LENGTH-Qualifizierer (falls verwendet) und nach der Typdeklaration (falls verwendet) stehen müssen. |
| DO<br><statements><br>LOOP | Diese Struktur führt eine Endlosschleife aus; 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 Subroutine) aus der Schleife herausgeführt werden. |
| DO WHILE expression<br>< statements ><br>LOOP | Loopt, solange der „Ausdruck“ (expression) wahr ist (dies entspricht der älteren WHILE-WEND-Schleife). Wenn der Ausdruck zu Beginn falsch ist, werden die Anweisungen in der Schleife nicht ausgeführt, nicht einmal. |
| DO<br>< statements ><br>LOOP UNTIL expression | Wiederholt sich, bis der Ausdruck nach UNTIL wahr ist. Da die Prüfung am Ende der Schleife erfolgt, werden die Anweisungen innerhalb der Schleife mindestens einmal ausgeführt, selbst wenn der Ausdruck wahr ist. |
| DO<br>< statements ><br>LOOP WHILE expression | Wiederholt sich, bis der Ausdruck nach WHILE falsch ist. Da die Prüfung am Ende der Schleife erfolgt, werden die Anweisungen innerhalb der Schleife mindestens einmal ausgeführt, auch wenn der Ausdruck falsch ist. |
| DRAW3D | NICHT IN DER WEBSITE-VERSION VERFÜGBAR<br>Die 3D-Engine enthält Befehle zur Bearbeitung von 3D-Bildern, darunter das Einstellen der Kamera, das Erstellen, Ausblenden, Drehen usw.<br>Eine vollständige Beschreibung findest du im Dokument *„3D_Graphics_User_Manual.*pdf“ im PicoMite-Firmware-Download. |

| | |
|---|---|
| DRIVE drive$ | Setzt das aktive Laufwerk auf „drive$“. „drive$“ kann „A:“ oder „B:“ sein, wobei A das Flash-Laufwerk und B die SD-Karte ist, sofern konfiguriert |
| EDIT<br>oder<br>EDIT fname$<br>oder<br>EDIT FILE fname$ | Rufe den Vollbild-Editor auf.<br>Wenn ein Dateiname angegeben wird, lädt der Editor die Datei von der aktuellen Diskette (A: oder B:) zur Bearbeitung und speichert sie beim Beenden mit F1 oder F2 auf der Diskette. Wenn die Datei nicht existiert, wird sie beim Beenden erstellt. Das aktuell im Flash-Speicher gespeicherte Programm bleibt davon unberührt. Beim Bearbeiten einer bestehenden Datei wird beim Beenden zusätzlich eine Sicherungskopie mit der Endung .bak erstellt.<br>Wenn fname$ eine andere Erweiterung als .bas enthält, wird die Farbcodierung während der Bearbeitung vorübergehend deaktiviert.<br>Wenn keine Erweiterung angegeben ist, geht die Firmware von .bas aus<br>Das Bearbeiten einer Datei von der Festplatte ermöglicht es, Nicht-Basic-Dateien wie HTML- oder Sprite-Dateien zu bearbeiten, ohne dass diese während des Tokenisierungsprozesses, der beim Speichern im Flash-Speicher stattfindet, beschädigt werden.<br>EDIT und EDIT fname$ können nur an der Befehlszeile aufgerufen werden.<br>Wenn du eine Datei in einem Programm bearbeiten möchtest, kannst du den Befehl EDIT FILE fname$ verwenden. Der Befehl muss im Hauptprogramm verwendet werden und darf nicht aus einer Subroutine heraus aufgerufen werden.<br>EDIT FILE fname$ unterscheidet sich von EDIT fname$ dadurch, dass es automatisch den gesamten Variablenbereich auf dem Laufwerk A: speichert und beim Beenden wiederherstellt. Der Befehl schlägt fehl, wenn auf dem Laufwerk A: nicht genügend freier Speicherplatz vorhanden ist. Bei einem RP2350 mit PSRAM wird der Variablenbereich in einem reservierten Bereich im PSRAM gespeichert und das Laufwerk A: wird nicht verwendet.<br>Weitere Informationen zur Verwendung des Editors findest du im Kapitel *„Vollbild-Editor*“. |
| ELSE | Führt eine optionale Standardbedingung in einer mehrzeiligen IF-Anweisung ein.<br>Weitere Details findest du bei der mehrzeiligen IF-Anweisung. |
| ELSEIF expression THEN<br>oder<br>ELSE IF expression THEN | Fügt eine optionale sekundäre Bedingung in eine mehrzeilige IF-Anweisung ein.<br>Weitere Details findest du bei der mehrzeiligen IF-Anweisung. |
| END [noend]<br>oder<br>END cmd$ | Beendet das laufende Programm und kehrt zur Eingabeaufforderung zurück. Wenn im Programm eine Subroutine namens MM.END vorhanden ist, wird diese ausgeführt, sobald das Programm mit einem tatsächlichen oder impliziten END-Befehl endet. Sie wird nicht ausgeführt, wenn das Programm mit dem Unterbrechungszeichen (d. h. Strg-C) endet.<br>Der optionale Parameter „noend“ kann verwendet werden, um die Ausführung der Subroutine MM.END zu blockieren, z. B. „END noend“<br>Wenn „cmd$“ angegeben ist, wird es nach Beendigung des Programms so ausgeführt, als ob es an der Befehlszeile stünde. Hinweis: Wenn „END cmd$“ verwendet wird, aber eine Subroutine MM.END existiert, wird diese ausgeführt und cmd$ ignoriert. |
| END CSUB | Markiert das Ende einer C-Subroutine. Siehe den Befehl CSUB.<br>Jedes CSUB muss genau eine entsprechende END CSUB-Anweisung haben. |

| | |
|---|---|
| END FUNCTION | Markiert das Ende einer benutzerdefinierten Funktion. Siehe den Befehl FUNCTION.<br>Jede Funktion muss genau eine passende END FUNCTION-Anweisung haben. Verwende EXIT FUNCTION, wenn du aus einer Funktion heraus zurückkehren möchtest. |
| ENDIF<br>oder<br>END IF | Beendet eine mehrzeilige IF-Anweisung.<br>Weitere Details findest du bei der mehrzeiligen IF-Anweisung. |
| END SUB | Markiert das Ende einer benutzerdefinierten Subroutine. Siehe den SUB-Befehl.<br>Jede Subroutine muss genau eine passende END SUB-Anweisung haben. Verwende EXIT SUB, wenn du aus einer Subroutine heraus zurückkehren möchtest. |
| END TYPE | Markiert das Ende einer benutzerdefinierten Struktur. Siehe den Befehl TYPE. |
| ERASE variable [,variable].. | Löscht globale Variablen und gibt den ihnen zugewiesenen Speicher frei. Dies funktioniert sowohl bei Array-Variablen als auch bei normalen (Nicht-Array-)Variablen. Arrays können durch leere Klammern (z. B. `dat()`) oder einfach durch Angabe des Variablennamens (z. B. `dat`) angegeben werden.<br>Verwende CLEAR, um alle Variablen gleichzeitig zu löschen (einschließlich Arrays). |
| ERROR [error_msg$] | Löst einen Fehler aus und beendet das Programm. Dies wird normalerweise beim Debuggen oder zum Abfangen von Ereignissen verwendet, die nicht auftreten sollten.<br>'error_msg$' ist optional und ist die Meldung, die auf der Konsole angezeigt werden soll. |
| EXECUTE command$ | Dies führt den Basic-Befehl „command$“ aus. Die Verwendung sollte auf Basic-Befehle beschränkt werden, die sequenziell ausgeführt werden; beispielsweise funktioniert die GOTO-Anweisung nicht richtig<br>Zu den getesteten und funktionierenden Befehlen gehören GOSUB, Unterprogrammaufrufe und andere einfache Anweisungen (wie PRINT und einfache Zuweisungen).<br>Mehrere Anweisungen, die durch : getrennt sind, sind nicht erlaubt und führen zu einem Fehler<br>Der Befehl setzt vor der Ausführung des angeforderten Befehls einen internen Watchdog. Wenn die Steuerung nicht zum Befehl zurückkehrt, wie beispielsweise bei einer GOTO-Anweisung, läuft der Timer ab. In diesem Fall erhältst du die Meldung „Befehls-Timeout“.<br>Du kannst EXECUTE nicht aus Code heraus aufrufen, der bereits mit EXECUTE ausgeführt wurde.<br>RUN ist ein Sonderfall und bricht den Timer ab, sodass du den Befehl bei Bedarf zum Verketten von Programmen verwenden kannst. |
| EXIT DO<br>EXIT FOR<br>EXIT FUNCTION<br>EXIT SUB | EXIT DO ermöglicht einen vorzeitigen Abbruch einer DO..LOOP<br>EXIT FOR ermöglicht einen vorzeitigen Abbruch einer FOR..NEXT-Schleife.<br>EXIT FUNCTION ermöglicht einen vorzeitigen Abbruch einer definierten Funktion.<br>EXIT SUB ermöglicht einen vorzeitigen Abbruch einer definierten Subroutine.<br>Der alte Standard von EXIT allein (Beenden einer do-Schleife) wird ebenfalls unterstützt. |

| | |
|---|---|
| FILES [fspec$] [,sort] | Listet Dateien in beliebigen Verzeichnissen auf dem Standard-Flash-Dateisystem oder der SD-Karte auf.<br>'fspec$' (falls angegeben) kann einen Pfad und Suchplatzhalter im Dateinamen enthalten. Fragezeichen (?) stehen für ein beliebiges Zeichen und ein Sternchen (*) für eine beliebige Anzahl von Zeichen. Wenn nichts angegeben wird, werden alle Dateien aufgelistet.<br>Zum Beispiel:<br>* Alle Einträge finden<br>*.TXT Alle Einträge mit der Erweiterung TXT suchen<br>E*.* Alle Einträge finden, die mit E beginnen<br>X?X.* Alle dreistelligen Dateinamen finden, die mit X beginnen und enden<br>mydir/* Alle Einträge im Verzeichnis mydir suchen<br>Hinweis: Wenn du Platzhalter im Pfadnamen verwendest, führt das zu einem Fehler<br>„sort“ legt die Sortierreihenfolge wie folgt fest:<br>Größe nach aufsteigender Größe<br>Zeit in absteigender Reihenfolge<br>name nach Dateinamen (Standard, wenn nicht anders angegeben)<br>Typ nach Dateiendung |
| FILL x, y, fillcolour [,bordercolour] | Füllt einen Bereich der Anzeige mit einer Farbe.<br>Wenn der Befehl ohne das optionale „bordercolour“ verwendet wird, liest er die Farbe an der Position „x“, „y“ in der Anzeige und füllt dann den Bereich ab diesem Punkt, an dem die aktuelle Farbe mit der neuen Farbe „fillcolour“ übereinstimmt.<br>Wenn der optionale Parameter „bordercolour“ angegeben ist, ersetzt „fillcolour“ die dort bereits vorhandene Farbe, bis die angegebene „bordercolour“-Farbe erreicht wird.<br>Beachte, dass dies auf TFT-Displays mit ungepufferten Treibern langsam sein wird. |
| FLAG(n%)= value | Setzt ein Bit in einem System-Flag-Register. N% kann zwischen 0 und 63 liegen (d. h. es stehen 64 Flag-Bits zur Verfügung). Der Wert (value) kann 0 oder 1 sein.<br>Siehe auch den Befehl FLAGS, die Funktion FLAG und MM.FLAGS |
| FLAGS= value% | Setzt alle Bits im System-Flag-Register auf den angegebenen Wert.<br>Siehe auch den Befehl FLAG, die Funktion FLAGS und MM.FLAGS |
| FLASH | Verwaltet die Speicherung von Programmen im Flash-Speicher. Bis zu drei Programme können im Flash-Speicher gespeichert und bei Bedarf abgerufen werden. Beachte, dass diese gespeicherten Programme bei einem Firmware-Upgrade gelöscht werden.<br>Einer dieser Flash-Speicherplätze kann mit dem Befehl OPTION AUTORUN n automatisch geladen und ausgeführt werden, sobald die Stromversorgung eingeschaltet wird. Im Folgenden steht „n“ für eine Zahl zwischen 1 und 3. |
| FLASH LIST | Zeigt eine Liste aller Flash-Speicherplätze einschließlich der ersten Zeile des Programms an. |
| FLASH LIST n [,all] | Listet das in Speicherplatz n gespeicherte Programm auf. Verwende ALL, um die Liste ohne Seitenumbrüche anzuzeigen. |
| FLASH ERASE n | Löscht einen Flash-Programmspeicherplatz. |
| FLASH ERASE ALL | Löscht alle Flash-Programmspeicherplätze. |
| FLASH SAVE n | Speichere das aktuelle Programm an dem angegebenen Flash-Speicherplatz. |

| | |
|---|---|
| FLASH LOAD n | Lade ein Programm vom angegebenen Flash-Speicherplatz in den Programmspeicher. |
| FLASH RUN n | Führt das Programm am Flash-Speicherplatz n aus, löscht alle Variablen. Ändert den Programmspeicher nicht. |
| FLASH CHAIN n | Führt das Programm am Flash-Speicherplatz n aus, wobei alle Variablen erhalten bleiben (ermöglicht ein Programm, das viel größer ist als der Programmspeicher). Ändert den Programmspeicher nicht. Hinweis: Wenn das verkettete Programm den Befehl READ verwendet, muss es vor dem ersten Lesevorgang RESTORE aufrufen. |
| FLASH OVERWRITE n | Lösche einen Flash-Programmspeicherplatz und speichere dann das aktuelle Programm an dem angegebenen Flash-Speicherplatz. |
| FLASH DISK LOAD n, fname$ [,O[VERWRITE]] | Lädt den Inhalt der Datei fname$ als Binärbild in den Flash-Speicherplatz n. Die Datei kann mit LIBRARY DISK SAVE erstellt werden. Außerdem kann jede extern erstellte Datei mit Daten, die von einem Programm benötigt werden, geladen und mit Befehlen wie PEEK und MEMORY COPY unter Verwendung der Adresse des Flash-Speicherplatzes abgerufen werden.<br>Wenn der optionale Parameter OVERWRITE (oder O) angegeben wird, wird der Inhalt des Flash-Slots überschrieben, ohne dass ein Fehler ausgelöst wird. |
| FLASH LOAD IMAGE n, filename$ [,O/OVERWRITE] | Dieser Befehl lädt eine angegebene BMP-Datei im RGB121-Format in den Flash-Speicherplatz.<br>Der Flash-Slot sollte zuvor gelöscht worden sein; andernfalls erzwingt die Angabe des optionalen Parameters O oder OVERWRITE ein Löschen.<br>Beachte, dass die Firmware die ersten beiden Wörter im Flash-Speicherplatz auf die Breite und Höhe des gespeicherten Bildes setzt.<br>Die Bildgröße muss nicht mit den Abmessungen der aktuellen Anzeige übereinstimmen. |
| FLUSH [#]fnbr | Bewirkt, dass alle gepufferten Schreibvorgänge in eine zuvor mit der Dateinummer „#fnbr“ geöffnete Datei auf die Festplatte geschrieben werden. Das # ist optional. Die Verwendung dieses Befehls stellt sicher, dass keine Daten verloren gehen, falls nach einem Schreibbefehl ein Stromausfall auftritt. |
| FONT [#]font-number, scaling | Damit wird die Standardschriftart für die Anzeige von Text auf einem LCD-Bildschirm oder der Videoausgabe festgelegt.<br>Schriftarten werden als Zahl angegeben. Zum Beispiel #2 (das # ist optional). Siehe das Kapitel *„Grafikbefehle und -funktionen“* für Details zu den verfügbaren Schriftarten.<br>„Skalierung“ kann zwischen 1 und 15 liegen und vergrößert die Pixel, wodurch das angezeigte Zeichen entsprechend breiter und höher wird. Bei einer Skalierung von 2 werden beispielsweise Höhe und Breite verdoppelt. |
| FOR counter = start TO finish [STEP increment] | Startet eine FOR-NEXT-Schleife, wobei der 'Zähler' zunächst auf 'Start' gesetzt ist und in Schritten von 'Inkrement' (Standard ist 1) erhöht wird, bis der 'Zähler' größer als 'Ende' ist.<br>Der Wert für „increment“ kann eine ganze Zahl oder eine Gleitkommazahl sein. Beachte, dass die Verwendung einer Gleitkommazahl für „increment“ zu Rundungsfehlern in „counter“ führen kann, wodurch die Schleife vorzeitig oder verspätet beendet werden könnte.<br>'increment' kann negativ sein; in diesem Fall sollte 'finish' kleiner als 'start' sein, und die Schleife zählt rückwärts.<br>Siehe auch den Befehl NEXT. |

| | |
|---|---|
| FRAME | Nur RP2350<br>Der Befehl „frame“ unterstützt den Programmierer beim Erstellen textbasierter Benutzeroberflächen – siehe das Handbuch „FRAME_User_Manual.pdf“ |
| FRAMEBUFFER | KEINE HDMI- UND VGA-VERSIONEN<br>Mit dem Framebuffer-Befehl kannst du einen Teil des variablen Speichers entweder einem Framebuffer, einer zweiten Anzeigeebene oder beiden zuweisen und diese dann auf interessante Weise nutzen, um Tearing-Artefakte zu vermeiden und/oder Grafikelemente über die Hintergrundanzeige abzuspielen. |
| FRAMEBUFFER CREATE | Erstellt einen Framebuffer „F“ mit einem RGB121-Farbraum und einer Auflösung, die dem konfigurierten SPI-Farbdisplay entspricht |
| FRAMEBUFFER LAYER | Erstellt einen Framebuffer „L“ mit einem RGB121-Farbraum und einer Auflösung, die dem konfigurierten SPI-Farbdisplay entspricht |
| FRAMEBUFFER WRITE where/where$ | Legt das Ziel für nachfolgende Grafikbefehle fest.<br>„where“ kann N, F oder L sein, wobei N das eigentliche Display ist. Es kann eine Zeichenfolgenvariable oder ein Literal verwendet werden |
| FRAMEBUFFER CLOSE [which] | Schließt einen Framebuffer und gibt den Speicher frei. Der optionale Parameter „which“ kann F oder L sein. Wenn er weggelassen wird, werden beide geschlossen. |
| FRAMEBUFFER COPY from, to [,b] | Führt eine hochoptimierte Vollbildkopie von einem Framebuffer in einen anderen durch.<br>„from“ und „to“ können N, F oder L sein, wobei N das physische Display ist. Du kannst nur von N auf Displays kopieren, die BLIT und transparenten Text unterstützen.<br>Die Firmware komprimiert oder erweitert die RGB-Auflösung automatisch beim Kopieren zwischen nicht kompatiblen Framebuffern.<br>Natürlich gehen beim Kopieren von RGB565 nach RGB121 Informationen verloren, aber für viele Anwendungen (z. B. Spiele) sind 16 Farbstufen mehr als ausreichend.<br>Beim Kopieren auf ein LCD-Display kann der optionale Parameter „b“ verwendet werden (FRAMEBUFFER COPY F/L, N, B). Dies weist die Firmware an, den Kopiervorgang mit der zweiten CPU im Raspberry Pi Pico durchzuführen, und die Steuerung kehrt sofort zum Basic-Programm zurück |
| FRAMEBUFFER WAIT | Hält die Verarbeitung an, bis das LCD-Display in die Bildausblendung wechselt. Implementiert für ILI9341-, ST7789_320- und ILI9488-Displays. Wird verwendet, um Artefakte beim Schreiben auf den Bildschirm zu reduzieren |
| FRAMEBUFFER MERGE [colour] [,mode] [,updaterate] | Kopiert den Inhalt des Layer-Puffers und des Framebuffers auf das LCD-Display, wobei alle Pixel einer bestimmten Farbe ausgelassen werden.<br>Voraussetzungen für den Befehl<br>sind, dass sowohl FRAMEBUFFER als auch LAYERBUFFER angelegt sind<br>FRAMEBUFFER MERGE – schreibt den Inhalt des Framebuffers auf das physische Display und überschreibt dabei alle Pixel im Framebuffer, die im Layerbuffer gesetzt sind (nicht Null)<br>FRAMEBUFFER MERGE col – schreibt den Inhalt des Framebuffers auf das physische Display und überschreibt dabei alle Pixel im Framebuffer, die im Layerbuffer nicht auf die transparente Farbe „col“ gesetzt sind. Die Farbe wird als Zahl zwischen 0 und 15 angegeben, die Folgendes darstellt: |

| | |
|---|---|
| | 0:BLACK,1:BLUE,2:MYRTLE,3:COBALT,4:MIDGREEN,5:CERULEAN,6:GREEN,7:CYAN,8:RED,9:MAGENTA,10:RUST,11:FUCHSIA,12:BROWN,13:LILAC,14:YELLOW,15:WHITE<br>FRAMEBUFFER MERGE col,B – wie oben, nur dass die Übertragung auf den physischen Bildschirm auf der zweiten CPU erfolgt und die Steuerung sofort an Basic zurückgegeben wird<br>FRAMEBUFFER MERGE col,R [,updaterate] – stellt die zweite CPU so ein, dass sie das physische Display kontinuierlich mit der Zusammenführung der beiden Puffer aktualisiert.<br>Setzt automatisch FRAMEBUFFER WRITE F, falls F oder L nicht bereits gesetzt sind. Standardmäßig wird der Bildschirm so schnell wie möglich aktualisiert (bei 200 MHz aktualisiert sich ein ILI9341 im SPI-Modus etwa 13 Mal pro Sekunde, im 8-Bit-Parallelmodus erreicht der ILI9341 27 FPS)<br>Wenn „updaterate“ gesetzt ist, wird der Bildschirm mit der in Millisekunden angegebenen Rate aktualisiert (es sei denn, diese liegt unter der schnellsten auf dem Display erreichbaren Rate)<br>Hinweis: FRAMEBUFFER WRITE kann nicht auf N gesetzt werden, solange die kontinuierliche zusammengeführte Aktualisierung aktiv ist.<br>FRAMEBUFFER MERGE col,A – bricht die kontinuierlichen Aktualisierungen ab<br>Außerdem wird die automatische Aktualisierung durch das Löschen des Layerbuf oder Framebuffers, durch Strg-C oder durch END ebenfalls abgebrochen. |
| FRAMEBUFFER SYNC | Wartet darauf, dass die letzte Hintergrundzusammenführung oder der letzte Hintergrundkopiervorgang auf der zweiten CPU abgeschlossen ist, um ein Zeichnen ohne Bildreißen zu ermöglichen |
| FRAMEBUFFER | <u>NUR HDMI- UND VGA-VERSIONEN</u><br>Mit dem Framebuffer-Befehl kannst du einen Teil des variablen Speichers für Framebuffer, Layer-Buffer oder beides reservieren und diese dann auf interessante Weise nutzen, um Tearing-Artefakte zu vermeiden und/oder Grafikelemente über die Hintergrundanzeige zu legen. |
| FRAMEBUFFER CREATE | Erstellt einen Framebuffer „F“ mit einem Farbraum und einer Auflösung, die dem aktuellen Anzeigemodus entsprechen |
| FRAMEBUFFER CREATE 2 | Nur RP2350: Erstellt einen zweiten Framebuffer „2“ mit einem Farbraum und einer Auflösung, die dem aktuellen Anzeigemodus entsprechen |
| FRAMEBUFFER LAYER [colour] | Erstellt einen Layer-Puffer „L“ mit einem Farbraum und einer Auflösung, die dem aktuellen Anzeigemodus entsprechen. Der optionale Parameter „colour“ wird als Zahl zwischen 0 und 15 (Modi 2 und 3), als RGB888-Farbe (Modus 4) oder als Wert zwischen 0 und 255 (Modus 5) angegeben und legt eine Farbe fest, die ignoriert wird, wenn die Ebene auf die Anzeige angewendet wird. In Anzeigemodi, in denen die automatische Anwendung von Ebenen nicht unterstützt wird, fungiert ein Ebenenpuffer als weiterer Framebuffer. |
| FRAMEBUFFER LAYER TOP [colour] | Nur RP2350: Erstellt einen zweiten Layer-Puffer „T“ mit einem Farbraum und einer Auflösung, die dem aktuellen Anzeigemodus entsprechen. Der optionale Parameter „colour“ wird als Zahl zwischen 0 und 15 (Modi 2 und 3) oder 0 und 255 (Modus 5) angegeben und legt eine Farbe fest, die ignoriert wird, wenn die Ebene auf das Display angewendet wird. In Anzeigemodi, in denen die automatische Anwendung der$^{zweiten}$Ebene nicht unterstützt wird, fungiert dieser als weiterer Framebuffer. |

| | |
|---|---|
| FRAMEBUFFER WRITE where/where$ | Legt das Ziel für nachfolgende Grafikbefehle fest.<br>„where“ kann N, F, 2, T oder L sein, wobei N die eigentliche Anzeige ist. Es kann eine Zeichenfolgenvariable verwendet werden |
| FRAMEBUFFER CLOSE [which] | Schließt einen Framebuffer und gibt den Speicher frei. Der optionale Parameter „which“ kann F, 2, T oder L sein. Wenn er weggelassen wird, werden alle geschlossen. |
| FRAMEBUFFER COPY from, to [,b] | Führt eine hochoptimierte Vollbildkopie von einem Framebuffer in einen anderen durch. „from“ und „to“ können N, F, 2, T oder L sein, wobei N die physische Anzeige ist.<br>Wenn der optionale Parameter „b“ angegeben wird, wird die Verarbeitung angehalten, bis der Monitor in die Bildausblendung wechselt. |
| FRAMEBUFFER WAIT | Hält die Verarbeitung an, bis die nächste Bildausblendung beginnt |
| FUNCTION xxx (arg1 [,arg2, …]) [AS <type>}<br><statements><br><statements><br>xxx = <return value><br>END FUNCTION | Definiert eine aufrufbare Funktion. Das entspricht dem Hinzufügen einer neuen Funktion zu MMBasic, während dein Programm läuft.<br>'xxx' ist der Funktionsname und muss den Namenskonventionen für Variablen entsprechen. Der Typ der Funktion kann durch ein Typ-Suffix (z. B. xxx$) oder durch Angabe des Typs mit AS <Typ> am Ende der Funktionsdefinition festgelegt werden. Zum Beispiel:<br>`FUNCTION xxx (arg1, arg2) AS STRING`<br>'arg1', 'arg2' usw. sind die Argumente oder Parameter der Funktion (die Klammern sind immer erforderlich, auch wenn keine Argumente vorhanden sind). Ein Array wird durch „ “ unter Verwendung leerer Klammern angegeben, d. h. arg3(). Der Typ des Arguments kann durch einen Typ-Suffix (z. B. arg1$) oder durch die Angabe des Typs mit „`AS <Typ>“ (z. B. arg1 AS STRING)` festgelegt werden.<br>Das Argument kann auch eine andere definierte Funktion oder dieselbe Funktion sein, wenn Rekursion verwendet werden soll (der Rekursionsstapel ist begrenzt).<br>Um den Rückgabewert der Funktion festzulegen, weist du dem Namen der Funktion den Wert zu. Zum Beispiel:<br>`FUNCTION SQUARE(a)`<br>`  SQUARE = a * a`<br>`END FUNCTION`<br>Jede Definition muss eine END FUNCTION-Anweisung enthalten. Wenn diese erreicht wird, gibt die Funktion ihren Wert an den Ausdruck zurück, von dem aus sie aufgerufen wurde. Der Befehl EXIT FUNCTION kann für einen vorzeitigen Abbruch verwendet werden.<br>Du verwendest die Funktion, indem du ihren Namen und ihre Argumente in einem Programm verwendest, genau wie bei einer normalen MMBasic-Funktion. Zum Beispiel:<br>`PRINT SQUARE(56.8)`<br>Wenn die Funktion aufgerufen wird, wird jedes Argument im Aufrufer dem entsprechenden Argument in der Funktionsdefinition zugeordnet. Diese Argumente sind nur innerhalb der Funktion verfügbar.<br>Funktionen können mit einer variablen Anzahl von Argumenten aufgerufen werden. Alle in der Funktionsliste ausgelassenen Argumente werden auf Null oder eine leere Zeichenkette gesetzt.<br>Argumente in der Liste des Aufrufers, 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 Beendigung der Funktion abgerufen werden können. Dem Argument kann das Präfix BYVAL |

| | |
|---|---|
| | vorangestellt werden, wodurch dieser Mechanismus verhindert wird und nur der Wert verwendet wird. Alternativ weist das Präfix BYREF MMBasic an, dass eine Referenz erforderlich ist, und es wird ein Fehler ausgegeben, wenn dies nicht möglich ist.<br>Arrays werden durch Angabe des Array-Namens in leeren Klammern (z. B. arg()) übergeben; sie werden immer per Referenz übergeben und müssen den richtigen Typ haben.<br>Du darfst nicht mit Befehlen wie GOTO in eine Funktion springen oder aus ihr herausspringen. Das hat undefinierte Nebenwirkungen, darunter auch, dass es dir den Tag ruiniert. |
| GAMEPAD COLOUR channel, colour | Ändert die Farbe des Displays auf einem PS4-Controller am USB-Kanal „channel“. „colour“ wird als Standard-RGB888-Wert festgelegt, z. B. RGB(RED) |
| GAMEPAD HAPTIC channel left, right | Löst die Aktivierung der linken und rechten Vibrationsmotoren eines PS4-Controllers am USB-Kanal „channel“ aus. „left“ und „right“ müssen Zahlen zwischen 0 (aus) und 255 (maximal) sein. |
| GAMEPAD INTERRUPT ENABLE channel, int [,mask] | Aktiviert Interrupts bei Tastendrücken auf einem USB-Gamecontroller. Der optionale Parameter „mask“ legt fest, welche der Tasten den Interrupt auslösen (Standardmäßig alle). „mask“ ist eine Bitmap, die dem Ausgang der Funktion DEVICE(GAMEPAD channel,B) entspricht. |
| GAMEPAD INTERRUPT DISABLE channel | Deaktiviert Interrupts vom Gamepad auf dem angegebenen Kanal |
| GAMEPAD MONITOR | Verwende GAMEPAD MONITOR, bevor du ein Gamepad anschließt; sobald es angeschlossen ist, zeigt es bei jeder Tastenänderung den Vorher-Nachher-Bericht an |
| GAMEPAD CONFIGURE vid,pid,i0,c0,i1,c1,i2,c2,i3,c3,i4,c4,i5,c5,i6,c6,i7,c7,i8,c8,i9,c9,i10,c10,i11,c11,i12,c12,i13,c13,i14,c14,i15,c15 | Verwende dies, um ein Gamepad zu konfigurieren, das von der Firmware nicht unterstützt wird. Führe den Befehl aus, bevor du das Gamepad anschließt. Alle 34 Parameter sind obligatorisch. In jedem Fall definieren die i/c-Parameter den Index im Bericht und die Bit- -Nummer an diesem Index für die Daten, die dem entsprechenden Bit entsprechen. Siehe DEVICE(GAMEPAD n,B) für weitere Informationen zur Bit-Verwendung (0-15) |
| GOTO target | Springt mit der Programmausführung zum Ziel, das eine Zeilennummer oder ein Label sein kann. |
| GUI BITMAP x, y, bits [, width] [, height] [, scale] [, c] [, bc] | Zeigt die Bits einer Bitmap auf einem VGA-/HDMI-Monitor oder LCD-Panel an, beginnend bei 'x' und 'y' auf einem angeschlossenen Gerät.<br>'height' und 'width' sind die Abmessungen der Bitmap, wie sie auf dem Gerät angezeigt werden, und sind standardmäßig auf 8x8 eingestellt.<br>'scale' ist optional und entspricht standardmäßig dem Wert, der mit dem Befehl FONT festgelegt wurde.<br>'c' ist die Zeichenfarbe und 'bc' ist die Hintergrundfarbe. Sie sind optional und standardmäßig auf die aktuellen Vordergrund- und Hintergrundfarben gesetzt.<br>Die Bitmap kann eine Ganzzahl- oder Zeichenfolgenvariable oder -konstante sein und wird gezeichnet, indem das erste Byte als erste Bits der obersten Zeile verwendet wird (zuerst Bit 7, dann Bit 6 usw.), gefolgt vom nächsten Byte usw. Wenn die oberste Zeile gefüllt ist, beginnt die nächste Zeile der angezeigten Bitmap mit dem nächsten Bit in der Ganzzahl oder Zeichenfolge. |

| | |
|---|---|
| | Eine Definition der Farben und Grafikkoordinaten findest du im Kapitel *„Grafikbefehle und -funktionen“*. |
| GUI CALIBRATE<br>oder<br>GUI CALIBRATE a,b,c,d,d | NICHT BEI VGA- UND HDMI-VERSIONEN<br>Dieser Befehl dient zur Kalibrierung der Touch-Funktion eines LCD-Bildschirms. Er zeigt eine Reihe von Zielmarken auf dem Bildschirm an und wartet darauf, dass jede einzelne präzise berührt wird.<br>Der Befehl kann auch mit fünf Argumenten verwendet werden, die die Kalibrierungswerte angeben; in diesem Fall erfolgt die Kalibrierung, ohne dass Zielpunkte angezeigt werden oder eine Eingabe vom Benutzer erforderlich ist. Um die Werte zu ermitteln, verwende die OPTIONSLISTE, **nachdem** du das Display normal kalibriert hast. Beachte, dass diese Werte spezifisch für dieses Display sind und erheblich variieren können. |
| GUI TEST LCDPANEL | Testet ein Anzeigegerät (LCD, VGA usw.). Es zeichnet kontinuierlich eine animierte Darstellung aus farbigen Kreisen auf dem Bildschirm. |
| GUI RESET LCDPANEL | NICHT VGA- UND HDMI-VERSIONEN<br>Initialisiert das konfigurierte LCD-Panel neu. Die Initialisierung erfolgt automatisch beim Start der PicoMite-Firmware, aber unter bestimmten Umständen kann es notwendig 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 TOUCH | NICHT VGA- UND HDMI-VERSIONEN<br>Wird die Touch-Funktion des Displays auf einem LCD-Bildschirm testen.<br>Der Bildschirm wird gelöscht und MMBasic wartet auf eine Berührung, woraufhin ein weißer Punkt auf dem Display erscheint, der die genaue Berührungsposition auf dem Bildschirm markiert. Jedes Zeichen, das an der Konsole eingegeben wird, beendet den Test. |
| HELP searchtext | Der Befehl „help“ sucht auf dem Laufwerk A: nach einer Datei namens „help.txt“. Diese kann vom Benutzer selbst erstellt oder von der Community entwickelt worden sein und muss ein bestimmtes Format aufweisen.<br>Für jeden Hilfseintrag muss die erste Zeile eine Suchzeichenfolge sein, der ein ~-Zeichen vorangestellt ist. Dies wird von der Hilfefunktion verwendet, um einen Eintrag zu finden, und wird nicht angezeigt. Der „searchtext“ kann ein ? für die Ersetzung eines einzelnen Zeichens oder ein * für die Ersetzung mehrerer Zeichen (oder keiner) enthalten.<br>Im Anschluss an den Suchbegriff gibt die nächste Zeile typischerweise die Syntax eines bestimmten Befehls oder einer bestimmten Funktion an. Alle folgenden Zeilen dienen der weiteren Erläuterung.<br>z. B.<br>*~COLOR*<br>*COLOR fore [, back]*<br>*Legt die Standardfarbe für Befehle (PRINT usw.) fest*<br>*die auf dem angeschlossenen LCD-Bildschirm angezeigt werden.*<br>*„fore“ ist die Vordergrundfarbe, „back“ ist die Hintergrundfarbe.*<br>*Der Hintergrund ist optional und wird standardmäßig auf Schwarz gesetzt, wenn er nicht angegeben wird.*<br>Der Befehl gibt alle Einträge zurück, die mit dem „Suchtext“ übereinstimmen, und diese werden entsprechend dem Konsolengerät seitenweise angezeigt. Verschiedene Versionen von help.txt sind unter https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=17865 verfügbar |

| | |
|---|---|
| HUMID pin, tvar, hvar [,DHT11] | Gibt die Temperatur und Luftfeuchtigkeit mithilfe des DHT22-Sensors zurück. Alternative Versionen des DHT22 sind der AM2303 oder der RHT03 (alle sind kompatibel).<br>„pin“ ist der mit dem Sensor verbundene I/O-Pin. Jeder beliebige I/O-Pin kann verwendet werden.<br>'tvar' ist die Variable, die die gemessene Temperatur enthält, und 'hvar' ist die entsprechende Variable für die Luftfeuchtigkeit. Beide müssen vorhanden sein und beide müssen Gleitkommavariablen sein.<br>Zum Beispiel: `HUMID 3, TEMP!, HUMIDITY!`<br>Die Temperatur wird in °C gemessen und die Luftfeuchtigkeit in Prozent relativer Luftfeuchtigkeit. Beide werden mit einer Auflösung von 0,1 gemessen. Tritt ein Fehler auf (Sensor nicht angeschlossen oder fehlerhaftes Signal), sind beide Werte 1000,0.<br>Normalerweise sollte der DHT22 mit 3,3 V versorgt werden, um seine Ausgangsspannung für den Raspberry Pi Pico unter 3,6 V zu halten (der Pico 2 hat dieses Problem nicht), und der Signalpin sollte über einen 1K- bis 10K-Widerstand (4,7K empfohlen) auf 3,3 V hochgezogen werden.<br>Der optionale DHT11-Parameter passt die Timings an, damit es mit dem DHT11 funktioniert. Setze ihn auf 1 für DHT11 und auf 0 oder lass ihn weg für DHT22. |
| I2C | Weitere Details findest du in Anhang B |
| I2C OPEN speed, timeout | Aktiviert das erste $I^2C$-Modul im Master-Modus. „speed“ ist die zu verwendende Taktrate (in kHz) und muss einer der Werte 100, 400 oder 1000 sein.<br>„timeout“ ist ein Wert in Millisekunden, nach dessen Ablauf die Sende- und Empfangsbefehle des Masters unterbrochen werden, wenn sie nicht abgeschlossen sind. Der Mindestwert beträgt 100. Ein Wert von Null deaktiviert das Timeout (dies wird jedoch nicht empfohlen). |
| I2C WRITE addr, option, sendlen, senddata [,sendata ..] | Sende Daten an das $I^2C$-Slave-Gerät. „addr“ ist die $I^2C$-Adresse des Slaves.<br>„option“ kann 0 für den normalen Betrieb oder 1 sein, um nach dem Befehl die Kontrolle über den Bus zu behalten (es wird nach Abschluss des Befehls kein Stopp-Signal gesendet)<br>„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).<br>Hinweise:<br>• Die Daten können als einzelne Bytes in der Befehlszeile angegeben werden.<br>Beispiel: `I2C WRITE &H6F, 0, 3, &H23, &H43, &H25`<br>• Die Daten können in einem eindimensionalen Array vorliegen, das mit leeren Klammern angegeben wird (d. h. keine Dimensionen). Es werden „sendlen“ Bytes des Arrays gesendet, beginnend mit dem ersten Element.<br>Beispiel: `I2C WRITE &H6F, 0, 3, ARRAY()`<br>Die Daten können eine Zeichenfolgenvariable sein (keine Konstante).<br>Beispiel: `I2C WRITE &H6F, 0, 3, STRING$` |

| | |
|---|---|
| I2C READ addr, option, rcvlen, rcvbuf | Hole Daten vom I2C-Slave-Gerät. „addr“ ist die I2C-Adresse des Slaves.<br>„option“ kann 0 für den normalen Betrieb oder 1 sein, um nach dem Befehl die Kontrolle über den Bus zu behalten (nach Abschluss des Befehls wird kein Stopp-Signal gesendet)<br>„rcvlen“ ist die Anzahl der zu empfangenden Bytes.<br>„rcvbuf“ ist die Variable oder das Array, in dem die empfangenen Daten gespeichert werden – dies kann sein:<br>• Eine Zeichenfolgenvariable. Bytes werden als aufeinanderfolgende Zeichen gespeichert.<br>• Ein eindimensionales Array aus Zahlen, angegeben mit leeren Klammern. Die empfangenen Bytes werden in aufeinanderfolgenden Elementen des Arrays gespeichert, beginnend mit dem ersten.<br>Beispiel: `I2C READ &H6F, 0, 3, ARRAY()`<br>Eine normale numerische Variable (in diesem Fall muss rcvlen 1 sein). |
| I2C CHECK addr | Setzt die schreibgeschützte Variable MM.I2C auf 0, wenn ein Gerät unter der Adresse „addr“ antwortet. MM.I2C wird auf 1 gesetzt, wenn keine Antwort erfolgt. |
| I2C CLOSE | Deaktiviert das Master-I$^2$C-Modul. Dieser Befehl sendet außerdem ein Stopp-Signal, falls der Bus noch belegt ist. |
| I2C-SLAVE | Siehe Anhang B |
| I2C2 | Es handelt sich um denselben Befehlssatz wie für I2C (oben), der jedoch auf den zweiten I$^2$C-Kanal angewendet wird. |
| I2CLCD | Diese Befehlsreihe steuert HD44780-kompatible Zeichen-LCD-Displays über ein PCF8574-I2C-I/O-Erweiterungsmodul. Diese Adaptermodule sind häufig als Tochterplatinen auf 16x2- und 20x4-Zeichen-LCDs zu finden und ermöglichen die I2C-Steuerung mit nur zwei Leitungen (plus Stromversorgung). |
| I2CLCD INIT address | Initialisiere das LCD an der angegebenen I2C-Adresse. Die Adresse lautet bei den meisten Modulen typischerweise &H27 oder &H3F. PCF8574 verwendet &H20–&H27, PCF8574A verwendet &H38–&H3F. Erfordert, dass OPTION SYSTEM I2C zuvor konfiguriert wurde. Ein Gradzeichen (°) wird automatisch unter dem Zeichencode 0 erstellt. |
| I2CLCD CLOSE | Schließe das LCD und schalte die Hintergrundbeleuchtung aus. |
| I2CLCD CLEAR | Lösche die Anzeige und setze den Cursor auf die Ausgangsposition zurück. |
| I2CLCD BACKLIGHT state | Schalte die Hintergrundbeleuchtung ein (1) oder aus (0). |
| I2CLCD CURSOR state [, BLINK] | Steuere die Sichtbarkeit des Cursors. Der Status kann ON/OFF oder 1/0 sein. Der optionale Parameter BLINK (oder 1) aktiviert das Blinken des Cursors. |
| I2CLCD CREATECHAR Code, d0, d1, d2, d3, d4, d5, d6, d7 | Definiere ein benutzerdefiniertes Zeichen unter Code (0-7). Die 8 Datenbytes (d0-d7) definieren das 5x8-Pixel-Muster, wobei jedes Byte eine Zeile darstellt (Werte 0-31, nur die unteren 5 Bits werden verwendet). |
| I2CLCD CMD Byte [, Byte ...] | Sende(n) Rohbefehlsbyte(s) direkt an den LCD-Controller. |
| I2CLCD DATA-Byte [, Byte ...] | Sende Rohdatenbyte(s) direkt an den LCD-Controller. |
| I2CLCD line, position, string$ | Zeige Text in Zeile (1–4) beginnend bei Position (1–40) an. |

| | |
|---|---|
| I2CLCD line, Cn, string$ | Zeige zentrierten Text in Zeile (1–4) an. Cn gibt die Anzeigebreite an: C8, C16, C20 oder C40.<br><br>Hinweis:<br>Hardware-Konfiguration:<br>Der PCF8574-I/O-Expander muss gemäß der Standardkonfiguration an das LCD angeschlossen werden:<br>P0 = RS (Register Select)<br>P1 = RW (Lesen/Schreiben)<br>P2 = EN (Aktivieren)<br>P3 = Hintergrundbeleuchtungssteuerung<br>P4 = D4<br>P5 = D5<br>P6 = D6<br>P7 = D7<br>Dies ist die Standardverdrahtung, die von praktisch allen LCD-I2C-Backpack-Modulen verwendet wird.<br>Voraussetzungen:<br>Die SYSTEM-I2C-Schnittstelle muss vor der Verwendung dieses Befehls konfiguriert werden:<br>OPTION SYSTEM I2C sda_pin, scl_pin |
| IF expr THEN stmt [: stmt]<br>oder<br>IF expr THEN stmt ELSE stmt | Wertet den Ausdruck „expr“ aus und führt die Anweisung nach dem Schlüsselwort THEN aus, wenn er wahr ist, oder springt zur nächsten Zeile, wenn er falsch ist. Wenn weitere Anweisungen in der Zeile stehen (getrennt durch Doppelpunkte (:)), werden diese ebenfalls ausgeführt, wenn der Ausdruck wahr ist, oder übersprungen, wenn er falsch ist. Das Schlüsselwort ELSE ist optional; wenn es vorhanden ist, werden die darauf folgenden Anweisungen ausgeführt, wenn „expr“ als falsch ausgewertet wird.<br>Die „THEN-Anweisung“ kann auch ersetzt werden durch:<br>GOTO Zeilennummer \| Label.<br>Diese Art von IF-Anweisung steht komplett in einer Zeile. |
| IF expression THEN<br><statements><br>[ELSEIF expression THEN<br><statements>]<br>[ELSE<br><statements>]<br>ENDIF | Mehrzeilige IF-Anweisung mit optionalen ELSE- und ELSEIF-Fällen, die mit ENDIF endet. Jede Komponente steht in einer eigenen Zeile.<br>Wertet „Ausdruck“ aus und führt die auf THEN folgenden Anweisungen aus, wenn der Ausdruck wahr ist, oder optional die auf die ELSE-Anweisung folgenden Anweisungen, wenn er falsch ist. Die ELSEIF-Anweisung (falls vorhanden) wird ausgeführt, wenn die vorherige Bedingung falsch ist, und sie startet eine neue IF-Kette mit weiteren ELSE- und/oder ELSEIF-Anweisungen, je nach Bedarf. Ein ENDIF wird verwendet, um die mehrzeilige IF-Anweisung zu beenden. |
| INC var [,increment] | Erhöht die Variable „var“ entweder um 1 oder, falls angegeben, um den Wert in „increment“. „increment“ kann negativ sein, was zu einer Verringerung führt.<br>Das funktioniert genauso wie „`var = var + increment`“, wird aber viel schneller verarbeitet |

| | |
|---|---|
| INPUT ["prompt$";] var1 [,var2 [, var3 [, usw.]] | Nimmt eine durch Kommas (,) getrennte Liste von Werten entgegen, die an der Konsole eingegeben wurden, und weist sie einer fortlaufenden Liste von Variablen zu.<br>Wenn der Befehl zum Beispiel lautet: INPUT a, b, c<br>Und Folgendes wird über die Tastatur eingegeben: 23, 87, 66<br>Dann ist a = 23 und b = 87 und c = 66<br>Die Liste der Variablen kann eine Mischung aus Float-, Integer- oder String-Variablen sein. Die an der Konsole eingegebenen Werte müssen dem Typ der Variablen entsprechen.<br>Wenn ein einzelner Wert eingegeben wird, ist kein Komma erforderlich (dieser Wert darf jedoch kein Komma enthalten).<br>„prompt$“ ist eine String-Konstante (keine Variable oder kein Ausdruck) und wird, falls angegeben, zuerst ausgegeben. Normalerweise endet die Eingabeaufforderung mit einem Semikolon (;) und in diesem Fall wird nach der Eingabeaufforderung ein Fragezeichen ausgegeben. Wenn die Eingabeaufforderung mit einem Komma (,) statt mit einem Semikolon (;) endet, wird das Fragezeichen unterdrückt. |
| INPUT #nbr,<br>list of variables | Wie oben, außer dass die Eingabe von einem seriellen Port oder einer Datei gelesen wird, die zuvor für INPUT als „nbr“ geöffnet wurde. Siehe den Befehl OPEN. |
| INTERRUPT [myint] | Dieser Befehl löst einen Software-Interrupt aus. Der Interrupt wird mit INTERRUPT „myint“ eingerichtet, wobei „myint“ der Name einer Subroutine ist, die ausgeführt wird, wenn der Interrupt ausgelöst wird.<br>Verwende INTERRUPT 0, um den Interrupt zu deaktivieren<br>Verwende INTERRUPT ohne Parameter, um den Interrupt auszulösen.<br>Hinweis: Der Interrupt kann auch aus einer CSUB heraus ausgelöst werden |
| IR dev, key, int<br>or<br>IR CLOSE | Dekodiert Infrarot-Fernbedienungssignale von NEC oder Sony.<br>Ein IR-Empfängermodul ist erforderlich, um das IR-Licht zu erfassen und das Signal zu demodulieren. Es kann an jeden beliebigen Pin angeschlossen werden, dieser Pin muss jedoch zuvor mit dem Befehl konfiguriert werden: `SETPIN n, IR`<br>Die Dekodierung des IR-Signals erfolgt im Hintergrund, und das Programm läuft nach diesem Befehl ohne Unterbrechung weiter. 'dev' und 'key' sollten numerische Variablen sein, und ihre Werte werden aktualisiert, sobald ein neues Signal empfangen wird ('dev' ist der von der Fernbedienung gesendete Gerätecode und 'key' ist die gedrückte Taste).<br>'int' ist eine benutzerdefinierte Subroutine, 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' überprüfen und entsprechende Maßnahmen ergreifen.<br>Der Befehl „IR CLOSE“ beendet den IR-Decoder.<br>Beachte, dass beim NEC-Protokoll die Bits in „dev“ und „key“ vertauscht sind. Beispielsweise sollte in „key“ Bit 0 Bit 7 sein, Bit 1 sollte Bit 6 sein usw. Dies hat keinen Einfluss auf den normalen Gebrauch, aber wenn du nach einem bestimmten numerischen Code suchst, der von einem Hersteller bereitgestellt wird, solltest du die Bits umkehren. Hier wird beschrieben, wie das geht: http://www.thebackshed.com/forum/forum_posts.asp?TID=8367<br>Weitere Details findest du im Kapitel *„Spezielle Hardwaregeräte*“. |

| | |
|---|---|
| IR SEND pin, dev, key | Erzeuge ein 12-Bit-Infrarotsignal nach dem Sony-Fernbedienungsprotokoll.<br>„pin“ ist der zu verwendende I/O-Pin. Dies kann ein beliebiger I/O-Pin sein, der automatisch als Ausgang konfiguriert wird und an eine Infrarot-LED angeschlossen werden sollte. Im Ruhezustand ist der Pegel niedrig; ein hoher Pegel zeigt an, wann die LED eingeschaltet werden soll.<br>'dev' ist das zu steuernde Gerät und eine Zahl zwischen 0 und 31, 'key' ist der simulierte Tastendruck und eine Zahl zwischen 0 und 127.<br>Das IR-Signal wird mit etwa 38 kHz moduliert und das Senden des Signals dauert etwa 25 ms, währenddessen die Programmausführung angehalten wird. |
| KEYPAD var, int, r1, r2, r3, r4, c1, c2, c3 [, c4]<br>or<br>KEYPAD CLOSE | Überwache und entschlüssele Tastendrücke auf einem 4x3- oder 4x4-Tastenfeld.<br>Die Überwachung des Tastenfelds erfolgt im Hintergrund, und das Programm wird nach diesem Befehl ohne Unterbrechung fortgesetzt. 'var' sollte eine numerische Variable sein, deren Wert bei jedem erkannten Tastendruck aktualisiert wird.<br>'int' ist eine benutzerdefinierte Subroutine, die aufgerufen wird, sobald ein neuer Tastendruck empfangen wird. In der Interrupt-Subroutine kann das Programm die Variable 'var' überprüfen und entsprechende Maßnahmen ergreifen.<br>r1, r2, r3 und r4 sind Pin-Nummern für die vier Zeilenanschlüsse des Tastenfelds und c1, c2, c3 und c4 sind die Spaltenanschlüsse. c4 ist optional und wird nur bei 4x4-Tastenfeldern verwendet. Dieser Befehl konfiguriert diese Pins automatisch wie erforderlich.<br>Bei einem Tastendruck ist der Wert, der „var“ zugewiesen wird, die Nummer einer Zifferntaste (z. B. gibt „6“ den Wert 6 zurück) oder 10 für die *-Taste und 11 für die #-Taste. Bei 4x4-Tastaturen wird für A der Wert 20, für B der Wert 21, für C der Wert 22 und für D der Wert 23 zurückgegeben.<br>Der Befehl KEYPAD CLOSE beendet die Tastaturfunktion und versetzt den I/O-Pin wieder in einen nicht konfigurierten Zustand.<br>Weitere Details findest du im Abschnitt *„Spezielle Hardwaregeräte*“. |
| KEYPAD keymapmap!(), var!,int, startcolpin, nocols, startrowpin, norows | <u>NUR RP2350-VERSIONEN</u><br>Konfiguriere und verwende ein Tastenfeld mit beliebig vielen Zeilen und Spalten und weise jeder Taste benutzerdefinierte Rückgabecodes zu.<br>'keymapmap!()' ist ein zweidimensionales Array aus Float-Werten, das die vom Benutzer festgelegte Zuordnung für jede Taste enthält. Die Abmessungen des Arrays müssen mit der angegebenen Anzahl an Spalten und Zeilen übereinstimmen.<br>'var!' ist die Float-Variable, die den zurückgegebenen Wert enthält. 'int' ist die Interrupt-Routine, die aufgerufen wird, wenn eine Taste gedrückt wird.<br>'startcolpin' ist der Pin, der den Bereich zusammenhängender GP-Pins für den Spalten-Scan beginnt, während 'nocols' die Anzahl der Pins in diesem Bereich angibt. 'startrowpin' ist der Pin, der den Bereich von 'norows' zusammenhängenden GP-Pins für den Zeilen-Scan beginnt.<br>Beispiel: Bei einer Tastatur mit 3 Spalten und 4 Zeilen (d. h. 123 / 456 / 789 / *0#). Verbinde GP1 mit der linken Spalte, GP2 mit der mittleren und GP3 mit der rechten, dann die 4 Zeilen von oben nach unten mit GP4 bis GP7.<br>Das Programm könnte lauten:<br>`OPTION BASE 1`<br>`DIM keymap(3,4)=(1,4,7,10, 4,5,6,0, 3,6,9,11)`<br>`KEYPAD keymap(), keyret, myint, GP1, 3, GP4, 4`<br>`DO`<br>`  ' Hauptprogramm-Verarbeitungsschleife`<br>`LOOP`<br>`SUB myint` |

| | |
|---|---|
| | PRINT keyret<br>END SUB |
| KILL file$ [,all] | Löscht die durch „file$“ angegebene Datei. Eine Dateiendung muss angegeben werden.<br>Ein Massenlöschvorgang wird ausgelöst, wenn fname$ ein „*“- oder ein „?“-Zeichen enthält. Wenn der optionale Parameter „all“ verwendet wird, wirst du einmalig zur Bestätigung aufgefordert. Wenn „all“ nicht angegeben ist, wirst du bei jeder Datei zur Bestätigung aufgefordert. |
| LCD INIT d4, d5, d6, d7, rs, en<br>or<br>LCD line, pos, text$<br>or<br>LCD CLEAR<br>or<br>LCD CLOSE | Zeigt Text auf einem LCD-Zeichenmodul an. Dieser Befehl funktioniert mit den meisten 1-zeiligen, 2-zeiligen oder 4-zeiligen LCD-Modulen, die den KS0066-, HD44780- oder SPLC780-Controller verwenden (dies kann jedoch nicht garantiert werden).<br>Der Befehl „LCD INIT“ dient dazu, das LCD-Modul für den Einsatz zu initialisieren. „d4“ bis „d7“ sind die I/O-Pins, die mit den Eingängen D4 bis D7 am LCD-Modul verbunden werden (die Eingänge D0 bis D3 sollten mit Masse verbunden werden). „rs“ ist der Pin, der mit dem Registerauswahl-Eingang am Modul verbunden ist (manchmal auch als CMD bezeichnet). „en“ ist der Pin, der mit dem Freigabe- oder Chipauswahl-Eingang am Modul verbunden ist. Der R/W-Eingang am Modul sollte immer auf Masse gelegt werden. Die oben genannten I/O-Pins werden durch diesen Befehl automatisch auf Ausgangsmodus gesetzt.<br>Wenn das Modul initialisiert wurde, können mit dem LCD-Befehl Daten darauf geschrieben werden. „line“ ist die Zeile auf dem Display (1 bis 4) und „pos“ ist die Zeichenposition in der Zeile (die erste Position ist 1). „text$“ ist eine Zeichenkette, die den Text enthält, der auf das LCD-Display geschrieben werden soll.<br>„pos“ kann auch C8, C16, C20 oder C40 sein; in diesem Fall wird die Zeile gelöscht und der Text auf einem 8-, 16-, 20- oder 40-zeiligen Display zentriert. Zum Beispiel:<br>`LCD 1, C16, „Hallo“`<br>LCD CLEAR löscht alle auf dem LCD angezeigten Daten und LCD CLOSE beendet die LCD-Funktion und versetzt alle I/O-Pins in den nicht konfigurierten Zustand.<br>Weitere Details findest du im Kapitel *„Spezielle Hardwaregeräte*“. |
| LCD CMD d1 [, d2 [, etc]]<br>or<br>LCD DATA d1 [, d2 [, etc]] | Diese Befehle senden ein oder mehrere Bytes an ein LCD-Display, entweder als Befehl (LCD CMD) oder als Daten (LCD DATA). Jedes Byte ist eine Zahl zwischen 0 und 255 und muss durch Kommas getrennt werden. Das LCD muss zuvor mit dem Befehl LCD INIT initialisiert worden sein (siehe oben).<br>Diese Befehle können verwendet werden, um ein nicht standardmäßiges LCD im „Raw-Modus“ anzusteuern, oder um spezielle Funktionen wie Bildlauf, Cursor und benutzerdefinierte Zeichensätze zu aktivieren. Die erforderlichen Befehls- und Datenwerte findest du im Datenblatt deines LCDs. |
| LET variable = expression | Weist der Variablen den Wert von „expression“ zu. LET wird automatisch angenommen, wenn eine Anweisung nicht mit einem Befehl beginnt. Zum Beispiel:<br>`Var = 56` |
| LIBRARY SAVE<br>or<br>LIBRARY DELETE<br>or | Die Bibliothek ist ein spezieller Bereich des Programmspeichers, der Programmcode wie Unterprogramme, Funktionen und CFunktionen enthalten kann. Diese Routinen sind für den Programmierer nicht sichtbar, stehen dem laufenden Programm jedoch zur Verfügung und verhalten sich genauso wie die integrierten Befehle und Funktionen in MMBasic. |

| | |
|---|---|
| LIBRARY LIST<br><br>or<br><br>LIBRARY LIST ALL<br><br>or<br><br>LIBRARY DISK SAVE fname$<br><br>or<br><br>LIBRARY DISK LOAD fname$ | Jeder Code in der Bibliothek, der nicht in einer Subroutine oder Funktion enthalten ist, wird unmittelbar vor dem Start des Programms ausgeführt. Dies kann zum Initialisieren von Konstanten, zum Setzen von Optionen usw. verwendet werden. Eine ausführliche Erklärung findest du unter der Überschrift *„Die Bibliothek“* in diesem Handbuch.<br>Die Bibliothek wird im Programmspeicher-Flash-Slot 3 gespeichert, der dann nicht mehr zum Speichern eines Programms zur Verfügung steht (die Slots 1 bis 2 bleiben weiterhin verfügbar).<br>LIBRARY SAVE nimmt den Inhalt des normalen Programmspeichers, komprimiert ihn (entfernt redundante Daten wie Kommentare) und hängt ihn an den Bibliotheksbereich an (der Hauptprogrammspeicher ist dann leer). Der Code in der Bibliothek wird in LIST oder EDIT nicht angezeigt und wird nicht gelöscht, wenn ein neues Programm geladen oder NEW verwendet wird.<br>LIBRARY DELETE löscht die Bibliothek und gibt Flash-Slot 3 wieder für den normalen Gebrauch frei (OPTION RESET bewirkt dasselbe).<br>LIBRARY LIST listet den Inhalt der Bibliothek auf. Verwende ALL, um die Liste ohne Seitenbestätigungen anzuzeigen.<br>LIBRARY DISK SAVE fname$ speichert die aktuelle Bibliothek als Binärdatei, sodass du sie später mit LIBRARY DISK LOAD fname$ wiederherstellen kannst. Zusammen ermöglichen diese Befehle das einfache Speichern, Wiederherstellen und Verteilen von Bibliotheken, die für einzelne Programme spezifisch sind. Abgesehen von der Verwendung versionsspezifischer Funktionen in der Bibliothek (WEB, VGA, GUI) können Bibliotheken zwischen verschiedenen Versionen gemeinsam genutzt werden. |
| LINE x1, y1, x2, y2 [,[-] LW [, C]] | Zeichnet eine Linie auf dem Bildschirm, die an den Koordinaten „x1“ und „y1“ beginnt und bei „x2“ und „y2“ endet.<br>„LW“ ist die Breite der Linie. Der Standardwert ist 1, wenn nichts anderes angegeben wird. Bei Linien mit definierter Breite bestimmen die Koordinaten x1 und y1 den oberen linken Pixel der dicken Linie. Das heißt, die Linie befindet sich auf dem Bildschirm rechts von der angegebenen Position oder darunter.<br>Wenn die Breite als negativer Wert angegeben wird, gilt die Breite für Linien in alle Richtungen, und diese werden auf die angegebenen Start- und Endkoordinaten zentriert.<br>„C“ ist eine Ganzzahl, die die Farbe angibt, und ist standardmäßig die aktuelle Vordergrundfarbe.<br>Alle Parameter können als Arrays angegeben werden, und die Software zeichnet die Anzahl der Linien, die durch die Dimensionen des kleinsten Arrays bestimmt wird. „x1“, „y1“, „x2“ und „y2“ müssen alle Arrays oder alle einzelne Variablen/Konstanten sein, andernfalls wird ein Fehler ausgegeben. „lw“ und „c“ können entweder Arrays oder einzelne Variablen/Konstanten sein. |
| LINE AA x1, y1, x2, y2 [, LW [, C]] | Zeichnet eine Linie mit Anti-Aliasing. Die Parameter entsprechen denen des obigen LINE-Befehls. Diese Version verwendet jedoch variable Intensitätswerte der angegebenen Farbe, um den „stufigen“ Effekt bei diagonalen Linien zu reduzieren. Außerdem kann diese Version diagonale Linien beliebiger Breite zeichnen. Beachte, dass sie keine Arrays als Parameter akzeptiert. |
| LINE GRAPH x(), y(), colour | Dieser Befehl erzeugt ein Liniendiagramm aus den in „x()“ und „y()“ angegebenen Koordinatenpaaren. Das Diagramm hat n-1 Segmente, wenn die Arrays x und y n Elemente enthalten. |

| | |
|---|---|
| LINE INPUT [prompt$,] string-variable$ | Liest eine ganze Zeile aus der Konsoleneingabe in „string-variable$“ ein.<br>„prompt$“ ist eine Zeichenfolgenkonstante (keine Variable oder kein Ausdruck) und wird, falls angegeben, zuerst ausgegeben. Ein Fragezeichen wird nur ausgegeben, wenn es Teil von „prompt$“ ist. Im Gegensatz zu INPUT liest dieser Befehl eine ganze Zeile ein, ohne bei durch Kommas getrennten Datenelementen anzuhalten. |
| LINE INPUT #nbr, string-variable$ | Wie oben, außer dass die Eingabe von einem seriellen Kommunikationsport oder einer Datei gelesen wird, die zuvor für INPUT als „nbr“ geöffnet wurde. Siehe den Befehl OPEN. |
| LINE PLOT ydata() [,nbr] [,xstart] [,xinc] [,ystart] [,yinc] [,colour] | Zeichnet ein Liniendiagramm aus einem Array von Datenpunkten auf der y-Achse.<br>„ydata()“ ist ein Array aus Fließkommazahlen oder Ganzzahlen, die gezeichnet werden sollen<br>„nbr“ ist die Anzahl der zu zeichnenden Liniensegmente – standardmäßig ist dies der kleinere Wert aus<br>Array-Größe und MM.HRES-2<br>„xstart“ ist die x-Koordinate, an der mit dem Zeichnen begonnen werden soll – Standardwert ist 0<br>„xinc“ ist der Schritt entlang der x-Achse, um jede Koordinate zu zeichnen – Standardwert ist 1<br>„ystart“ ist die Position in ydata, an der die Darstellung beginnen soll – Standardwert ist der Anfang des Arrays. Hinweis: berücksichtigt OPTION BASE<br>‘yinc’ ist der Schritt, um den der Index in ydata für jeden zu zeichnenden Punkt erhöht wird<br>„colour“ ist die Farbe, in der die Linie gezeichnet wird |
| LIST [fname$]<br>or<br>LIST ALL [fname$] | Zeige ein Programm auf der Konsole an.<br>LIST allein listet das Programm mit einer Pause nach jeder Bildschirmseite auf.<br>LIST ALL listet das Programm ohne Pausen auf. Das ist nützlich, wenn du das Programm auf einen Terminalemulator auf einem PC übertragen möchtest, der die Eingabeströme in eine Datei speichern kann. Wenn<br>Wenn das optionale „fname$“ angegeben wird, wird die entsprechende Datei auf dem Flash-Dateisystem oder der SD-Karte aufgelistet. |
| LIST PINS | Listet den aktuellen Status aller Pins des Prozessors auf |
| LIST SYSTEM I2C | Listet eine Übersicht aller an den System-I2C-Bus angeschlossenen I2C-Geräte auf |
| LIST COMMANDS<br>oder<br>LIST FUNCTIONS | Listet alle gültigen Befehle oder Funktionen auf |
| LIST VARIABLES [s%()] | Listet alle globalen Variablen und Konstanten auf und, falls in einer Subroutine aufgerufen, die von dieser Subroutine verwendete Variable sowie alle Subroutinen, die sie aufgerufen haben. Wenn der optionale Parameter s$() verwendet wird, werden die Variablen in s%() aufgelistet, das als Longstring behandelt wird (siehe Befehl LONGSTRING). |

| | |
|---|---|
| LMID(array%(),start [,num])=string$ | Fügt „string$“ an der Position „start“ in den Longstring „array%()“ ein und ersetzt dabei „num“ vorhandene Zeichen. Wenn „num“ nicht angegeben wird, wird es aus der Länge von „string$“ berechnet.<br>„num“ kann 0 sein; in diesem Fall wird „string$“ an der angegebenen Position eingefügt. |
| LOAD file$ [,R] | Lädt ein Programm namens „file$“ vom Flash-Dateisystem oder der SD-Karte in den Programmspeicher.<br>Wenn das optionale Suffix ‚R hinzugefügt wird, wird das Programm sofort ohne Abfrage ausgeführt (in diesem Fall muss „file$“ eine String-Konstante sein). Der Befehl RUN macht dasselbe und erlaubt die Verwendung einer String-Variablen.<br>Wenn keine Erweiterung angegeben wird, wird „.BAS“ an den Dateinamen angehängt. |
| LOAD CONTEXT [KEEP] | Stellt den Variablenbereich auf den zuvor gespeicherten Zustand zurück und bewahrt optional die gespeicherten Variablen, um bei Bedarf einen zweiten LOAD-Befehl zu ermöglichen.<br>Siehe auch SAVE CONTEXT |
| LOAD DATA fname$, address | Lädt die Datei „fname$“ als Binärdaten und schreibt sie an die angegebene Speicheradresse.<br>Beachte, dass die Adresse nicht überprüft wird, daher muss der Befehl mit Vorsicht verwendet werden. |
| LOAD IMAGE/BMP fname$ [,x] [,y] [,mode] [,ximage] [,yimage] | Lädt eine BMP-Datei aus „fname$“ und schreibt sie auf das aktuelle Ausgabegerät (Bildschirm oder Framebuffer), beginnend bei „x“, „y“ (Standardwert ist 0,0).<br>Der optionale Parameter „mode“ gibt an, ob die Ausgabe gedithert werden soll. Die Bits 0 und 1 legen das Ausgabeformat fest, und Bit 2 bestimmt die Art des zu verwendenden Dithering.<br>Standardmäßig ist „mode“ auf -1 gesetzt, was bedeutet, dass kein Dithering angewendet werden soll.<br>DITHER_FLOYD_STEINBERG_RGB121 0<br>DITHER_FLOYD_STEINBERG_RGB222 1<br>DITHER_FLOYD_STEINBERG_RGB332 2<br>DITHER_ATKINSON_RGB121 4<br>DITHER_ATKINSON_RGB222 5<br>DITHER_ATKINSON_RGB332 6<br>„ximage“ und „yimage“ geben an, an welcher Stelle in der Bilddatei mit dem Schreiben auf den Bildschirm begonnen werden soll (Standardwert ist 0,0)<br>Wenn keine Dateiendung angegeben wird, wird „.BMP“ an den Dateinamen angehängt.<br>Alle Arten des BMP-Formats werden unterstützt, einschließlich Schwarz-Weiß- und 24-Bit-True-Color-Bilder. |
| LOAD JPG file$ [, x] [, y] [,mode] [,ximage] [,yimage] | Lädt eine JPG-Datei aus „file$“ und schreibt sie auf das aktuelle Ausgabegerät (Bildschirm oder Framebuffer), beginnend bei „x“, „y“ (Standardwert ist 0,0).<br>Der optionale Parameter „mode“ legt fest, ob die Ausgabe gedithert werden soll.<br>Die Bits 0 und 1 legen das Ausgabeformat fest, und Bit 2 legt die Art des zu verwendenden Ditherings fest.<br>Standardmäßig ist „mode“ auf -1 gesetzt, was bedeutet, dass kein Dithering angewendet werden soll.<br>DITHER_FLOYD_STEINBERG_RGB121 0<br>DITHER_FLOYD_STEINBERG_RGB222 1<br>DITHER_FLOYD_STEINBERG_RGB332 2<br>DITHER_ATKINSON_RGB121 4<br>DITHER_ATKINSON_RGB222 5 |

| | |
|---|---|
| | DITHER_ATKINSON_RGB332 6<br>„ximag“ und „yimage“ geben an, an welcher Stelle in der Bilddatei mit dem Schreiben auf den Bildschirm begonnen werden soll (Standardwert ist 0,0)<br>Wenn keine Dateiendung angegeben wird, wird „.JPG“ an den Dateinamen angehängt.<br>Progressive JPG-Bilder werden nicht unterstützt. |
| LOAD PNG fname$ [, x] [, y] [,transparent] [,alphacut] | NUR RP2350-VERSIONEN<br>Lädt und zeigt eine PNG-Datei „fname“ an<br>Wenn keine Dateiendung angegeben wird, wird automatisch „.png“ an den Dateinamen angehängt.<br>Die Datei muss im RGBA8888-Format vorliegen, was die normale Standardeinstellung ist. Falls angegeben, geben „x“ und „y“ an, wo auf dem Bildschirm oder im Framebuffer das Bild erscheinen soll.<br>Der optionale Parameter „transparent“ (Standardwert 0) gibt einen der Farbcodes (0–15) an, der den Pixeln in der PNG-Datei mit einem Alpha-Wert unter „alphacut“ (Standardwert 20) zugewiesen wird. Wenn „transparent“ auf -1 gesetzt ist, wird das PNG-Bild so geschrieben, dass Pixel mit einem Alpha-Wert unter „alphacut“ komplett ausgelassen werden. |
| LOCAL variable [, variables]<br>See DIM for the full syntax. | Definiert eine Liste von Variablennamen als lokal für die Subroutine oder Funktion. Dieser Befehl verwendet genau dieselbe Syntax wie DIM und erstellt Variablen, die nur innerhalb der Subroutine oder Funktion sichtbar sind. Sie werden automatisch verworfen, wenn die Subroutine oder Funktion beendet wird. |
| LOCATION | Legt einen manuellen Standort (Breitengrad/Längengrad) und eine Uhrzeit für die Verwendung mit dem ASTRO-Befehl fest.<br>Dies ist einer aus einer Reihe von Befehlen, die hochpräzise astronomische Berechnungen durchführen, die sich für die Ausrichtung von Teleskopen oder zur Navigation eignen, und wird ausführlich in der Datei *GPS_Astro_Reference.pdf* beschrieben, die im ZIP-Download der Firmware enthalten ist. |
| LONGSTRING | Mit den LONGSTRING-Befehlen kannst du Zeichenfolgen bearbeiten, die länger sind als die normale MMBasic-Grenze von 255 Zeichen.<br>Variablen für lange Zeichenfolgen müssen als eindimensionale Integer-Arrays definiert werden, wobei die Anzahl der Elemente auf die für die maximale Zeichenfolgenlänge erforderliche Zeichenanzahl geteilt durch acht gesetzt wird. Der Grund für die Division durch acht ist, dass jedes Integer in einem MMBasic-Array acht Bytes belegt. Beachte, dass die Routinen für lange Zeichenfolgen nicht auf Überlauf bei der Länge der Zeichenfolgen prüfen. Wenn versucht wird, eine Zeichenfolge zu erstellen, die länger ist als die Größe einer Variablen für lange Zeichenfolgen, ist das Ergebnis undefiniert. |
| LONGSTRING AES128 ENCRYPT/DECRYPT CBC/ECB/CTR key$/key[!/%](), in%(), out%() [,iv$/iv[!/%]()] | Verschlüsselt oder entschlüsselt den Longstring in in%() und speichert das Ergebnis in out%()<br>Bei den Modi CBC und CTR generiert die Verschlüsselung einen zufälligen Initialisierungsvektor und hängt diesen an out%() an. Wenn ein expliziter IV angegeben wird, wird dieser anstelle des zufälligen Vektors verwendet und an out%() angehängt<br>Bei der CBC- und CTR-Entschlüsselung geht die Firmware davon aus, dass die ersten 16 Bytes von in%() den Initialisierungsvektor bilden.<br>Falls du eine Nachricht ohne IV übertragen möchtest, kannst du LONGSTRING RIGHT verwenden, um den IV vor dem Senden der Nachricht zu entfernen. In diesem Fall muss der Empfänger sowohl den IV als auch den Schlüssel kennen und einen vollständigen Longstring erstellen, bevor er |

| | |
|---|---|
| | DECRYPT verwendet. Dies kann durch die Verwendung von LONGSTRING CONCAT erfolgen, um die eingehende Nachricht an einen Longstring anzuhängen, der den IV enthält. |
| LONGSTRING APPEND array%(), string$ | Fügt eine normale MMBasic-Zeichenkette an eine Longstring-Variable an. „array%()“ ist eine Longstring-Variable, während „string$“ ein normaler MMBasic-Zeichenkettenausdruck ist. |
| LONGSTRING BASE64 ENCODE/DECODE in%(), out%() | Diese Funktion codiert oder decodiert den Longstring in in%() mit BASE64 und speichert das Ergebnis in out%(). Das als Ausgabe verwendete Array muss im Verhältnis zur Eingabe und zur Richtung groß genug sein. Die Codierung erhöht die Länge um 4/3, die Decodierung verringert sie um 3/4 |
| LONGSTRING CLEAR array%() | Löscht die Longstring-Variable „array%()“, d. h., sie wird auf eine leere Zeichenkette gesetzt. |
| LONGSTRING COPY dest%(), src%() | Kopiert eine lange Zeichenkette in eine andere. „dest%()“ ist die Zielvariable und „src%()“ die Quellvariable. Der bisherige Inhalt von „dest%()“ wird überschrieben. |
| LONGSTRING CONCAT dest%(), src%() | Verkettet eine lange Zeichenkette mit einer anderen. „dest%()“ ist die Zielvariable und „src%()“ die Quellvariable. „src%()“ wird an das Ende von „dest%()“ angehängt (das Ziel wird **nicht** überschrieben). |
| LONGSTRING LCASE array%() | Wandelt alle Großbuchstaben in „array%()“ in Kleinbuchstaben um. „array%()“ muss eine Variable für lange Zeichenfolgen sein. |
| LONGSTRING LEFT dest%(), src%(), nbr | Kopiert die ersten „nbr“ Zeichen von „src%()“ nach „dest%()“ und überschreibt dabei den bisherigen Inhalt von „dest%()“. , d. h. es wird vom Anfang von „src%()“ kopiert. „src%()“ und „dest%()“ müssen lange Zeichenfolgenvariablen sein. „nbr“ muss eine ganzzahlige Konstante oder ein Ausdruck sein. |
| LONGSTRING LOAD array%(), nbr, string$ | Kopiert „nbr“ Zeichen aus „string$“ in die lange Zeichenfolgenvariable „array%()“ und überschreibt dabei den bisherigen Inhalt von „array%()“. |
| LONGSTRING MID dest%(), src%(), start, nbr | Kopiert 'nbr' Zeichen von ‘src%()’ nach ‘dest%()’, beginnend an der Zeichenposition 'start', und überschreibt dabei den bisherigen Inhalt von ‘dest%()’. D. h. es wird ab der Mitte von ‘src%()’ kopiert. 'nbr' ist optional; wenn es weggelassen wird, werden die Zeichen von 'start' bis zum Ende der Zeichenkette kopiert. ‘src%()’ und ‘dest%()’ müssen Long-String-Variablen sein. 'start' und 'nbr' müssen ganzzahlige Konstanten oder Ausdrücke sein. |
| LONGSTRING PRINT [#n,] src%() [;] | Gibt den in „src%()“ gespeicherten Langstring in die als „#n“ geöffnete Datei oder den COM-Port aus. Wenn „#n“ nicht angegeben ist, wird die Ausgabe an die Konsole gesendet. Füge ein Semikolon hinzu, um CR/LF zu unterdrücken. |
| LONGSTRING REPLACE array%() , string$, start | Ersetzt Zeichen in der normalen MMBasic-Zeichenkette „string$“ in einer bestehenden Longstring-Zeichenkette „array%()“, beginnend an der Position „start“ in der Longstring-Zeichenkette. |
| LONGSTRING RESIZE addr%(), nbr | Stellt die Größe des Longstrings auf „nbr“ ein. Dies überschreibt die durch andere Longstring-Befehle festgelegte Größe und sollte daher mit Vorsicht verwendet werden. Eine typische Anwendung wäre die Verwendung eines Longstrings als Byte-Array. |
| LONGSTRING RIGHT dest%(), src%(), nbr | Kopiert die rechten „nbr“-Zeichen von „src%()“ nach „dest%()“ und überschreibt dabei den bisherigen Inhalt von „dest%()“. D. h. es wird vom Ende von „src%()“ kopiert. „src%()“ und „dest%()“ müssen Long-String-Variablen sein. „nbr“ muss eine ganzzahlige Konstante oder ein Ausdruck sein. |

| | |
|---|---|
| LONGSTRING SETBYTE addr%(), nbr, data | Setzt das Byte „nbr“ auf den Wert „data“, wobei „nbr“ die Option BASE berücksichtigt |
| LONGSTRING TRIM array%(), nbr | Entfernt „nbr“ Zeichen vom Anfang einer langen Zeichenkette. „array%()“ muss eine lange Zeichenkette sein. „nbr“ muss eine ganzzahlige Konstante oder ein Ausdruck sein. |
| LONGSTRING UCASE array%() | Wandelt alle Kleinbuchstaben in „array%()“ in Großbuchstaben um. „array%()“ muss eine Variable vom Typ „long string“ sein. |
| LOOP [UNTIL expression] | Beendet eine Programmschleife: siehe DO. |
| MAP | NUR HDMI-VERSION<br>Mit den MAP-Befehlen kann der Programmierer die Farben festlegen, die in 4- oder 8-Bit-Farbmodi verwendet werden. Jeder Wert in der 4- oder 8-Bit-Farbpalette kann auf eine unabhängige 24-Bit-Farbe (d. h. im RGB555-Format) gesetzt werden. Weitere Informationen findest du unter der MAP-Funktion |
| MAP( n ) = rgb% | Dadurch wird allen Pixeln, deren 4- oder 8-Bit-Farbwert „n“ beträgt, die 24-Bit-Farbe „rgb%“ zugewiesen. Die Änderung wird nach dem Befehl „MAP SET“ aktiviert. |
| MAP MAXIMITE | Dadurch wird die Farbzuordnung auf die Farben eingestellt, die im ursprünglichen Colour Maximite implementiert sind. |
| MAP GREYSCALE | Damit wird die Farbzuordnung auf 16 oder 32 Graustufen gesetzt (abhängig vom MODE). MAP GRAYSCALE ist ebenfalls gültig. |
| MAP SET | Dadurch aktualisiert MMBasic die Farbzuordnung (festgelegt mit MAP(n)=rgb%) während des nächsten Bildausblendintervalls. |
| MAP RESET | Damit wird die Farbkarte auf die Standardfarben zurückgesetzt |
| MAP | NUR FÜR PICOMITE RP2350-TREIBER MIT Puffer<br>Mit den MAP-Befehlen kann der Programmierer die im 8-Bit-RGB332-Farbmodus verwendeten Farben festlegen. Jeder Wert in der 8-Bit-Farbpalette kann auf eine unabhängige 24-Bit-Farbe (d. h. im RGB888-Format) eingestellt werden. Weitere Informationen findest du unter der MAP-Funktion |
| MAP( n ) = rgb% | Dadurch wird allen Pixeln mit dem 8-Bit-Farbwert „n“ die 24-Bit-Farbe „rgb%“ zugewiesen. Die Änderung wird nach dem Befehl MAP SET aktiviert. |
| MAP SET | Dadurch aktualisiert MMBasic die Farbzuordnung (festgelegt mit MAP(n)=rgb%) während des nächsten Bildausblendintervalls. |
| MAP RESET | Dies setzt die Farbkarte auf die Standardfarben zurück |
| MAP | NUR VGA-VERSION<br>Mit den MAP-Befehlen kann der Programmierer die Farben auswählen, die in 4-Bit-Farbmodi verwendet werden. Jeder Wert in der 4-Bit-Farbpalette kann auf eine der 16 verfügbaren Farben gesetzt werden. Weitere Informationen findest du unter der MAP-Funktion. |

MAP( n ) = rgb%

Dadurch wird allen Pixeln mit dem 4-Bit-Farbwert „n“ die 24-Bit-Farbe „rgb%“ zugewiesen. Der RGB-Wert wird in eine der 16 verfügbaren VGA-RGB121-Farben umgewandelt, wie sie durch das Widerstandsnetzwerk festgelegt sind. Die Änderung wird nach dem Befehl MAP SET aktiviert.

MAP MAXIMITE

Damit wird die Farbzuordnung auf die Farben eingestellt, die im ursprünglichen Colour Maximite implementiert waren.

MAP SET

Dadurch aktualisiert MMBasic die Farbzuordnung (eingerichtet mit MAP(n)=rgb%) während des nächsten Bildausblendintervalls.

MAP RESET

Damit wird die Farbzuordnung auf die Standardfarben zurückgesetzt, die im 4-Bit-Modus wie folgt lauten:

| ‘n’ | Colour | Value |
|---|---|---|
| 15 | WHITE | RGB(255, 255, 255) |
| 14 | YELLOW | RGB(255, 255, 0) |
| 13 | LILAC | RGB(255, 128, 255) |
| 12 | BROWN | RGB(255, 128, 0) |
| 11 | FUCHSIA | RGB(255, 64, 255) |
| 10 | RUST | RGB(255, 64, 0) |
| 9 | MAGENTA | RGB(255, 0, 255) |
| 8 | RED | RGB(255, 0, 0) |
| 7 | CYAN | RGB(0, 255, 255) |
| 6 | GREEN | RGB(0, 255, 0) |
| 5 | CERULEAN | RGB(0, 128, 255) |
| 4 | MIDGREEN | RGB(0, 128, 0) |
| 3 | COBALT | RGB(0, 64, 255) |
| 2 | MYRTLE | RGB(0, 64, 0) |
| 1 | BLUE | RGB(0, 0, 255) |
| 0 | BLACK | RGB(0, 0, 0) |

MATH

Der Befehl „math“ führt viele einfache mathematische Berechnungen durch, die man in BASIC programmieren könnte, aber es gibt Geschwindigkeitsvorteile, wenn man Schleifenstrukturen in der Firmware codiert, und es hat den Vorteil, dass sie nach dem Debuggen für alle da sind, ohne das Rad neu erfinden zu müssen. Hinweis: Zweidimensionale mathematische Matrizen werden immer als DIM matrix(n_columns, n_rows) angegeben, und natürlich richten sich die Dimensionen nach OPTION BASE. Quaternionen werden als 5-Element-Array w, x, y, z, magnitude gespeichert.

MATH RANDOMIZE [n]

Setzt den Mersenne-Twister-Algorithmus.
Wenn n nicht angegeben ist, ist der Startwert die Zeit in Mikrosekunden seit dem Booten
Der Mersenne-Twister-Algorithmus liefert viel bessere Zufallszahlen als die integrierte Funktion der C-Bibliothek. Hinweis: Der RP2350 verfügt über einen Hardware-Zufallszahlengenerator.

**Einfache Array-Arithmetik**

MATH SET nbr, array()

Siehe ARRAY SET

MATH SCALE in(), scale ,out()

Dies skaliert die Matrix in() mit dem Skalar scale und speichert das Ergebnis in out(). Funktioniert für Arrays beliebiger Dimension sowohl mit ganzzahligen als auch mit Fließkommazahlen und kann zwischen diesen konvertieren. Die Einstellung von b auf 1 ist optimiert und stellt die schnellste Methode zum Kopieren eines gesamten Arrays dar.

| | |
|---|---|
| MATH ADD in(), num, out() | Siehe ARRAY ADD |
| MATH INTERPOLATE in1(), in2(), ratio, out() | Dieser Befehl wendet die folgende Gleichung auf jedes Array-Element an:<br>out = (in2 - in1) * ratio + in1<br>Arrays können beliebig viele Dimensionen haben, müssen jedoch unterschiedlich sein und die gleiche Gesamtanzahl an Elementen aufweisen. Der Befehl funktioniert sowohl mit Integer- als auch mit Gleitkomma-Arrays in beliebiger Mischung |
| MATH WINDOW in(), minout, maxout, out() [,minin!, maxin!] | Dieser Befehl nimmt das „in“-Array und skaliert es zwischen „minout“ und „maxout“, wobei das Ergebnis in „out“ zurückgegeben wird. Optional kann er auch die minimalen und maximalen Gleitkommawerte aus den Originaldaten zurückgeben („minin!“ und „maxin!“). Hinweis: „minout“ kann größer als „maxout“ sein; in diesem Fall werden die Daten sowohl skaliert als auch invertiert.<br>Dieser Befehl kann das Skalieren von Daten für Diagramme usw. erheblich vereinfachen.<br>Beispiel:<br>DIM IN(2)=(1,2,3)<br>DIM OUT(2)<br>MATH WINDOW IN(),7,3,OUT(),LOW,HIGH<br>Gibt OUT(0)=7, OUT(1)=5, OUT(2)=3, LOW=1, HIGH=3 zurück |
| MATH SLICE sourcearray(), [d1] [,d2] [,d3] [,d4] [,d5] , destinationarray() | Siehe ARRAY SLICE |
| MATH INSERT targetarray(), [d1] [,d2] [,d3] [,d4] [,d5] , sourcearray() | Siehe ARRAY INSERT |
| MATH POWER inarray(), power, outarray() | Erhebt jedes Element in „inarray()“ auf die definierte „Potenz“ und speichert das Ergebnis in „outarray()“ |
| MATH SHIFT inarray%(), nbr, outarray%() [,U] | Dieser Befehl führt eine Bitverschiebung für alle Elemente von inarray%() durch und speichert das Ergebnis in outarray%() (kann mit inarray%() identisch sein). nbr kann zwischen -63 und 63 liegen. Positive Zahlen bedeuten eine Linksverschiebung (Multiplikation mit einer 2er-Potenz). Negative e Zahlen bedeuten eine Rechtsverschiebung. Der optionale Parameter ,U erzwingt eine vorzeichenlose Verschiebung. |
| **Matrixarithmetik** | |
| MATH M_INVERSE array!(), inversearray!() | Dies gibt die Inverse von array!() in inversearray!() zurück. Das Array muss quadratisch sein, und du erhältst eine Fehlermeldung, wenn das Array nicht invertiert werden kann (Determinante = 0). array!() und inversearray!() dürfen nicht identisch sein. |
| MATH M_PRINT array() | Schnelle Methode, um eine 2D-Matrix mit einer Zeile pro Zeile auszugeben. |
| MATH M_TRANSPOSE in(), out() | Transponiere die Matrix in() und speichere das Ergebnis in der Matrix out(). Beide Arrays müssen zweidimensional sein, müssen aber nicht quadratisch sein. Wenn sie nicht quadratisch sind, müssen die Arrays die Dimensionen in(m,n) out(n,m) haben |
| MATH M_MULT in1(), in2(), out() | Multipliziere die Arrays in1() und in2() und speichere das Ergebnis in out()c. Alle Arrays müssen zweidimensional sein, müssen aber nicht quadratisch sein. |

| | |
|---|---|
| | Wenn sie nicht quadratisch sind, müssen die Arrays die Dimensionen in1(m,n), in2(p,m) und out(p,n) haben |
| **Vektor-Arithmetik** | |
| MATH V_PRINT array() [,hex] | Schneller Mechanismus, um ein kleines Array in einer einzigen Zeile auszugeben. „hex“ gibt es im Hexadezimalformat aus. |
| MATH V_NORMALISE inV(), outV() | Konvertiert einen Vektor inV() auf Einheitsskala und speichert das Ergebnis in outV()<br>(sqr(x*x + y*y +....)=1<br>Die Anzahl der Elemente im Vektor ist unbegrenzt |
| MATH V_MULT matrix(), inV(), outV() | Multipliziert matrix() und den Vektor inV() und gibt den Vektor outV() zurück. Die Vektoren und die 2D-Matrix können beliebig groß sein, müssen aber die gleiche Kardinalität haben. |
| MATH V_CROSS inV1(), inV2(), outV() | Berechnet das Kreuzprodukt zweier Vektoren mit jeweils drei Elementen inV1() und inV2() und speichert das Ergebnis in outV() |
| MATH V_ROTATE x, y, a, xin(), yin(), xout(), yout() | Dieser Befehl dreht die Koordinatenpaare in „xin()“ und „yin()“ um den durch „x“ und „y“ definierten Mittelpunkt um den Winkel „a“ und speichert die Ergebnisse in „xout()“ und „yout()“. Hinweis: Die Eingabe- und Ausgabe-Arrays können identisch sein, und der Drehwinkel ist standardmäßig in Radianten angegeben, dies kann jedoch mit dem Befehl OPTION ANGLE geändert werden. |
| **Quaternion-Arithmetik** | |
| MATH Q_INVERT inQ(), outQ() | Invertiere das Quaternion in inQ() und speichere das Ergebnis in outQ() |
| MATH Q_VECTOR x, y, z, outVQ() | Konvertiert einen durch x, y und z angegebenen Vektor in einen normalisierten Quaternion-Vektor outVQ(), wobei die ursprüngliche Größe gespeichert wird |
| MATH Q_CREATE theta, x, y, z, outRQ() | Erzeugt ein normalisiertes Rotationsquaternion outRQ(), um Quaternion-Vektoren um die Achsen x, y, z um einen Winkel theta zu drehen. Theta wird in Radianten angegeben. |
| MATH Q_EULER yaw, pitch, roll, outRQ() | Erzeugt ein normiertes Rotationsquaternion outRQ(), um Quaternion-Vektoren gemäß den Gier-, Nick- und Rollwinkeln zu drehen<br>Wenn der Vektor vor dem „Betrachter“ liegt, blickt Yaw von der Spitze des Vektors aus und dreht im Uhrzeigersinn, Pitch dreht die Spitze von der Kamera weg und Roll dreht um die z-Achse im Uhrzeigersinn.<br>Die Gier-, Nick- und Rollwinkel sind standardmäßig in Radianten angegeben, berücksichtigen aber die Einstellung von OPTION ANGLE |
| MATH Q_MULT inQ1(), inQ2(), outQ() | Multipliziert zwei Quaternionen inQ1() und inQ2() und speichert das Ergebnis in outQ() |
| MATH Q_ROTATE , RQ(), inVQ(), outVQ() | Dreht den Quell-Quaternion-Vektor inVQ() um den Dreh-Quaternion RQ() und speichert das Ergebnis in outVQ() |

| | |
|---|---|
| MATH C_ADD array1(), array2(), array3()<br>MATH C_SUB array1(), array2(), array3()<br>MATH C_MUL array1(), array2(), array3()<br>MATH C_DIV array1(), array2(), array3()<br>MATH C_AND array1(), array2(), array3()<br>MATH C_OR array1(), array2(), array3()<br>MATH C_XOR array1(), array2(), array3() | Diese Befehle führen zellenweise Operationen an Arrays durch.<br>array1 und array2 sind die Parameter und array3 ist die Ausgabe. Alle Arrays müssen dieselbe Größe und denselben Typ haben (Float oder Integer).<br>Es gibt keine Einschränkungen hinsichtlich der Anzahl der Dimensionen und keine Einschränkungen bei der Verwendung desselben Arrays zweimal oder sogar dreimal in den Parametern.<br>z. B. MATH C_MUL a%(),a%(),a%()<br>quadriert alle Werte im Array a%() |
| MATH FFT signalarray!(), FFTarray!() | Führt eine schnelle Fourier-Transformation der Daten in „signalarray!“ durch. „signalarray“ muss ein Gleitkomma-Array sein und die Größe muss eine Potenz von 2 sein (z. B. s(1023), vorausgesetzt, OPTION BASE ist Null)<br>„FFTarray“ muss ein Gleitkomma-Wert sein und die Dimension 2*N haben, wobei N dem Signal-Array entspricht (z. B. f(1,1023), vorausgesetzt, OPTION BASE ist Null)<br>Der Befehl gibt die FFT als komplexe Zahlen zurück, wobei der Realteil in f(0,n) und der Imaginärteil in f(1,n) enthalten ist |
| MATH FFT INVERSE FFTarray!(), signalarray!() | Führt eine inverse schnelle Fourier-Transformation der Daten in „FFTarray!“ durch. „FFTarray“ muss ein Gleitkomma-Wert sein und die Dimension 2*N haben, wobei N eine Potenz von 2 sein muss (z. B. f(1,1023), vorausgesetzt, OPTION BASE ist Null), wobei der Realteil in f(0,n) und der Imaginärteil in f(1,n) enthalten ist.<br>„signalarray“ muss ein Gleitkomma-Array sein und die einzige Dimension muss mit der des FFT-Arrays übereinstimmen.<br>Der Befehl gibt den Realteil der inversen Transformation in „signalarray“ zurück. |
| MATH FFT MAGNITUDE signalarray!(),magnitudearray!() | Erzeugt Amplituden für die Frequenzen der Daten in „signalarray!“<br>„signalarray“ muss ein Gleitkomma-Wert sein und die Größe muss eine Potenz von 2 sein (z. B. s(1023), vorausgesetzt, OPTION BASE ist Null). „magnitudearray“ muss ein Gleitkomma-Wert sein und die Größe muss mit der des Signal-Arrays übereinstimmen<br>Der Befehl gibt die Amplitude des Signals bei verschiedenen Frequenzen gemäß der folgenden Formel zurück:<br>Frequenz an Array-Position N = N * Abtastfrequenz / Anzahl der Abtastwerte |
| MATH FFT PHASE signalarray!(), phasearray!() | Erzeugt Phasen für Frequenzen der Daten in „signalarray!“.<br>„signalarray“ muss ein Gleitkomma-Wert sein und die Größe muss eine Potenz von 2 sein (z. B. s(1023), vorausgesetzt, OPTION BASE ist Null). „phasearray“ muss ein Gleitkomma-Wert sein und die Größe muss mit der des Signal-Arrays übereinstimmen<br>Der Befehl gibt den Phasenwinkel des Signals bei verschiedenen Frequenzen gemäß der obigen Formel zurück. |
| MATH SENSORFUSION Typ ax, ay, az, gx, gy, gz, mx, my, mz, pitch, roll, yaw [,p1] [,p2] | Typ kann MAHONY oder MADGWICK sein<br>Ax, ay und az sind die Beschleunigungen in den drei Richtungen und sollten in Einheiten der Standardbeschleunigung angegeben werden.<br>Gx, gy und gz sind die Momentanwerte der Drehgeschwindigkeit, die in Radiant pro Sekunde angegeben werden sollten.<br>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 bei den drei Eingaben konsistent sind. So lautet beispielsweise bei Verwendung des MPU-9250 die korrekte Eingabe für „ “ ax, ay, az, gx, gy, gz, my, mx, -mz, basierend auf den Messwerten des Sensors.<br>Pitch, Roll und Yaw sollten Gleitkommavariablen sein und die Ergebnisse der Sensorfusion enthalten.<br>Die Routine SENSORFUSION misst automatisch die Zeit zwischen aufeinanderfolgenden Aufrufen und verwendet diese für ihre internen Berechnungen.<br>Der Madwick-Algorithmus nimmt einen optionalen Parameter p1 entgegen. Dieser wird in der Berechnung als Beta verwendet. Wenn er nicht angegeben wird, ist der Standardwert 0,5<br>Der Mahony-Algorithmus benötigt zwei optionale Parameter p1 und p2. Diese werden in der Berechnung als Kp und Ki verwendet. Wenn sie nicht angegeben werden, sind die Standardwerte 10,0 bzw. 0,0.<br>Ein vollständig ausgearbeitetes Beispiel für die Verwendung des Codes findest du im BackShed-Forum unter: https://www.thebackshed.com/forum/ViewTopic.php?TID=13459&PID=166962#166962 |
| MATH SINC x_in(), y_in(), n, m, window, freq, x_out(), y_out()<br><br>MATH SINC x_in(), y_in(), n, window, freq, x_out(), y_out() | Wende einen Sinc-Filter mit Fensterfunktion an, um Koordinatendaten zu glätten oder zu interpolieren.<br>Ein Sinc-Filter ist ein idealer Tiefpassfilter, der hochfrequentes Rauschen entfernt und dabei die Signalform beibehält. Er eignet sich besonders gut für das Resampling (Interpolation).<br>Parameter:<br>x_in(), y_in() — Eingabe-Koordinaten-Arrays (float)<br>n — Anzahl der Eingangspunkte<br>m — Anzahl der Ausgabepunkte (optional). Wenn weggelassen oder m=n: Glättungsmodus (Ausgabegröße n). Wenn m!=n: Interpolations-/Resampling-Modus (Ausgabegröße m)<br>window — Filterkernelgröße (muss ungerade sein; gerade Werte werden erhöht). Typische Werte: 15 bis 101. Größere Werte sorgen für eine schärfere Flankensteilheit, sind aber langsamer.<br>freq — Normalisierte Grenzfrequenz (0,0 < freq <= 0,5). 0,5 ist die Nyquist-Frequenz (keine Filterung). Typische Werte: 0,1 (stark geglättet) bis 0,4 (leicht geglättet).<br>x_out(), y_out() — Ausgangskoordinaten-Arrays (müssen groß genug sein, um den Ausgang zu fassen) |
| MATH PID INIT channel, pid_params!(), callback | Dieser Befehl richtet einen PID-Regler ein, der automatisch im Hintergrund arbeiten kann. Es können bis zu 8 PID-Regler gleichzeitig laufen (Kanäle 1 bis 8)<br>„callback“ ist eine MMbasic-Subroutine, die mit der durch die Abtastzeit definierten Frequenz aufgerufen wird. Siehe die Funktion MATH(PID …) für Details dazu, was in der Subroutine enthalten sein sollte.<br>Das Array pid_params!() muss für alle aufgeführten Elemente dimensioniert sein, einschließlich der Regler-Speicherparameter (DIM pid_params!(13)), und mit den erforderlichen Einstellungen initialisiert werden.<br>PID-Konfiguration<br>Element 0 = Kp<br>Element 1 = Ki<br>Element 2 = Kd |

| | |
|---|---|
| | Element 3 = tau ' Zeitkonstante des Ableitungs-Tiefpassfilters<br>Element 4 = limMin 'Ausgangsgrenzen<br>Element 5 = limMax<br>Element 6 = limMinInt 'Integrator-Grenzwerte<br>Element 7 = limMaxInt<br>Element 8 = T 'Abtastzeit (in Sekunden)<br>Controller-„Speicher“<br>Element 9 = Integrator<br>Element 10 = prevError<br>Element 11 = Differenzierer<br>Element 12 = vorherigeMessung<br>Element 13 = out |
| MATH PID START channel | Startet einen zuvor initialisierten PID-Regler auf dem angegebenen Kanal |
| MATH PID STOP channel | Stoppt einen zuvor initialisierten PID-Regler auf dem angegebenen Kanal und löscht die internen Datenstrukturen<br>Siehe https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=17263<br>Ein Beispiel für die Einrichtung und den Betrieb eines PID-Reglers |
| MATH AES128 ENCRYPT/DECRYPT CBC/ECB/CTR key$/key(), in$/in(), out$/out() [,iv$/iv()] | Dieser Befehl verschlüsselt oder entschlüsselt die Daten in 'in' und speichert das Ergebnis in 'out' unter Verwendung der angegebenen AES128-Verschlüsselungsmethode<br>Die Parameter können jeweils entweder eine Zeichenkette, ein Integer-Array oder ein Float-Array sein, wobei jede beliebige Kombination möglich ist<br>Der Schlüssel muss 16 Elemente lang sein (16*8=128 Bit), „in“ und „out“ müssen ein Vielfaches von 16 Elementen lang sein. Falls „out“ als Zeichenkette angegeben wird (z. B. „out$“), muss die Zeichenkettenvariable existieren und sollte auf leer gesetzt sein (DIM out$="")<br>Die maximale Anzahl der Elemente in 'in' und 'out' ist durch den Speicher begrenzt, wenn sie als Arrays definiert sind. Zeichenfolgen für die Verschlüsselung sind auf 240 Byte (EBR) und 224 Byte (CTR und CBC) begrenzt.<br>Bei der CBC- und CTR-Verschlüsselung kannst du optional einen Initialisierungsvektor „iv“ angeben. „iv“ muss 16 Elemente lang sein (16 × 8 = 128 Bit). Wenn kein Initialisierungsvektor angegeben wird, generiert die Verschlüsselung einen zufälligen Initialisierungsvektor.<br>In beiden Fällen wird der IV der Ausgabe vorangestellt.<br>Bei CBC und CTR setzt die Entschlüsselung voraus, dass die ersten 16 Elemente der Eingabe den Initialisierungsvektor bilden.<br>Falls du eine Nachricht ohne IV übertragen möchtest, solltest du den IV vor dem Senden der Nachricht mithilfe von Standard-MMBasic-Manipulationen entfernen. In diesem Fall muss der Empfänger sowohl den IV als auch den Schlüssel kennen und eine vollständige Nachricht erstellen, bevor er DECRYPT verwendet, indem er den IV an den Anfang der eingehenden Nachricht setzt. |
| MEMORY | Zeige die derzeit belegte Speichermenge an. Beispiel:<br>`Program:`<br>`0K ( 0%) Program (0 lines)`<br>`180K (100%) Free`<br>`Saved Variables:`<br>`16K (100%) Free` |

| | |
|---|---|
| | RAM:<br>0K ( 0%) 0 Variables<br>0K ( 0%) General<br>228K (100%) Free<br><br>Hinweise:<br>• Die Speicherauslastung wird auf die nächsten 1K Byte gerundet.<br>• Der allgemeine Speicher (RAM) wird von Arrays, Strings, seriellen E/A-Puffern usw. genutzt. |
| MEMORY SET address, byte, numberofbytes<br><br>MEMORY SET BYTE address, byte, numberofbytes<br><br>MEMORY SET SHORT address, short, numberofshorts<br><br>MEMORY SET WORD address, word, numberofwords<br><br>MEMORY SET INTEGER address, integervalue ,numberofintegers [,increment]<br><br>MEMORY SET FLOAT address, floatingvalue ,numberofloats [,increment] | Dieser Befehl setzt einen Speicherbereich auf einen Wert.<br>BYTE = Ein Byte pro Speicheradresse.<br>SHORT = Zwei Bytes pro Speicheradresse.<br>WORD = Vier Bytes pro Speicheradresse.<br>FLOAT = Acht Bytes pro Speicheradresse.<br>„increment“ ist optional und steuert die Schrittweite des Zeigers „address“ während der Ausführung der Operation. Wenn beispielsweise increment=3 ist, wird nur jedes dritte Element des Ziels gesetzt. Der Standardwert ist 1. |
| MEMORY COPY sourceaddress, destinationaddres, numberofbytes [,sourceincrement][,destination increment]<br><br>MEMORY COPY INTEGER sourceaddress, destinationaddress, numberofintegers [,sourceincrement][,destination increment]<br><br>MEMORY COPY FLOAT sourceaddress, destinationaddress, numberoffloats [,sourceincrement][,destination increment] | Dieser Befehl kopiert einen Speicherbereich in einen anderen.<br>COPY INTEGER und FLOAT kopieren acht Bytes pro Operation.<br>„sourceincrement“ ist optional und steuert die Inkrementierung des Zeigers „sourceaddress“ während der Ausführung des Befehls. Wenn beispielsweise sourceincrement=3 ist, wird nur jedes dritte Element der Quelle kopiert. Der Standardwert ist 1.<br>„destinationincrement“ funktioniert ähnlich und wirkt sich auf den Zeiger „destinationaddress“ aus. |

| | |
|---|---|
| MEMORY PRINT #]fnbr , nbr, address%/array()<br><br>MEMORY INPUT [#]fnbr , nbr, address%/array() | Diese Befehle speichern oder lesen „nbr“ Datenbytes aus dem Speicher oder in den Speicher bzw. aus einer oder in eine geöffnete Datei.<br>Der zu speichernde Speicher kann als Ganzzahl-Array angegeben werden; in diesem Fall wird die Anzahl der zu speichernden oder zu lesenden Bytes mit der Array-Größe abgeglichen. Alternativ kann eine Speicheradresse verwendet werden; in diesem Fall findet keine Überprüfung statt und Benutzerfehler könnten zu einem Absturz der Firmware führen. |
| MEMORY PACK source%()/sourceaddress%, dest%()/destaddress%, number, size<br><br>MEMORY UNPACK source%()/sourceaddress%, dest%()/destaddress%, number, size | Mit „Memory pack“ und „unpack“ können ganzzahlige Werte aus einem Array in ein anderes komprimiert oder von einem in das andere dekomprimiert werden.<br>Die beiden Arrays sind immer normale Integer-Arrays, aber das gepackte Array kann 2, 4, 8, 16 oder 64 Werte „in sich gepackt“ haben. Somit könnte ein einzelnes Integer-Array-Element 2 32-Bit-Wörter, 4 16-Bit-Werte, 8 Bytes, 16 Nibbles oder 64 Boolesche Werte (Bits) speichern.<br>„number“ gibt die Anzahl der zu packenden oder zu entpackenden Werte an und „size“ gibt die Anzahl der Bits (1, 4, 8, 16 oder 32) an<br>Alternativ können Speicheradressen verwendet werden; in diesem Fall findet keine Überprüfung statt und Benutzerfehler könnten zu einem Absturz der Firmware führen. |
| MKDIR dir$ | Erstelle das Verzeichnis „dir$“ auf dem Standard-Flash-Dateisystem oder der SD-Karte. |
| MID$( str$, start, num) = str2$ | Die „num“ Zeichen in „str$“, beginnend an der Position „start“, werden durch die Zeichen in „str2$“ ersetzt. „num“ kann auf 0 gesetzt werden, d. h. die neue Zeichenfolge wird an der angegebenen Position eingefügt. Wenn „num“ nicht angegeben wird, wird standardmäßig die Länge von „str2$“ verwendet |
| MODE 1<br>or<br>MODE 2<br>Or<br>MODE 3 (RP2350 only) | <u>NUR VGA-VERSIONEN</u><br>VGA-Video unterstützt eine Reihe von Auflösungen (siehe OPTION RESOLUTION). Dieser Befehl wählt je nach Auflösung den Modus „n“ aus:<br><u>OPTION RESOLUTION 640 x 480</u><br>MODE 1 640 x 480 x 2 Farben (monochrom). Standard beim Start. Die Kachelbreite ist fest auf 8 Pixel eingestellt. Die Kachelhöhe beträgt standardmäßig 12 Pixel, kann aber zwischen 8 und MM.HRES liegen. Die Kachelfarben werden mit der Standard-RGB888-Notation angegeben. Diese wird in RGB121 umgewandelt. Es können ein Framebuffer (F) und ein Layer-Buffer (L) erstellt werden. Diese haben keinen Einfluss auf die Anzeige und belegen keinen Benutzerspeicher, können aber beide zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden.<br>MODE 2 320 x 240 x 16 Farben. RGB121-Format (d. h. 1 Bit für Rot, 2 Bits für Grün und 1 Bit für Blau). Es kann ein Framebuffer (F) erstellt werden. Dieser hat keinen Einfluss auf die Anzeige und belegt keinen Benutzerspeicher, kann aber zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Buffer erstellt werden. Auch dieser beansprucht keinen Benutzerspeicher. Alle in den Layer-Puffer geschriebenen Pixel erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptbildschirmpuffer befindet. Es kann eine Farbe festgelegt werden (0–15; Standard ist 0), die nicht angezeigt wird, sodass der Hauptbildschirmpuffer durchscheinen kann. Es gibt eine Zuordnungsfunktion, mit der die |

| | |
|---|---|
| | Standardfarben der 16 verfügbaren Farben überschrieben werden können. Die Hardware ist auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind<br>MODE 3 640 x 480 x 16 Farben.<br>RGB121-Format (d. h. 1 Bit für Rot, 2 Bits für Grün und 1 Bit für Blau). Ein Framebuffer (F) kann erstellt werden. Dieser hat keinen Einfluss auf die Anzeige und belegt keinen Benutzerspeicher, kann aber zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Puffer erstellt werden. Auch dieser beansprucht keinen Benutzerspeicher. Alle in den Layer-Puffer geschriebenen Pixel erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptbildschirmpuffer befindet. Es kann eine Farbe festgelegt werden (0–15; Standard ist 0), die nicht angezeigt wird, sodass der Hauptbildschirmpuffer durchscheinen kann. Es gibt eine Zuordnungsfunktion, mit der die Standardfarben der 16 verfügbaren Farben überschrieben werden können. Die Hardware ist auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind<br>OPTION RESOLUTION 720 x 400<br>MODE 1 720 x 400 x 2 Farben (monochrom). Standard beim Start.<br>Die Kachelbreite ist auf 8 Pixel festgelegt. Die Kachelhöhe beträgt standardmäßig 12 Pixel, kann aber zwischen 8 und MM.HRES liegen. Die Kachelfarben werden mit der Standard-RGB888-Notation angegeben. Diese wird in RGB121 umgewandelt. Es können ein Framebuffer (F) und ein Layer-Buffer (L) erstellt werden. Diese haben keinen Einfluss auf die Anzeige und belegen keinen Benutzerspeicher, aber beide können zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden<br>MODE 2 360 x 200 x 16 Farben.<br>RGB121-Format (d. h. 1 Bit für Rot, 2 Bits für Grün und 1 Bit für Blau). Es kann ein Framebuffer (F) erstellt werden. Dieser hat keinen Einfluss auf die Anzeige und belegt keinen Benutzerspeicher, kann aber zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Buffer erstellt werden. Auch dieser belegt keinen Benutzerspeicher. Alle in den Layer-Puffer geschriebenen Pixel erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptbildschirmpuffer befindet. Es kann eine Farbe festgelegt werden (0–15; Standard ist 0), die nicht angezeigt wird, sodass der Hauptbildschirmpuffer durchscheinen kann. Es gibt eine Zuordnungsfunktion, um die Standardfarben der 16 verfügbaren Farben zu überschreiben. Bei VGA ist die Hardware auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind<br>MODE 3 720 x 400 x 16 Farben.<br>RGB121-Format (d. h. 1 Bit für Rot, 2 Bits für Grün und 1 Bit für Blau). Ein Framebuffer (F) kann erstellt werden. Dieser hat keinen Einfluss auf die Anzeige und belegt keinen Benutzerspeicher, kann aber zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Puffer erstellt werden. Auch dieser belegt keinen Benutzerspeicher. Alle in den Layer-Puffer geschriebenen Pixel erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptbildschirmpuffer befindet. Es kann eine Farbe festgelegt werden (0–15; Standard ist 0), die nicht angezeigt wird, sodass der Hauptbildschirmpuffer durchscheinen |

kann. Es gibt eine Zuordnungsfunktion, um die Standardfarben der 16 verfügbaren Farben zu überschreiben. Bei VGA ist die Hardware auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind

OPTION RESOLUTION 800 x 600 (nur RP2350)

MODE 1 800 x 600 x 2 Farben (monochrom). Standard beim Start. Die Kachelbreite ist auf 8 Pixel festgelegt. Die Kachelhöhe beträgt standardmäßig 12 Pixel, kann aber zwischen 8 und MM.HRES liegen. Die Kachelfarben werden mit der Standard-RGB888-Notation angegeben. Diese wird in RGB121 umgewandelt. Es können ein Framebuffer (F) und ein Layer-Buffer (L) erstellt werden. Diese haben keinen Einfluss auf die Anzeige und belegen keinen Benutzerspeicher, können aber beide zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden.

MODE 2 400 x 300 x 16 Farben.
RGB121-Format (d. h. 1 Bit für Rot, 2 Bits für Grün und 1 Bit für Blau). Es kann ein Framebuffer (F) erstellt werden. Dieser hat keinen Einfluss auf die Anzeige und belegt keinen Benutzerspeicher, kann aber zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Buffer erstellt werden. Auch dieser beansprucht keinen Benutzerspeicher. Alle in den Layer-Puffer geschriebenen Pixel erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptbildschirmpuffer befindet. Es kann eine Farbe festgelegt werden (0–15; Standard ist 0), die nicht angezeigt wird, sodass der Hauptbildschirmpuffer durchscheinen kann. Es steht eine Zuordnungsfunktion zur Verfügung, um die Standardfarben der 16 verfügbaren Farben zu überschreiben. Die Hardware ist auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind

MODE 3 800 x 600 x 16 Farben.
RGB121-Format (d. h. 1 Bit für Rot, 2 Bits für Grün und 1 Bit für Blau). Es kann ein Framebuffer (F) erstellt werden. Dieser hat keinen Einfluss auf die Anzeige und belegt keinen Benutzerspeicher, kann aber zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Buffer erstellt werden. Auch dieser beansprucht keinen Benutzerspeicher. Alle in den Layer-Buffer geschriebenen Pixel erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptbildschirm-Buffer von befindet. Es kann eine Farbe festgelegt werden (0–15; Standard ist 0), die nicht angezeigt wird, sodass der Hauptbildschirm-Buffer durchscheinen kann. Es gibt eine Zuordnungsfunktion, um die Standardfarben der 16 verfügbaren Farben zu überschreiben. Die Hardware ist auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind

OPTION RESOLUTION 848 x 480 (nur RP2350)

MODE 1 848 x 480 x 2 Farben (monochrom). Standard beim Start. Die Kachelbreite ist auf 8 Pixel festgelegt. Die Kachelhöhe beträgt standardmäßig 12 Pixel, kann aber zwischen 8 und MM.HRES liegen. Die Kachelfarben werden mit der Standard-RGB888-Notation angegeben. Diese wird in RGB121 umgewandelt. Es können ein Framebuffer (F) und ein Layer-Buffer (L) erstellt werden. Diese haben keinen Einfluss auf die Anzeige und belegen keinen Benutzerspeicher, können aber beide

| | |
|---|---|
| | zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden.<br>MODE 2 424 x 240 x 16 Farben.<br>RGB121-Format (d. h. 1 Bit für Rot, 2 Bits für Grün und 1 Bit für Blau). Es kann ein Framebuffer (F) erstellt werden. Dieser hat keinen Einfluss auf die Anzeige und belegt keinen Benutzerspeicher, kann aber zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Buffer erstellt werden. Auch dieser beansprucht keinen Benutzerspeicher. Alle in den Layer-Puffer geschriebenen Pixel erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptbildschirmpuffer befindet. Es kann eine Farbe festgelegt werden (0–15; Standard ist 0), die nicht angezeigt wird, sodass der Hauptbildschirmpuffer durchscheinen kann. Es steht eine Zuordnungsfunktion zur Verfügung, um die Standardfarben der 16 verfügbaren Farben zu überschreiben. Die Hardware ist auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind<br>MODE 3 848 x 48 x 16 Farben.<br>RGB121-Format (d. h. 1 Bit für Rot, 2 Bits für Grün und 1 Bit für Blau). Ein Framebuffer (F) kann erstellt werden. Dieser hat keinen Einfluss auf die Anzeige und belegt keinen Benutzerspeicher, kann aber zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Puffer erstellt werden. Auch dieser beansprucht keinen Benutzerspeicher. Alle in den Layer-Puffer geschriebenen Pixel erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptbildschirmpuffer befindet. Es kann eine Farbe festgelegt werden (0–15; Standard ist 0), die nicht angezeigt wird, sodass der Hauptbildschirmpuffer durchscheinen kann. Es gibt eine Zuordnungsfunktion, um die Standardfarben der 16 verfügbaren Farben zu überschreiben. Die Hardware ist auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind |
| MODE n | NUR HDMI-VERSIONEN<br>HDMI-Video unterstützt eine Reihe von Auflösungen (siehe OPTION RESOLUTION). Dieser Befehl wählt je nach Auflösung den Modus „n“ aus:<br>OPTION RESOLUTION 640 x 480<br>MODE 1 640 x 480 x 2 Farben (monochrom). Standard beim Start. Verwende den Befehl TILE wie gewohnt. Die Breite der Kacheln ist auf 8 Pixel festgelegt. Die Höhe der Kacheln beträgt standardmäßig 12 Pixel, kann aber zwischen 8 und MM.HRES liegen. Die Farben der Kacheln werden mit der Standard-RGB888-Notation angegeben. Diese wird in RGB555 umgewandelt. Es können ein Framebuffer (F) und ein Layer-Buffer (L) erstellt werden. Diese können zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden<br>MODE 2 320 x 240 x 16 Farben.<br>Es kann ein Framebuffer (F) erstellt werden. Dieser kann zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Buffer erstellt werden. Alle Pixel, die in den Layer- -Buffer geschrieben werden, erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptanzeigebuffer befindet. Es kann eine Farbe festgelegt |

werden (0–15: Standardwert ist 0), die nicht angezeigt wird, sodass der Hauptanzeigebuffer durchscheinen kann. Es gibt eine Zuordnungsfunktion, um die Standardfarben zu überschreiben.

MODE 3 640 x 480 x 16 Farben.
Farbzuordnung zur RGB555-Palette. Ein Framebuffer (F) kann erstellt werden. Er kann zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Puffer erstellt werden. Alle in den Layer-Puffer geschriebenen Pixel erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptanzeigepuffer befindet. Es kann eine Farbe angegeben werden (0–15: Standardwert ist 0), die nicht angezeigt wird, sodass der Hauptanzeigepuffer durchscheint.

MODE 4 320 x 240 x 32.768 Farben.
Dies ist volles RGB555, wodurch Farbbilder in guter Qualität angezeigt werden können. Ein Framebuffer (F) und ein Layer-Puffer (L) können erstellt werden. Diese haben keinen Einfluss auf die Anzeige und können zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Es kann nur einer erstellt werden

MODE 5 320 x 240 x 256 Farben.
Es kann ein Framebuffer (F) erstellt werden. Dieser hat keinen Einfluss auf die Anzeige. Er kann zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Buffer erstellt werden. Dieser belegt keinen Benutzerspeicher. Alle in den Layer-Puffer geschriebenen Pixel erscheinen automatisch auf dem Display und liegen über allem, was sich im Hauptanzeigepuffer befindet. Es kann eine Farbe festgelegt werden (0–255; Standardwert ist 0), die nicht angezeigt wird, sodass der Hauptanzeigepuffer durchscheinen kann. Es steht eine Zuordnungsfunktion zur Verfügung, um die Standardfarben der 256 verfügbaren Farben zu überschreiben. Jede der 256 Farben kann einer beliebigen RGB555-Farbe zugeordnet werden.

<u>OPTION RESOLUTION 720 x 400</u>

MODE 1 720 x 400 x 2 Farben (monochrom). Standard beim Start. Verwende den Befehl TILE wie gewohnt. Die Breite der Kacheln ist auf 8 Pixel festgelegt. Die Höhe der Kacheln beträgt standardmäßig 12 Pixel, kann aber zwischen 8 und MM.HRES liegen. Die Farben der Kacheln werden mit der Standard-RGB888-Notation angegeben. Diese wird in RGB555 umgewandelt. Es können ein Framebuffer (F) und ein Layer-Buffer (L) erstellt werden. Diese können zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden

MODE 2 360 x 200 x 16 Farben.
Es kann ein Framebuffer (F) erstellt werden. Dieser kann zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Buffer erstellt werden. Alle in den Layer-Buffer geschriebenen Pixel erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptanzeigepuffer befindet. Es kann eine Farbe festgelegt werden (0–15: Standardwert ist 0), die nicht angezeigt wird, sodass der Hauptanzeigepuffer durchscheinen kann. Es steht eine Zuordnungsfunktion zur Verfügung, um die Standardfarben zu überschreiben.

| | |
|---|---|
| | MODE 3 720 x 400 x 16 Farben.<br>Farbzuordnung zur RGB555-Palette. Ein Framebuffer (F) kann erstellt werden. Er kann zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Puffer erstellt werden. Alle in den Layer-Puffer geschriebenen Pixel erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptanzeigepuffer befindet. Es kann eine Farbe festgelegt werden (0–15: Standard t 0), die nicht angezeigt wird, sodass der Hauptanzeigepuffer durchscheint.<br>MODE 4 360 x 200 x 32768 Farben.<br>Dies ist volles RGB555, wodurch Farbbilder in guter Qualität angezeigt werden können. Ein Framebuffer (F) und ein Layer-Puffer (L) können erstellt werden. Diese haben keinen Einfluss auf die Anzeige und können zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Es kann nur einer erstellt werden<br>MODE 5 360 x 200 x 256 Farben.<br>Es kann ein Framebuffer (F) erstellt werden. Dieser hat keinen Einfluss auf die Anzeige. Er kann zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Buffer erstellt werden. Dieser belegt keinen Benutzerspeicher. Alle in den Layer-Puffer geschriebenen Pixel erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptbildschirmpuffer befindet. Es kann eine Farbe festgelegt werden (0–255; Standardwert ist 0), die nicht angezeigt wird, sodass der Hauptbildschirmpuffer durchscheinen kann. Es steht eine Zuordnungsfunktion zur Verfügung, um die Standardfarben der 256 verfügbaren Farben zu überschreiben. Jede der 256 Farben kann einer beliebigen RGB555-Farbe zugeordnet werden.<br><u>OPTION RESOLUTION 800 x 600 (Hinweis: reduziert den verfügbaren Heap-Speicher)</u><br>MODE 1 800 x 600 x 2 Farben mit RGB332-Kacheln (verwende den Befehl TILE wie gewohnt)<br>MODE 2 400 x 300 x 16 Farben mit optionaler Ebene und Farbabbildung auf die RGB332-Palette<br>MODE 3 800 x 400 x 16 Farben mit optionaler Ebene und Farbabbildung auf die RGB332-Palette<br>MODUS 5 400 x 300 x 256 Farben mit optionaler Ebene (kein Speicherverbrauch)<br><u>OPTION RESOLUTION 848 x 480 (Hinweis: reduziert den verfügbaren Heap-Speicher)</u><br>MODE 1 848 x 480 x 2 Farben mit RGB332-Kacheln (verwende den Befehl TILE wie gewohnt)<br>MODE 2 424 x 240 x 16 Farben mit optionaler Ebene und Farbabbildung auf die RGB332-Palette<br>MODE 3 848 x 480 x 16 Farben mit optionaler Ebene und Farbabbildung auf die RGB332-Palette<br>MODE 5 424 x 240 x 256 Farben mit optionaler Ebene (kein Speicherverbrauch)<br><u>OPTION RESOLUTION 1280 x 720</u><br>MODE 1 1280 x 720 x 2 Farben mit RGB332-Kacheln (verwende den Befehl TILE wie gewohnt) |

| | |
|---|---|
| | MODE 2 320 x 180 x 16 Farben mit 2 optionalen Ebenen und Farbabbildung auf die RGB332-Palette<br>MODE 3 640 x 360 x 16 Farben mit optionaler Ebene und Farbabbildung auf die RGB332-Palette<br>MODE 5 320 x 180 x 256 Farben mit optionaler Ebene (kein Speicherverbrauch)<br><br>OPTION RESOLUTION 1024 x 768<br>MODE 1 1024 x 768 x 2 Farben mit RGB332-Kacheln (verwende den Befehl TILE wie gewohnt)<br>MODE 2 256 x 192 x 16 Farben mit 2 optionalen Ebenen und Farbabbildung auf die RGB332-Palette<br>MODE 3 512 x 384 x 16 Farben<br>MODE 5 256 x 192 x 256 Farben mit optionaler Ebene und Farbabbildung auf die RGB332-Palette<br><br>OPTION RESOLUTION 1024 x 600<br>MODE 1 1024 x 600 x 2 Farben mit RGB332-Kacheln (verwende den Befehl TILE wie gewohnt)<br>MODE 2 256 x 150 x 16 Farben mit 2 optionalen Ebenen und Farbabbildung auf die RGB332-Palette<br>MODE 3 512 x 300 x 16 Farben mit optionaler Ebene und Farbabbildung auf die RGB332-Palette<br>MODE 5 256 x 150 x 256 Farben mit optionaler Ebene und Farbabbildung auf die RGB332-Palette<br><br>OPTION RESOLUTION 800 x 480 (Hinweis: reduziert den verfügbaren Heap-Speicher)<br>MODE 1 800 x 480 x 2 Farben mit RGB332-Kacheln (verwende den Befehl TILE wie gewohnt)<br>MODE 2 400 x 240 x 16 Farben mit 2 optionalen Ebenen und Farbabbildung auf die RGB332-Palette<br>MODE 3 800 x 480 x 16 Farben mit optionaler Ebene und Farbabbildung auf die RGB332-Palette<br>MODUS 5 400 x 240 x 256 Farben mit optionaler Ebene und MODE auf die RGB332-Palette |
| MODE 400<br>oder<br>MODE 800 | NUR RP2350-PICOMITE-VERSIONEN<br>Umschalten zwischen den Modi 400x240 und 800x480 für gepufferte SSD1963-Treiber |
| MOUSE<br><br><br><br>MOUSE INTERRUPT ENABLE channel, int<br><br>MOUSE INTERRUPT DISABLE channel | Für alle Varianten des Befehls. Bei USB-Firmware ist „channel“ der USB-Anschluss, an den die Maus angeschlossen ist (1–4). Weitere Informationen findest du unter MM.INFO(USB n). Bei PS2-Firmware ist „channel“ fest auf den Wert 2 gesetzt<br><br>„int“ ist eine benutzerdefinierte Subroutine, die aufgerufen wird, wenn die linke Maustaste gedrückt wird.<br><br>Deaktiviert einen Interrupt für die linke Maustaste |

| | |
|---|---|
| MOUSE SET channel, x-coord, y-coord [, wheel-count] | Legt die aktuelle Position fest, die von der Maus zurückgegeben wird: x, y und optional die Radpositionen |
| MOUSE OPEN channel, CLKpin, DATApin | NICHT-USB-VERSIONEN – NUR FÜR EINE PS2-MAUS<br>Öffnet eine Verbindung zu einer PS2-Maus, die an die beiden angegebenen Pins angeschlossen ist. Dieser Befehl kann in einem Programm verwendet werden, um die Maus während der Programmausführung zu konfigurieren, im Gegensatz zu OPTION MOUSE, das die Maus dauerhaft konfiguriert.<br>Der Kanal ist zur Kompatibilität mit USB-Mausfunktionen enthalten und muss auf 2 gesetzt werden. Wenn keine Maus angeschlossen ist, erhältst du eine Fehlermeldung und der Befehl kann erneut aufgerufen werden, sobald die Maus angeschlossen ist |
| MOUSE CLOSE channel | Schließt den Zugriff auf die Maus und stellt die Pins wieder auf den normalen Gebrauch zurück. Der Befehl führt zu einem Fehler, wenn OPTION MOUSE gesetzt wurde. |
| NEW | Löscht den Programmspeicher und alle Variablen, einschließlich gespeicherter Variablen. |
| NEXT [counter-variable] [, counter-variable], etc | NEXT steht am Ende einer FOR-NEXT-Schleife; siehe FOR.<br>Die „Zählervariable“ gibt genau an, auf welche Schleife angewendet wird. Wenn keine „Zählervariable“ angegeben wird, bezieht sich NEXT standardmäßig auf die innerste Schleife. Es ist auch möglich, mehrere Zählervariablen anzugeben, wie in:<br>`NEXT x, y, z` |
| ON ERROR ABORT<br>or<br>ON ERROR IGNORE<br>or<br>ON ERROR SKIP [nn]<br>or<br>ON ERROR CLEAR<br>Or<br>ON ERROR RESTART | Dies steuert die Aktion, die ausgeführt wird, wenn während der Ausführung eines Programms ein Fehler auftritt, und gilt für alle von MMBasic erkannten Fehler, einschließlich Syntaxfehler, falscher Daten, fehlender Hardware usw.<br>ON ERROR ABORT bewirkt, dass MMBasic eine Fehlermeldung anzeigt, das Programm abbricht und zur Eingabeaufforderung zurückkehrt. Dies ist das normale Verhalten und die Standardeinstellung, wenn ein Programm gestartet wird.<br>ON ERROR RESTART bewirkt, dass MMBasic einen Hardware-Reset durchführt, und wenn du OPTION AUTORUN eingestellt hast, startet das Programm natürlich anschließend sauber neu<br>ON ERROR IGNORE bewirkt, dass alle Fehler ignoriert werden.<br>ON ERROR SKIP ignoriert einen Fehler in einer bestimmten Anzahl von Befehlen (angegeben durch die Zahl „nn“), die nach diesem Befehl ausgeführt werden. „nn“ ist optional; die Standardeinstellung, falls nicht angegeben, ist eins. Nachdem die Anzahl der Befehle abgeschlossen ist (mit oder ohne Fehler), kehrt das Verhalten von MMBasic zu ON ERROR ABORT zurück.<br>Wenn ein Fehler auftritt und ignoriert/übersprungen wird, wird die schreibgeschützte Variable MM.ERRNO auf einen Wert ungleich Null gesetzt und MM.ERRMSG$ auf die Fehlermeldung, die normalerweise generiert würde. Diese werden durch ON ERROR CLEAR auf Null und eine leere Zeichenkette zurückgesetzt. Sie werden auch gelöscht, wenn das Programm ausgeführt wird und wenn ON ERROR IGNORE und ON ERROR SKIP verwendet werden.<br>ON ERROR IGNORE kann das Debuggen eines Programms erheblich erschweren, daher wird dringend empfohlen, nur ON ERROR SKIP zu verwenden. |
| ON KEY target<br>or<br>ON KEY ASCIIcode, target | Die erste Version des Befehls setzt einen Interrupt, der die benutzerdefinierte Subroutine „target“ aufruft, sobald ein oder mehrere Zeichen im Eingabepuffer der seriellen Konsole warten. |

| | |
|---|---|
| | Beachte, dass alle im Eingabepuffer wartenden Zeichen in der Interrupt-Subroutine gelesen werden sollten, da sonst automatisch ein weiterer Interrupt generiert wird, sobald das Programm aus dem Interrupt zurückkehrt.<br>Die zweite Version ermöglicht es dir, eine Interrupt-Routine mit einem bestimmten Tastendruck zu verknüpfen. Dies funktioniert auf einer niedrigen Ebene für die serielle Konsole, und wenn aktiviert, wird die Taste nicht in den Eingabepuffer geschrieben, sondern löst lediglich den Interrupt aus. Es verwendet einen separaten Interrupt vom einfachen ON KEY-Befehl, sodass beide bei Bedarf gleichzeitig verwendet werden können.<br>In beiden Varianten verwende zur Deaktivierung des Interrupts die Ziffer Null als Ziel, d. h.:<br>ON KEY 0. oder ON KEY ASCIIcode, 0 |
| ON PS2 target | Dies löst einen Interrupt aus, sobald die PicoMite-Firmware eine Nachricht von der PS2-Schnittstelle erkennt.<br>Verwende MM.info(PS2), um die empfangene Rohnachricht auszugeben. So kann der Programmierer sowohl das Drücken als auch das Loslassen einer Taste erfassen.<br>Die Scan-Codes (Set 2) findest du unter https://wiki.osdev.org/PS/2_Keyboard. |
| ONEWIRE RESET pin<br>or<br>ONEWIRE WRITE pin, flag, length, data [, data…]<br>or<br>ONEWIRE READ pin, flag, length, data [, data…] | Befehle zur Kommunikation mit 1-Wire-Geräten.<br>ONEWIRE RESET setzt den 1-Wire-Bus zurück<br>ONEWIRE WRITE sendet eine bestimmte Anzahl von Bytes<br>ONEWIRE READ liest eine bestimmte Anzahl von Bytes<br>'pin' ist der zu verwendende I/O-Pin (befindet sich am hinteren Anschluss). Es kann jeder Pin sein, der für digitale Ein- und Ausgänge geeignet ist.<br>„flag“ ist eine Kombination aus den folgenden Optionen:<br>1 – Reset vor dem Befehl senden<br>2 – Reset nach dem Befehl senden<br>4 – Nur ein Bit statt eines Datenbytes senden/empfangen<br>8 – Rufe nach dem Befehl einen starken Pull-up auf (der Pin wird auf High gesetzt und Open Drain deaktiviert)<br>'length' ist die Länge der zu sendenden oder zu empfangenden Daten<br>'data' sind die zu sendenden Daten oder die zu empfangende Variable. Die Anzahl der Datenelemente muss mit dem Längenparameter übereinstimmen.<br>Siehe auch *Anhang C*. |
| ONESHOT triggerPin, sense, outputPin, prePulseDelay, pulseWidth [, quiescentPeriod] | Konfiguriert einen One-Shot-Impulsgenerator im Hintergrund.<br>Wenn die ausgewählte Triggerflanke an „triggerPin“ erkannt wird, startet MMBasic eine Verzögerung von „prePulseDelay“ Mikrosekunden und schaltet dann „outputPin“ um. Nach „pulseWidth“ Mikrosekunden schaltet es „outputPin“ erneut um. Dadurch entsteht ein Impuls in entgegengesetzter Richtung zum Ruhezustand des Ausgangspins.<br>Optional kann eine „quiescentPeriod“ (in Mikrosekunden) angegeben werden. Ist diese vorhanden, ignoriert ONESHOT für diesen Zeitraum nach Beendigung des Impulses weitere Auslösungen.<br>„sense“ wählt den Trigger-Flankenübergang aus und kann POSITIVE (auch RISING) oder NEGATIVE (auch FALLING) sein. Die Triggerung erfolgt flankenbasiert (Übergang), nicht pegelbasiert.<br>Das Timing ist interruptgesteuert und nicht blockierend. Verzögerung und Impulsbreite werden in Mikrosekunden angegeben. |
| ONESHOT DISABLE | ONESHOT DISABLE stoppt den Hintergrundbetrieb und bricht anstehende Timing-Ereignisse ab. |

| | |
|---|---|
| ONESHOT R triggerPin, sense, outputPin, prePulseDelay, pulseWidth [, quiescentPeriod] | ONESHOT R verwendet dieselben Parameter wie ONESHOT, ist in diesem Fall jedoch nachtriggerbar. Im nachtriggerbaren Modus:<br>Wenn während der Vorimpulsverzögerung ein Trigger eintrifft, wird die Vorimpulsverzögerung ab diesem Trigger neu gestartet. Trigger werden nicht in eine Warteschlange gestellt. Während der Vorimpulsverzögerung ersetzt jeder neue gültige Trigger die vorherige ausstehende Verzögerung und startet das Timing neu. Das bedeutet, dass der Impuls erst nach einer Ruhephase beginnt, die mindestens so lang ist wie `prePulseDelay`, und zwar nach dem letzten gültigen Trigger.<br>Wenn Trigger weiterhin häufiger als `prePulseDelay` eintreffen, kann der Ausgangsübergang auf unbestimmte Zeit verzögert werden, bis die Trigger aufhören.<br>Wenn während des Impulses ein Trigger eintrifft, wird der Impulsbreiten-Timer ab diesem Trigger neu gestartet (der Ausgang bleibt in seinem aktiven Impulszustand).<br>Wenn während der Ruhephase ein Trigger eintrifft, wird ein neuer Zyklus gestartet (Vorimpulsverzögerung, dann Impuls). |
| OPEN fname$ FOR mode AS [#]fnbr | Öffnet eine Datei zum Lesen oder Schreiben.<br>„fname“ ist der Dateiname mit einer optionalen Erweiterung, getrennt durch einen Punkt (.). Lange Dateinamen mit Groß- und Kleinbuchstaben werden unterstützt. Die Dateisysteme sowohl auf der SD-Karte als auch auf den Flash-Laufwerken unterscheiden NICHT zwischen Groß- und Kleinschreibung.<br>Ein Verzeichnispfad kann mit dem Backslash als Verzeichnistrennzeichen angegeben werden. Das übergeordnete Verzeichnis des aktuellen Verzeichnisses kann durch den Verzeichnisnamen „..“ (zwei Punkte) und das aktuelle Verzeichnis durch „.“ (ein einzelner Punkt) angegeben werden.<br>Beispiel: OPEN ".\dir1\dir2\filename.txt" FOR INPUT AS #1<br>„mode“ ist INPUT, OUTPUT, APPEND oder RANDOM.<br>INPUT öffnet die Datei zum Lesen und gibt einen Fehler aus, wenn die Datei nicht existiert. OUTPUT öffnet die Datei zum Schreiben und überschreibt automatisch jede vorhandene Datei mit demselben Namen.<br>APPEND öffnet die Datei ebenfalls zum Schreiben, überschreibt jedoch keine vorhandene Datei; stattdessen werden alle Schreibvorgänge an das Ende der Datei angehängt. Wenn keine vorhandene Datei mit dem Namen „ “ existiert, verhält sich der APPEND-Modus genauso wie der OUTPUT-Modus (d. h. die Datei wird erstellt und dann zum Schreiben geöffnet).<br>RANDOM öffnet die Datei sowohl zum Lesen als auch zum Schreiben und ermöglicht den wahlfreien Zugriff mit dem Befehl SEEK. Beim Öffnen wird der Lese-/Schreibzeiger am Ende der Datei positioniert. Wenn die Datei nicht existiert, wird sie erstellt.<br>„fnbr“ ist die Dateinummer (1 bis 10). Das # ist optional. Es können bis zu 10 Dateien gleichzeitig geöffnet sein, die sich entweder auf dem Laufwerk A: oder B: oder auf beiden befinden können. Die Befehle INPUT, LINE INPUT, PRINT, WRITE und CLOSE sowie die Funktionen EOF() und INPUT$() verwenden alle „fnbr“, um die zu bearbeitende Datei zu identifizieren.<br>Siehe auch ON ERROR und MM.ERRNO zur Fehlerbehandlung. |
| OPEN comspec$ AS [#]fnbr | Öffnet einen seriellen Kommunikationsport zum Lesen und Schreiben. Es stehen zwei Ports zur Verfügung (COM1: und COM2:) und beide können gleichzeitig geöffnet sein. Eine vollständige Beschreibung mit Beispielen findest du in *Anhang A*.<br>Mit „fnbr“ kann der Port mit jedem Befehl oder jeder Funktion, die eine Dateinummer verwendet, beschrieben und gelesen werden. |

| | |
|---|---|
| OPEN comspec$ AS GPS [,timezone_offset] [,monitor] | Öffnet einen seriellen Kommunikationsport zum Lesen von Daten eines GPS-Empfängers. Die Daten können dann mit der Funktion GPS() abgerufen werden.<br>Dies wird ausführlich in der Datei *„Option_GPS_User_Manual.pdf“* beschrieben, die im ZIP-Download der Firmware enthalten ist. |
| OPTION | Siehe den Abschnitt *Options* weiter oben in diesem Handbuch. |
| PAUSE delay | Halte die Ausführung des laufenden Programms für „delay“ ms an. Das kann auch ein Bruch sein. Zum Beispiel entspricht 0,2 200 µs. Die maximale Verzögerung beträgt 2147483647 ms (etwa 24 Tage).<br>Beachte, dass Interrupts während einer Pause erkannt und verarbeitet werden. |
| PIN( pin ) = value | Bei einem als digitalen Ausgang konfigurierten „Pin“ wird der Ausgang auf Low gesetzt („Wert“ ist Null) oder auf High („Wert“ ungleich Null). Du kannst einen Ausgang auf High oder Low setzen, bevor er als Ausgang konfiguriert wird, und diese Einstellung wird zum Standardausgang, sobald der Befehl SETPIN wirksam wird.<br>Siehe die Funktion PIN() zum Auslesen eines Pins und den Befehl SETPIN zum Konfigurieren desselben. Eine allgemeine Beschreibung der Ein-/Ausgabefunktionen der PicoMite-Firmware findest du im Kapitel *„Verwendung der I/O-Pins*“. |
| PIO | Der Prozessorchip im Raspberry Pi Pico mit den RP2040-Prozessoren enthält ein programmierbares I/O-System mit zwei identischen PIO-Geräten (pio%=0 oder pio%=1), die wie spezialisierte CPU-Kerne funktionieren.<br>Der Raspberry Pi Pico 2 mit den RP2350-Prozessoren verfügt über drei PIO-Geräte.<br>Eine detailliertere Beschreibung der Programmierung von PIOs findest du in Anhang F.<br>Die PIO-Befehle werden als MMBasic-Befehle behandelt.<br>Gültige Befehle sind:<br>jmp (<cond>) <target><br>wait <polarity> gpio <gpio_num><br>wait <polarity> pin <pin_num><br>wait <polarity> irq (prev \| next) <irq_num> (rel)<br>wait <polarity> jmppin (+ <pin_offset>)<br>in <source>, <bit_count><br>out <destination>, <bit_count><br>push (iffull)<br>push (iffull) block<br>push (iffull) noblock<br>pull (ifempty)<br>pull (ifempty) block<br>pull (ifempty) noblock<br>mov <destination>, (op) <source><br>irq (prev \| next) <irq_num> (rel)<br>irq (prev \| next) set <irq_num> (rel)<br>irq (prev \| next) nowait <irq_num> (rel)<br>irq (prev \| next) wait <irq_num> (rel)<br>irq (prev \| next) clear <irq_num> (rel)<br>set <destination>, <value> |

| | |
|---|---|
| | RP2350 only<br>mov rxfifo[y], isr<br>mov rxfifo[<index>], isr<br>mov osr, rxfifo[y] NB<br>mov osr, rxfifo[<index>] NB: RP2350 only<br><br>Valid directives:<br>.program name<br>.line n/next<br>.label name<br>.wrap target<br>.wrap<br>.side set n<br>.end program [list]<br><br>PIO-Befehle werden zwischen einem 'PIO ASSEMBLE n'-Befehl und einer '.end program'-Anweisung verwendet. Dazwischen können auch andere Anweisungen stehen. |
| PIO assemble pio,linedata$ | Dieser Befehl assembliert und lädt textbasierten PIO-Assembler-Code einschließlich Labels für Sprünge.<br>Verwende: PIO assemble pio,".program anything", um den Assembler zu initialisieren.<br>Verwende: PIO assemble pio,".side_set n [opt] [pindirs]", wenn du Side Set verwendest. Dies ist zwingend erforderlich, um die Op-Codes korrekt zu erstellen, wenn ein oder mehrere Side-Set-Pins verwendet werden.<br>Das pinctrl-Register wird nicht geladen, da dieses spezifisch für die Zustandsmaschine ist.<br>Beachte außerdem, dass der Parameter „opt" den Op-Code bei Befehlen mit einem Side-Parameter ändert.<br>Verwendung: PIO assemble pio,".line n", um ab einer anderen Zeile als 1 zu asemblieren – dies ist optional.<br>Verwende: PIO assemble pio,".end program [list]", um die Assemblierung zu beenden und das PIO zu programmieren. Der optionale Parameter LIST bewirkt einen Hex-Dump der Op-Codes auf das Terminal.<br>Verwende: PIO assemble pio,"label:" um ein Label zu definieren. Dies muss als separater Befehl erscheinen.<br>Verwendung: PIO assemble „.wrap target", um anzugeben, wohin das Programm zurückspringen soll. Siehe PIO(.wrap target) für die Verwendung.<br>Verwende: PIO assemble „.wrap", um anzugeben, wohin das Programm von „.wrap target" zurückspringen soll. Siehe PIO(.wrap) für die Verwendung.<br>Verwendung: PIO assemble pio "instruction [parameters]" um die eigentlichen PIO-Befehle zu definieren, die in Maschinencode umgewandelt werden sollen. |
| PIO DMA RX pio, sm, nbr, data%() [,completioninterrupt] [,transfersize] [,loopbackcount]<br>PIO DMA TX pio, sm, nbr, data%() [,completioninterrupt] | Richtet DMA-Übertragungen vom PIO zum MMBasic-Speicher ein.<br>pio gibt an, welche der beiden PIO-Instanzen verwendet werden soll (0 oder 1).<br>sm gibt an, welche Zustandsmaschine verwendet werden soll (0–3).<br>nbr gibt an, wie viele 32-Bit-Wörter übertragen werden sollen. Siehe unten für den Sonderfall, bei dem nbr auf Null gesetzt wird.<br>data%() ist das Array, das die PIO-Daten entweder bereitstellt oder empfängt. |

| | |
|---|---|
| [,transfersize]<br>[,loopbackcount] | Der optionale Parameter completioninterrupt ist der Name einer MMBasic-Subroutine, die aufgerufen wird, wenn der DMA abgeschlossen ist und im Fall von DMA_OUT der FIFO geleert wurde.<br>Wenn der optionale Interrupt nicht verwendet wird, kann der Status des DMA mit den folgenden Funktionen überprüft werden:<br>MM.INFO(PIO RX DMA)<br>MM.INFO(PIO TX DMA)<br>Der optionale Parameter „transfersize“ ermöglicht es dir, die normalen 32-Bit-Übertragungen zu überschreiben und 8, 16 oder 32 zu wählen.<br>Der optionale Parameter loopbackcount gibt an, wie viele Datenelemente gelesen oder geschrieben werden sollen, bevor der DMA wieder am Anfang des Puffers startet.<br>Der Parameter muss eine Potenz von 2 zwischen 2 und 32768 sein.<br>Aufgrund einer Einschränkung im RP2040/RP2350 muss das MMBasic-Array im Speicher an die Anzahl der Bytes in der Schleife ausgerichtet sein, wenn „loopbackcount“ verwendet wird.<br>Wenn das Array also 64 Ganzzahlen lang ist, was 512 Byte entspricht, muss das Array im Speicher an einer 512-Byte-Grenze ausgerichtet sein.<br>Alle MMBasic-Arrays sind an einer 256-Byte-Grenze ausgerichtet, aber um ein Array zu erstellen, das garantiert an einer 512-Byte-Grenze oder größer ausgerichtet ist, muss der Befehl PIO MAKE RING BUFFER verwendet werden.<br>Wenn loopbackcount gesetzt ist, kann „nbr“ auf 0 gesetzt werden. In diesem Fall läuft die Übertragung kontinuierlich weiter und füllt den Puffer wiederholt, bis sie explizit gestoppt wird.<br>Wenn sowohl „nbr“ als auch „loopbackcount“ angegeben und identisch sind, startet der DMA nach Abschluss automatisch neu (nur TX) |
| PIO DMA RX OFF<br>PIO DMA TX OFF | Bricht einen laufenden DMA ab. |
| PIO-INTERRUPT pio, sm [,RXinterrupt] [,TXinterrupt] | Setzt grundlegende Interrupts für PIO-Aktivitäten.<br>Verwende den Wert 0 für „RXinterrupt“ oder „TXinterrupt“, um einen Interrupt zu deaktivieren.<br>Lass nicht benötigte Werte weg.<br>Der RX-Interrupt wird ausgelöst, sobald ein Wort vom PIO-Code in den angegebenen FIFO „geschoben“ wurde. Die Daten MÜSSEN im Interrupt gelesen werden, um ihn zu löschen.<br>Der TX-Interrupt wird ausgelöst, sobald der angegebene FIFO FULL war und der PIO-Code ihn nun „herausgeholt“ hat |
| PIO INIT MACHINE pio%, statemachine%, clockspeed [,pinctrl] [,execctrl] [,shiftctrl] [,startinstruction] [,sideout [,setout] [,outout] | Initialisiert PIO 'pio%' mit der Zustandsmaschine 'statemachine%'. 'clockspeed' ist die Taktrate der Zustandsmaschine in kHz. Die ersten vier optionalen Argumente sind Variablen, die Initialisierungswerte der Zustandsmaschinenregister und die Adresse des ersten auszuführenden Befehls enthalten (Standardwert ist Null). Diese bestimmen, wie der PIO arbeiten wird.<br>sideout, setout und outout können auf 0 (Standard) oder 1 gesetzt werden, um anzugeben, ob die in pinctrl definierten Pins als Eingänge (0) oder Ausgänge (1) initialisiert werden sollen |
| PIO EXECUTE pio, state_machine, instruction% | Führt den Befehl sofort auf dem angegebenen PIO und der angegebenen Zustandsmaschine aus. |

| | |
|---|---|
| PIO WRITE pio, state_machine, count, data0 [,data1..] | Schreibt die Datenelemente in den angegebenen PIO und die angegebene Zustandsmaschine. Der Schreibvorgang ist blockierend, daher muss die Zustandsmaschine in der Lage sein, die bereitgestellten Daten zu verarbeiten<br>Hinweis: Dieser Befehl wird in zukünftigen Versionen wahrscheinlich zusätzliche Funktionen benötigen |
| PIO WRITEFIFO a,b,c,d | Schreibt in eines der 4 einzelnen FIFO-Register.<br>„a“ ist der PIO (0 oder 1), „b“ ist die Zustandsmaschine (0...3), „c“ ist das FIFO-Register (*0…3), „d“ ist der Datenwert (32-Bit-Ganzzahlwert). |
| PIO READ pio, state_machine, count, data%[()] | Liest die Datenelemente aus dem angegebenen PIO und der angegebenen Zustandsmaschine. Der Lesevorgang ist nicht blockierend, daher muss die Zustandsmaschine in der Lage sein, die angeforderten Daten bereitzustellen. Wenn count gleich eins ist, kann eine Ganzzahl zum Empfangen der Daten verwendet werden, andernfalls sollte ein Ganzzahl-Array angegeben werden.<br>Hinweis: Dieser Befehl wird in zukünftigen Versionen wahrscheinlich zusätzliche Funktionen benötigen. |
| PIO START pio, statemachine | Starte eine bestimmte Zustandsmaschine auf pio. |
| PIO STOP pio, statemachine | Stopp einer bestimmten Zustandsmaschine auf pio. |
| PIO CLEAR pio | Dadurch wird das auf pio angegebene pio auf allen Zustandsmaschinen angehalten und die Steuerregister sowie die Interrupt-Flags für die Zustandsmaschinen PINCTRL, EXECTRL und SHIFTCTRL auf die Standardwerte zurückgesetzt. |
| PIO PROGRAM pio | Gibt an, dass die folgenden Zeilen PIO-Assembler-Befehle für das PIO „pio“ sind, bis eine „.end program“-Anweisung folgt |
| PIO PROGRAM pio,array%() | Programmiert den gesamten PIO-Programmspeicher mit den Daten in array%(). Siehe Anhang F. |
| PIO PROGRAM LINE pio, line, instruction | Programmiert nur die angegebene Zeile in einem PIO-Programm. |
| PIO SYNC pio, statemachines, [,prevstatemachines] [,nextstatemachines] | Synchronisiert die Uhren der PIO-Zustandsmaschinen<br>„pio“ gibt den Referenz-PIO für den Befehl an<br>„statemachines“, „prevstatemachines“ und „nextstatemachines“ sind Bitmaps (0 bis 15), die angeben, welche Zustandsmaschinentakte synchronisiert werden sollen.<br>Um also festzulegen, dass alle Zustandsmaschinen von PIO0 und PIO1 synchronisiert werden sollen, könntest du Folgendes verwenden:<br>PIO SYNC 0,15,,15<br>oder<br>PIO SYNC 1,15,15 |
| PIO SET BASE 0/16 | PIO-Befehle funktionieren nur mit 32 GPIO-Ports. Beim RP2350B weist dieser Befehl das System an, GP0-GP31 (0) oder GP16-GP47 (16) zu verwenden |
| PIO CONFIGURE pio, sm, clock [,startaddress] [,sidesetbase] [,sidesetno] [,sidesetout] [,setbase] [,setno] [,setout] [,outbase] [,outno] [,outout] | Die Parameter in diesem Befehl entsprechen im Wesentlichen denen, die du im Befehl „PIO INIT“ verwenden würdest, ergänzt um die Hilfsfunktionen PINCTRL, SHIFTCTRL und EXECCTRL, jedoch in einem einzigen Befehl zusammengefasst. Dies ist erforderlich, da das Pico SDK im Hintergrund einige sehr clevere Verarbeitungsschritte durchführt, um den RP2350B zu steuern |

| | |
|---|---|
| [,inbase]<br>[,jmppin] [,wraptarget] [,wrap]<br>[,sideenable] [,sidepindir]<br>[,pushthreshold]<br>[,pullthreshold] [,autopush]<br>[,autopull] [,inshiftdir]<br>[,outshiftdir]<br>[,joinrxfifo] [,jointxfifo]<br>[,joinrxfifoget] [,joinrxfifoput] | „sidesetbase“, „sidebase outbase“, „inbase“ und „jmppin“ sind Pin-Definitionen. Du kannst diese entweder als GP-Nummer oder als Pin-Nummer angeben (z. B. GP3 oder 5). Gib in jedem Fall den tatsächlichen Pin an. Wenn also PIO SET BASE für diesen PIO auf 16 gesetzt ist, sind die Werte GP16 bis GP47 gültig. Wenn „PIO SET BASE“ nicht gesetzt oder auf 0 gesetzt ist, sind die Pins GP0 bis GP31 von „ “ gültig. Sie alle verwenden standardmäßig die Basis-Einstellung, außer „jmppin“ (Standardwert -1), das explizit gesetzt werden muss, wenn du ein „jmppin“ verwenden möchtest, da dies die Einstellung des erforderlichen Statusbits auslöst<br>„clock“ ist die gewünschte PIO-Taktrate in Hz<br>„startaddress“ ist die PIO-Anweisung, mit der die Ausführung beginnt – Standardwert ist 0<br>„sidesetno“, „setno“ und „outno“ geben die Anzahl der Pins an, die für diese Funktionen verwendet werden können – Standardwert ist 0<br>„sidesetout“, „setout“ und „outout“ geben an, ob diese Pins als Ausgänge konfiguriert werden sollen (1=ja, 0=nein) – Standardwert ist 0<br>„wraptarget“ und „wrap“ liegen im Bereich von 0 bis 31 und sind standardmäßig auf 0 bzw. 31 gesetzt<br>„inshiftdir“ und „outshiftdir“ sind standardmäßig auf 1 gesetzt – das Ausgangs-Schieberegister wird nach rechts verschoben und das Eingangs-Schieberegister wird nach rechts verschoben (Daten kommen von links herein).<br>Alle anderen Parameter sind Boolesche Werte, die eine bestimmte Funktion aktivieren können – 1 zum Aktivieren, 0 zum Deaktivieren – alle sind standardmäßig auf 0 gesetzt.<br>Einfaches Beispiel:<br>*'PIO Configure pio, sm, clock, startaddress,*<br>*'sidesetbase, sidesetno, sidesetout,*<br>*'setbase, setno, setout, outbase, outno, outout, inbase,*<br>*'jmppin, wraptarget, wrap, sideenable, sidepindir,*<br>*'pushthreshold, pullthreshold, autopush, autopull, inshiftdir, outshiftdir,*<br>*'joinrxfifo, jointxfifo, joinrxfifoget, joinrxfifoput*<br>*PIO assemble 1*<br>*.program test*<br>*.line 0*<br>*.wrap target*<br>*Set pins,1*<br>*Set pins,0*<br>*.wrap*<br>*.end program*<br>*SetPin gp45,pio1*<br>*PIO set base 1,16*<br>*PIO configure 1,0,1000000,,,,,gp45,1,1,,,,,,Pio(.wrap target),Pio(.wrap)*<br>*PIO start 1,0*<br>*Do*<br>*Loop*<br><br>Obwohl der Befehl PIO CONFIGURE viele Parameter hat, ist er sehr einfach zu verwenden, wenn du diesen einfachen Ansatz wählst: Kopiere die Kommentarzeilen aus dem Beispiel in dein Programm. Ersetze für jeden Parameter den gewünschten Wert oder lösche den Parameter, wobei du die Kommas beibehältst.<br>Sobald alle Ersetzungen vorgenommen sind, lösche alle nachgestellten Kommas<br>Da die Zeile für den Editor vermutlich zu lang ist, lösche die Zeilenumbrüche nacheinander, beginnend am Ende der vorletzten Zeile und nach oben arbeitend.<br>Auf diese Weise erhältst du einen gültigen Befehl, der sich leicht eingeben und bearbeiten lässt. |

| | |
|---|---|
| | Hinweis: Du kannst auch Fortsetzungszeilen verwenden, um die Bearbeitung zu vereinfachen (siehe OPTION CONTINUATION LINES) |
| PIXEL x, y [,c] | Setze ein Pixel auf einem Videoausgang oder einem angeschlossenen LCD-Bildschirm auf eine Farbe.<br>'x' ist die horizontale Koordinate und 'y' die vertikale Koordinate des Pixels. 'c' ist eine 24-Bit-Zahl, die die Farbe angibt. 'c' ist optional; wird es weggelassen, wird die aktuelle Vordergrundfarbe verwendet.<br>Alle Parameter können als Arrays angegeben werden, und die Software zeichnet die Anzahl der Pixel entsprechend den Abmessungen des kleinsten Arrays. „x“ und „y“ müssen beide Arrays oder beide einzelne Variablen/Konstanten sein, andernfalls wird ein Fehler ausgegeben. „c“ kann entweder ein Array oder eine einzelne Variable oder Konstante sein.<br>Eine Definition der Farben und Grafikkoordinaten findest du im Kapitel *„Grafikbefehle und -funktionen*“. |
| PLAY | Dieser Befehl erzeugt verschiedene Audioausgaben.<br>Siehe den Befehl OPTION AUDIO, um die für die Ausgabe zu verwendenden I/O-Pins einzustellen. Der Ton ist ein pulsweitenmoduliertes Signal (PWM), daher ist ein Tiefpassfilter erforderlich, um die Trägerfrequenz zu entfernen. |
| PLAY TONE left, right [,dur] [,interrupt] | Erzeugt zwei separate Frequenzen auf dem linken und rechten Kanal der Audioausgabe.<br>„left“ und „right“ sind die Frequenzen in Hz, die für den linken und rechten Kanal verwendet werden sollen. Der Ton wird im Hintergrund abgespielt (das Programm läuft nach diesem Befehl weiter) und „dur“ gibt die Anzahl der Millisekunden an, die der Ton erklingen soll. Wenn die Dauer nicht angegeben wird, ertönt der Ton so lange, bis er explizit gestoppt wird oder das Programm beendet wird.<br>'interrupt' ist eine optionale Subroutine, die aufgerufen wird, wenn die Wiedergabe endet.<br>Die Frequenz kann zwischen 1 Hz und 20 kHz liegen und ist sehr genau (sie basiert auf einem Quarzoszillator). Die Frequenz kann jederzeit durch einen neuen PLAY TONE-Befehl geändert werden. |

| | |
|---|---|
| PLAY FLAC file$ [, interrupt] | Spielt eine FLAC-Datei über den Soundausgang ab.<br>'file$' ist die abzuspielende FLAC-Datei (die Endung .flac wird angehängt, falls sie fehlt). Die Abtastrate kann bis zu 48 kHz in Stereo betragen (96 kHz, wenn der Pico übertaktet ist)<br>Die FLAC-Datei wird im Hintergrund abgespielt. 'interrupt' ist optional und bezeichnet den Namen einer Subroutine, die aufgerufen wird, sobald die Datei vollständig abgespielt wurde.<br>Wenn file$ ein Verzeichnis auf dem Laufwerk B: ist, spielt der Pico alle Dateien in diesem Verzeichnis nacheinander ab. |
| PLAY WAV file$ [, interrupt] | Spielt eine WAV-Datei über den Soundausgang ab.<br>„file$“ ist die abzuspielende WAV-Datei (die Erweiterung .wav wird angehängt, falls sie fehlt). Die WAV-Datei muss PCM-kodiert sein, in Mono oder Stereo mit 8- oder 16-Bit-Abtastung. Die Abtastrate kann bis zu 48 kHz in Stereo betragen (96 kHz, wenn der Pico übertaktet ist).<br>Die WAV-Datei wird im Hintergrund abgespielt. 'interrupt' ist optional und ist der Name einer Subroutine, die aufgerufen wird, wenn die Datei vollständig abgespielt wurde. |
| PLAY ARRAY l%(), r%(), freq [,start] [,end] [,terminationinterrupt] | Gibt Daten über ein oder zwei Arrays an den Audioausgang aus:<br>Dies erfordert gepackte Arrays „l%“ und „r%“ (könnte dasselbe Array für links und rechts sein) mit 16-Bit-Werten im Bereich von -32768 bis 32767 und gibt die Werte im Array mit der angegebenen Abtastfrequenz aus.<br>Wenn der optionale Parameter „start“ angegeben wird, werden die Arrays ab der gepackten Position „start“ ausgegeben<br>Wenn der optionale Parameter „end“ angegeben wird, werden die Arrays ausgegeben, bis die gepackte Position „end“ erreicht ist<br>Wenn der optionale Parameter „terminationinterrupt“ angegeben wird, wird die Subroutine<br>, sobald das letzte Array-Element ausgegeben wurde.<br>Es ist wichtig, den Parameter „freq“ zu verstehen. Dies ist die Rate, mit der jedes Array-Element ausgegeben wird. Wenn das Array beispielsweise 10 Zyklen einer Sinuswelle mit insgesamt 18000 Samples enthält, würde die Einstellung der Frequenz auf 1800 eine 1-Hz-Sinuswelle für 10 Sekunden ausgeben. Die Einstellung der Frequenz auf 18000 würde eine 10-Hz-Sinuswelle für 1 Sekunde ausgeben. |
| PLAY MP3 file$ [, interrupt] | Spielt eine MP3-Datei über den Soundausgang ab (NUR RP2350).<br>'file$' ist die abzuspielende MP3-Datei (die Endung .mp3 wird angehängt, falls sie fehlt). Die Abtastrate kann bis zu 48 kHz betragen.<br>Die MP3-Datei wird im Hintergrund abgespielt. 'interrupt' ist optional und bezeichnet den Namen einer Subroutine, die aufgerufen wird, sobald die Datei vollständig abgespielt wurde.<br>Wenn file$ ein Verzeichnis auf dem Laufwerk B: ist, spielt der Pico alle Dateien in diesem Verzeichnis nacheinander ab. |
| PLAY MODFILE file$ [,interrupt] | Spielt eine MOD-Datei über den Soundausgang ab.<br>„file$“ ist die abzuspielende MOD-Datei (die Endung .mod wird angehängt, falls sie fehlt).<br>Die MOD-Datei wird im Hintergrund abgespielt und läuft kontinuierlich in einer Schleife. Wenn das optionale „interrupt“ angegeben ist, wird dies aufgerufen, sobald die Datei einmal durchgespielt wurde, und die Wiedergabe wird dann beendet. Dieser Befehl nutzt vorrangig Speicherplatz im PSRAM, falls dieser für den Dateipuffer aktiviert ist (nur RP2350). In diesem Fall muss ein modbuffer nicht mit dem OPTION-Befehl aktiviert werden |

| | |
|---|---|
| PLAY MODSAMPLE Samplenum, channel [,volume | Spielt ein bestimmtes Sample aus der MOD-Datei auf dem angegebenen Kanal ab. Die Lautstärke ist optional und kann zwischen 0 und 64 liegen. Dieser Befehl kann nur verwendet werden, wenn bereits eine MOD-Datei abgespielt wird, und ermöglicht die Ausgabe von Soundeffekten, während die Hintergrundmusik noch läuft. |
| PLAY LOAD SOUND array%() | Lädt ein Array mit 1024 Elementen, das aus 4096 16-Bit-Werten zwischen 0 und 4095 besteht. Dies liefert die Daten für eine beliebige Wellenform, die mit dem Befehl PLAY SOUND abgespielt werden kann. Mit dem Befehl MEMORY PACK kannst du die Arrays aus einem normalen Integer-Array mit 4096 Elementen erstellen. |
| PLAY SOUND soundno, channelno, type [,frequency] [,volume] | Spielt eine Reihe von Tönen gleichzeitig über den Audioausgang ab.<br>„soundno“ ist die Soundnummer und kann zwischen 1 und 4 liegen, was vier gleichzeitige Sounds auf jedem Kanal ermöglicht.<br>'channelno' gibt den Ausgangskanal an. Er kann L (linker Lautsprecher), R (rechter Lautsprecher) oder B (beide Lautsprecher) sein.<br>„type“ gibt die zu verwendende Wellenform an. Es kann S (Sinuswelle), Q (Rechteckwelle), T (Dreieckwelle), W (Sägezahnwelle), O (Null-Ausgabe), P (periodisches Rauschen), N (Zufallsrauschen) oder U (benutzerdefiniert mit PLAY LOAD SOUND) sein.<br>'frequency' ist die Frequenz von 1 bis 20000 (Hz) und muss angegeben werden, außer wenn type O ist. Im Modus Type U kann dieser Parameter auch Dezimalwerte annehmen. Zum Beispiel spielen alle folgenden Befehle die Wellenform in ihrer ursprünglichen Tonhöhe ab:<br>Abtastrate von 4000, Frequenz = 1<br>Abtastrate von 8000, Frequenz = 2<br>Abtastrate von 16000, Frequenz = 4<br>„volume“ ist optional und muss zwischen 1 und 25 liegen. Der Standardwert ist 25<br>Wenn PLAY SOUND aufgerufen wird, werden alle anderen Audio-Vorgänge blockiert und bleiben blockiert, bis PLAY STOP aufgerufen wird. Die Ausgabe kann vorübergehend mit PLAY PAUSE und PLAY RESUME angehalten werden.<br>Der Aufruf von SOUND für eine bereits laufende „soundno“ ersetzt sofort die vorherige Ausgabe. Einzelne Sounds werden mit dem Typ „O“ ausgeschaltet<br>Das gleichzeitige Abspielen von 4 Sounds auf beiden Kanälen des Audioausgangs beansprucht etwa 23 % der CPU-Leistung. |
| PLAY PAUSE<br>PLAY RESUME<br>PLAY STOP | PLAY PAUSE hält die aktuell abgespielte Datei oder den Ton vorübergehend an.<br>PLAY RESUME setzt die Wiedergabe eines angehaltenen Sounds fort.<br>PLAY STOP beendet die Wiedergabe der Datei oder des Tons. Wenn das Programm aus irgendeinem Grund beendet wird, wird die Tonausgabe ebenfalls automatisch gestoppt. |
| PLAY VOLUME left, right | Stellt die Lautstärke der Audioausgabe ein.<br>„left“ und „right“ sind die Pegel für den linken und rechten Kanal und können zwischen 0 und 100 liegen, wobei 100 die maximale Lautstärke ist. Es besteht ein linearer Zusammenhang zwischen dem angegebenen Pegel und der Ausgabe. Die Lautstärke ist standardmäßig auf Maximum eingestellt, wenn ein Programm ausgeführt wird. |
| PLAY NEXT | Beendet die Wiedergabe der aktuellen Audiodatei und startet die nächste im Verzeichnis |

| | |
|---|---|
| PLAY PREVIOUS | Beendet die Wiedergabe der aktuellen Audiodatei und startet die vorherige im Verzeichnis |
| PLAY SAMPLE left%(), right%(), freq, attack, decay, sustain, release [, freqR, attackR, decayR, sustainR, releaseR] [, interrupt]<br>oder<br>PLAY RELEASE | Nur RP2350.<br>Spielt eine geloopte Ein-Zyklus-Wellenform aus Integer-Arrays mit einer ADSR-Hüllkurve ab. Der optionale Parametersatz für den rechten Kanal ermöglicht eine unabhängige Steuerung von Stereofrequenz und ADSR.<br>Verwende PLAY RELEASE, um in die Release-Phase zu wechseln und die Wiedergabe zu beenden. |
| | VS1053-SPEZIFISCHE PLAY-BEFEHLE |
| PLAY MP3 file$ [, interrupt] | Spielt eine MP3-Datei über den Soundausgang ab.<br>'file$' ist die abzuspielende MP3-Datei (die Endung .mp3 wird angehängt, falls sie fehlt). Die Abtastrate sollte 44100 Hz Stereo betragen.<br>Die MP3-Datei wird im Hintergrund abgespielt. 'interrupt' ist optional und ist der Name einer Subroutine, die aufgerufen wird, wenn die Datei vollständig abgespielt wurde.<br>Wenn „file$“ ein Verzeichnis auf dem Laufwerk B: ist, spielt der Pico alle Dateien in diesem Verzeichnis nacheinander ab. |
| PLAY HALT | Dieser Befehl funktioniert, während eine MP3-Datei abgespielt wird. Er stoppt die Wiedergabe und speichert die aktuelle Dateiposition, damit die Wiedergabe an derselben Stelle fortgesetzt werden kann. Dieser Befehl wurde speziell zur Unterstützung von MP3-Hörbüchern entwickelt |
| PLAY CONTINUE track$ | Setzt die Wiedergabe des angegebenen MP3-Titels fort. „track$“ ist der Name der Datei, die zum Zeitpunkt des Anhaltens abgespielt wurde, wobei alle Dateiattribute entfernt werden<br>z. B.<br>PLAY MP3 „B:/mp3/mymp3.mp3“<br>etwas später<br>PLAY HALT<br>noch später<br>PLAY CONTINUE „mymp3“ |
| PLAY MIDIFILE file$ [, interrupt] | Spielt eine MIDI-Datei über den Soundausgang ab.<br>'file$' ist die abzuspielende MIDI-Datei (die Endung .mid wird angehängt, falls sie fehlt).<br>Die MIDI-Datei wird im Hintergrund abgespielt. 'interrupt' ist optional und bezeichnet den Namen einer Subroutine, die aufgerufen wird, sobald die Datei vollständig abgespielt wurde.<br>Wenn „file$“ ein Verzeichnis auf Laufwerk B: ist, spielt der Pico alle Dateien in diesem Verzeichnis nacheinander ab. |
| PLAY MIDI | Startet den Echtzeit-MIDI-Modus. In diesem Modus können MIDI-Befehle an den VS1053 gesendet werden, um auszuwählen, welche Instrumente auf welchen Kanälen gespielt werden sollen, sowie Noten in Echtzeit ein- und auszuschalten |
| PLAY MIDI CMD cmd%, data1%, data2% | Sendet einen MIDI-Befehl im Echtzeit-MIDI-Modus. Ein Beispiel wäre die Zuweisung eines Instruments zu einem Kanal. Z. B.<br>PLAY MIDI CMD &B11000001,4 ‘Kanal 1 auf Instrument 4 setzen |
| PLAY MIDI TEST n | Spielt eine MIDI-Testsequenz ab, n=0 bis 3, 0 = normale Echtzeit, die anderen spielen Noten- und Instrumentensamples |

| | |
|---|---|
| PLAY NOTE ON channel%, note%, velocity% | Schaltet die Note auf dem angegebenen Kanal ein, wenn du im Echtzeit-MIDI-Modus bist |
| PLAY NOTE OFF channel%, note% [,velocity%] | Schaltet die Note auf dem angegebenen Kanal im Echtzeit-MIDI-Modus aus |
| PLAY STREAM buffer%(), readpointer%, writepointer% | Sendet Daten aus dem Ringpuffer „buffer%“ an den VS1053-CODEC. Dieser Befehl initiiert einen Hintergrund-Ausgabestream, bei dem alles, was sich im Puffer zwischen dem Lesezeiger und dem Schreibzeiger befindet, an den VS1053 gesendet wird, wobei der Lesezeiger dabei fortlaufend aktualisiert wird. Kann für die Ausgabe beliebiger Wellenformen verwendet werden. |
| POKE BYTE addr%, byte<br>or<br>POKE SHORT addr%, short%<br>or<br>POKE WORD addr%, word%<br>or<br>POKE INTEGER addr%, int%<br>or<br>POKE FLOAT addr%, float!<br>or<br>POKE VAR var, offset, byte<br>or<br>POKE VARTBL, offset, byte | Setzt ein Byte oder ein Wort im Speicherbereich des Pico. Wenn mehr als ein Byte geschrieben wird, muss die Adresse genau durch die Anzahl der Bytes teilbar sein: 2, 4 oder 8, andernfalls wird ein Fehler gemeldet.<br>POKE BYTE setzt das Byte (d. h. 8 Bit) an der Speicheradresse „addr%“ auf „byte“. „addr%“ sollte eine Ganzzahl sein.<br>POKE SHORT setzt die kurze Ganzzahl (d. h. 16 Bit) an der Speicheradresse 'addr%' auf 'word%'. 'addr%' und 'word%' sollten Ganzzahlen sein.<br>POKE WORD setzt das Wort (d. h. 32 Bit) an der Speicheradresse 'addr%' auf 'word%'. 'addr%' und 'word%' sollten Ganzzahlen sein.<br>POKE INTEGER setzt die MMBasic-Ganzzahl (d. h. 64 Bit) an der Speicheradresse 'addr%' auf 'int%'. 'addr%' und 'int%' sollten Ganzzahlen sein.<br>POKE FLOAT setzt das Wort (d. h. 64 Bit) an der Speicheradresse 'addr%' auf 'float!'. 'addr%' sollte eine Ganzzahl und 'float!' eine Gleitkommazahl sein.<br>POKE VAR setzt ein Byte an 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 der ±Offset vom Anfang der Variablentabelle. Beachte, dass nach dem Schlüsselwort VARTBL ein Komma erforderlich ist. |
| POKE DISPLAY command [,data1] [,data2] [,datan] | Dieser Befehl sendet Befehle und zugehörige Daten an den Display-Controller eines angeschlossenen Displays. So kannst du als Programmierer die Konfigurationsparameter des Displays ändern. Beispielsweise schaltet POKE DISPLAY &H28 ein SSD1963-Display aus und POKE DISPLAY &H29 schaltet es wieder ein.<br>Funktioniert bei allen Displays außer dem ST7790. |
| POKE DISPLAY HRES n<br>POKE DISPLAY VRES n | Diese Befehle ändern den gespeicherten Wert von MM.HRES und MM.VRES, sodass der Programmierer nicht standardmäßige Displays konfigurieren kann. |
| POLYGON n, xarray%(), yarray%() [, bordercolour] [, fillcolour]<br>POLYGON n(), xarray%(), yarray%() [, bordercolour()] [, fillcolour()]<br>POLYGON n(), xarray%(), yarray%() [, bordercolour] [, fillcolour] | Zeichnet ein gefülltes oder umrandetes Polygon mit „n“ xy-Koordinatenpaaren in „xarray%()“ und „yarray%()“. Wenn „fillcolour“ weggelassen wird, wird nur die Polygonumrandung gezeichnet. Wenn „bordercolour“ weggelassen wird, wird standardmäßig die aktuelle Standard-Vordergrundfarbe verwendet.<br>Wenn das letzte xy-Koordinatenpaar nicht mit dem ersten übereinstimmt, erstellt die Firmware automatisch ein zusätzliches xy-Koordinatenpaar, um das Polygon zu vervollständigen. Die Größe der Arrays sollte mindestens so groß sein wie die Anzahl der x,y-Koordinatenpaare.<br>'n' kann ein Array sein, und die Farben können optional ebenfalls Arrays sein, wie folgt:<br>POLYGON n(), xarray%(), yarray%() [, bordercolour()] [, fillcolour()]<br>POLYGON n(), xarray%(), yarray%() [, bordercolour] [, fillcolour] |

| | |
|---|---|
| | Die Elemente von „array n()“ legen die Anzahl der xy-Koordinatenpaare in jedem der Polygone fest. Z. B. würde DIM n(1)=(3,3) festlegen, dass 2 Polygone mit jeweils drei Eckpunkten gezeichnet werden sollen. Die Größe des n-Arrays bestimmt die Anzahl der Polygone, die gezeichnet werden, es sei denn, ein Element mit dem Wert Null wird 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 enthalten wie die Summe der Werte im n-Array.<br>Jedes Polygon kann geschlossen sein, wobei das erste und das letzte Element identisch sind. Wenn das letzte Element nicht mit dem ersten übereinstimmt, erstellt die Firmware automatisch ein zusätzliches x,y-Koordinatenpaar, um das Polygon zu vervollständigen. Wenn die Füllfarbe weggelassen wird, werden nur die Polygonkonturen gezeichnet.<br>Die Farbparameter können ein einzelner Wert sein – in diesem Fall werden alle Polygone in derselben Farbe gezeichnet – oder sie können Arrays mit derselben Kardinalität wie „n“ sein. In diesem Fall kann jedes gezeichnete Polygon eine andere Farbe für den Rand und/oder die Füllung haben.<br>Zum Beispiel werden damit 3 Dreiecke in Gelb, Grün und Rot gezeichnet:<br>`DIM c%(2)=(3,3,3)`<br>`DIM x%(8)=(100,50,150,100,50,150,100,50,150)`<br>`DIM y%(8)=(50,100,100,150,200,200,250,300,300)`<br>`DIM fc%(2)=(rgb(yellow),rgb(green),rgb(red))`<br>`POLYGON c%(),x%(),y%(),fc%(),fc%()` |
| PORT(start, nbr [,start, nbr]…) = value | Setze mehrere I/O-Pins gleichzeitig (d. h. mit einem Befehl).<br>'start' ist eine I/O-Pin-Nummer, und das niedrigste Bit in 'value' (Bit 0) wird verwendet, um diesen Pin zu setzen. Bit 1 wird verwendet, um den Pin 'start' plus 1 zu setzen, Bit 2 setzt den Pin 'start'+2 und so weiter für die Anzahl der Bits 'nbr'. Die verwendeten I/O-Pins müssen fortlaufend nummeriert sein, und jeder I/O-Pin, der ungültig oder nicht als Ausgang konfiguriert ist, verursacht einen Fehler. Das Paar „start/nbr“ kann wiederholt werden, wenn eine zusätzliche Gruppe von Ausgangspins hinzugefügt werden muss.<br>Zum Beispiel: `PORT(15, 4, 23, 4) = &B10000011`<br>Damit werden acht I/O-Pins gesetzt. Die Pins 15 und 16 werden auf High gesetzt, während 17, 18, 23, 24 und 25 auf Low gesetzt werden und schließlich 26 auf High.<br>Dieser Befehl kann verwendet werden, um bequem mit parallelen Geräten wie LCD-Displays zu kommunizieren. 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.<br>Hinweis: Wenn die Pins mit der GPn-Syntax definiert werden, ignoriert die Firmware ungültige Pins, sodass `PORT(GP0, 8) = &B10000011` acht I/O-Pins setzt. Die Pins GP0, GP1 und GP7 werden auf High gesetzt, während GP2, GP3, GP4, GP5 und GP6 auf Low gesetzt werden.<br>Siehe die PORT-Funktion, um gleichzeitig von mehreren Pins zu lesen. |
| PRINT expression [[,; ]expression] … etc | Gibt Text an die serielle Konsole aus, gefolgt von einem Wagenrücklauf/Zeilenumbruch-Paar. Es können mehrere Ausdrücke verwendet werden, die entweder durch ein:<br>• Komma (,), das das Tabulatorzeichen ausgibt<br>• Semikolon (;), das nichts ausgibt (es dient nur zur Trennung von Ausdrücken).<br>• Nichts oder ein Leerzeichen, das sich wie ein Semikolon verhält. |

| | |
|---|---|
| | Ein Semikolon (;) oder ein Komma (,) am Ende der Ausdrucksliste unterdrückt die Ausgabe des Wagenrücklauf-/Zeilenumbruch-Paares am Ende einer print-Anweisung.<br>Beim Ausgeben wird einer Zahl ein Leerzeichen vorangestellt, wenn sie positiv ist, oder ein Minuszeichen (-), wenn sie negativ ist, aber es folgt kein Leerzeichen. Ganzzahlen werden ohne Dezimalpunkt ausgegeben, während Brüche mit Dezimalpunkt und den signifikanten Dezimalstellen ausgegeben werden. Große oder kleine Gleitkommazahlen werden automatisch im wissenschaftlichen Notationsformat ausgegeben.<br>Die Funktion TAB() kann verwendet werden, um den Text an einer bestimmten Spalte auszurichten, und die Funktion STR$() kann verwendet werden, um Zeichenfolgen auszurichten oder anderweitig zu formatieren. |
| PRINT #nbr, expression [[,; ]expression] … etc | Wie oben, außer dass die Ausgabe an einen seriellen Kommunikationsport oder eine für OUTPUT oder APPEND geöffnete Datei mit der Dateinummer „nbr“ geleitet wird. Siehe den Befehl OPEN. |
| PRINT #GPS, expression [[,; ]expression] … etc | Gibt eine NMEA-Zeichenkette an ein geöffnetes GPS-Gerät aus. Die Zeichenkette muss mit einem $-Zeichen beginnen und mit einem *-Zeichen enden. Die Prüfsumme wird automatisch berechnet und zusammen mit den CR/LF-Zeichen an die Zeichenkette angehängt. |
| PRINT @(x [, y]) expression<br>or<br>PRINT @(x, [y], m) expression | Funktioniert auf der Terminal-Konsole eines angeschlossenen Computers oder über VGA/HDMI-Video oder auf dem Display, wenn OPTION LCDPANEL CONSOLE aktiviert ist.<br>Entspricht dem Standardbefehl PRINT, außer dass der Cursor an den Koordinaten x, y positioniert wird, ausgedrückt in Pixeln. Wenn y weggelassen wird, wird der Cursor an „x“ in der aktuellen Zeile positioniert.<br>Beispiel: PRINT @(150, 45) „Hello World“<br>Die @-Funktion kann an beliebiger Stelle in einem Druckbefehl verwendet werden.<br>Beispiel: PRINT @(150, 45) "Hallo" @(150, 55) "Welt"<br>Mit der Funktion @(x,y) kannst du den Cursor an einer beliebigen Stelle auf oder außerhalb des Bildschirms positionieren. Zum Beispiel zeigt PRINT @(-10, 0) „Hello“ nur „llo“ an, da die ersten beiden Zeichen nicht angezeigt werden konnten, weil sie außerhalb des Bildschirms lagen.<br>Die Funktion @(x,y) unterdrückt automatisch den automatischen Zeilenumbruch, der normalerweise erfolgt, wenn der Cursor über den rechten Bildschirmrand hinausgeht.<br>Wenn „m“ angegeben wird, verhält sich der Videomodus wie folgt:<br>m = 0 Normaler Text (weiße Buchstaben, schwarzer Hintergrund)<br>m = 1 Der Hintergrund wird nicht gezeichnet (d. h. transparent)<br>m = 2 Das Video wird invertiert (schwarze Buchstaben, weißer Hintergrund)<br>m = 5 Aktuelle Pixel werden invertiert (transparenter Hintergrund) |

| | |
|---|---|
| PULSE pin, width | Erzeugt einen Impuls am „Pin“ mit einer Dauer von „width“ ms. „width“ kann ein Bruch sein. Zum Beispiel entspricht 0,01 10 µs, was die Erzeugung sehr schmaler Impulse ermöglicht.<br>Der erzeugte Impuls hat die entgegengesetzte Polarität zum Zustand des I/O-Pins zum Zeitpunkt der Befehlsausführung. Wenn der Ausgang beispielsweise auf High gesetzt ist, erzeugt der PULSE-Befehl einen fallenden Impuls.<br>Hinweise:<br>• „pin“ muss als Ausgang konfiguriert sein.<br>• Bei einem Impuls von weniger als 3 ms beträgt die Genauigkeit ± 1 µs.<br>• Bei einem Impuls von 3 ms oder mehr beträgt die Genauigkeit ± 0,5 ms.<br>• Ein Impuls von 3 ms oder mehr läuft im Hintergrund. Bis zu fünf verschiedene und gleichzeitige Impulse können im Hintergrund laufen, und die Zeit jedes einzelnen kann durch einen neuen PULSE-Befehl geändert oder durch einen PULSE-Befehl mit dem Wert Null für „width“ beendet werden. |
| PWM channel, frequency, [dutyA] [,dutyB][,phase][,defer] | Es stehen 8 separate PWM-Frequenzen zur Verfügung (Kanäle 0 bis 7) und bis zu 16 Ausgänge mit individuell steuerbarem Tastverhältnis. Du kannst für jeden Kanal entweder PWMnA oder PWMnB oder beide nutzen – ohne Einschränkung. Tastverhältnisse werden als Prozentsatz angegeben, und du kannst einen negativen Wert verwenden, um den Ausgang zu invertieren (-100,0 <= duty <= 100,0).<br>Um nur Kanal B zu verwenden, benutze die Syntax: „PWM-Kanal, Frequenz, , dutyB“ – beachte das doppelte Komma vor dem gewünschten Tastverhältnis.<br>Mindestfrequenz = (cpuspeed + 1) / (2^24) Hz. Die maximale Geschwindigkeit beträgt OPTION CPUSPEED/4. Bei sehr hohen Geschwindigkeiten werden die Tastverhältnisse zunehmend eingeschränkt.<br>„Phase“ ist ein Parameter, der dafür sorgt, dass die Wellenformen so zentriert werden, dass eine Wellenform mit kürzerem Tastverhältnis im gleichen Abstand von einer längeren Wellenform beginnt und endet. Verwende 1, um diesen Modus zu aktivieren, und 0 (oder lass den Parameter weg), um den normalen Betrieb beizubehalten<br>Der Parameter „deferredstart“ konfiguriert die PWM-Kanäle~~,~~ wenn er auf 1 gesetzt ist~~,~~ startet den Ausgang jedoch nicht. Sie können ~~dann~~ mit dem Befehl PWM SYNC gestartet werden. Dies kann verwendet werden, um unerwünschte Startartefakte zu vermeiden<br>Der PWM-Befehl kann auch Servos wie folgt ansteuern:<br>`PWM 1,50,(position_as_a_percentage * 0.05 + 5)` |
| PWM SYNC s0 [,s1][,s2][,s3][,s4][,s5][,s6][,s7 ] | Dies initiiert den PWM auf Kanälen, für die ein verzögerter Start definiert wurde, oder synchronisiert einfach bereits laufende Kanäle. Die Stärke liegt jedoch in der Möglichkeit, die Kanäle gegeneinander zu versetzen (definiert als Prozentsatz der Zeitdauer gemäß dem Tastverhältnis – kann ein Float-Wert sein)<br>Du kannst einen Versatz von -1 verwenden, um einen Kanal von der Synchronisation auszuschließen |
| PWM channel, OFF | Ausgang auf „Kanal“ stoppen. |

| | |
|---|---|
| RAM | Nur RP2350 mit aktiviertem PSRAM<br>Der RAM-Befehl ermöglicht den Zugriff auf bis zu 5 RAM-Programmspeicherplätze (ähnlich wie Flash-Speicherplätze). RAM-Speicherplätze bleiben nach einem Hardware- und Software-Reset erhalten, jedoch nicht nach einem Neustart. |
| RAM LIST | Zeigt eine Liste aller RAM-Speicherplätze einschließlich der ersten Zeile des Programms an. |
| RAM LIST n [,all] | Zeigt das in Speicherplatz n gespeicherte Programm an. Verwende ALL, um die Liste ohne Seitenumbrüche anzuzeigen. |
| RAM ERASE n | Lösche einen RAM-Programmspeicherplatz. |
| RAM ERASE ALL | Lösche alle RAM-Programmspeicherplätze. |
| RAM SAVE n | Speichere das aktuelle Programm an dem angegebenen RAM-Speicherplatz. |
| RAM LOAD n | Lade ein Programm vom angegebenen RAM-Speicherplatz in den Programmspeicher. |
| RAM RUN n | Führt das Programm an RAM-Speicherplatz n aus, löscht alle Variablen. Ändert den<br>Programmspeicher. |
| RAM CHAIN n | Führt das Programm an RAM-Speicherplatz n aus und lässt alle Variablen unverändert (ermöglicht ein<br>Programm, das viel größer ist als der Programmspeicher). Verändert den Programmspeicher. Hinweis: Wenn das verkettete Programm den Befehl READ verwendet, muss es vor dem ersten Lesevorgang RESTORE aufrufen. |
| RAM OVERWRITE n | Lösche einen RAM-Programmspeicherplatz und speichere dann das aktuelle Programm an der<br>angegebenen RAM-Speicherplatz. |
| RAM FILE LOAD n, fname$ [,O[VERWRITE]] | Lädt die MMBasic-Datei fname$ in den angegebenen RAM-Speicherplatz. Wenn der optionale Parameter OVERWRITE (oder O) angegeben wird, wird der Inhalt des<br>Flash-Speicherplatzes wird überschrieben, ohne dass ein Fehler ausgelöst wird. |
| RANDOMIZE nbr | Setze den Zufallszahlengenerator mit „nbr“.<br>Auf dem RP2040 wird der Zufallszahlengenerator beim Hochfahren mit Null initialisiert und erzeugt jedes Mal dieselbe Zufallszahlensequenz. Um jedes Mal eine andere Zufallssequenz zu erzeugen, musst du einen anderen Wert für „nbr“ verwenden (die TIMER-Funktion ist dafür praktisch).<br>Dieser Befehl hat auf dem RP2350 keine Wirkung, da dieser über einen Hardware-Zufallsgenerator verfügt, der keine Initialisierung benötigt. |
| RAY | Nur RP2350<br>Befehl zur Entwicklung von Spielen mit Raycasting-Techniken. Siehe das separate Handbuch „Raycaster_User_manual.pdf“ |
| RBOX x, y, w, h [, r] [, c] [, fill] | Zeichnet auf dem Videoausgang oder dem angeschlossenen LCD-Bildschirm ein Rechteck mit abgerundeten Ecken, beginnend bei 'x' und 'y', das 'w' Pixel breit und 'h' Pixel hoch ist.<br>'r' ist der Radius der Ecken des Kastens. Der Standardwert ist 10.<br>'c' gibt die Farbe an und wird standardmäßig auf die Standard-Vordergrundfarbe gesetzt, wenn nichts anderes angegeben wird. 'fill' ist die |

| | |
|---|---|
| | Füllfarbe. Sie kann weggelassen oder auf -1 gesetzt werden; in diesem Fall wird das Feld nicht gefüllt.<br>Alle Parameter können als Arrays angegeben werden, und die Software zeichnet die Anzahl der Kästchen, die durch die Dimensionen des kleinsten Arrays bestimmt wird. 'x', 'y', 'w' und 'h' müssen alle Arrays oder alle einzelne Variablen/Konstanten sein, andernfalls wird ein Fehler ausgegeben. 'r', 'c' und 'fill' können entweder Arrays oder einzelne Variablen/Konstanten sein.<br>Eine Definition der Farben und Grafikkoordinaten findest du im Kapitel *„Grafikbefehle und -funktionen*“. |
| REDIM [PRESERVE] array1(dimensions) [, array2(dimensions]…[,arrayn( dimensions] | Dies passt die Größe des/der angegebenen Arrays mit den angegebenen Dimensionen an.<br>Wenn der optionale Unterbefehl PRESERVE angegeben wird, werden die vorhandenen Daten in das neue Array kopiert.<br>Das neue Array kann größer oder kleiner als das Original sein.<br>Bei Zeichenfolgen-Arrays bleibt die ursprünglich angegebene LÄNGE erhalten.<br>Beachte, dass bei mehrdimensionalen Arrays nur die letzte Dimension geändert werden kann, wenn PRESERVE verwendet wird.<br>Bei Verwendung von PRESERVE muss genügend Speicherplatz vorhanden sein, damit sowohl das ursprüngliche Array als auch dessen geänderte Version gleichzeitig existieren können. Der dem ursprünglichen Array zugewiesene Speicher wird beim Beenden des Befehls freigegeben. |
| READ variable[, variable].. | Liest Werte aus DATA-Anweisungen und weist diese Werte den genannten Variablen zu. Die Variablentypen in einer READ-Anweisung müssen mit den Datentypen in den DATA-Anweisungen übereinstimmen, während sie gelesen werden.<br>Arrays können als Variablen verwendet werden (angegeben mit leeren Klammern, z. B. a()) und in diesem Fall bestimmt die Größe des Arrays, wie viele Elemente gelesen werden sollen. Wenn das Array mehrdimensional ist, wird die Dimension ganz links am schnellsten durchlaufen.<br>Siehe auch DATA und RESTORE. |
| READ SAVE<br>or<br>READ RESTORE | READ SAVE speichert den virtuellen Zeiger, den der READ-Befehl verwendet, um auf das nächste zu lesende DATA zu zeigen. READ RESTORE stellt den zuvor gespeicherten Zeiger wieder her.<br>Dadurch können Unterprogramme Daten lesen und anschließend den Lesezeiger wiederherstellen, um andere Teile des Programms nicht zu stören, die möglicherweise dieselben Datenanweisungen lesen. Diese Befehle können verschachtelt werden. |
| REFRESH | Löst eine Aktualisierung des Bildschirms für E-Ink-Schwarz-Weiß-Displays aus.<br>Diese können jeweils nur bildschirmweise aktualisiert werden, und wenn OPTION AUTOREFRESH auf OFF steht, kann dieser Befehl verwendet werden, um den Schreibvorgang auszulösen.<br>Dieser Befehl funktioniert mit den folgenden Displays: N5110, SSD1306I2C, SSD1306I2C32, SSD1306SPI, ST7920. |
| REM string | REM ermöglicht es, Anmerkungen in ein Programm einzufügen.<br>Beachte, dass die Microsoft-Konvention mit dem einfachen Anführungszeichen (‘) zur Kennzeichnung von Kommentaren ebenfalls unterstützt wird und bevorzugt wird. |

| | |
|---|---|
| RENAME old$ AS new$ | Benenne eine Datei oder ein Verzeichnis von „old$“ in „new$“ um. Beide sind Zeichenfolgen.<br>Sowohl in „old$“ als auch in „new$“ kann ein Verzeichnispfad verwendet werden. Wenn sich die Pfade unterscheiden, wird die in „old$“ angegebene Datei unter dem angegebenen Dateinamen in den in „new$“ angegebenen Pfad verschoben. |
| RESTORE [Zeile] | Setzt die Zeilen- und Positionszähler für die READ-Anweisung zurück.<br>Wenn „line“ angegeben ist, werden die Zähler auf den Anfang der angegebenen Zeile zurückgesetzt. „line“ kann eine Zeilennummer, eine Beschriftung oder eine Variable mit diesen Werten sein.<br>Wenn „Zeile“ nicht angegeben ist, werden die Zähler auf den Anfang des Programms zurückgesetzt. |
| RMDIR dir$ | Entferne oder lösche das Verzeichnis „dir$“ auf dem standardmäßig ausgewählten Laufwerk. |
| RTC GETTIME<br><br>RTC SETTIME year, month, day, hour, minute, second<br><br>RTC SETREG reg, value<br>RTC GETREG reg, var | RTC GETTIME ruft das aktuelle Datum/die aktuelle Uhrzeit von einer Echtzeituhr vom Typ PCF8563, DS1307, DS3231 oder RV3028 ab und stellt die interne MMBasic-Uhr ein. Das Datum/die Uhrzeit kann dann mit den Funktionen DATE$ und TIME$ abgerufen werden.<br>RTC SETTIME stellt die Zeit im Uhrenchip ein. „Stunde“ muss im 24-Stunden-Format angegeben werden. „year“ kann zwei- oder vierstellig sein. Der Befehl RTC SETTIME akzeptiert auch ein einzelnes String-Argument im Format *dd/mm/yy hh:mm*. Das bedeutet, dass das Datum/die Uhrzeit vom Benutzer über eine GUI-FORMATBOX mit dem Format DATETIME2 eingegeben werden kann (siehe *Advanced Graphics Functions.pdf*).<br>Die Befehle RTC SETREG und GETREG können verwendet werden, um den Inhalt von Registern im Echtzeituhr-Chip zu setzen oder auszulesen. 'reg' ist die Registernummer, 'value' ist der Wert, der im Register gespeichert werden soll, und 'var' ist eine Variable, die den aus dem Register ausgelesenen Wert erhält. Diese Befehle sind für den normalen Betrieb nicht erforderlich, können aber zur Steuerung spezieller Funktionen des Chips (Alarme, Ausgangssignale usw.) verwendet werden. Sie sind auch nützlich, um temporäre Informationen im batteriegepufferten RAM des Chips zu speichern.<br>Diese Chips sind I²C-Geräte und müssen gemäß der Option SYSTEM I2C mit geeigneten Pull-up-Widerständen an die beiden I²C-Pins angeschlossen werden.<br>Siehe auch den Befehl OPTION RTC AUTO ENABLE. |
| RUN<br>or<br>RUN [file$] [, cmdline$] | Führe ein Programm aus.<br>Wenn „file$“ nicht angegeben wird, wird das aktuell im Programmspeicher befindliche Programm ausgeführt.<br>Wenn „file$“ angegeben wird, führe die angegebene Datei aus dem Dateisystem des Flash-Speichers oder der SD-Karte aus. Wenn „file$“ keine „.BAS“-Erweiterung enthält, wird automatisch eine hinzugefügt.<br>Wenn „cmdline$“ angegeben wird, übergebe seinen Wert an die Konstante MM.CMDLINE$ des Programms, wenn es ausgeführt wird. Wenn „cmdline$“ nicht angegeben wird, wird ein leerer String-Wert an MM.CMDLINE$ übergeben. Hinweise:<br>• Sowohl „file$“ als auch „cmdline$“ können als Zeichenfolgenausdrücke angegeben werden.<br>• Verwende FLASH RUN n, um ein in einem Flash-Slot gespeichertes Programm auszuführen. |

| | |
|---|---|
| SAVE file$ | Speichert das Programm im aktuellen Arbeitsverzeichnis des Flash-Dateisystems oder auf der SD-Karte als „file$“. Beispiel: SAVE „TEST.BAS“<br>Wenn keine Erweiterung angegeben wird, wird „.BAS“ an den Dateinamen angehängt.<br>Siehe auch FLASH SAVE *n* zum Speichern in einem Flash-Slot. |
| SAVE CONTEXT [CLEAR] | Speichert den Variablenbereich und löscht ihn optional – der Befehl sollte im Hauptprogramm und nicht innerhalb einer Subroutine verwendet werden. Dadurch wird der gesamte Variablenbereich auf dem Laufwerk A: gespeichert. Der Befehl schlägt fehl, wenn auf dem Laufwerk A: nicht genügend freier Speicherplatz vorhanden ist. Bei einem RP2350 mit PSRAM wird der Variablenbereich in einem reservierten Bereich im PSRAM gespeichert und das Laufwerk A: wird nicht verwendet.<br>Siehe auch LOAD CONTEXT |
| SAVE DATA fname$, address, nBytes | Schreibt „nBytes“ von der Speicheradresse „address“ in die Datei fname$.<br>Beachte, dass die Adresse nicht überprüft wird; achte daher darauf, dass die Adresse innerhalb der Pico-Adresszuordnung gültig ist. |
| SAVE IMAGE file$ [,x, y, w, h]<br>or<br>SAVE COMPRESSED IMAGE file$ [,x, y, w, h] | Speichert das aktuelle Bild auf dem Videoausgang oder dem LCD-Panel als BMP-Datei. Jedes LCD-Panel muss lesbar sein, zum Beispiel ein Panel auf ILI9341-Basis oder ein VIRTUAL_M- oder VIRTUAL_C-Panel.<br>'file$' ist der Name der Datei. Wenn keine Erweiterung angegeben wird, wird „.BMP“ an den Dateinamen angehängt. Das Bild wird als 24-Bit-True-Color-Bild gespeichert.<br>‘x’, ‘y’, ‘w’ und ‘h’ sind optional und geben die Koordinaten (‘x’ und ‘y’ sind die Koordinaten der oberen linken Ecke) sowie die Abmessungen (Breite und Höhe) des zu speichernden Bereichs an. Wenn nichts angegeben wird, wird der gesamte Bildschirm gespeichert. Beachte, dass ‘width’, falls verwendet, ein Vielfaches von 2 sein muss.<br>SAVE COMPRESSED IMAGE funktioniert genauso, außer dass RLE-Komprimierung verwendet wird, um die Dateigröße zu reduzieren. |
| SAVE PERSISTENT n% | Speichert den Wert n% an einem speziellen Speicherort, der einen Watchdog-Reset oder einen physischen Reset übersteht, jedoch keinen Stromausfall<br>Siehe auch MM.PERSISTENT oder MM.INFO(PERSISTENT) |
| SEEK [#]fnbr, pos | Positioniert den Lese-/Schreibzeiger in einer Datei, die auf dem Flash-Dateisystem oder der SD-Karte geöffnet wurde, für den RANDOM-Zugriff auf das Byte „pos“.<br>Das erste Byte in einer Datei ist mit eins nummeriert, daher positioniert SEEK #5,1 den Lese-/Schreibzeiger am Anfang der als #5 geöffneten Datei. |
| SELECT CASE value<br>CASE testexp [[, testexp] …]<br><statements><br><statements><br>CASE test-n [[, test-n] …]<br><statements><br><statements><br>CASE ELSE<br><statements><br><statements><br>END SELECT | Führt je nach Wert eines Ausdrucks eine von mehreren Anweisungsgruppen aus. 'value' ist der zu prüfende Ausdruck. Es kann sich um eine Zahlen- oder Zeichenfolgenvariable oder einen komplexen Ausdruck handeln.<br>'testexp' (oder test-n) ist der Wert, mit dem verglichen werden soll. Das kann sein:<br>• Ein einzelner Ausdruck (z. B. 34, „string“ oder PIN(4)*5), dem er entsprechen darf<br>• Ein Wertebereich in Form von zwei einzelnen Ausdrücken, getrennt durch das Schlüsselwort „TO“ (z. B. 5 TO 9 oder „aa“ TO „cc“)<br>• Ein Vergleich, der mit dem Schlüsselwort „IS“ beginnt (was optional ist). Zum Beispiel: IS > 5, IS <= 10.<br>Wenn mehrere Testausdrücke (durch Kommas getrennt) verwendet werden, ist die CASE-Anweisung wahr, wenn einer dieser Tests wahr ist. |

| | |
|---|---|
| | Wenn „value“ nicht mit einem „testexp“ abgeglichen werden kann, wird es automatisch mit dem CASE ELSE abgeglichen. Wenn kein CASE ELSE vorhanden ist, führt das Programm keine <Anweisungen> aus und fährt mit dem Code nach dem END SELECT fort. Wenn eine Übereinstimmung gefunden wird, werden die <Anweisungen> nach der CASE-Anweisung ausgeführt, bis END SELECT oder ein weiteres CASE erreicht wird; dann fährt das Programm mit dem Code nach dem END SELECT fort.<br>Es kann eine unbegrenzte Anzahl von CASE-Anweisungen verwendet werden, aber es darf nur ein CASE ELSE geben, und dieses sollte das letzte vor dem END SELECT sein.<br>Beispiel:<br>`SELECT CASE nbr%`<br>`  CASE 4, 9, 22, 33 TO 88`<br>`  Anweisungen`<br>`  CASE IS < 4, IS > 88, 5 TO 8`<br>`  Anweisungen`<br>`  CASE ELSE`<br>`  Anweisungen`<br>`END SELECT`<br>Jedes SELECT CASE muss genau eine passende END SELECT-Anweisung haben. Es können beliebig viele SELECT…CASE-Anweisungen innerhalb der CASE-Anweisungen anderer SELECT…CASE-Anweisungen verschachtelt werden. |
| SERVO channel [positionA] [,positionB] | Steuere einen Standard-Servo an.<br>„positionA“ und „positionB“ können zwischen -20 und 120 liegen und erzeugen ein 50-Hz-Signal zwischen 800 µs und 2,2 ms<br>Wie beim PWM-Befehl müssen die Pins mit SETPIN n,PWM eingerichtet werden<br>Um nur Kanal B zu verwenden, benutze die Syntax: SERVO Kanal,,Position.<br>Beachte die beiden Kommas, um anzugeben, dass Kanal A nicht eingestellt wird.<br>Sieh dir die Pinbelegung an, um den Kanal und den Unterkanal (A oder B) für jeden Pin zu ermitteln.<br>Beispiele:<br>90°-Servo: 0 = 0° und 90 = 90°<br>180°-Servo: 0 = 0° und 90 = 180°<br>Hinweis: Bei Werten < 0 oder > 90 kann der Strom ansteigen, wenn das Servo seine Endposition erreicht.<br>360°-Servo: Geschwindigkeit 50 = Stopp, Links = L->H:51-100, Rechts = L->H:49-0. |
| SETPIN pin, cfg [, option] | Konfiguriert einen externen I/O-Pin. Eine allgemeine Beschreibung der Ein-/Ausgabefunktionen des Pico findest du im Kapitel *„Verwendung der I/O-Pins*“.<br>'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' ist ein Schlüsselwort und kann einer der folgenden Werte sein:<br>OFF — Nicht konfiguriert oder inaktiv<br>AIN — Analoger Eingang (d. h. Messung der Spannung am Eingang).<br>ARAW — Schneller Analogeingang, der einen Wert zwischen 0 und 4095 zurückgibt.<br>DIN — Digitaler Eingang<br>Wenn „option“ weggelassen wird, ist der Eingang hochohmig<br>Wenn „option“ das Schlüsselwort „PULLUP“ oder „PULLDOWN“ ist, wird ein konstanter Strom von etwa 50 µA |

| | |
|---|---|
| | verwendet, um den Eingangspin auf 3,3 V hoch- oder herunterzuziehen. Aufgrund eines Fehlers in den RP2350-Chips wird empfohlen, einen Pulldown mit einem Widerstand von 8,2 kΩ oder weniger zu implementieren.<br>FIN Frequenzeingang<br>Mit „option“ kannst du die Gate-Zeit festlegen (die Zeitspanne, über die die Eingangszyklen gezählt werden). Dies kann eine beliebige Zahl zwischen 10 ms und 100000 ms sein. Die Funktion PIN() gibt die Frequenz immer korrekt in Hz zurück, unabhängig von der verwendeten Gate-Zeit. Wenn „option“ weggelassen wird, beträgt die Gate-Zeit 1 Sekunde.<br>Die Pins können GP6, GP7, GP8 oder GP9 sein (kann mit OPTION COUNT geändert werden).<br>PIN Periodeneingabe<br>Mit „option“ kannst du die Anzahl der Eingangszyklen angeben, über die die Periodenmessung gemittelt werden soll. Es kann eine beliebige Zahl zwischen 1 und 10000 sein. Die Funktion PIN() gibt immer die durchschnittliche Periode eines Zyklus korrekt skaliert in ms zurück, unabhängig von der Anzahl der für den Durchschnitt verwendeten Zyklen. Wenn „option“ weggelassen wird, wird die Periode von nur einem Zyklus verwendet.<br>Die Pins können GP6, GP7, GP8 oder GP9 sein (kann mit OPTION COUNT geändert werden).<br>CIN Zählender Eingang<br>Mit „option“ kannst du festlegen, welche Flanke den Zählvorgang auslöst<br>und ob ein Pull-up oder Pull-down aktiviert ist<br>2 gibt eine fallende Flanke mit Pull-up an,<br>3 gibt an, dass sowohl eine fallende als auch eine steigende Flanke einen<br>Zählvorgang auslösen, ohne dass ein Pull-up angewendet wird,<br>5 gibt beide Flanken an, jedoch mit Pull-up.<br>Wenn „option“ weggelassen wird, löst eine steigende Flanke den Zählvorgang aus.<br>Aufgrund eines Fehlers in den RP2350-Chips wird Pulldown nicht empfohlen.<br>Die Pins können GP6, GP7, GP8 oder GP9 sein (kann mit OPTION COUNT geändert werden).<br>DOUT Digitaler Ausgang<br>„option“ wird in diesem Modus nicht verwendet.<br>Die Funktionen PIN() und PORT() können ebenfalls verwendet werden, um den Wert an einem oder mehreren Ausgangspins zurückzugeben. Siehe die Funktion PIN() zum Lesen von Eingängen und die Anweisung PIN()= zum Setzen eines Ausgangs. Siehe den folgenden Befehl, wenn ein Interrupt konfiguriert ist. |
| SETPIN pin, cfg, target [, option] | Konfiguriert „pin“ so, dass er gemäß „cfg“ einen Interrupt generiert. Jeder I/O-Pin, der für digitale Eingänge geeignet ist, kann so konfiguriert werden, dass er einen Interrupt generiert, wobei maximal zehn Interrupts gleichzeitig konfiguriert werden können.<br>'cfg' ist ein Schlüsselwort und kann einer der folgenden Werte sein:<br>OFF Nicht konfiguriert oder inaktiv<br>INTH Interrupt bei steigendem Pegel<br>INTL Interrupt bei High-zu-Low-Eingang<br>INTB Interrupt bei beiden (d. h. bei jeder Änderung des Eingangs) |

| | |
|---|---|
| | „target“ ist eine benutzerdefinierte Subroutine, die aufgerufen wird, wenn das Ereignis eintritt. Die Rückkehr aus dem Interrupt erfolgt über die Befehle END SUB oder EXIT SUB. „option“ entspricht der in SETPIN Pin DIN (oben) verwendeten Option.<br>In diesem Modus wird der Pin außerdem als digitaler Eingang konfiguriert, sodass der Wert des Pins jederzeit mit der Funktion PIN() abgerufen werden kann.<br>Eine allgemeine Beschreibung findest du im Kapitel *„Verwendung der I/O-Pins*“. |
| SETPIN GP25, DOUT \| HEARTBEAT | NICHT IN DER WEBMITE-VERSION<br>Diese Version von SETPIN steuert die integrierte LED.<br>Wenn er als DOUT konfiguriert ist, kann er programmgesteuert ein- und ausgeschaltet werden.<br>Wenn sie als HEARTBEAT konfiguriert ist, blinkt sie bei eingeschalteter Stromversorgung kontinuierlich 1 Sekunde an, 1 Sekunde aus. Dies ist der Standardzustand, der wiederhergestellt wird, wenn das Anwenderprogramm nicht mehr läuft. |
| SETPIN p1[, p2 [, p3]], device | Diese Befehle dienen der Pin-Zuweisung für spezielle Geräte.<br>Die Pins müssen aus dem Pinbelegungsdiagramm ausgewählt und zugewiesen werden, bevor die Geräte verwendet werden können. Beachte, dass die Pins (z. B. rx, tx usw.) in beliebiger Reihenfolge deklariert werden können und dass auf die Pins über ihre Pin-Nummer (z. B. 1, 2) oder GP-Nummer (z. B. GP0, GP1) verwiesen werden kann.<br>Beachte, dass bei der WebMite-Version:<br>• SPI1 und SPI2 sind auf GP20 bis GP28 nicht verfügbar<br>• COM1 und COM2 sind auf P20 bis GP28 nicht verfügbar<br>• I2C ist auf Pin 34 (GP28) nicht verfügbar<br>• Folgende sind nicht verfügbar: GP29, GP25, GP24 und GP23 |
| SETPIN rx, tx, COM1 | Weise die Pins zu, die für den seriellen Port COM1 verwendet werden sollen.<br>Gültige Pins sind RX: GP1, GP13 oder GP17<br>TX: GP0, GP12, GP16 oder GP28 |
| SETPIN rx, tx, COM2 | Weise die Pins zu, die für den seriellen Port COM2 verwendet werden sollen.<br>Gültige Pins sind RX: GP5, GP9 oder GP21<br>TX: GP4, GP8 oder GP20 |
| SETPIN rx, tx, clk, SPI | Weise die Pins zu, die für den SPI-Port SPI verwendet werden sollen.<br>Gültige Pins sind RX: GP0, GP4, GP16 oder GP20<br>TX: GP3, GP7 oder GP19<br>CLK: GP2, GP6 oder GP18 |
| SETPIN rx, tx, clk, SPI2 | Weise die Pins zu, die für den SPI-Port SPI2 verwendet werden sollen.<br>Gültige Pins sind RX: GP8, GP12 oder GP28<br>TX: GP11, GP15 oder GP27<br>CLK: GP10, GP14 oder GP26 |
| SETPIN sda, scl, I2C | Weise die Pins zu, die für den $I^2C$-Port I2C verwendet werden sollen.<br>Gültige Pins sind SDA: GP0, GP4, GP8, GP12, GP16, GP20 oder GP28<br>SCL: GP1, GP5, GP9, GP13, GP17 oder GP21 |

| | |
|---|---|
| SETPIN sda, scl, I2C2 | Weise die Pins zu, die für den I²C-Port I2C2 verwendet werden sollen.<br>Gültige Pins sind SDA: GP2, GP6, GP10, GP14, GP18, GP22 oder GP26<br>SCL: GP3, GP7, GP11, GP15, GP19 oder GP27 |
| SETPIN-Pin, PWM[nx] | Weise den Pin PWMnx zu<br>„n“ ist die PWM-Nummer (0 bis 7) und „x“ ist der Kanal (A oder B). n und x sind optional.<br>Der SETPIN-Pin kann geändert werden, bis der PWM-Befehl ausgegeben wird. Ab diesem Zeitpunkt ist der Pin für PWM gesperrt, bis PWMn,OFF ausgegeben wird. |
| SETPIN Pin, IR | Weise Pins für die Infrarot-Kommunikation (IR) zu (kann jeder beliebige Pin sein). |
| SETPIN pin, PIOn | Reserviere einen Pin für die Verwendung durch PIO0, PIO1 oder PIO2 (nur RP2350) (siehe *Anhang F* für Details zu PIO). |
| SETPIN GP1, FFIN [,gate] | <u>NUR RP2350</u><br>Legt GP1 als schnellen Frequenzeingang fest.<br>Es können Eingänge bis zur Hälfte der CPU-Geschwindigkeit erfasst werden.<br>Mit „gate“ kannst du die Gate-Zeit festlegen (die Zeitspanne, in der die Eingangszyklen gezählt werden). Dies kann eine beliebige Zahl zwischen 10 ms und 100000 ms sein. Die Funktion PIN() gibt die Frequenz immer korrekt in Hz zurück, unabhängig von der verwendeten Gate-Zeit. Wenn „option“ weggelassen wird, beträgt die Gate-Zeit 1 Sekunde.<br>Die Funktion nutzt PWM-Kanal 0 zum Zählen, daher ist sie mit jeder anderen Nutzung dieses PWM-Kanals unvereinbar. |
| SETTICK period, target [, nbr] | Dies richtet einen periodischen Interrupt (oder „Tick“) ein.<br>Es stehen vier Tick-Timer zur Verfügung („nbr“ ist 1, 2, 3 oder 4). „nbr“ ist optional; wenn nicht angegeben, wird Timer Nummer 1 verwendet.<br>Die Zeit zwischen den Interrupts beträgt „period“ Millisekunden und „target“ ist die Interrupt-Subroutine, die aufgerufen wird, wenn das zeitgesteuerte Ereignis eintritt.<br>Die Periode kann zwischen 1 und 2147483647 ms (etwa 24 Tage) liegen.<br>Diese Interrupts können deaktiviert werden, indem „period“ auf Null gesetzt wird<br>(d. h. SETTICK 0, 0, 3 deaktiviert den Tick-Timer Nummer 3). |
| SETTICK PAUSE, target [, nbr]<br>or<br>SETTICK RESUME, target [, nbr] | Den angegebenen Timer anhalten oder fortsetzen. Im angehaltenen Zustand wird der Interrupt verzögert, der aktuelle Zählstand bleibt jedoch erhalten. |
| SORT array() [,indexarray()] [,flags] [,startposition] [,elementstosort] | Dieser Befehl nimmt ein Array beliebigen Typs (Integer, Float oder String) entgegen und sortiert es an Ort und Stelle in aufsteigender Reihenfolge.<br>Er verfügt über einen optionalen Parameter „indexarray()“. Wenn dieser verwendet wird, muss es sich um ein Integer-Array handeln, das dieselbe Größe wie das zu sortierende Array hat. 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 ermöglicht das Sortieren von verbundenen Arrays.<br>Der Parameter „flag“ ist optional und gültige Flag-Werte sind: |

| | |
|---|---|
| | bit0: 0 (Standard, wenn weggelassen) normale Sortierung – 1 umgekehrte Sortierung<br>bit1: 0 (Standard) groß-/kleinschreibungsabhängig – 1 Sortierung ist groß-/kleinschreibungsunabhängig (nur Zeichenketten).<br>bit2: 0 (Standard) normale Sortierung – 1 leere Zeichenfolgen werden an das Ende des Arrays verschoben<br>Die optionale „startposition“ legt fest, bei welchem Element im Array die Sortierung beginnen soll. Standard ist 0 (OPTION BASE 0) oder 1 (OPTION BASE 1)<br>Der optionale Parameter „elementstosort“ legt fest, wie viele Elemente im Array sortiert werden sollen. Standardmäßig sind dies alle Elemente nach der „startposition“.<br>Jeder der optionalen Parameter kann weggelassen werden. Um beispielsweise nur die ersten 50 Elemente eines Arrays zu sortieren, könntest du Folgendes verwenden:<br>`SORT array(), , , ,50`<br>Beispiel:<br>Das Array city$() könnte die Namen von Städten weltweit enthalten und lässt sich mit dem Befehl ganz einfach in aufsteigender alphabetischer Reihenfolge sortieren: SORT city$()<br>Der Befehl SORT funktioniert mit Zeichenketten, Gleitkommazahlen und Ganzzahlen, das zu sortierende Array muss jedoch eindimensional sein.<br>Häufig werden Daten in mehreren Arrays gespeichert, zum Beispiel könnte der Name jeder Stadt im Array city$() gespeichert sein, die Einwohnerzahl im Array pop%() und die Größe der Stadt im Array area!(). Der gleiche Index würde sich auf den Namen, die Einwohnerzahl und die Fläche der Stadt beziehen.<br>Das Sortieren und Abrufen dieser Daten ist etwas komplexer, lässt sich aber relativ einfach mit einem optionalen Parameter für den Sortierbefehl wie folgt bewerkstelligen:<br>SORT array(), indexarray%()<br>indexarray%() muss ein eindimensionales Ganzzahl-Array sein, das dieselbe Größe hat wie das zu sortierende Array. Nach dem Sortieren enthält indexarray%() den entsprechenden Index zu den ursprünglichen Daten vor dem Sortieren. (Alles, was zuvor in indexarray%() stand, wird überschrieben).<br>Um auf die sortierten Daten zuzugreifen, kopierst du zunächst das Array, das den Hauptschlüssel enthält, in ein temporäres Array und sortierst dieses unter Angabe von indexarray%(). Nach dem Sortieren kann indexarray%() verwendet werden, um die ursprünglichen Arrays zu indizieren.<br>Zum Beispiel:<br>`DIM city$(100),pop%(100),area!(100),sindex%(100),t$(100)`<br>`FOR i = 0 to 100`<br>`  t$(i) = city$(i)  ' temporäre Kopie der Schlüssel`<br>`NEXT i`<br>`SORT t$(), sindex%()  ' das temporäre Array sortieren,`<br>`FOR i = 0 bis 100`<br>`  k = sindex%(i)  ' Index zum ursprünglichen Array`<br>`  PRINT city$(k),pop%(k),area!(k)' in sortierter Reihenfolge ausgeben`<br>`NEXT i` |
| SPI OPEN speed, mode, bits<br>or<br>SPI READ nbr, array() | Kommunikation über einen SPI-Kanal. Details findest du in *Anhang D*.<br>'nbr' ist die Anzahl der zu sendenden oder zu empfangenden Datenelemente<br>'data1', 'data2' usw. können Float- oder Integer-Werte sein und im Fall von WRITE eine Konstante oder ein Ausdruck. |

| | |
|---|---|
| or<br>SPI WRITE nbr, data1, data2, data3, … usw.<br>or<br>SPI WRITE nbr, string$<br>or<br>SPI WRITE nbr, array()<br>or<br>SPI CLOSE | Wenn 'string$' verwendet wird, werden 'nbr' Zeichen gesendet.<br>'array' muss ein eindimensionales Float- oder Integer-Array sein, und es werden 'nbr' Elemente gesendet oder empfangen. |
| SPI2 | Der gleiche Befehlssatz wie für SPI (oben), gilt jedoch für den zweiten SPI-Kanal. |
| SPRITE | Siehe Anhang G und das separate PDF-Dokument „SPRITE_User_Manual“ |
| STAR | Berechnet die Position eines Himmelsobjekts anhand der aktuellen Uhrzeit und des Standorts, die von einem angeschlossenen GPS-Modul geliefert werden.<br>Dies ist einer aus einer Reihe von Befehlen, die hochpräzise astronomische Berechnungen durchführen, die sich für die Ausrichtung von Teleskopen oder zur Navigation eignen, und wird ausführlich in der Datei *„GPS_Astro_Reference.pdf“* beschrieben*, die* in der ZIP-Datei mit dem Firmware-Download enthalten ist. |
| STATIC variable [, variables]<br>Siehe DIM für die vollständige Syntax. | Definiert eine Liste von Variablennamen, die lokal für die Subroutine oder Funktion gelten. Diese Variablen behalten ihren Wert zwischen Aufrufen der Subroutine oder Funktion bei (im Gegensatz zu Variablen, die mit dem Befehl LOCAL erstellt wurden).<br>Dieser Befehl verwendet genau dieselbe Syntax wie DIM. Der einzige Unterschied besteht darin, dass die Länge des von STATIC erstellten Variablennamens und die Länge des Unterprogramm- oder Funktionsnamens zusammen nicht mehr als 31 Zeichen betragen dürfen.<br>Statische Variablen können mit einem Wert initialisiert werden. Diese Initialisierung wirkt sich nur beim ersten Aufruf der Subroutine aus (nicht bei nachfolgenden Aufrufen). |
| STEPPER | Dieser Befehl bietet ein umfassendes System zur Steuerung von bis zu 3 Schrittmotorachsen (X, Y, Z) mit Unterstützung für G-Code-Ausführung, Beschleunigungsplanung und Hardware-Endschalter.<br>Dies wird ausführlich in der Datei *Stepper_Reference.pdf* beschrieben, die im ZIP-Download der Firmware enthalten ist. |
| STRUCT | Dieser Befehl ermöglicht die Bearbeitung von Strukturen (auch als benutzerdefinierte Typen bekannt), mit denen verwandte Variablen unterschiedlicher Typen unter einem einzigen Namen zusammengefasst werden können.<br>Dies wird ausführlich in der Datei *„MMBasic_Structures_Manual.pdf“* beschrieben, die im ZIP-Archiv des Firmware-Downloads enthalten ist. |
| SUB xxx (arg1 [,arg2, …])<br><statements><br><statements><br>END SUB | Definiert eine aufrufbare Subroutine. Das entspricht dem Hinzufügen eines neuen Befehls zu MMBasic, während dein Programm läuft.<br>„xxx“ ist der Name der Subroutine und muss den Namenskonventionen für Variablen entsprechen.<br>'arg1', 'arg2' usw. sind die Argumente oder Parameter der Subroutine. Ein Array wird durch leere Klammern angegeben, z. B. arg3(). Der Typ des Arguments kann durch ein Typ-Suffix (z. B. arg1$) oder durch die Angabe des Typs mit `AS <Typ> (z. B. arg1 AS STRING)` festgelegt werden. |

| | |
|---|---|
| | Argumente in der Liste des Aufrufers, die eine Variable sind und den richtigen Typ haben, werden per Referenz an die Subroutine übergeben. Das bedeutet, dass alle Änderungen am entsprechenden Argument in der Subroutine auch in die Variable des Aufrufers kopiert werden und daher nach Beendigung der Subroutine abgerufen werden können. Dem Argument kann das Präfix BYVAL vorangestellt werden, wodurch dieser Mechanismus verhindert wird und nur der Wert verwendet wird. Alternativ weist das Präfix BYREF MMBasic an, dass eine Referenz erforderlich ist, und es wird ein Fehler ausgegeben, wenn dies nicht möglich ist.<br>Arrays werden durch Angabe des Array-Namens in leeren Klammern (z. B. arg()) übergeben; sie werden immer per Referenz übergeben und müssen den richtigen Typ haben.<br>Jede Definition muss eine END SUB-Anweisung enthalten. Wenn diese erreicht wird, kehrt das Programm zur nächsten Anweisung nach dem Aufruf der Subroutine zurück. Der Befehl EXIT SUB kann für einen vorzeitigen Abbruch verwendet werden.<br>Du verwendest die Subroutine, indem du ihren Namen und ihre Argumente in einem Programm verwendest, genau wie bei einem normalen Befehl. Zum Beispiel: `MySub a1, a2`<br>Wenn die Subroutine aufgerufen wird, wird jedes Argument im Aufrufer dem entsprechenden Argument in der Subroutinendefinition zugeordnet. Diese Argumente sind nur innerhalb der Subroutine verfügbar. Subroutinen können mit einer variablen Anzahl von Argumenten aufgerufen werden. Alle in der Argumentliste der Subroutine ausgelassenen Argumente werden auf Null oder eine leere Zeichenkette gesetzt.<br>Klammern um die Argumentliste sind sowohl im Aufrufer als auch in der Definition optional. |
| SYNC time% [,period]<br>or<br>SYNC | Mit dem SYNC-Befehl kannst du sehr präzise getaktete, sich wiederholende Aktionen ausführen (Genauigkeit von 1–2 Mikrosekunden).<br>Um dies zu aktivieren, wird der Befehl zunächst mit dem Parameter time% aufgerufen. Dadurch wird ein Wiederholungstimer für time% Mikrosekunden eingerichtet. Der optionale Parameter „period“ modifiziert die Zeit und kann „U“ für Mikrosekunden, „M“ für Millisekunden oder „S“ für Sekunden lauten.<br>Sobald der Takt eingerichtet ist, wird das Programm mit dem Befehl SYNC ohne Parameter darauf synchronisiert. Dieser wartet, bis die Taktperiode abgelaufen ist. Bei Perioden unter 2 ms ist dies nicht unterbrechbar. Bei Perioden über 2 ms reagiert das Programm auf Strg-C, jedoch nicht auf MMBasic-Interrupts.<br>Typischerweise wird die Uhr außerhalb einer Schleife eingestellt und dann am Anfang der Schleife der Befehl SYNC ohne Parameter aufgerufen. Das bedeutet, dass der Inhalt der Schleife genau einmal pro eingestellter Taktperiode ausgeführt wird. Zum Beispiel würde Folgendes einen Servo mit der erforderlichen präzisen 50-Hz-Taktung ansteuern:<br>`SYNC 20, M`<br>`DO`<br>`  SYNC`<br>`  PULSE GP0,n`<br>`LOOP` |
| TEMPR START pin [, precision] [,timeout] | Mit diesem Befehl kannst du eine Messung an einem DS18B20-Temperatursensor starten, der an „pin“ angeschlossen ist. Normalerweise reicht die Funktion TEMPR() allein für eine Temperaturmessung aus, daher ist die Verwendung dieses Befehls optional. Weitere Details findest du im Abschnitt *„Temperaturmessung*“.<br>Dieser Befehl startet die Messung am Temperatursensor. Das Programm kann sich dann anderen Aufgaben widmen, während die Messung läuft, und später |

| | |
|---|---|
| | die Funktion TEMPR() verwenden, um den Messwert abzurufen. Wenn die Funktion TEMPR() aufgerufen wird, bevor die Umwandlungszeit abgelaufen ist, wartet die Funktion die verbleibende Umwandlungszeit ab, bevor sie den Wert zurückgibt.<br>Es können beliebig viele dieser Umwandlungen (an verschiedenen Pins) gestartet werden und gleichzeitig laufen.<br>„precision“ ist die Auflösung der Messung und optional. Es handelt sich um eine Zahl zwischen 0 und 3 mit folgender Bedeutung:<br>0 = 0,5 °C Auflösung, 100 ms Umwandlungszeit.<br>1 = 0,25 °C Auflösung, 200 ms Umwandlungszeit (dies ist die Standardeinstellung).<br>2 = 0,125 °C Auflösung, 400 ms Umwandlungszeit.<br>3 = 0,0625 °C Auflösung, 800 ms Umwandlungszeit.<br>Der optionale Timeout-Parameter überschreibt die oben genannten Umwandlungszeiten, um langsame Geräte zu berücksichtigen. |
| TEXT x, y, string$ [,alignment$] [, font] [, scale] [, c] [, bc] | Zeigt eine Zeichenkette auf dem Videoausgang oder dem angeschlossenen LCD-Bildschirm an, beginnend bei 'x' und 'y'.<br>‘string$’ ist die anzuzeigende Zeichenkette. Numerische Daten sollten in eine Zeichenkette konvertiert und mit der Funktion Str$() formatiert werden.<br>'alignment$' ist ein Zeichenfolgenausdruck oder eine Zeichenfolgenvariable, die aus 0, 1 oder 2 Buchstaben besteht, wobei der erste Buchstabe die horizontale Ausrichtung um 'x' angibt und L, C oder R für LEFT, CENTER, RIGHT sein kann, und der zweite Buchstabe die vertikale Ausrichtung um 'y' angibt und T, M oder B für TOP, MIDDLE, BOTTOM sein kann. Die Standardausrichtung ist links/oben.<br>Beispiel: „CM“ zentriert den Text vertikal und horizontal.<br>Die Zeichenfolge „alignment$“ kann eine Konstante (z. B. „CM“) oder eine Zeichenfolgenvariable sein. Zur Abwärtskompatibilität mit früheren Versionen von MMBasic kann die Zeichenfolge auch ohne Anführungszeichen stehen (z. B. CM).<br>Ein dritter Buchstabe kann in der Ausrichtungszeichenfolge verwendet werden, um die Drehung des Textes anzugeben. Dies kann „N“ für normale Ausrichtung sein, „V“ für vertikalen Text, bei dem jedes Zeichen unter dem vorherigen steht und von oben nach unten verläuft, „I“ für invertierten Text (d. h. auf dem Kopf stehend), „U“ für eine Drehung des Textes um 90° gegen den Uhrzeigersinn und „D“ für eine Drehung des Textes um 90° im Uhrzeigersinn<br>„font“ und „scale“ sind optional und entsprechen standardmäßig den Einstellungen des FONT-Befehls.<br>„c“ ist die Zeichenfarbe und „bc“ die Hintergrundfarbe. Sie sind optional und entsprechen standardmäßig den aktuellen Vordergrund- und Hintergrundfarben.<br>Eine Definition der Farben und Grafikkoordinaten findest du im Kapitel *„Grafikbefehle und -funktionen*“. |

| | |
|---|---|
| TILE x, y [,foreground] [,background] [,nbr_tiles_wide] [,nbr_tiles_high]<br><br><br><br><br><br><br><br><br><br><br><br>TILE HEIGHT n | <u>NUR MODUS 1 DER VGA- ODER HDMI-VERSIONEN</u><br>Legt die Farbe für eine oder mehrere Kacheln auf dem Bildschirm fest.<br>Im standardmäßigen Monochrom-Modus ist der Bildschirm in 80x40 Kacheln mit jeweils 8x12 Pixeln unterteilt. Dies entspricht Schriftart 1 und ermöglicht eine vollständige Farbcodierung im Editor im Monochrom-Modus.<br>Jeder Kachel kann eine andere Vordergrund- und Hintergrundfarbe aus den folgenden Optionen zugewiesen werden: Weiß, Gelb, Lila, Braun, Fuchsia, Rostrot, Magenta, Rot, Cyan, Grün, Cerulean, Mittelgrün, Kobalt, Myrte, Blau und Schwarz.<br>„x“ und „y“ sind die Koordinaten des Startblocks (0–79, 0–39)<br>„foreground“ und „background“ sind die neu ausgewählten Farben. 'nbr_tiles_wide' und 'nbr_tiles_high' sind die Anzahl der zu ändernden Kacheln.<br>Die Änderung erfolgt sofort und hat keinen Einfluss auf den Text oder die Grafiken, die derzeit in den Kacheln angezeigt werden (nur auf die Farben).<br>Legt die Höhe der Kacheln fest. „n“ kann zwischen 12 und 480 (RP2040) oder zwischen 8 und 480 (RP2350) liegen |
| TIME$ = "HH:MM:SS"<br>or<br>TIME$ = "HH:MM"<br>or<br>TIME$ = "HH" | Stellt die Zeit der internen Uhr ein. MM und SS sind optional und werden standardmäßig auf Null gesetzt, wenn sie nicht angegeben werden. Zum Beispiel stellt TIME$ = "14:30" die Uhr auf 14:30 Uhr mit null Sekunden ein.<br>Mit OPTION RTC AUTO ENABLE startet die PicoMite-Firmware mit dem in der RTC programmierten TIME$. Ohne OPTION RTC AUTO ENABLE startet die Firmware mit TIME$="00:00:00" |
| TILEMAP | TILEMAP ist ein leistungsstarker Befehl zum Schreiben von Spielen. Weitere Details findest du im separaten Dokument TILEMAP_User_Manual.pdf |
| TIMER = msec | Setzt den Timer auf eine bestimmte Anzahl von Millisekunden zurück. Normalerweise wird dies nur verwendet, um den Timer auf Null zurückzusetzen, aber du kannst ihn auf jede beliebige positive Zahl einstellen.<br>Weitere Details findest du bei der TIMER-Funktion. |
| TRACE ON<br>or<br>TRACE OFF<br>or<br>TRACE LIST nn | TRACE ON/OFF schaltet die Trace-Funktion ein bzw. aus. Diese Funktion gibt während der Programmausführung die Nummer jeder Zeile (gezählt ab dem Beginn des Programms) in eckigen Klammern aus. Das ist beim Debuggen von Programmen nützlich.<br>TRACE LIST listet die letzten „nn“ ausgeführten Zeilen im oben beschriebenen Format auf. MMBasic protokolliert immer die ausgeführten Zeilen, daher ist diese Funktion immer verfügbar (d. h., sie muss nicht extra aktiviert werden). |
| TRIANGLE X1, Y1, X2, Y2, X3, Y3 [, C [, FILL]] | Zeichnet ein Dreieck auf dem angeschlossenen Videoausgang oder LCD-Display mit den Eckpunkten bei X1, Y1 und X2, Y2 sowie X3, Y3. 'C' ist die Farbe des Dreiecks und ist standardmäßig die aktuelle Vordergrundfarbe. 'FILL' ist die Füllfarbe und ist standardmäßig auf keine Füllung gesetzt (sie kann auch auf -1 gesetzt werden, um keine Füllung zu verwenden).<br>Alle Parameter können als Arrays angegeben werden, und die Software zeichnet die Anzahl der Dreiecke, die durch die Dimensionen des kleinsten Arrays bestimmt wird, es sei denn, X1 = Y1 = X2 = Y2 = X3 = Y3 = -1; in diesem Fall wird die Verarbeitung an dieser Stelle beendet. 'x1', 'y1', 'x2', 'y2', 'x3' und 'y3' müssen alle Arrays oder alle einzelne Variablen/Konstanten sein, andernfalls wird ein Fehler ausgegeben. 'c' und 'fill' können entweder Arrays oder einzelne Variablen/Konstanten sein. |

| | |
|---|---|
| TRIANGLE SAVE [#]n, x1,y1,x2,y2,x3,y3<br><br>TRIANGLE RESTORE [#]n | Speichert einen dreieckigen Bereich des Bildschirms im Puffer #n.<br><br>Stellt einen gespeicherten dreieckigen Bereich des Bildschirms wieder her und löscht den gespeicherten Puffer. |
| TURTLE | Die Firmware implementiert eine vollständige Turtle-Grafik-Engine. Siehe Anhang H |
| TYPE struct-name<br>struct-mber1, struct-mber2<br>…<br>END TYPE | Definiere eine Struktur (auch als benutzerdefinierter Typ bekannt), mit der verwandte Variablen unterschiedlicher Typen unter einem einzigen Namen zusammengefasst werden können.<br>Dies wird ausführlich in der Datei *MMBasic_Structures_Manual.pdf* beschrieben, die im ZIP-Archiv des Firmware-Downloads enthalten ist. |
| UPDATE FIRMWARE | NICHT BEI USB-VERSIONEN<br>Versetzt die PicoMite-Firmware in den Firmware-Update-Modus (entspricht dem Einschalten bei gedrückter BOOTSEL-Taste). Dieser Befehl ist nur an der Befehlszeile verfügbar. |
| VAR SAVE var [, var]…<br>or<br>VAR RESTORE<br>or<br>VAR CLEAR | VAR SAVE speichert eine oder mehrere Variablen in einem nichtflüchtigen Flash-Speicher, von wo sie später wiederhergestellt werden können (normalerweise nach einem Stromausfall).<br>„var“ kann eine beliebige Anzahl von numerischen oder Zeichenfolgenvariablen und/oder Arrays sein. Arrays werden durch leere Klammern angegeben. Zum Beispiel: var()<br>Der Befehl VAR SAVE kann wiederholt verwendet werden. Zuvor gespeicherte Variablen werden mit ihrem neuen Wert aktualisiert, und alle neuen Variablen (die zuvor nicht gespeichert wurden) werden zur Liste der gespeicherten Variablen hinzugefügt, um später wiederhergestellt zu werden.<br>VAR RESTORE ruft die zuvor gespeicherten Variablen ab und fügt sie (und ihre Werte) in die Variablentabelle ein.<br>VAR CLEAR löscht alle gespeicherten Variablen.<br>Dieser Befehl wird normalerweise verwendet, um Kalibrierungsdaten, Optionen und andere Daten zu speichern, die sich nicht oft ändern, aber über einen Stromausfall hinweg erhalten bleiben müssen. Normalerweise wird der Befehl VAR RESTORE an den Anfang des Programms gesetzt, damit zuvor gespeicherte Variablen wiederhergestellt werden und dem Programm beim Start sofort zur Verfügung stehen. Hinweise:<br>• Der für diesen Befehl verfügbare Speicherplatz beträgt 16 KB.<br>• Die Verwendung von VAR RESTORE ohne vorherige Speicherung hat keine Wirkung und löst keinen Fehler aus.<br>• Wenn bei der Verwendung von RESTORE bereits eine Variable mit demselben Namen existiert, wird deren Wert überschrieben.<br>• Gespeicherte Arrays müssen (mit DIM) deklariert werden, bevor sie wiederhergestellt werden können.<br>• Beachte, dass Zeichenfolgen-Arrays den diesem Befehl zugewiesenen Speicherplatz schnell aufbrauchen können. Der Qualifizierer LENGTH kann bei der Deklaration eines Zeichenfolgen-Arrays verwendet werden, um die Größe des Arrays zu reduzieren (siehe den Befehl DIM). Bei gewöhnlichen Zeichenfolgenvariablen ist dies nicht erforderlich.<br>• Die gespeicherten Variablen werden durch ein Firmware-Upgrade, durch den Befehl NEW oder beim Laden eines neuen Programms über AUTOSAVE, XMODEM usw. automatisch gelöscht. |

| | |
|---|---|
| WATCHDOG timeout<br>or<br>WATCHDOG OFF<br>or<br>WATCHDOG HW timeout<br>or<br>WATCHDOG HW OFF | Startet den Watchdog-Timer, der die Prozessoren automatisch neu startet, wenn die Zeit abgelaufen ist. Dies kann genutzt werden, um sich von einem Ereignis zu erholen, das das laufende Programm deaktiviert hat (wie z. B. eine Endlosschleife oder ein Programmier- oder anderer Fehler, der ein laufendes Programm zum Stillstand bringt). Dies kann in einer unbeaufsichtigten Steuerungssituation wichtig sein.<br>Das Ablaufen der Zeit kann entweder im System-Timer-Interrupt (WATCHDOG-Befehl) oder als echter CPU-/Hardware-Watchdog (WATCHDOG HW-Befehl) verarbeitet werden.<br>Wenn der Hardware-Watchdog verwendet wird, beträgt die maximale Zeit des Timers 8,3 Sekunden. Für den Software-Watchdog gibt es keine solche Begrenzung.<br>„timeout“ ist die Zeit in Millisekunden (ms), bevor ein Neustart erzwungen wird. Dieser Befehl sollte an strategischen Stellen im laufenden BASIC-Programm platziert werden, um den Watchdog-Timer ständig zurückzusetzen (auf „timeout“) und so zu verhindern, dass er bis auf Null herunterzählt.<br>Wenn der Timer-Zähler tatsächlich Null erreicht (vielleicht weil das BASIC-Programm nicht mehr läuft), wird die PicoMite-Firmware automatisch neu gestartet und die automatische Variable MM.WATCHDOG auf „true“ (d. h. 1) gesetzt, was anzeigt, dass ein Fehler aufgetreten ist. Bei einem normalen Start wird MM.WATCHDOG auf „false“ (d. h. 0) gesetzt. Beachte, dass OPTION AUTORUN angegeben werden muss, damit das Programm neu gestartet wird.<br>Mit WATCHDOG OFF kannst du den Watchdog-Timer deaktivieren (dies ist die Standardeinstellung bei einem Reset oder beim Einschalten). Der Timer wird auch ausgeschaltet, wenn das Break-Zeichen (STRG-C) auf der Konsole verwendet wird, um ein laufendes Programm zu unterbrechen. |
| WII [CLASSIC] OPEN [,interrupt] | Öffnet einen WII Classic-Controller und führt eine Hintergrundabfrage des Geräts durch. Der Wii Classic muss an die durch OPTION SYSTEM I2C angegebenen Pins angeschlossen sein – das ist eine Voraussetzung.<br>Open versucht, mit dem Wii Classic zu kommunizieren, und gibt einen Fehler zurück, wenn es nicht gefunden wird. Wird es gefunden, abtastet die Firmware die Wii-Daten im Hintergrund mit einer Rate von 50 Hz. Wenn ein optionaler Benutzer-Interrupt angegeben ist, wird dieser ausgelöst, sobald sich einer der Tastenstatus ändert (sowohl Ein- als auch Ausschalten)<br>Siehe die DEVICE-Funktion, um zu erfahren, wie du Daten vom Wii Classic ausliest. |
| WII [CLASSIC] CLOSE | CLOSE beendet die Hintergrundabfrage und deaktiviert alle angegebenen Interrupts |
| WII NUNCHUCK OPEN [,interrupt] | Öffnet einen Wii Nunchuck-Controller und implementiert die Hintergrundabfrage des Geräts. Der Wii Nunchuck muss an die von OPTION SYSTEM I2C angegebenen Pins angeschlossen sein – das ist eine Voraussetzung.<br>Open versucht, mit dem Wii Nunchuck zu kommunizieren, und gibt einen Fehler zurück, wenn es nicht gefunden wird. Wird es gefunden, erfasst die Firmware die Wii-Daten im Hintergrund mit einer Rate von 50 Hz. Wenn ein optionaler Benutzer-Interrupt angegeben ist, wird dieser ausgelöst, sobald sich einer der Tastenzustände ändert (sowohl Ein- als auch Ausschalten)<br>Siehe die Funktion DEVICE, um zu erfahren, wie du Daten vom Wii Nunchuck ausliest. |
| WII NUNCHUCK CLOSE | CLOSE beendet die Hintergrundabfrage und deaktiviert alle angegebenen Interrupts |

| | |
|---|---|
| | NUR WEBMITE |
| WEB | Die WEB-Befehle dienen zur Verwaltung der Internetfunktionen des WebMite. |
| WEB CONNECT [ssid$, passwd$, [name$] [,ipaddress$, mask$, gateway$]] | Dieser Befehl stellt ohne optionale Parameter, sofern möglich, eine Verbindung zum Standardnetzwerk her (wie zuvor mit OPTION WIFI festgelegt) oder verbindet sich mit den optionalen Parametern mit dem angegebenen Netzwerk und richtet zudem die OPTION WIFI für die zukünftige Nutzung ein. |
| WEB MQTT CONNECT addr$, port, user$, passwd$ [, interrupt] | Verbinde dich mit einem MQTT-Broker.<br>'addr$' ist die IP-Adresse, 'port' ist die zu verwendende Portnummer, 'user$' ist der Benutzername, 'passwd$' ist das Passwort des Kontos und 'interrupt' ist optional; wenn angegeben, ist dies die Subroutine, die bei Empfang einer Nachricht aufgerufen wird.<br>WEB CONNECT trennt die Verbindung zu einem zuvor verbundenen Netzwerk nicht, sollte also nur verwendet werden, wenn zuvor noch nichts eingerichtet wurde oder wenn ein zuvor konfiguriertes Netzwerk nicht aktiv ist oder die Verbindung zu einem zuvor konfigurierten Netzwerk beim Booten fehlgeschlagen ist (keine Parameter) |
| WEB MQTT PUBLISH topic$, msg$, [,qos] [,retain] | Veröffentliche Inhalte in einem MQTT-Broker-Topic.<br>'topic$' ist der Topic-Name und 'msg$' ist die Nachricht/<br>'qos' ist die optionale Dienstgüte mit den Werten 0, 1 oder 2 (Standard ist 1). |
| WEB MQTT SUBSCRIBE topic$ [,qos] | Ein MQTT-Broker-Thema abonnieren.<br>„topic$“ ist der Name des Themas und „qos“ ist die optionale Dienstgüte mit den Werten 0, 1 oder 2 (Standard ist 1). |
| WEB MQTT UNSUBSCRIBE topic$ | Ein MQTT-Broker-Thema abbestellen.<br>'topic$' ist der Name des Themas. |
| WEB MQTT CLOSE | Schließe eine persistente MQTT-Verbindung. |
| WEB NTP [timeoffset [, NTPserver$]] [,timeout]]] | Rufe Datum und Uhrzeit von einem NTP-Server ab und stelle die interne WebMite-Uhr ein.<br>'timeoffset' ist die lokale Zeitzone; wird sie weggelassen, werden Datum und Uhrzeit auf GMT gesetzt. 'NTPserver$' ist der zu verwendende Zeitserver; wird er weggelassen, wird standardmäßig ein internationaler Zeitserver-Pool verwendet. 'timeout' ist die optionale Zeitüberschreitung in Millisekunden und beträgt standardmäßig 5000. |
| WEB OPEN TCP CLIENT address$, port | Öffnet eine TCP-Client-Verbindung zu einem Webserver.<br>'address$' ist eine Zeichenkette und gibt die Adresse des Servers an, mit dem eine Verbindung hergestellt werden soll. Dies kann entweder eine URL (z. B. „api.openweathermap.org“) oder eine IP-Adresse (z. B. „192.168.1.111“) sein. 'port' ist die Nummer des zu verwendenden Ports.<br>Wird zusammen mit WEB TCP CLIENT REQUEST verwendet, um den Server abzufragen.<br>Beachte, dass nur eine CLIENT-Verbindung zulässig ist. |
| WEB OPEN TCP STREAM address$, port | Öffnet eine TCP-Client-Verbindung zu einem WEB-Server wie bei WEB OPEN TCP CLIENT, verbindet jedoch die Empfängerlogik von WEB TCP CLIENT STREAM anstelle der Logik für WEB TCP CLIENT REQUEST.<br>'address$' ist eine Zeichenkette und gibt die Adresse des Servers an, mit dem eine Verbindung hergestellt werden soll. Dies kann entweder eine URL (z. B. „api.openweathermap.org“) oder eine IP-Adresse (z. B. „192.168.1.111“) sein. 'port' ist die Nummer des zu verwendenden Ports. |

| | |
|---|---|
| | Beachte, dass nur eine CLIENT-Verbindung zulässig ist. |
| WEB SCAN [array%()] | Sucht nach allen verfügbaren WLAN-Verbindungen. Wenn „array%()“ angegeben wird, wird die Ausgabe in einem Longstring gespeichert, andernfalls erfolgt die Ausgabe auf der Konsole. Der Befehl kann verwendet werden, unabhängig davon, ob bereits eine Netzwerkverbindung aktiv ist oder nicht. |
| WEB TCP CLIENT REQUEST request$, buff%() [,timeout] | Sendet eine Anfrage an den mit WEB OPEN TCP CLIENT geöffneten Remote-Server und wartet auf eine Antwort.<br>'request$' ist eine Zeichenkette und stellt die an den Server zu sendende Anfrage dar. 'buff%()' ist ein Integer-Array, das die Antwort als LONGSTRING empfängt. Die Größe dieses Puffers begrenzt die vom Server empfangene Datenmenge.<br>'timeout' ist die optionale Zeitüberschreitung in Millisekunden und beträgt standardmäßig 5000.<br>Wenn die Anfrage abläuft, tritt ein Fehler auf, andernfalls werden die empfangenen Daten im LONGSTRING 'buff%()' gespeichert. Wenn die empfangenen Daten eine JSON-Zeichenkette sind, kann die Funktion JSON$() verwendet werden, um sie zu parsen. |
| WEB TCP CLIENT STREAM command$, buffer%(), readpointer%, writepointer% | Stellt eine Verbindung zu einem Server her, der zuvor mit WEB OPEN TCP STREAM geöffnet wurde.<br>'command$' ist eine Zeichenkette und stellt die Anfrage dar, die an den Server gesendet werden soll.<br>'buffer%()' ist ein Integer-Array, das die laufenden Antworten empfängt und als Ringpuffer für die empfangenen Bytes fungiert.<br>Die Firmware verwaltet den Parameter „writepointer%“, sobald die Daten vom Server eintreffen.<br>„readpointer%“ sollte vom Basic-Programm verwaltet werden, da es Daten aus dem Ringpuffer entfernt.<br>Wenn „writepointer%“ zu „readpointer%“ aufholt, wird „readpointer%“ erhöht, um immer ein Byte voraus zu sein, und eingehende Daten gehen verloren.<br>Dieser Befehl ist so konzipiert, dass er mit dem Befehl PLAY STREAM kompatibel ist, um die Implementierung von Streaming-Internet-Audio zu ermöglichen. |
| WEB CLOSE TCP CLIENT | Schließt die mit WEB OPEN TCP CLIENT geöffnete Verbindung zum Remote-Server. Dies muss geschehen, bevor ein weiterer Verbindungsaufbau versucht wird. |
| WEB TCP INTERRUPT InterruptSub | Starte den TCP-Server. „InterruptSub“ ist die Subroutine, die aufgerufen wird, wenn eine Anfrage an den TCP-Server gestellt wird (d. h. ein Interrupt).<br>Beachte, dass zuerst der Befehl OPTION WIFI und anschließend der Befehl OPTION TCP SERVER PORT verwendet werden muss, um den TCP-Server zu aktivieren. |
| WEB TCP READ cb%, buff%() | Lies die Daten aus einer potenziellen TCP-Verbindung „cb%“ ein. „buff%()“ ist ein Array, um alle Daten aus dieser Verbindung als Longstring zu empfangen. Die Größe dieses Puffers begrenzt die Datenmenge, die vom Remote-Client empfangen wird. Wenn über diese Verbindung nichts empfangen wird, gibt dies eine leere Zeichenkette zurück (d. h. `LLEN(buff%())=0`).<br>Wenn Daten empfangen wurden, muss das BASIC-Programm mit einem der WEB TRANSMIT-Befehle antworten, um die Verbindung zu schließen. |

| | |
|---|---|
| WEB TCP SEND cb%, data%()<br>WEB TCP CLOSE cb% | Diese beiden Befehle ermöglichen eine größere Flexibilität bei der Nutzung des TCP-Servers. Im Gegensatz zu WEB TRANSMIT PAGE oder WEB TRANSMIT FILE erstellt WEB TCP SEND keinerlei Header und schließt die TCP-Verbindung nach der Übertragung auch nicht. Es sendet einfach genau das, was sich in der LONGSTRING-Variablen data%() befindet, und es liegt am BASIC-Programmierer, die Verbindung zum geeigneten Zeitpunkt zu schließen. |
| WEB TRANSMIT CODE cb%, nnn% | Sendet eine numerische Antwort an die offene TCP-Verbindung 'cb%' und schließt anschließend die Verbindung.<br>Typischerweise würde man `TRANSMIT CODE cb%, 404` verwenden, um anzuzeigen, dass die Seite nicht gefunden wurde. |
| WEB TRANSMIT FILE cb%, filename$, content-type$ | Erstellt einen HTTP 1.1-Header mit dem angegebenen 'content-type$', sendet ihn und überträgt anschließend den Inhalt der Datei an die offene TCP-Verbindung cb%; nach Abschluss wird die Verbindung geschlossen.<br>„content-type$" ist ein MIME-Typ, der als Zeichenkette angegeben wird. Z. B. „image/jpeg" |
| WEB TRANSMIT PAGE cb%, filename$ [,buffersize] | Erstellt einen HTTP 1.1-Header, sendet ihn und überträgt anschließend den Inhalt der Datei an die offene TCP-Verbindung cb%. Nach Abschluss wird die Verbindung geschlossen.<br>MMBasic ersetzt alle in der Datei definierten MMBasic-Variablen oder -Ausdrücke innerhalb von geschweiften Klammern, z. B. {myvar%}, durch aktuelle Werte. Variablen können einfache Werte, Array-Elemente oder Ausdrücke sein.<br>Eine öffnende geschweifte Klammer kann mithilfe von {{ in die Ausgabe eingefügt werden.<br>Standardmäßig reserviert der Befehl einen Puffer in der Größe der Datei + 4096 Byte, um die zu übertragende Seite zu erstellen. Wenn die Seite jedoch komplex ist und viele MMBasic-Variablen enthält, deren Text länger ist als der Variablenname, kann es sein, dass der Puffer nicht groß genug ist. In diesem Fall kannst du den benötigten zusätzlichen Speicherplatz angeben (Standardwert ist 4096, wenn nicht anders angegeben) |
| WEB UDP INTERRUPT intname | Richtet eine BASIC-Interrupt-Routine ein, die immer dann ausgelöst wird, wenn ein UDP-Datagramm empfangen wird. Der Inhalt wird in MM.MESSAGE$ gespeichert. Die IP-Adresse des Absenders wird in MM.ADDRESS$ gespeichert. |
| WEB UDP SEND addr$, port, data$ | Wird verwendet, um ein Datagramm an einen entfernten Empfänger zu senden. In diesem Fall muss die IP-Adresse angegeben werden; dies kann entweder eine numerische Adresse (z. B. „192.168.1.147") oder eine normale Textadresse (z. B. „google.com") sein. Die Portnummer des Empfängers muss ebenfalls angegeben werden, ebenso wie die Nachricht selbst. Der SEND-Befehl kann als Antwort auf eine eingehende Nachricht oder eigenständig verwendet werden. |
| WS2812 Typ, Pin, Anzahl, Wert%[()] | Dieser Befehl steuert einen oder mehrere WS2812-LED-Chips an, die an „pin" angeschlossen sind. Beachte, dass der Pin auf einen digitalen Ausgang gesetzt sein muss, bevor dieser Befehl verwendet wird.<br>„type" ist ein einzelnes Zeichen, das den Typ des angesteuerten Chips angibt:<br>O = Original WS2812<br>B = WS2812B<br>S = SK6812<br>W = SK6812W (RGBW) |

| | |
|---|---|
| | „nbr“ ist die Anzahl der LEDs in der Kette (1 bis 256). Das Array „value%()“ sollte ein Integer-Array sein, dessen Größe genau der Anzahl der anzusteuernden LEDs entspricht.<br>Bei den ersten drei Varianten sollte jedes Element im Array die Farbe im normalen RGB888-Format enthalten (d. h. 0 bis &HFFFFFF).<br>Verwende für den Typ W einen RGBW-Wert (0–&HFFFFFFFF).<br>Wenn nur eine LED angeschlossen ist, sollte für „value%“ eine einzelne Ganzzahl verwendet werden (d. h. kein Array). |
| XMODEM SEND<br>or<br>XMODEM SEND file$<br>or<br>XMODEM RECEIVE<br>or<br>XMODEM RECEIVE file$<br>or<br>XMODEM CRUNCH | Überträgt ein BASIC-Programm unter Verwendung des XModem-Protokolls zu oder von einem Remote-Computer. Die Übertragung erfolgt über die USB-Konsolenverbindung.<br>XMODEM SEND sendet das aktuell im Programmspeicher des PicoMite befindliche Programm an das Remote-Gerät.<br>XMODEM RECEIVE nimmt ein vom Remote-Gerät gesendetes Programm entgegen und speichert es im Programmspeicher des PicoMite, wobei das dort aktuell befindliche Programm überschrieben wird.<br>In beiden Fällen kannst du auch „file$“ angeben, wodurch die Daten zu/von einer Datei auf dem Flash-Dateisystem oder der SD-Karte übertragen werden. Wenn die Datei bereits existiert, wird sie beim Empfang einer Datei überschrieben.<br>Beachte, dass die Daten im RAM gepuffert werden, was die maximale Übertragungsgröße begrenzt. Dieser Befehl erstellt außerdem eine Sicherungskopie des Programms im Flash-Speicher, die automatisch wiederhergestellt wird, wenn die CPU zurückgesetzt wird oder die Stromversorgung ausfällt.<br>Die Option CRUNCH funktioniert wie RECEIVE, entfernt jedoch vor dem Speichern alle Kommentare, Leerzeilen und unnötigen Leerzeichen aus dem Programm. Dies kann bei großen Programmen genutzt werden, damit sie in den begrenzten Speicher passen.<br>SEND, RECEIVE und CRUNCH können mit S, R und C abgekürzt werden.<br>Das XModem-Protokoll erfordert ein kompatibles Softwareprogramm, das auf dem Remote-Computer läuft und mit dessen serieller Schnittstelle verbunden ist. Es wurde mit Tera Term unter Windows getestet und es wird empfohlen, dieses zu verwenden.<br>Nachdem du den XMODEM-Befehl in MMBasic ausgeführt hast, wähle:<br>Datei -> Übertragen -> XMODEM -> Empfangen/Senden<br>aus dem Tera Term-Menü, um die Übertragung zu starten.<br>Es kann bis zu 15 Sekunden dauern, bis die Übertragung beginnt. Wenn der XMODEM-Befehl keine Verbindung herstellen kann, kehrt er nach 60 Sekunden zur MMBasic-Eingabeaufforderung zurück und lässt den Programmspeicher unverändert.<br>Lade Tera Term von http://ttssh2.sourceforge.jp/ herunter |
| YMODEM SEND<br>or<br>YMODEM SEND file$<br>or<br>YMODEM RECEIVE<br>or<br>YMODEM RECEIVE file$<br>or<br>YMODEM CRUNCH | NUR RP2350-VERSIONEN OHNE USB<br>Dieser Befehl ahmt XMODEM nach, überträgt Dateien jedoch viel schneller (>10x) über eine serielle CDC-Verbindung<br>YMODEM unterscheidet sich von XMODEM dadurch, dass die Nachricht den Dateinamen enthält.<br>Das bedeutet, dass beim Senden einer Datei vom Pico an einen Computer das empfangende Programm automatisch denselben Namen verwendet, den du gesendet hast.<br>Falls die Datei bereits existiert, erhöht der Empfänger automatisch die Versionsnummer im Namen der empfangenen Datei.<br>fred1.bas, fred2.bas usw.<br>Beim Senden aus dem Speicher mit YMODEM S gibt es zwei Möglichkeiten. |

| | |
|---|---|
| | Wurde die Datei vom Laufwerk A: oder B: geladen, enthält sie einen versteckten Dateinamen, der verwendet wird.<br>Bei einer Datei, die über den Editor erstellt oder automatisch gespeichert wurde, erhält die empfangene Datei automatisch den Namen FILEn.DAT<br>Es liegt in der Verantwortung des empfangenden Programms, anzugeben, wo die Datei gespeichert wird.<br>Bei Teraterm ist dies eine Option, die du im Bildschirm „Allgemeine Einstellungen“ unter „Ordner für Dateiübertragung“ festlegst<br>Wenn du eine Datei auf dem Pico empfängst, kannst du den Namen und das Verzeichnis ganz normal frei festlegen<br>Hinweis: YMODEM reagiert sehr empfindlich auf die Verbindungsqualität und die USB-Implementierung auf dem Host. Wenn es bei dir nicht funktioniert, verwende weiterhin XMODEM |

# Funktionen

Beachte, dass die Funktionen im Zusammenhang mit Kommunikationsfunktionen ($I^2C$, 1-Wire und SPI) hier nicht aufgeführt sind, sondern in den Anhängen am Ende dieses Dokuments beschrieben werden.
Eckige Klammern zeigen an, dass der Parameter oder die Zeichen optional sind.

| | |
|---|---|
| ABS( number ) | Gibt den Absolutwert des Arguments „number“ zurück (d. h., ein eventuelles Minuszeichen wird entfernt und eine positive Zahl zurückgegeben). |
| ACOS( number ) | Gibt den Arkuskosinus des Arguments „number“ in Radianten zurück. |
| ASC( string$ ) | Gibt den ASCII-Code (d. h. den Byte-Wert) für den ersten Buchstaben in „string$“ zurück. |
| ASIN( number ) | Gibt den Invers-Sinus-Wert des Arguments „number“ in Radianten zurück. |
| ATN( number ) | Gibt den Arkustangens des Arguments „number“ in Radianten zurück. |
| ATAN2( y, x ) | Gibt den Arkustangens der beiden Zahlen x und y als Winkel in<br>Radianten.<br>Das entspricht der Berechnung des Arkustangens von y / x, nur dass die Vorzeichen<br>beider Argumente verwendet werden, um den Quadranten des Ergebnisses zu bestimmen. |
| BIN$( number [, chars]) | Gibt eine Zeichenkette zurück, die den Binärwert (Basis 2) für die „number“ angibt.<br>'chars' ist optional und gibt die Anzahl der Zeichen in der Zeichenkette an, wobei Null als führendes Auffüllzeichen dient. |
| BIN2STR$(type, value [,BIG]) | Gibt eine Zeichenfolge zurück, die die binäre Darstellung von „value“ enthält.<br>„type“ kann sein:<br>INT64 vorzeichenbehaftete 64-Bit-Ganzzahl, konvertiert in eine 8-Byte-Zeichenkette<br>UINT64 vorzeichenlose 64-Bit-Ganzzahl, konvertiert in eine 8-Byte-Zeichenkette<br>INT32 Vorzeichenbehaftete 32-Bit-Ganzzahl, konvertiert in eine 4-Byte-Zeichenkette<br>UINT32 32-Bit-Ganzzahl ohne Vorzeichen, konvertiert in eine 4-Byte-Zeichenkette<br>INT16 Vorzeichenbehaftete 16-Bit-Ganzzahl, konvertiert in eine 2-Byte-Zeichenkette<br>UINT16 vorzeichenlose 16-Bit-Ganzzahl, konvertiert in eine 2-Byte-Zeichenkette<br>INT8 Vorzeichenbehaftete 8-Bit-Ganzzahl, konvertiert in eine 1-Byte-Zeichenkette<br>UINT8 vorzeichenlose 8-Bit-Ganzzahl, konvertiert in eine 1-Byte-Zeichenkette<br>SINGLE Gleitkommazahl mit einfacher Genauigkeit, konvertiert in eine 4-Byte-Zeichenkette<br>DOUBLE Gleitkommazahl mit doppelter Genauigkeit, konvertiert in eine 8-Byte-Zeichenkette<br>Standardmäßig enthält die Zeichenkette die Zahl im Little-Endian-Format (d. h. das niedrigstwertige Byte steht an erster Stelle in der Zeichenkette). Wenn du |

| | |
|---|---|
| | den dritten Parameter auf „BIG“ setzt, wird die Zeichenkette im Big-Endian-Format zurückgegeben (d. h. das höchstwertige Byte steht an erster Stelle in der Zeichenkette). Bei der Konvertierung von Ganzzahlen wird ein Fehler ausgegeben, wenn der „Wert“ nicht in den „Typ“ passt (z. B. beim Versuch, den Wert 400 in einem INT8 zu speichern).<br>Diese Funktion erleichtert die Aufbereitung von Daten für effiziente binäre Datei-E/A oder die Vorbereitung von Zahlen für die Ausgabe an Sensoren und das Speichern im Flash-Speicher.<br>Siehe auch die Funktion STR2BIN |
| BIT(var%, bitno) | 'gibt den Wert eines bestimmten Bits (0–63) in einer Ganzzahlvariablen zurück (0 oder 1).<br>Siehe auch den Befehl BIT |
| BOUND(array() [,dimension] | Dies gibt die Obergrenze des Arrays für die angeforderte Dimension zurück.<br>Die Dimension ist standardmäßig 1, wenn sie nicht angegeben wird. Die Angabe eines Dimensionswerts von 0 gibt den aktuellen Wert von OPTION BASE zurück.<br>Nicht verwendete Dimensionen geben den Wert Null zurück.<br>Zum Beispiel:<br>DIM myarray(44,45)<br>BOUND(myarray(),2) gibt 45 zurück |
| BYTE(var$, byteno) | Gibt den ganzzahligen Wert eines bestimmten Bytes in einer Zeichenkette zurück (0–255). Dies entspricht ASC(MID$(var$,byteno,1)), arbeitet aber wesentlich schneller.<br>Siehe auch den Befehl BYTE |
| CALL(userfunname$, [,userfunparameters,..]) | Dies ist eine effiziente Methode, um benutzerdefinierte Funktionen programmgesteuert aufzurufen. (Siehe auch den Befehl CALL). In vielen Fällen kann damit komplexe SELECT- und IF THEN ELSEIF ENDIF-Klauseln vermieden werden, und die Verarbeitung erfolgt wesentlich effizienter.<br>„userfunname$“ kann eine beliebige Zeichenkette, Variable oder Funktion sein, die sich auf den Namen einer normalen Benutzerfunktion auflöst (kein integrierter Befehl). „userfunparameters“ sind dieselben Parameter, die zum direkten Aufruf der Funktion verwendet würden.<br>Eine typische Anwendung für diesen Befehl könnte das Schreiben einer beliebigen Art von Emulator sein, bei dem je nach einer bestimmten Variablen eine von vielen Funktionen aufgerufen werden soll. Er bietet auch eine Möglichkeit, einen Funktionsnamen als Variable an eine andere Subroutine oder Funktion zu übergeben. |
| CHOICE(condition, ExpressionIfTrue, ExpressionIfFalse) | Mit dieser Funktion kannst du einfache Entweder-oder-Auswahlen effizienter und schneller durchführen als mit IF THEN ELSE ENDIF-Klauseln.<br>Die Bedingung ist alles, was zu einem Wert ungleich Null (wahr) oder Null (falsch) führt.<br>Die Ausdrücke können alles sein, was du normalerweise einer Variablen zuweisen oder in einem Befehl verwenden würdest, und können Ganzzahlen, Gleitkommazahlen oder Zeichenfolgen sein.<br>Beispiele:<br>PRINT CHOICE(1, "hello","bye") gibt „Hello“ aus<br>PRINT CHOICE (0, „hello“, „bye“) gibt „Bye“ aus<br>a=1 : b=1 : PRINT CHOICE (a=b, 4, 5) gibt 4 aus |

| | |
|---|---|
| CHR$( number ) | Gibt eine Zeichenkette mit einem Zeichen zurück, das dem ASCII-Code (d. h. dem Byte-Wert) entspricht, der durch das Argument „number“ angegeben wird. |
| CINT( number ) | Rundet Zahlen mit Bruchteilen auf oder ab auf die nächste ganze Zahl oder den nächsten ganzzahligen Wert.<br>Zum Beispiel 45,47 wird auf 45 gerundet<br>45,57 wird auf 46 gerundet<br>-34,45 wird auf -34 gerundet<br>-34,55 wird auf -35 gerundet<br>Siehe auch INT() und FIX(). |
| COS( number ) | Gibt den Kosinus des Arguments „number“ in Radianten zurück. |
| CWD$ | Gibt das aktuelle Arbeitsverzeichnis auf dem Flash-Dateisystem oder der SD-Karte zurück. Gilt nicht für das exFAT-Format.<br>Das Format lautet: `A:/Verzeichnis1/Verzeichnis2`. |
| DATE$ | Gibt das aktuelle Datum basierend auf der internen Uhr von MMBasic als Zeichenkette im Format „TT-MM-JJJJ“ zurück. Zum Beispiel „28-07-2012“.<br>Die interne Uhr/der interne Kalender verfolgt die Uhrzeit und das Datum einschließlich Schaltjahren. Um das Datum einzustellen, verwende den Befehl DATE$ =. |
| DATETIME$(n) | Gibt das Datum und die Uhrzeit zurück, die der Epochenzahl „n“ entsprechen (Anzahl der Sekunden, die seit Mitternacht GMT am 1. Januar 1970 vergangen sind).<br>Das Format der zurückgegebenen Zeichenfolge lautet „TT-MM-JJJJ hh:mm:ss“. Verwende den Text NOW, um die aktuelle Datums- und Zeitzeichenfolge zu erhalten, d. h. DATETIME$(NOW) |
| DAY$(date$) | Gibt den Wochentag für ein bestimmtes Datum als Zeichenfolge zurück. Zum Beispiel „Montag“, „Dienstag“ usw.<br>„date$“ ist eine Zeichenfolge und ihr Format kann DD-MM-YY, DD-MM-YYYY oder YYYY-MM-DD lauten. Du kannst auch NOW verwenden, um den Tag für das aktuelle Datum zu erhalten, z. B. PRINT DAY$(NOW) |
| DEG( radians ) | Konvertiert „radians“ in degrees / Grad. |
| DEVICE(GAMEPAD channel, funct) | Gibt Daten von einem USB-PS3- oder PS4-Controller zurück.<br>'funct' ist ein 1- oder 2-stelliger Code, der die zurückzugebenden Informationen wie folgt angibt:<br>LX die Position der x-Achse des linken Analog-Joysticks<br>LY die Position der Y-Achse des linken Analog-Joysticks<br>RX die Position der x-Achse des rechten Analog-Joysticks<br>RY die Position der Y-Achse des rechten analogen Joysticks<br>GX der Messwert vom X-Achsen-Gyro (sofern unterstützt)<br>GY der Messwert vom Y-Achsen-Gyro (sofern unterstützt)<br>GZ der Messwert vom Z-Achsen-Gyro (sofern unterstützt)<br>AX der Messwert vom X-Achsen-Beschleunigungsmesser (sofern unterstützt)<br>AY den Messwert des Beschleunigungssensors auf der Y-Achse (sofern unterstützt)<br>AZ der Messwert vom Beschleunigungsmesser der Z-Achse (sofern unterstützt)<br>L die Position des linken Analogknopfes<br>R die Position des rechten Analogknopfes |

| | |
|---|---|
| | B ein Bitmap des Zustands aller Tasten. Ein Bit wird auf 1 gesetzt, wenn die Taste gedrückt ist.<br>T die ID-Nummer des Controllers<br>Die Tasten-Bitmap sieht wie folgt aus:<br>BIT 0 Taste R/R1<br>BIT 1 Taste Start/Optionen<br>BIT 2 Taste „Home“<br>BIT 3 Taste „Auswählen/Teilen“<br>BIT 4 Taste L/L1<br>BIT 5 Taste Cursor nach unten<br>BIT 6 Taste Cursor rechts<br>BIT 7 Cursor nach oben<br>BIT 8 Taste Cursor links<br>BIT 9 Rechte Schultertaste 2/R2<br>BIT 10 Taste X/Dreieck<br>BIT 11 Taste A/Kreis<br>BIT 12 Taste Y/Quadrat<br>BIT 13 Taste b/Kreuz<br>BIT 14 Linker Schultertaste 2/L2<br>BIT 15 Touchpad |
| DEVICE(MOUSE channel, funct) | Gibt Daten von einer über den „Kanal“ angeschlossenen Maus zurück.<br>Einer PS2-Maus wird immer Kanal 2 zugewiesen. Normalerweise wird auch einer USB-Maus Kanal 2 zugewiesen, dies kann jedoch variieren. Weitere Informationen findest du unter MM.INFO(USB n).<br>„funct“ ist ein einstelliger Code, der die zurückzugebenden Informationen wie folgt angibt:<br>X die X-Koordinate (0 bis MM.HRES-1)<br>Y die Y-Koordinate (0 bis MM.VRES-1)<br>L der Status der linken Maustaste<br>R der Status der rechten Maustaste<br>M der Status der mittleren Maustaste (Räderklick)<br>D 1, wenn ein Doppelklick mit der linken Maustaste erfolgt ist<br>W gibt die Änderung der Radposition seit dem letzten Aufruf an (PS2-Maus oder USB-Maus mit aktivierter Option „Mausempfindlichkeit“)<br>B den Status aller Tasten als Bitmap (nur USB-Maus) |
| DEVICE(WII [CLASSIC] funct) | Gibt Daten von einem Wii Classic Controller zurück.<br>'funct' ist ein 1- oder 2-stelliger Code, der die zurückzugebenden Informationen wie folgt angibt:<br>LX die Position der x-Achse des linken Analog-Joysticks<br>LY die Position des linken Analog-Joysticks auf der Y-Achse<br>RX die Position der x-Achse des rechten Analog-Joysticks<br>RY die Position der Y-Achse des rechten analogen Joysticks<br>L die Position des linken Analogknopfs<br>R die Position des rechten analogen Knopfes<br>B eine Bitmap, die den Status aller Schaltflächen angibt. Ein Bit wird auf 1 gesetzt, wenn die Schaltfläche gedrückt ist.<br>T Der ID-Code des Controllers – sollte hex &HA4200101 lauten<br>Die Schaltflächen-Bitmap sieht wie folgt aus:<br>BIT 0 Taste R |

| | |
|---|---|
| | BIT 1 Taste Start<br>BIT 2 Taste „Home“<br>BIT 3 Auswahltaste<br>BIT 4 Taste L<br>BIT 5 Taste Cursor nach unten<br>BIT 6 Taste Cursor rechts<br>BIT 7 Cursor nach oben<br>BIT 8 Taste Cursor links<br>BIT 9 ZR-Taste<br>BIT 10 Taste X<br>BIT 11 Taste a<br>BIT 12 Taste y<br>BIT 13 Taste b<br>BIT 14 Taste ZL |
| DEVICE(NUNCHUCK funct) | Gibt Daten von einem Nunchuck-Controller zurück.<br>„funct“ ist ein 1- oder 2-stelliger Code, der die zurückzugebenden Informationen wie folgt angibt:<br>AX die Beschleunigung auf der x-Achse<br>AY die Beschleunigung auf der y-Achse<br>AZ die Beschleunigung in der Z-Achse<br>JX die Position der Joystick-X-Achse<br>JY die Position der Y-Achse des Joysticks<br>C der Status der C-Taste<br>Z der Status der Z-Taste<br>T die ID-Nummer des Controllers – sollte im Hexadezimalformat lauten: &HA4200000 |
| DIR$( fspec, type )<br>or<br>DIR$( fspec )<br>or<br>DIR$( ) | Durchsucht das Standard-Flash-Dateisystem oder die SD-Karte nach Dateien und gibt die Namen der gefundenen Einträge zurück.<br>'fspec' ist eine Dateispezifikation mit Platzhaltern, wie sie auch vom Befehl FILES verwendet wird. Z. B. gibt „*.*“ alle Einträge zurück, „*.TXT“ gibt Textdateien zurück. Beachte, dass der Platzhalter *.* keine Dateien oder Ordner ohne Erweiterung findet.<br>'type' ist der Typ des zurückzugebenden Eintrags und kann einer der folgenden sein:<br>ALL Suche nach allen Dateien und Verzeichnissen<br>DIR Nur nach Verzeichnissen suchen<br>FILE Nur nach Dateien suchen (Standard, wenn „type“ nicht angegeben ist)<br>Die Funktion gibt den ersten gefundenen Eintrag zurück. Um nachfolgende Einträge abzurufen, verwende die Funktion ohne Argumente, d. h. DIR$( ). Die Rückgabe einer leeren Zeichenkette bedeutet, dass keine weiteren Einträge mehr vorhanden sind.<br>Dieses Beispiel gibt alle Dateien in einem Verzeichnis aus:<br>`f$ = DIR$("*.*", FILE)`<br>`DO WHILE f$ <> ""`<br>`PRINT f$`<br>`f$ = DIR$()`<br>`LOOP`<br>Du musst in das gewünschte Verzeichnis wechseln, bevor du diesen Befehl aufrufst. |

| | |
|---|---|
| DISTANCE( trigger, echo )<br>or<br>DISTANCE( trig-echo ) | Miss die Entfernung zu einem Ziel mit dem Ultraschall-Entfernungssensor HC-SR04.<br>Vierpolige Sensoren verfügen über separate Trigger- und Echo-Anschlüsse. „trigger“ ist der I/O-Pin, der mit dem „trig“-Eingang des Sensors verbunden ist, und „echo“ ist der Pin, der mit dem „echo“-Ausgang des Sensors verbunden ist.<br>Dreipolige Sensoren verfügen über einen kombinierten Trigger- und Echo-Anschluss; in diesem Fall musst du nur einen I/O-Pin für die Anbindung an den Sensor angeben.<br>Beachte, dass der HC-SR04 ein 5-V-Gerät ist, sodass bei Pico-Prozessoren (RP2040) eine Pegelumsetzung erforderlich ist, bei Pico 2-Prozessoren (RP2350) jedoch nicht.<br>Die I/O-Pins werden durch diese Funktion automatisch konfiguriert, und es können mehrere Sensoren an verschiedenen I/O-Pins verwendet werden.<br>Der zurückgegebene Wert ist die Entfernung in Zentimetern zum Ziel oder -1, wenn kein Ziel erkannt wurde, oder -2, wenn ein Fehler aufgetreten ist (z. B. Sensor nicht angeschlossen). |
| Draw3D(arg) | NICHT IN DER WEBSITE-VERSION VERFÜGBAR<br>Die DRAW3D-Funktion kann verwendet werden, um die Grenzen eines 3D-Objekts zu bestimmen, und könnte dazu dienen, einen Bereich zu definieren, der vor dem Neuzeichnen gelöscht werden soll.<br>Eine vollständige Beschreibung findest du im Dokument *„3D_Graphics_User_Manual.*pdf“ im PicoMite-Firmware-Download. |
| EOF( [#]fnbr ) | Gibt „true“ zurück, wenn die zuvor auf dem Flash-Dateisystem oder der SD-Karte für INPUT geöffnete Datei mit der Dateinummer „#fnbr“ am Ende der Datei positioniert ist.<br>Das # ist optional. Siehe auch die Befehle OPEN, INPUT und LINE INPUT sowie die Funktion INPUT$. |
| EPOCH(DATETIME$) | Gibt die Epochenzahl (Anzahl der Sekunden, die seit Mitternacht GMT am 1. Januar 1970 vergangen sind) für die angegebene DATETIME$-Zeichenkette zurück.<br>Das Format für DATETIME$ lautet „dd-mm-yyyy hh:mm:ss“, „dd-mm-yy hh:mm:ss“ oder „yyyy-mm-dd hh:mm:ss“. Verwende NOW, um die Epochenzahl für das aktuelle Datum und die aktuelle Uhrzeit zu erhalten, z. B. PRINT EPOCH(NOW) |
| EVAL( string$ ) | wertet 'string$' so aus, als wäre es ein BASIC-Ausdruck, und gibt das Ergebnis zurück.<br>'string$' kann eine Konstante, eine Variable oder ein Zeichenfolgenausdruck sein. Der Ausdruck kann alle Operatoren, Funktionen, Variablen, Unterprogramme usw. verwenden, die zum Zeitpunkt der Ausführung bekannt sind. Der zurückgegebene Wert ist je nach Ergebnis der Auswertung eine Ganzzahl, eine Gleitkommazahl oder eine Zeichenfolge.<br>Beispiel: S$ = "COS(RAD(30)) * 100" : PRINT EVAL(S$)<br>Zeigt an: 86,6025 |
| EXP( number ) | Gibt den Exponentialwert von „number“ zurück, d. h. e^x, wobei x „number“ ist. |
| FIELD$( string1, nbr, string2 [, string3] ) | Gibt ein bestimmtes Feld in einer Zeichenkette zurück, wobei die Felder durch Trennzeichen getrennt sind. Beachte, dass ein Leerzeichen nicht als Trennzeichen verwendet werden kann.<br>„nbr“ ist das Feld, das zurückgegeben werden soll (das erste ist nbr 1). „string1“ ist die zu durchsuchende Zeichenfolge und „string2“ ist eine |

| | |
|---|---|
| | Zeichenfolge, die die Trennzeichen enthält (es können mehrere verwendet werden). Das Leerzeichen darf nicht als Trennzeichen verwendet werden.<br>'string3' ist optional und enthält, falls angegeben, Zeichen, die in 'string1' zur Einfassung von Text verwendet werden (d. h., in eingefasstem Text wird nicht nach einem Trennzeichen gesucht).<br>Zum Beispiel:<br>S$ = "foo, boo, zoo, doo"<br>r$ = FIELD$(s$, 2, ",")<br>ergibt r$ = "boo". Während:<br>s$ = "foo, 'boo, zoo', doo"<br>r$ = FIELD$(s$, 2, ",", "'")<br>ergibt r$ = "boo, zoo". |
| FIX( number ) | Runde eine Zahl auf eine ganze Zahl ab, indem du den Dezimalpunkt und alle Zeichen rechts davon entfernst.<br>Beispielsweise ergibt 9,89 den Wert 9 und -2,11 den Wert -2.<br>Der wesentliche Unterschied zwischen FIX() und INT() besteht darin, dass FIX() eine echte Ganzzahlfunktion bereitstellt (d. h. bei negativen Zahlen nicht wie INT() die nächstniedrigere Zahl zurückgibt). Dieses Verhalten dient der Kompatibilität mit Microsoft.<br>Siehe auch CINT(). |
| FLAG(n%) | Gibt den Wert (0 oder 1) des Bits n% (0–63) im System-Flag-Register zurück.<br>Siehe auch MM.FLAGS und die Befehle FLAG und FLAGS |
| FORMAT$( nbr [, fmt$] ) | Gibt eine Zeichenkette zurück, die „nbr“ darstellt, formatiert gemäß den Angaben in der Zeichenkette „fmt$“.<br>Die Formatangabe beginnt mit einem %-Zeichen und endet mit einem Buchstaben. Alles außerhalb dieser Konstruktion wird unverändert in die Ausgabe kopiert.<br>Die Struktur einer Formatangabe lautet:<br>% [flags] [width] [.precision] type<br>Wobei „flags“ Folgendes sein kann:<br>- Den Wert innerhalb einer bestimmten Feldbreite linksbündig ausrichten<br>0 Verwende 0 für das Füllzeichen anstelle eines Leerzeichens<br>+ Erzwingt die Anzeige des +-Zeichens bei positiven Zahlen<br>Leerzeichen Lässt bei einem positiven Wert ein Leerzeichen anstelle des Vorzeichens anzeigen. Bei negativen Werten wird weiterhin das Minuszeichen angezeigt<br>„width“ ist die Mindestanzahl der auszugebenden Zeichen; bei weniger Zeichen wird die Zahl aufgefüllt, bei mehr Zeichen wird die Breite erweitert.<br>„precision“ gibt die Anzahl der Dezimalstellen an, die beim Typ e oder f generiert werden sollen, oder die maximale Anzahl der signifikanten Stellen beim Typ g; der Standardwert ist 4 Stellen. Wenn angegeben, muss der Präzisionswert mit einem Punkt (.) eingeleitet werden.<br>„type“ kann einer der folgenden Werte sein:<br>g Die Zahl wird automatisch für die beste Darstellung formatiert.<br>f Formatiert die Zahl mit dem Dezimalpunkt und den nachfolgenden Ziffern<br>e Die Zahl im Exponentialformat formatieren |

| | |
|---|---|
| | Wenn ein großes G oder F verwendet wird, wird in der Exponentialdarstellung ein großes E verwendet. Wenn keine Formatangabe gemacht wird, wird „%g“ angenommen.<br>Beispiele: format$(45) gibt 45 zurück<br>format$(45, „%g“) gibt 45 zurück |
| GETSCANLINE | NUR VGA- UND HDMI-VERSIONEN<br>Dies gibt die Zeile an, die gerade auf dem VGA-/HDMI-Monitor gezeichnet wird, im Bereich von 0 bis 525. Dies ist unabhängig vom aktuellen MODE.<br>Wenn du das nutzt, um Aktualisierungen auf dem Bildschirm zu timen, kannst du Timing-Effekte vermeiden, die durch Aktualisierungen entstehen, während der Bildschirm gerade aktualisiert wird.<br>Die erste sichtbare Zeile gibt den Wert 0 zurück. Jede Zeilennummer über 479 liegt in der Bildausblendphase. |
| GPS() | Die GPS-Funktion gibt Daten von einem seriellen Kommunikationskanal zurück, der mit den Befehlen OPEN GPS oder OPTION GPS als GPS geöffnet wurde.<br>Diese Funktion wird ausführlich in der Datei *„Option_GPS_User_Manual.pdf“* beschrieben, die im ZIP-Download der Firmware enthalten ist.<br>Die Funktion GPS(VALID) sollte vor dem Zugriff auf die Daten überprüft werden, um sicherzustellen, dass der zurückgegebene Wert gültig ist. |
| HEX$( number [, chars]) | Gibt eine Zeichenkette zurück, die den Hexadezimalwert (Basis 16) für die „number“ angibt.<br>„chars“ ist optional und gibt die Anzahl der Zeichen in der Zeichenkette an, wobei Nullen als führende Auffüllzeichen dienen. |
| INKEY$ | Überprüft den Konsoleneingabepuffer und entfernt, falls ein oder mehrere Zeichen in der Warteschlange stehen, das erste Zeichen und gibt es als einzelnes Zeichen in einer Zeichenkette zurück.<br>Ist der Eingabepuffer leer, gibt diese Funktion sofort eine leere Zeichenkette (d. h. „“) zurück. |
| INPUT$(nbr, [#]fnbr) | Gibt eine Zeichenkette zurück, die aus „nbr“ Zeichen besteht, die aus einer Datei oder einem seriellen Kommunikationsport gelesen wurden, der als „fnbr“ geöffnet wurde. Diese Funktion gibt so viele Zeichen zurück, wie sich in der Datei oder im Empfangspuffer befinden, bis zu „nbr“. Wenn keine Zeichen verfügbar sind, gibt sie sofort eine leere Zeichenkette zurück.<br>Es kann #0 verwendet werden, was sich auf den Eingabepuffer der Konsole bezieht.<br>Das # ist optional. Siehe auch den Befehl OPEN. |
| INSTR( [start-position,] string-searched$, string-pattern$ [,size] ) | Gibt die Position zurück, an der „string-pattern$“ in „string-searched$“ vorkommt, beginnend bei 'start-position'. Wenn 'start-position' nicht angegeben wird, ist der Standardwert 1.<br>Sowohl die zurückgegebene Position als auch 'start-position' verwenden 1 für das erste Zeichen, 2 für das zweite usw.<br>Die Funktion gibt Null zurück, wenn 'string-pattern$' nicht gefunden wird.<br>Wenn der optionale Parameter „size“ angegeben wird, wird „string-pattern“ als regulärer Ausdruck behandelt. Details findest du in *Anhang E*. |
| INT( number ) | Rundet einen Ausdruck auf die nächste ganze Zahl ab, die kleiner oder gleich dem Argument ist. Zum Beispiel gibt 9,89 den Wert 9 zurück und -2,11 den Wert -3. |

| | |
|---|---|
| | Dieses Verhalten dient der Kompatibilität mit Microsoft; die Funktion FIX() bietet eine echte Ganzzahlfunktion. Siehe auch CINT() . |
| JSON$(array%(), string$) | Gibt eine Zeichenkette zurück, die ein bestimmtes Element aus der im longstring-Array %() gespeicherten JSON-Eingabe darstellt. Beachte, dass viele JSON-Datensätze recht groß sind und möglicherweise zu groß sind, um mit dem verfügbaren Speicher geparst zu werden.<br>Beispiele (entnommen aus api.openweathermap.org):<br>JSON$(a%(), „name")<br>JSON$(a%(), „coord.lat")<br>JSON$(a%(), „weather[0].description")<br>JSON$(a%(), „list[4].weather[0].description |
| KEYDOWN(n) | Gibt den dezimalen ASCII-Wert der Taste zurück, die gerade gedrückt gehalten wird, oder Null, wenn keine Taste gedrückt ist. Diese Funktion gilt für USB-Tastaturen und PS2-Tastaturen (in MMBasic 6.02.01 oder höher).<br>Beachte, dass PS2-Tastaturen die Modifikatortasten nicht auf dieselbe Weise unterstützen wie USB-Tastaturen. PS2-Tastaturen unterscheiden nicht zwischen linker und rechter ALT- oder STRG-Taste, und MMBasic kann die Windows- und/oder Funktionswahltasten (sofern vorhanden) nicht verarbeiten.<br>Die Dezimalwerte für die Funktions- und Pfeiltasten sind in Anhang I aufgeführt.<br>Diese Funktion meldet mehrere gleichzeitig gedrückte Tasten, und der Parameter „n" gibt die Nummer der zu meldenden Tastenbetätigung an. KEYDOWN(0) gibt die Anzahl der gedrückten Tasten zurück<br>Wenn beispielsweise „c", „g" und „p" gleichzeitig gedrückt werden, gibt KEYDOWN(0) 3 zurück, KEYDOWN(1) gibt 99 zurück, KEYDOWN(2) gibt 103 zurück usw. Die Tasten müssen nicht gleichzeitig gedrückt werden und werden in der Reihenfolge ihrer Betätigung gemeldet. Wenn du einen Finger von einer Taste nimmst, wird die nächste gedrückte Taste zur Nummer 1.<br>Die erste Taste ('n' = 1) wird in den Tastaturpuffer geschrieben (zugänglich über INKEY$), während die Tasten 2 bis 6 nur über diese Funktion abgerufen werden können. Die Verwendung dieser Funktion löscht den Konsoleneingabepuffer.<br>KEYDOWN(7) gibt alle gedrückten Modifikatortasten zurück. Diese Tasten werden nicht zur Zählung in keydown(0) hinzugefügt<br>Der Rückgabewert ist eine Bitmaske wie folgt:<br>lalt = 1, lctrl = 2, lgui = 4, lshift = 8, ralt = 16, rctrl = 32, rgui = 64, rshift = 128<br>KEYDOWN(8) gibt den aktuellen Status der Sperrtasten zurück. Diese Tasten werden nicht zur Zählung in keydown(0) hinzugefügt<br>Der Rückgabewert ist eine Bitmaske wie folgt:<br>caps_lock = 1, num_lock = 2, scroll_lock = 4<br>Beachte, dass manche Tastaturen die Anzahl der aktiven Tasten begrenzen, über die sie berichten können. |
| LCASE$( string$ ) | Gibt „string$" in Kleinbuchstaben zurück. |
| LCOMPARE(array1%(), array2%()) | Vergleicht den Inhalt der beiden langen Zeichenfolgenvariablen „array1%()" und „array2%()". Der Rückgabewert ist eine Ganzzahl und beträgt -1, wenn „array1%()" kleiner ist als „array2%()". Er beträgt Null, wenn Länge und Inhalt identisch sind, und +1, wenn „array1%()" größer ist als „array2%()". Der Vergleich verwendet den ASCII-Zeichensatz und unterscheidet zwischen Groß- und Kleinschreibung. |

| | |
|---|---|
| LEFT$( string$, nbr ) | Gibt eine Teilzeichenfolge von „string$“ mit „nbr“ Zeichen vom linken Anfang der Zeichenfolge zurück. |
| LEN( string$ ) | Gibt die Anzahl der Zeichen in 'string$' zurück. |
| LGETBYTE(array%(), n) | Gibt den numerischen Wert des „n“-ten Bytes in dem in „array%()“ enthaltenen LONGSTRING zurück. Diese Funktion berücksichtigt die Einstellung von OPTION BASE bei der Bestimmung des zurückzugebenden Bytes. |
| LGETSTR$(array%(), start, length) | Gibt einen Teil einer langen Zeichenkette, die in „array%()“ gespeichert ist, als normale MMBasic-Zeichenkette zurück. Die Parameter start und length definieren den Teil der Zeichenkette, der zurückgegeben werden soll. |
| LINSTR(array%(), search$ [,start] [,size])) | Gibt die Position einer Suchzeichenfolge in einer langen Zeichenfolge zurück.<br>Der zurückgegebene Wert ist eine Ganzzahl und beträgt Null, wenn die Teilzeichenfolge nicht gefunden werden kann. ‘array%()’ ist die zu durchsuchende Zeichenfolge und muss eine lange Zeichenfolgenvariable sein. ‘search$’ ist die zu suchende Teilzeichenfolge und muss eine normale MMBasic-Zeichenfolge oder ein Ausdruck sein (keine lange Zeichenfolge). Bei der Suche wird die Groß-/Kleinschreibung beachtet.<br>Normalerweise beginnt die Suche am ersten Zeichen in 'array%()', aber mit dem optionalen dritten Parameter kannst du die Startposition der Suche festlegen.<br>Wenn der optionale Parameter „size“ angegeben wird, wird „search$“ als regulärer Ausdruck behandelt. Details findest du in *Anhang E*. |
| LLEN(array%()) | Gibt die Länge einer in „array%()“ gespeicherten langen Zeichenkette zurück. |
| LINPUT(array%(),fnbr,nbr) | Dies liest „nbr“ Bytes aus einer als „fnbr“ geöffneten Datei in den LONGSTRING „array%()“.<br>Die Funktion gibt die Anzahl der tatsächlich gelesenen Bytes zurück; wenn du dich also nahe am<br>Ende der Datei, kann die Zahl geringer sein als die angeforderte.<br>Das funktioniert nur mit Datei-E/A und nicht mit serieller oder Konsolen-E/A |
| LOC( [#]fnbr ) | Bei einer Datei im Flash-Dateisystem oder auf der SD-Karte, die als „fnbr“ geöffnet wurde, gibt diese Funktion die aktuelle Position des Lese-/Schreibzeigers in der Datei zurück. Beachte, dass das erste Byte einer Datei mit 1 nummeriert ist.<br>Bei einem seriellen Kommunikationsport, der als „fnbr“ geöffnet wurde, gibt diese Funktion die Anzahl der empfangenen Bytes zurück, die im Empfangspuffer darauf warten, gelesen zu werden. Du kannst #0 verwenden, was sich auf den Eingabepuffer der Konsole bezieht.<br>Das # ist optional. |
| LOF( [#]fnbr ) | Bei einer Datei auf dem Flash-Dateisystem oder der SD-Karte, die als „fnbr“ geöffnet wurde, gibt diese Funktion die aktuelle Länge der Datei in Bytes zurück.<br>Bei einem als „fnbr“ geöffneten seriellen Kommunikationsport gibt diese Funktion den verbleibenden Platz (in Zeichen) im Sendepuffer zurück. Beachte, dass MMBasic beim Hinzufügen eines neuen Zeichens pausiert, wenn der Puffer voll ist, und wartet, bis wieder Platz verfügbar ist; diese Funktion kann verwendet werden, um dies zu vermeiden.<br>Das # ist optional. |
| LOG( number ) | Gibt den natürlichen Logarithmus des Arguments „Zahl“ zurück. |

| | |
|---|---|
| MAP( n ) | NUR FÜR HDMI-, VGA- UND PICOMITE RP2350-BUFFERED-TREIBER<br>Gibt den 24-Bit-RGB-Wert für den Index „n“ in der Farbkartentabelle zurück.<br>Siehe den Befehl MAP. Dies ermöglicht es dem Basic-Programmierer, eine durch den Befehl MAP angegebene Farbe zu verwenden<br>z. B.<br>MAP(8) = RGB(100,100,100)<br>MAP SET<br>Pixel x,y,map(8)<br><br>Hinweis: Bei VGA werden alle durch den MAP-Befehl festgelegten Farben in die nächstgelegene RGB121-Farbe umgewandelt, wie sie durch das VGA-Widerstandsnetzwerk bestimmt wird. Bei HDMI-Displays werden Farben in die nächstgelegene RGB555-Farbe (Auflösung 640x480) oder RGB332-Farbe (Auflösung 1024x768 oder 1280x720) umgewandelt. Bei Displays mit gepufferten Treibern des PIcoMite RP2350 werden die Farben in die nächstgelegene RGB332-Farbe umgewandelt. |
| MATH<br><br>**Simple functions** | Die Math-Funktion führt viele einfache mathematische Berechnungen durch, die man in Basic programmieren könnte, aber es gibt Geschwindigkeitsvorteile, wenn man Schleifenstrukturen in C programmiert, und es hat den Vorteil, dass sie nach dem Debuggen für alle da sind, ohne das Rad neu erfinden zu müssen. |
| 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 Tangens von a zurück |
| MATH(CRCn data [,length] [,polynome] [,startmask] [,endmask] [,reverseIn] [,reverseOut] | Berechnet den CRC auf n Bit (8, 12, 16, 32) von „data“. „data“ kann ein Integer- oder Fließkomma-Array oder eine Zeichenfolgenvariable sein. „Length“ ist optional; wenn nicht angegeben, wird die Größe des Arrays oder die Länge der Zeichenfolge verwendet. Die Standardwerte für startmask, endmask, reverseIn und reversOut sind alle Null. reverseIn und reversOut sind beide Boolesche Werte und nehmen den Wert 1 oder 0 an. Die Standardwerte für die Polynome sind CRC8=&H07, CRC12=&H80D, CRC16=&H1021, crc32=&H04C11DB7<br>z. B. für crc16_CCITT verwende MATH(CRC16 array(), n,, &HFFFF) |
| MATH(RAND) | Gibt eine Zufallszahl zwischen 0,0 und 1,0 zurück, unter Verwendung des „Mersenne-Twister-Algorithmus“. Wenn nicht mit MATH RANDOMIZE initialisiert, wird bei der ersten Verwendung die Zeit seit dem Booten in Mikrosekunden als Startwert verwendet. Hinweis: Der RP2350 verfügt über einen Hardware-Zufallszahlengenerator. |
| **Simple Statistics** | |
| MATH(CHI a()) | Gibt den Pearson-Chi-Quadrat-Wert des zweidimensionalen Arrays a()) zurück |

| | |
|---|---|
| MATH(CHI_p a()) | Gibt die zugehörige Wahrscheinlichkeit in % des Pearson-Chi-Quadrat-Werts des zweidimensionalen Arrays a()) zurück |
| MATH(CROSSING array() [,level] [,direction] | Dies gibt den Array-Index zurück, an dem die Werte im Array den „Level“ in der angegebenen Richtung überschreiten. Level ist standardmäßig 0. Richtung ist standardmäßig 1 (gültige Werte sind -1 oder 1) |
| 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 eine beliebige Anzahl von Dimensionen haben. Wenn die ganzzahlige Variable angegeben wird, wird sie mit dem Index des Maximalwerts im Array aktualisiert. Dies ist nur bei eindimensionalen Arrays verfügbar |
| MATH(MEAN a()) | Gibt den Durchschnitt aller Werte im Array a() zurück; a() kann eine beliebige Anzahl von Dimensionen haben |
| MATH(MEDIAN a()) | Gibt den Median aller Werte im Array a() zurück; a() kann eine beliebige Anzahl von Dimensionen haben |
| MATH(MIN a(), [index%]) | Gibt das Minimum aller Werte im Array a() zurück; a() kann eine beliebige Anzahl von Dimensionen haben. Wenn die ganzzahlige Variable angegeben wird, wird sie mit dem Index des Minimalwerts im Array aktualisiert. Dies ist nur bei eindimensionalen Arrays verfügbar. |
| MATH(SD a()) | Gibt die Stichprobenstandardabweichung aller Werte im Array a() zurück; a() kann eine beliebige Anzahl von Dimensionen haben |
| MATH(SUM a()) | Gibt die Summe aller Werte im Array a() zurück; a() kann eine beliebige Anzahl von Dimensionen haben |
| **Vector Arithmetic** | |
| MATH(MAGNITUDE v()) | Gibt die Norm des Vektors v() zurück. Der Vektor kann eine beliebige Anzahl von Elementen haben |
| MATH(DOTPRODUCT v1(), v2()) | Gibt das Skalarprodukt der beiden Vektoren v1() und v2() zurück. Die Vektoren können eine beliebige Anzahl von Elementen haben, müssen aber die gleiche Kardinalität aufweisen |
| **Matrix Arithmetic** | |
| MATH(M_DETERMINANT array!()) | Gibt die Determinante des Arrays zurück. Das Array muss quadratisch sein. |

| | |
|---|---|
| **Creation**<br>complex% = MATH(C_CPLX r!, i!)<br>complex% = MATH(C_POLAR radius!, angle!)<br>**Floating returns**<br>real! = MATH(C_REAL complex%)<br>imag! = MATH(C_IMAG complex%)<br>arg! = MATH(C_ARG complex%) | MMBasic unterstützt eine umfassende Palette an Funktionen zur Bearbeitung komplexer Zahlen. In dieser Implementierung haben komplexe Zahlen einen 32-Bit-Realteil und einen 32-Bit-Imaginärteil. Damit dies in MMBasic funktioniert, werden diese Werte in Ganzzahlen (64-Bit) gespeichert. |

| | |
|---|---|
| mod! = MATH(C_MOD complex%)<br>phase! = MATH(C_PHASE complex%)<br>**Unary functions**<br>complex1% = MATH(C_CONJ complex2%)<br>complex1% = MATH(C_SIN complex2%)<br>complex1% = MATH(C_COS complex2%)<br>complex1% = MATH(C_TAN complex2%)<br>complex1% = MATH(C_ASIN complex2%)<br>complex1% = MATH(C_ACOS complex2%)<br>complex1% = MATH(C_ATAN complex2%)<br>complex1% = MATH(C_SINH complex2%)<br>complex1% = MATH(C_COSH complex2%)<br>complex1% = MATH(C_TANH complex2%)<br>complex1% = MATH(C_ASINH complex2%)<br>complex1% = MATH(C_ACOSH complex2%)<br>complex1% = MATH(C_ATANH complex2%)<br>complex1% = MATH(C_PROJ complex2%)<br>**Basic Arithmetic**<br>complex1% = MATH(C_ADD complex2%,complex3%)<br>complex1% = MATH(C_SUB complex2%,complex3%)<br>complex1% = MATH(C_MUL complex2%,complex3%)<br>complex1% = MATH(C_DIV complex2%,complex3%)<br>complex1% = MATH(C_POW complex2%,complex3%)<br>complex1% = MATH(C_AND complex2%,complex3%)<br>complex1% = MATH(C_OR complex2%,complex3%)<br>complex1% = MATH(C_XOR complex2%,complex3%) | |
| MATH(PID channel, setpoint!, measurement)) | Diese Funktion muss in der PID-Callback-Subroutine für den angegebenen „channel“ aufgerufen werden und gibt den Ausgang der Reglerfunktion zurück.<br>Der Wert „setpoint“ ist der Sollwert, den der Regler zu erreichen versucht. „measurement“ ist der aktuelle Wert aus der realen Welt.<br>https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=17263<br>Ein Beispiel für die Einrichtung und den Betrieb eines PID-Reglers |
| MATH(BASE64 ENCODE/DECODE in$/in(), out$/out()) | Gibt die Länge von out$/out() zurück. Dies codiert oder decodiert die Daten in 'in' nach Base64 und speichert das Ergebnis in 'out'. Wenn Arrays als Ausgabe verwendet werden, müssen sie im Verhältnis zur Eingabe und zur Richtung groß genug sein. Die Verschlüsselung erhöht die Länge um 4/3 und die Entschlüsselung verringert sie um 3/4. |
| MAX( arg1 [, arg2 [, …]] )<br>or<br>MIN( arg1 [, arg2 [, …]] ) | Gibt die größte oder kleinste Zahl in der Argumentliste zurück.<br>Beachte, dass es sich um einen Gleitkomma-Vergleich handelt (ganzzahlige Argumente werden in Gleitkommazahlen umgewandelt) und ein Gleitkommawert zurückgegeben wird. |
| MID$( string$, start )<br>or<br>MID$( string$, start, nbr ) | Gibt eine Teilzeichenfolge von „string$“ zurück, die bei „start“ beginnt und sich über „nbr“ Zeichen erstreckt. Das erste Zeichen in der Zeichenfolge ist die Nummer 1.<br>Wenn „nbr“ weggelassen wird, erstreckt sich die zurückgegebene Zeichenkette bis zum Ende von „string$“ |
| OCT$( number [, chars]) | Gibt eine Zeichenkette zurück, die die oktale (Basis 8) Darstellung von 'number' angibt.<br>'chars' ist optional und gibt die Anzahl der Zeichen in der Zeichenkette an, wobei Null als führendes Auffüllzeichen dient. |

| | |
|---|---|
| PEEK(BYTE addr%)<br>or<br>PEEK(SHORT addr%)<br>Or<br>PEEK(WORD addr%)<br>or<br>PEEK(INTEGER addr%)<br>or<br>PEEK(FLOAT addr%)<br>Or<br>PEEK(VARADDR var)<br>or<br>PEEK(VARHEADER var)<br>or<br>PEEK(CFUNADDR cfun)<br>or<br>PEEK(VAR var, ±offset)<br>or<br>PEEK( VARTBL, ±offset)<br>or<br>PEEK( PROGMEM, ±offset)<br><br><br><br><br>PEEK(BP n%)<br><br>PEEK(SP n%)<br><br>PEEK(WP n%) | Hinweis: Adressen werden abgerundet, um dem angeforderten Datentyp zu entsprechen (z. B. PEEK(SHORT) oder es wird ein Fehler ausgegeben, wenn nicht ausgerichtet, z. B. PEEK(SP<br>Gibt ein Byte oder ein Wort innerhalb des virtuellen Speicherbereichs des Prozessors zurück.<br>BYTE gibt das Byte (8 Bit) zurück, das sich an der Adresse 'addr%' befindet<br>SHORT gibt die kurze Ganzzahl (16 Bit) zurück, die sich an der Adresse 'addr%' befindet<br>WORD gibt das Wort (32 Bit) zurück, das sich an der Adresse 'addr%' befindet<br>INTEGER gibt die Ganzzahl (64 Bit) zurück, die sich an der Adresse 'addr%' befindet<br>FLOAT gibt die Gleitkommazahl (64 Bit) zurück, die sich an der Adresse 'addr%' befindet<br>VARADDR gibt die Adresse (32 Bit) der Variablen 'var' im Speicher zurück. Ein Array wird als var() angegeben.<br>VARHEADER gibt die Adresse (32 Bit) des Header der Variablen 'var' im Speicher zurück. Ein Array wird als var() angegeben. Dies ist die Adresse des ersten Bytes des Variablennamens.<br>CFUNADDR gibt die Adresse (32 Bit) der CFunction „cfun“ im Speicher zurück. Diese Adresse kann an eine andere CFunction übergeben werden, die sie dann aufrufen kann, um einen gemeinsamen Vorgang auszuführen.<br>VAR gibt ein Byte in dem Speicher zurück, der für 'var' reserviert ist. Ein Array wird als var() angegeben.<br>VARTBL gibt ein Byte in dem Speicher zurück, der der von MMBasic verwalteten Variablentabelle zugewiesen ist. Beachte, dass nach dem Schlüsselwort VARTBL ein Komma steht.<br>PROGMEM gibt ein Byte aus dem Speicher zurück, der dem Programm zugewiesen wurde. Beachte, dass nach dem Schlüsselwort PROGMEM ein Komma steht.<br>Beachte, dass „addr%“ eine ganze Zahl sein sollte.<br>PEEK(bp n%) ' gibt das Byte an der Adresse n% zurück und erhöht n%, um auf das nächste Byte zu zeigen.<br>PEEK(sp n%) ' gibt das Short an der Adresse n% zurück und erhöht n%, um auf das nächste Short zu zeigen.<br>PEEK(wp n%) ' gibt das Wort an der Adresse n% zurück und erhöht n%, um auf das nächste Wort zu zeigen. |
| PI | Gibt den Wert von Pi zurück. |
| PIN( pin ) | Gibt den Wert am externen I/O-„Pin“ zurück. Null bedeutet digital Low, 1 bedeutet digital High und bei analogen Eingängen wird die gemessene Spannung als Gleitkommazahl zurückgegeben.<br>Frequenzeingänge geben die Frequenz in Hz zurück. Ein Periodeneingang gibt die Periode in Millisekunden zurück, während ein Zählereingang den Zählstand seit dem Reset zurückgibt (die Zählung erfolgt an der positiven steigenden |

| | |
|---|---|
| | Flanke). Der Zählereingang kann auf Null zurückgesetzt werden, indem der Pin auf Zählereingang zurückgesetzt wird (auch wenn er bereits so konfiguriert ist).<br>Wenn ein Pin als Ausgang konfiguriert ist, gibt diese Funktion den Wert der Ausgangseinstellung zurück (d. h. High oder Low). Siehe auch die Befehle SETPIN und PIN(). Eine allgemeine Beschreibung der Ein-/Ausgangs-Fähigkeiten des PicoMite findest du im Kapitel *„ Verwendung der I/O-Pins*“. |
| PIN( BOOTSEL ) | Gibt den Status des Boot-Auswahlschalters zurück, sodass er in einem Programm als Benutzereingabe verwendet werden kann. |
| PIN( TEMP ) | Gibt die Temperatur des RP2040/RP2350-Chips zurück (Details findest du im Datenblatt). |
| PIO(DMA RX POINTER)<br>PIO(DMA TX POINTER) | Gibt das aktuelle Datenelement zurück, das gerade vom PIO geschrieben oder gelesen wird. |
| PIO (SHIFTCTRL push_threshold [,pull_threshold] [,autopush] [,autopull] [,in_shiftdir] [,out_shiftdir] [,fjoin_tx] [,fjoin_rx]) | Hilfsfunktion zur Berechnung des Werts von shiftctrl für den Befehl INIT MACHINE. |
| PIO (PINCTRL no_side_set_pins [,no_set_pins] [,no_out_pins] [,IN base] [,side_set_base] [,set_base][, out_base]) | Hilfsfunktion zur Berechnung des Werts von pinctrl für den Befehl INIT MACHINE. Hinweis: Die Pin-Parameter müssen im Format GPn angegeben werden. |
| PIO (EXECCTRL jmp_pin ,wrap_target, wrap [,side_pindir] [,side_en]) | Hilfsfunktion zur Berechnung des Werts von execctrl für den Befehl INIT MACHINE |
| PIO(READFIFO a, b, c) | Lesen aus einem PIO-FIFO<br>„a“ ist der PIO (0 oder 1), „b“ ist die Zustandsmaschine (0...3), „c“ ist das FIFO-Register (*0…3) |
| PIO (FDEBUG pio) | Gibt den Wert des FSDEBUG-Registers für den angegebenen 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<br>PIO(FLEVEL pio) |
| PIO(FLEVEL pio ,sm, DIR) | dir kann RX oder TX sein. Gibt den Füllstand des angegebenen FIFO zurück |
| PIO(.WRAP)<br>PIO(.WRAP TARGET) | gibt die Position der .wrap-Anweisung in PIO ASSEMBLE zurück<br>gibt die Position der .wrap-Zielanweisung in PIO ASSEMBLE zurück.<br>Diese können in der PIO(EXECCTRL-Funktion wie folgt verwendet werden:<br>PIO (EXECCTRL jmp_pin PIO(.WRAP TARGET), PIO(.WRAP) [,side_pindir] [,side_en]) |
| PIO(NEXT LINE) | Gibt den nächsten ungenutzten PIO-Befehlsslot nach einem durch END PROGRAM beendeten Block von PIO-Befehlen zurück |

| | |
|---|---|
| PIXEL( x, y) | Gibt die Farbe eines Pixels auf dem Videoausgang oder dem LCD-Display zurück. 'x' ist die horizontale Koordinate und 'y' die vertikale Koordinate des Pixels.<br>Wenn ein LCD-Display verwendet wird, muss es einen der Controller SSD1963, ILI9341, ILI9488 oder ST7789_320 verwenden. |
| PORT(start, nbr [,start, nbr]…) | Gibt den Wert einer Reihe von I/O-Pins in einem Schritt zurück.<br>'start' ist eine I/O-Pin-Nummer und ihr Wert wird als Bit 0 zurückgegeben. 'start'+1 wird als Bit 1 zurückgegeben, 'start'+2 als Bit 2 und so weiter für die Anzahl 'nbr' an Bits. Die verwendeten I/O-Pins müssen fortlaufend nummeriert sein, und jeder I/O-Pin, der ungültig oder nicht als Eingang konfiguriert ist, führt zu einem Fehler. Das Paar „start/nbr“ kann bis zu 25 Mal wiederholt werden, falls weitere Gruppen von Eingangspins hinzugefügt werden müssen.<br>Diese Funktion gibt auch den Ausgangszustand eines als Ausgang konfigurierten Pins zurück.<br>Dies kann genutzt werden, um bequem mit parallelen Geräten wie Speicherchips zu kommunizieren. 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.<br>Hinweis: Wenn die Pins mit der GPn-Syntax definiert werden, ignoriert die Firmware ungültige Pins, sodass `PORT(GP0, 8)` acht I/O-Pins liest: GP0, GP1, GP2, GP3, GP4, GP5, GP6 und GP7.<br>Siehe den PORT-Befehl, um gleichzeitig auf mehrere Pins auszugeben. |
| PULSIN( pin, polarity )<br>or<br>PULSIN( pin, polarity, t1 )<br>or<br>PULSIN( pin, polarity, t1, t2 ) | Misst die Breite eines Eingangsimpulses von 1 µs bis 1 Sekunde mit einer Auflösung von 0,1 µs.<br>'pin' ist der für die Messung zu verwendende I/O-Pin, er muss zuvor als digitaler Eingang konfiguriert worden sein. 'polarity' ist der Typ des zu messenden Impulses; bei Null gibt die Funktion die Breite des nächsten negativen Impulses zurück, bei einem Wert ungleich Null misst sie den nächsten positiven Impuls.<br>'t1' ist die Zeitüberschreitung, die beim Warten auf den Impuls angewendet wird, 't2' ist die Zeitüberschreitung, die während der Messung des Impulses verwendet wird. Beide Werte sind in Mikrosekunden (µs) angegeben und optional. Wenn „t2“ weggelassen wird, wird der Wert von „t1“ für beide Zeitlimits verwendet. Wenn sowohl „t1“ als auch „t2“ weggelassen werden, werden die Zeitlimits auf 100000 (d. h. 100 ms) gesetzt.<br>Diese Funktion gibt die Breite des Impulses in Mikrosekunden (µs) zurück oder -1, falls ein Timeout aufgetreten ist. Die Messung hat eine Genauigkeit von ±0,5 % und ±0,5 µs.<br>Beachte, dass diese Funktion das laufende Programm während der Messung anhält und Interrupts in diesem Zeitraum ignoriert werden. |
| RAD( degrees ) | Konvertiert „Grad“ in Radiant. |
| RGB(red, green, blue)<br>or<br>RGB(shortcut) | Erzeugt einen RGB-True-Color-Wert.<br>„rot“, „blau“ und „grün“ stehen für die Intensität der jeweiligen Farbe. Ein Wert von Null steht für Schwarz und 255 für volle Intensität.<br>Mit „shortcut“ kannst du gängige Farben durch ihre Namen angeben. Die Farben, die benannt werden können, sind white, black, blue, green, cyan, red, magenta, yellow, brown, white, orange, pink, gold, salmon, beige, lightgrey und grey (oder in der US-Schreibweise gray/lightgray). Zum Beispiel RGB(red) oder RGB(cyan). |
| RIGHT$( string$, number-of-chars ) | Gibt eine Teilzeichenfolge von „string$“ mit „number-of-chars“ Zeichen ab dem rechten Ende (Ende) der Zeichenfolge zurück. |

| | |
|---|---|
| RND( number )<br>or<br>RND | Gibt eine Zufallszahl (RP2350) oder eine Pseudozufallszahl (RP2040) im Bereich von 0 bis 0,999999 zurück. Der Wert „Zahl“ wird ignoriert, falls angegeben.<br>Siehe auch den Befehl RANDOMIZE (nur RP2040). |
| 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 „number“ in Radianten zurück. |
| SPACE$( number ) | Gibt eine Zeichenkette aus Leerzeichen mit der Länge von „number“ zurück. |
| SPI ( data )<br>or<br>SPI2 ( data ) | Sende und empfange Daten über einen SPI-Kanal.<br>Eine einzelne SPI-Transaktion sendet Daten und empfängt gleichzeitig Daten vom Slave. „data“ sind die zu sendenden Daten, und die Funktion gibt die während der Transaktion empfangenen Daten zurück. „data“ kann eine Ganzzahl, eine Gleitkommavariable oder eine Konstante sein. |
| SPRITE() | Siehe Anhang G und das separate PDF-Dokument „SPRITE_User_Manual“ |
| SQR( number ) | Gibt die Quadratwurzel des Arguments „number“ zurück. |
| STR$( number )<br>or<br>STR$( number, m )<br>or<br>STR$( number, m, n )<br>or<br>STR$( number, m, n, c$ ) | Gibt eine Zeichenkette in der dezimalen (Basis 10) Darstellung von 'number' zurück.<br>Wenn „m“ angegeben wird, werden am Anfang der Zahl genügend Leerzeichen hinzugefügt, um sicherzustellen, dass die Anzahl der Zeichen vor dem Dezimalpunkt (einschließlich des Vorzeichens) mindestens „m“ Zeichen beträgt. Wenn „m“ Null ist oder die Zahl mehr als „m“ signifikante Stellen hat, werden keine Auffüllzeichen hinzugefügt.<br>Wenn 'm' negativ ist, wird positiven Zahlen das Pluszeichen und negativen Zahlen das Minuszeichen vorangestellt. Wenn 'm' positiv ist, wird nur das Minuszeichen verwendet.<br>'n' ist die Anzahl der Stellen, die nach dem Dezimalpunkt folgen sollen. Wenn der Wert Null ist, wird die Zeichenkette ohne Dezimalpunkt zurückgegeben. Wenn der Wert negativ ist, wird die Ausgabe immer im Exponentialformat mit einer Auflösung von 'n' Stellen erfolgen. Wenn 'n' nicht angegeben wird, variieren die Anzahl der Dezimalstellen und das Ausgabeformat automatisch je nach Zahl.<br>'c$' ist eine Zeichenkette, und wenn sie angegeben wird, wird das erste Zeichen dieser Zeichenkette anstelle eines Leerzeichens als Füllzeichen verwendet (siehe das Argument 'm'). Beispiele:<br>STR$(123.456) gibt „123.456“ zurück<br>STR$(-123.456) gibt „-123,456“ zurück<br>STR$(123.456, 1) gibt „123.456“ zurück<br>STR$(123,456, -1) gibt „+123,456“ zurück<br>STR$(123.456, 6) gibt „ 123.456“ zurück<br>STR$(123.456, -6) gibt „ +123.456“ zurück<br>STR$(-123,456, 6) gibt „ -123,456“ zurück<br>STR$(-123,456, 6, 5) gibt „ -123,45600“ zurück<br>STR$(-123,456, 6, -5) gibt „ -1,23456e+02“ zurück<br>STR$(53, 6) gibt „ 53“ zurück<br>STR$(53, 6, 2) gibt „ 53,00“ zurück<br>STR$(53, 6, 2, "*") gibt „****53.00“ zurück |
| STR2BIN(type, string$ [,BIG]) | Gibt eine Zahl zurück, die der binären Darstellung in „string$“ entspricht.<br>„type“ kann sein: |

| | |
|---|---|
| | INT64 konvertiert eine 8-Byte-Zeichenkette, die eine vorzeichenbehaftete 64-Bit-Ganzzahl darstellt, in eine Ganzzahl<br>UINT64 konvertiert eine 8-Byte-Zeichenkette, die eine vorzeichenbehaftete 64-Bit-Ganzzahl darstellt, in eine Ganzzahl<br>INT32 konvertiert eine 4-Byte-Zeichenkette, die eine vorzeichenbehaftete 32-Bit-Ganzzahl darstellt, in eine Ganzzahl<br>UINT32 konvertiert eine 4-Byte-Zeichenkette, die eine vorzeichenlose 32-Bit-Ganzzahl darstellt, in eine Ganzzahl<br>INT16 konvertiert eine 2-Byte-Zeichenkette, die eine vorzeichenbehaftete 16-Bit-Ganzzahl darstellt, in eine Ganzzahl<br>UINT16 konvertiert eine 2-Byte-Zeichenkette, die eine vorzeichenlose 16-Bit-Ganzzahl darstellt, in eine Ganzzahl<br>INT8 konvertiert eine 1-Byte-Zeichenkette, die eine vorzeichenbehaftete 8-Bit-Ganzzahl darstellt, in eine Ganzzahl<br>UINT8 konvertiert eine 1-Byte-Zeichenkette, die eine vorzeichenlose 8-Bit-Ganzzahl darstellt, in eine Ganzzahl<br>SINGLE konvertiert eine 4-Byte-Zeichenkette, die eine Gleitkommazahl mit einfacher Genauigkeit darstellt, in eine Gleitkommazahl<br>DOUBLE konvertiert eine 8-Byte-Zeichenkette, die eine Gleitkommazahl mit einfacher Genauigkeit darstellt, in eine Gleitkommazahl<br>Standardmäßig muss die Zeichenkette die Zahl im Little-Endian-Format enthalten (d. h. das<br>niedrigstwertige Byte steht an erster Stelle in der Zeichenkette). Wenn du den dritten Parameter auf „BIG“ setzt, wird die Zeichenkette im Big-Endian-Format interpretiert (d. h. das höchstwertige<br>Byte ist das erste in der Zeichenkette).<br>Diese Funktion erleichtert das Auslesen von Daten aus Binärdateien, das Interpretieren<br>Zahlen von Sensoren zu interpretieren oder Binärdaten effizient von Flash-Speicherchips<br>.<br>Es wird ein Fehler ausgelöst, wenn die Zeichenkette die falsche Länge für die<br>gewünschten Konvertierung<br>Siehe auch die Funktion BIN2STR$ |
| STRING$( nbr, ascii )<br>or<br>STRING$( nbr, string$ ) | Gibt eine Zeichenkette mit einer Länge von 'nbr' Bytes zurück, die entweder aus dem ersten Zeichen von string$ oder aus dem Zeichen besteht, das den ASCII-Wert 'ascii' darstellt, wobei es sich um eine Ganzzahl oder eine Gleitkommazahl im Bereich von 0 bis 255 handelt. |
| STRUCT( arg, …) | Gibt Daten aus einer Struktur (auch als benutzerdefinierter Typ bezeichnet) zurück, mit der zusammengehörige Variablen unterschiedlicher Typen unter einem einzigen Namen zusammengefasst werden können.<br>Dies wird ausführlich in der Datei *„MMBasic_Structures_Manual.pdf“* beschrieben, die im ZIP-Archiv des Firmware-Downloads enthalten ist. |
| TAB( number ) | Gibt Leerzeichen aus, bis die durch „number“ angegebene Spalte in der Konsolenausgabe erreicht ist. |
| TAN( number ) | Gibt den Tangens des Arguments „number“ in Radianten zurück. |
| TEMPR( pin [,timeout]) | Gibt die Temperatur zurück, die von einem an „pin“ angeschlossenen DS18B20-Temperatursensor gemessen wurde (der nicht konfiguriert sein muss). |

| | |
|---|---|
| | Der Rückgabewert ist in Grad Celsius mit einer Standardauflösung von 0,25 °C. Tritt während der Messung ein Fehler auf, ist der Rückgabewert 1000.<br>Die für die gesamte Messung benötigte Zeit beträgt 200 ms, und Interrupts werden während dieser Zeit ignoriert.<br>Der optionale Parameter „timeout“ kann verwendet werden, um die Standardeinstellung (200 ms) zu überschreiben und so auch langsamere Geräte zu berücksichtigen.<br>Alternativ kannst du den Befehl TEMPR START verwenden, um die Messung zu starten, und dein Programm kann während der Umwandlung andere Aufgaben ausführen. Wenn diese Funktion aufgerufen wird, wird der Wert sofort zurückgegeben, vorausgesetzt, die Umwandlungszeit ist abgelaufen. Ist dies nicht der Fall, wartet diese Funktion die verbleibende Umwandlungszeit ab, bevor sie den Wert zurückgibt.<br>Der DS18B20 kann separat über eine 3-V- bis 5-V-Versorgung mit Strom versorgt werden oder über parasitäre Energie vom Raspberry Pi Pico betrieben werden.<br>Weitere Details findest du im Kapitel *„Spezielle Hardwaregeräte*“. |
| TIME$ | Gibt die aktuelle Uhrzeit basierend auf der internen Uhr von MMBasic als Zeichenkette im Format „HH:MM:SS“ im 24-Stunden-Format zurück. Zum Beispiel „14:30:00“.<br>Um die aktuelle Uhrzeit einzustellen, verwende den Befehl TIME$ = |
| TIMER | Gibt die seit dem Zurücksetzen verstrichene Zeit in Millisekunden (d. h. 1/1000 einer Sekunde) zurück.<br>Der Timer wird beim Hochfahren oder bei einem CPU-Neustart auf Null zurückgesetzt, und du kannst ihn auch mit dem Befehl TIMER zurücksetzen. Wenn er nicht ausdrücklich zurückgesetzt wird, zählt er unendlich weiter (es handelt sich um eine 64-Bit-Zahl, die daher erst nach 200 Millionen Jahren auf Null zurückspringt). |
| TOUCH(X)<br>oder<br>TOUCH(Y)<br>oder nur FT6336<br>TOUCH(X2)<br>oder<br>TOUCH(Y2) | Gibt die X- oder Y-Koordinate der Stelle zurück, die gerade auf einem LCD-Bildschirm berührt wird.<br>Wenn der Bildschirm nicht berührt wird, gibt die Funktion -1 zurück.<br>Beim FT6336 geben TOUCH(X2) und TOUCH(Y2) die Position einer zweiten Berührungsstelle zurück oder -1, wenn keine zweite Stelle berührt wird. |
| TRIM$(source$ [,mask$] [,where/where$]) | Diese Funktion kann Zeichen vom Anfang oder Ende einer Zeichenkette oder von beiden entfernen<br>‘source$’ ist die Eingabezeichenfolge<br>„mask$“ ist eine Zeichenkette, die eine Liste der zu entfernenden Zeichen enthält. Wird sie weggelassen, ist der Standardwert ein Leerzeichen<br>„where/where$“ kann L, R oder B sein oder eine Zeichenkette, die mit L, R oder B beginnt, um<br>anzugeben, ob Zeichen links von der Quelle,<br>rechts von der Quelle oder von beiden Seiten entfernt werden sollen. Wird weggelassen, ist die Standardeinstellung L |
| UCASE$( string$ ) | Gibt „string$“ in Großbuchstaben zurück. |
| VAL( string$ ) | Gibt den numerischen Wert von „string$“ zurück. Wenn „string$“ eine ungültige Zahl ist, gibt die Funktion Null zurück.<br>Diese Funktion erkennt das Präfix &H für eine Hexadezimalzahl, &O für Oktal und &B für Binär. |

# Veraltete Befehle und Funktionen

Diese Befehle und Funktionen dienen hauptsächlich dazu, die Konvertierung von Programmen zu erleichtern, die für Microsoft BASIC geschrieben wurden. Für neue Programme solltest du die entsprechenden modernen Befehle in MMBasic verwenden.

Beachte, dass diese Befehle/Funktionen in Zukunft entfernt werden können, um Speicherplatz für andere Funktionen freizugeben.

| | |
|---|---|
| BITBANG | Ersetzt durch den Befehl DEVICE. Aus Kompatibilitätsgründen kann BITBANG weiterhin in Programmen verwendet werden und wird automatisch in DEVICE umgewandelt |
| DEVICE CAMERA | In den Befehl CAMERA geändert |
| DEVICE GAMEPAD | Wurde in den Befehl GAMEPAD geändert |
| DEVICE HUMID | Wurde in den Befehl HUMID geändert |
| DEVICE KEYPAD | Wurde in den Befehl KEYPAD geändert |
| DEVICE MOUSE | Wurde in den Befehl MOUSE geändert |
| DEVICE LCD | Wurde in den Befehl LCD geändert |
| DEVICE WII | Wurde in den Befehl WII geändert |
| DEVICE WS2812 | Wurde in den Befehl WS2812 geändert |
| GOSUB target | Löst einen Unterprogrammaufruf zum Ziel aus, das eine Zeilennummer oder ein Label sein kann. Das Unterprogramm muss mit RETURN enden.<br>Neue Programme sollten definierte Unterprogramme verwenden (d. h. SUB…END SUB). |
| IF condition THEN linenbr | Aus Kompatibilitätsgründen mit Microsoft wird ein GOTO angenommen, wenn auf die THEN-Anweisung eine Zahl folgt. Ein Label ist in dieser Konstruktion ungültig.<br>Neue Programme sollten Folgendes verwenden: IF Bedingung THEN GOTO Zeilennr \| Label |
| IRETURN | Kehrt aus einem Interrupt zurück, wenn das Interrupt-Ziel eine Zeilennummer oder ein Label war.<br>Neue Programme sollten eine benutzerdefinierte Subroutine als Interrupt-Ziel verwenden. In diesem Fall bewirkt END SUB oder EXIT SUB eine Rückkehr aus dem Interrupt. |
| ON nbr GOTO \| GOSUB target[,target, target,..] | ON verzweigt entweder (GOTO) oder ruft eine Subroutine auf (GOSUB), basierend auf dem gerundeten Wert von 'nbr'; wenn dieser 1 ist, wird das erste Ziel aufgerufen, bei 2 das zweite Ziel usw. Ziel kann eine Zeilennummer oder ein Label sein.<br>Neue Programme sollten SELECT CASE verwenden. |
| POS | Gibt für die Konsole die aktuelle Cursorposition in der Zeile in Zeichen zurück. |
| RETURN | RETURN beendet eine durch GOSUB aufgerufene Subroutine und kehrt zur Anweisung nach dem GOSUB zurück. |

# Anhang A
# Serielle Kommunikation

Für die asynchrone serielle Kommunikation stehen zwei serielle Schnittstellen zur Verfügung. Sie sind mit COM1: und COM2: gekennzeichnet.

## I/O-Pins

Bevor eine serielle Schnittstelle verwendet werden kann, müssen die I/O-Pins mit dem folgenden Befehl für den ersten Kanal (bezeichnet als COM1) definiert werden:

SETPIN rx, tx, COM1

Gültige Pins sind 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 sind RX: GP5, GP9 oder GP21

TX: GP4, GP8 oder GP20

TX sind Daten vom Raspberry Pi Pico und RX sind Daten an ihn.

Beachte, dass bei der WebMite-Version COM1 und COM2 auf GP20 bis GP28 nicht verfügbar sind

Die Signalpolarität entspricht dem Standard für Geräte, die mit TTL-Spannungen arbeiten. Im Ruhezustand ist die Spannung hoch, das Startbit ist niedrig, Daten verwenden eine hohe Spannung für Logik 1 und das Stoppbit ist hoch. Diese Signalpegel ermöglichen dir den direkten Anschluss an Geräte wie GPS-Module (die in der Regel TTL-Spannungspegel verwenden).

## Befehle

Nach dem Öffnen erhält der serielle Port eine Dateinummer, und du kannst alle Befehle verwenden, die mit einer Dateinummer arbeiten, um Daten von ihm zu lesen oder auf ihn zu schreiben. Ein serieller Port kann mit dem Befehl CLOSE geschlossen werden.

Hier ein Beispiel:

```
SETPIN GP13, GP16, COM1     ' Weise die I/O-Pins für den ersten seriellen Port zu
OPEN "COM1:4800" AS #5      ' den ersten seriellen Port mit einer Geschwindigkeit von 4800 Baud öffnen
PRINT #5, "Hallo"           ' sende die Zeichenkette „Hallo“ über die serielle Schnittstelle
dat$ = INPUT$(20, #5)       ' bis zu 20 Zeichen vom seriellen Port abrufen
CLOSE #5                    ' Schließe den seriellen Port
```

## Der Befehl OPEN

Ein serieller Port wird mit dem folgenden Befehl geöffnet:

```
OPEN comspec$ AS #fnbr
```

„fnbr“ ist die zu verwendende Dateinummer. Sie muss im Bereich von 1 bis 10 liegen. Das # ist optional.

„comspec$“ ist die Kommunikationsspezifikation und eine Zeichenkette (sie kann eine Zeichenkette-Variable sein), die den zu öffnenden seriellen Port und optionale Parameter angibt. Die Standardeinstellung ist 9600 Baud, 8 Datenbits, keine Parität und ein Stoppbit.

Die Form lautet `"COMn: baud, buf, int, int-trigger, EVEN, ODD, S2, 7BIT"` , wobei:

- ‘n’ die Nummer des seriellen Ports ist, entweder COM1: oder COM2:.
- ‘baud’ ist die Baudrate. Dies kann eine beliebige Zahl zwischen 1200 und 921600 sein. Der Standardwert ist 9600.
- „buf“ ist die Größe des Empfangspuffers in Byte (Standardgröße ist 256). Der Sendepuffer ist fest auf 256 Byte eingestellt.
- „int“ ist die Interrupt-Subroutine, die aufgerufen wird, wenn der serielle Port 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 ein Parameter weggelassen wird, müssen auch alle folgenden Parameter weggelassen werden, und es werden die Standardwerte verwendet.

Am Ende von „comspec$“ können fünf Optionen hinzugefügt werden. Diese sind:

- 'S2' gibt an, dass nach jedem übertragenen Zeichen zwei Stoppbits gesendet werden.
- EVEN gibt an, dass ein gerades Paritätsbit verwendet wird; dies führt zu einer 9-Bit-Übertragung, sofern 7BIT nicht gesetzt ist.
- ODD gibt an, dass ein ungerades Paritätsbit verwendet wird; dies führt zu einer 9-Bit-Übertragung, sofern nicht 7BIT gesetzt ist
- 7BIT gibt an, dass 7 Datenbits vorhanden sind. Dies wird normalerweise mit EVEN oder ODD verwendet
- INV gibt an, dass die Ausgangssignale invertiert werden und die Eingabe als invertiert angenommen wird

## Beispiele

Öffnen eines seriellen Ports mit allen Standardeinstellungen:

```
OPEN "COM1:" AS #2
```

Öffnen eines seriellen Ports unter Angabe der Baudrate (4800 Bit pro Sekunde):

```
OPEN "COM1:4800" AS #1
```

Öffnen eines seriellen Ports unter Angabe der Baudrate (9600 Bit pro Sekunde) und der Größe des Empfangspuffers (1 KB):

```
OPEN "COM2:9600, 1024" AS #8
```

Genau wie oben, aber mit zwei Stop-Bits:

```
OPEN "COM2:9600, 1024, S2" AS #8
```

Ein Beispiel, bei dem alles festgelegt wird, einschließlich eines Interrupts, einer Interrupt-Stufe und zweier Stoppbits:

```
OPEN "COM2:19200, 1024, ComIntLabel, 256, S2" AS #5
```

## Lesen und Schreiben

Sobald ein serieller Port geöffnet wurde, kannst du jeden Befehl oder jede Funktion verwenden, die eine Dateinummer nutzt, um vom Port zu lesen und auf ihn zu schreiben. Daten, die über den seriellen Port empfangen werden, werden von MMBasic automatisch im Speicher gepuffert, bis sie vom Programm gelesen werden, und die Funktion INPUT$() ist der bequemste Weg, dies zu tun. Bei Verwendung der Funktion INPUT$() entspricht die angegebene Zeichenanzahl der maximalen Anzahl der zurückgegebenen Zeichen, sie kann jedoch geringer sein, wenn sich weniger Zeichen im Empfangspuffer befinden. Tatsächlich gibt die Funktion INPUT$() sofort eine leere Zeichenkette zurück, wenn im Empfangspuffer keine Zeichen verfügbar sind.

Die Funktion LOC() ist ebenfalls praktisch; sie gibt die Anzahl der im Empfangspuffer wartenden Zeichen zurück (d. h. die maximale Anzahl an Zeichen, die mit der Funktion INPUT$() abgerufen werden können). Beachte, dass der serielle Port bei einem Überlauf des Empfangspuffers durch eingehende Daten automatisch die ältesten Daten verwirft, um Platz für die neuen Daten zu schaffen.

Der Befehl PRINT dient zur Ausgabe an eine serielle Schnittstelle, und alle zu sendenden Daten werden in einem Speicherpuffer gehalten, während die serielle Schnittstelle sie sendet. Das bedeutet, dass MMBasic nach dem Befehl PRINT mit der Ausführung der Befehle fortfährt, während die Daten übertragen werden. Die einzige Ausnahme ist, wenn der Ausgabepuffer voll ist; in diesem Fall hält MMBasic an und wartet, bis genügend Platz vorhanden ist, bevor es fortfährt. Die Funktion LOF() gibt die verbleibende Größe des Sendepuffers zurück, und du kannst dies nutzen, um zu vermeiden, dass das Programm ins Stocken gerät, während es darauf wartet, dass Platz im Puffer frei wird.

Wenn du sichergehen willst, dass alle Daten gesendet wurden (vielleicht, weil du die Antwort vom Remote-Gerät lesen möchtest), solltest du warten, bis die Funktion LOF() den Wert 256 (die Größe des Sendepuffers) zurückgibt, was bedeutet, dass nichts mehr zu senden ist.

Serielle Schnittstellen können mit dem Befehl CLOSE geschlossen werden. Dadurch wird gewartet, bis der Sendepuffer geleert ist, anschließend wird der von den Puffern belegte Speicher freigegeben und der Interrupt (falls gesetzt) deaktiviert. Eine serielle Schnittstelle wird auch automatisch geschlossen, wenn Befehle wie RUN und NEW ausgegeben werden.

## Interrupts

Die Interrupt-Subroutine (falls angegeben) funktioniert genauso wie ein allgemeiner Interrupt an einem externen E/A-Pin (siehe das Kapitel *„Verwendung der E/A-Pins“* für eine Beschreibung).

Bei der Verwendung von Interrupts musst du beachten, dass es einige Zeit dauert, bis MMBasic auf den Interrupt reagiert, und dass in der Zwischenzeit weitere Zeichen eingetroffen sein könnten, insbesondere bei hohen Baudraten.  Wenn du beispielsweise die Interrupt-Schwelle auf 200 Zeichen und einen Puffer von 256 Zeichen festgelegt hast, kann es leicht passieren, dass der Puffer überläuft, bevor die Interrupt-Subroutine die Daten lesen kann. In diesem Fall sollte der Puffer auf 512 Zeichen oder mehr erhöht werden.

# Anhang B
# I²C-Kommunikation

Es gibt zwei I² C-Kanäle. Sie können im Master- oder Slave-Modus betrieben werden.

## I/O-Pins

Bevor die I² C-Schnittstelle genutzt werden kann, müssen die I/O-Pins mit dem folgenden Befehl für den ersten Kanal (bezeichnet als I2C) definiert werden:

SETPIN sda, scl, I2C

Gültige Pins sind SDA: GP0, GP4, GP8, GP12, GP16, GP20 oder GP28

SCL: GP1, GP5, GP9, GP13, GP17 oder GP21

Beachte, dass bei der WebMite-Version I2C SDA nicht auf GP28 verfügbar ist

Und der folgende Befehl für den zweiten Kanal (bezeichnet als I2C2):

SETPIN sda, scl, I2C2

Gültige Pins sind SDA: GP2, GP6, GP10, GP14, GP18, GP22 oder GP26

SCL: GP3, GP7, GP11, GP15, GP19 oder GP27

Wenn der I²C-Bus mit mehr als 100 kHz betrieben wird, kommt es auf die Verkabelung zwischen den Geräten an. Idealerweise sollten die Kabel so kurz wie möglich sein (um die Kapazität zu reduzieren) und die Daten- und Taktleitungen sollten nicht nebeneinander verlaufen, sondern durch eine Erdungsleitung voneinander getrennt sein (um Übersprechen zu reduzieren).

Wenn die Datenleitung bei hohem Takt nicht stabil ist oder die Taktleitung schwankt, können die I²C-Peripheriegeräte „verwirrt“ werden und den Bus blockieren (normalerweise, indem sie die Taktleitung auf Low halten). Wenn du keine höheren Geschwindigkeiten benötigst, ist der Betrieb bei 100 kHz die sicherste Wahl.

Mit dem Befehl I2C CHECK addr kannst du prüfen, ob an der Adresse „addr“ ein Gerät vorhanden ist. Dadurch wird die schreibgeschützte Variable MM.I2C auf 0 gesetzt, wenn ein Gerät antwortet, oder auf 1, wenn keine Antwort erfolgt.

## I² C-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

| | |
|---|---|
| I2C OPEN speed, timeout | Aktiviert das I²C-Modul im Master-Modus. Der Befehl I2C bezieht sich auf Kanal 1, während der Befehl I2C2 sich auf Kanal 2 bezieht, wobei die gleiche Syntax verwendet wird.<br>„speed“ ist die zu verwendende Taktrate (in kHz) und muss entweder 100, 400 oder 1000 betragen.<br>„timeout“ ist ein Wert in Millisekunden, nach dessen Ablauf die Sende- und Empfangsbefehle des Masters unterbrochen werden, wenn sie noch nicht abgeschlossen sind. Der Mindestwert beträgt 100. Ein Wert von Null deaktiviert das Timeout (dies wird jedoch nicht empfohlen). |
| I2C WRITE addr, option, sendlen, senddata [,sendata ..] | Sende Daten an das I²C-Slave-Gerät. Der Befehl I2C bezieht sich auf Kanal 1, während der Befehl I2C2 sich unter Verwendung derselben Syntax auf Kanal 2 bezieht.<br>„addr“ ist die I2C-Adresse des Slaves.<br>„option“ kann 0 für den Normalbetrieb oder 1 sein, um nach dem Befehl die Kontrolle über den Bus zu behalten (nach Abschluss des Befehls wird kein Stoppbefehl gesendet)<br>„sendlen“ ist die Anzahl der zu sendenden Bytes.<br>„senddata“ sind die zu sendenden Daten – diese können auf verschiedene Arten angegeben werden (alle gesendeten Daten werden als Bytes mit einem Wert zwischen 0 und 255 gesendet): |

- Die Daten können als einzelne Bytes in der Befehlszeile angegeben werden. Beispiel: I2C WRITE &H6F, 0, 3, &H23, &H43, &H25
- Die Daten können in einem eindimensionalen Array liegen, das mit leeren Klammern angegeben wird (d. h. keine Dimensionen). „sendlen“ Bytes des Arrays werden beginnend mit dem ersten Element gesendet. Beispiel: I2C WRITE &H6F, 0, 3, ARRAY()
- Die Daten können eine Zeichenfolgenvariable sein (keine Konstante). Beispiel: I2C WRITE &H6F, 0, 3, STRING$

I2C READ addr, option, rcvlen, rcvbuf

Daten vom I²C-Slave-Gerät abrufen. Der Befehl I2C bezieht sich auf Kanal 1, während der Befehl I2C2 sich mit derselben Syntax auf Kanal 2 bezieht.

„addr“ ist die I²C-Adresse des Slaves.

„option“ kann 0 für den normalen Betrieb oder 1 sein, um nach dem Befehl die Kontrolle über den Bus zu behalten (nach Abschluss des Befehls wird kein Stopp-Signal gesendet)

„rcvlen“ ist die Anzahl der zu empfangenden Bytes.

„rcvbuf“ ist die Variable oder das Array, in dem die empfangenen Daten gespeichert werden – dies kann sein:

- Eine String-Variable. Die Bytes werden als aufeinanderfolgende Zeichen im String gespeichert.
- Ein eindimensionales Array aus Zahlen, angegeben mit leeren Klammern. Die empfangenen 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-I²-C-Modul und versetzt die I/O-Pins in einen „nicht konfigurierten“ Zustand. Dieser Befehl sendet außerdem ein Stopp-Signal, falls der Bus noch belegt ist.

## I²C-Slave-Befehle

I2C SLAVE OPEN addr, send_int, rcv_int

Aktiviert das I²C-Modul im Slave-Modus. Der Befehl I2C bezieht sich auf Kanal 1, während der Befehl I2C2 sich mit derselben Syntax auf Kanal 2 bezieht.

„addr“ ist die I²C-Slave-Adresse.

„send_int“ ist die Subroutine, die aufgerufen wird, wenn das Modul erkannt hat, dass der Master Daten erwartet.

„rcv_int“ ist die Subroutine, die aufgerufen wird, wenn das Modul Daten vom Master empfangen hat. Beachte, dass dies beim Empfang des ersten Bytes ausgelöst wird, sodass dein Programm möglicherweise warten muss, bis alle Daten empfangen wurden.

I2C SLAVE WRITE sendlen, senddata [,sendata ..]

Sende die Daten an den I²C-Master. Der Befehl I2C bezieht sich auf Kanal 1, während der Befehl I2C2 sich mit derselben Syntax 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 Interrupt-Subroutine ein Flag gesetzt werden und der Befehl aus der Hauptprogrammschleife aufgerufen werden, sobald 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 die I2C-WRITE-Befehle für Details.

| | |
|---|---|
| I2C SLAVE READ rcvlen, rcvbuf, rcvd | Empfange Daten vom I²C-Master-Gerät. Der Befehl I2C bezieht sich auf Kanal 1, während der Befehl I2C2 sich mit derselben Syntax auf Kanal 2 bezieht.<br>Dieser Befehl sollte im Empfangsinterrupt verwendet werden (d. h. in der Subroutine „rcv_int“, wenn der Master Daten gesendet hat). Alternativ kann in der Empfangsinterrupt-Subroutine ein Flag gesetzt werden und der Befehl aus der Hauptprogrammschleife aufgerufen werden, wenn das Flag gesetzt ist.<br>„rcvlen“ ist die maximale Anzahl der zu empfangenden Bytes.<br>„rcvbuf“ ist die Variable, in der die Daten empfangen werden. Dies kann auf verschiedene Arten angegeben werden, siehe die I2C-READ-Befehle für Details.<br>„rcvd“ ist eine Variable, die nach Abschluss des Befehls die tatsächliche Anzahl der empfangenen Bytes enthält (die von „rcvlen“ abweichen kann). |
| I2C SLAVE CLOSE | Deaktiviert das I²C-Slave-Modul und versetzt die externen I/O-Pins in einen „nicht konfigurierten“ Zustand. Sie können dann mit SETPIN konfiguriert werden. |

## Fehler

Nach einem I²C-Schreib- oder Lesevorgang wird die automatische Variable MM.I2C wie folgt gesetzt, um das Ergebnis anzuzeigen:

0 = Der Befehl wurde ohne Fehler ausgeführt.
1 = Eine NACK-Antwort wurde empfangen
2 = Zeitüberschreitung beim Befehl

## 7-Bit-Adressierung

Die in diesen Befehlen verwendeten Standardadressen sind 7-Bit-Adressen (ohne Lese-/Schreib-Bit). MMBasic fügt das Lese-/Schreib-Bit hinzu und behandelt es bei der Datenübertragung entsprechend.

Einige Hersteller bieten 8-Bit-Adressen an, die das Lese-/Schreib-Bit enthalten. Du kannst feststellen, ob dies der Fall ist, da sie eine Adresse zum Schreiben an das Slave-Gerät und eine andere zum Lesen vom Slave bereitstellen. In diesen Situationen solltest du nur die oberen sieben Bits der Adresse verwenden. Beispiel: Wenn die Leseadresse 9B (hex) und die Schreibadresse 9A (hex) ist, ergibt die Verwendung nur der oberen sieben Bits die Adresse 4D (hex).

Ein weiterer Hinweis darauf, dass ein Hersteller 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 deine Slave-Adresse größer als dieser Bereich ist, hat dein Hersteller wahrscheinlich eine 8-Bit-Adresse bereitgestellt.

## Beispiele

Als Beispiel für eine einfache Kommunikation, bei der der Raspberry Pi Pico der Master ist, liest das folgende Programm die aktuelle Uhrzeit (Stunden und Minuten) aus, die von einem PCF8563-Echtzeituhr-Chip verwaltet wird, der an den zweiten I2C-Kanal angeschlossen ist, und zeigt sie an:

```
DIM AS INTEGER RData(2)              ' hier werden die empfangenen Daten gespeichert
SETPIN GP6, GP7, I2C2                ' Weist die I/O-Pins für I2C2 zu
I2C2 OPEN 100, 1000                  ' den I2C-Kanal öffnen
I2C2 WRITE &H51, 0, 1, 3             ' das erste Register auf 3 setzen
I2C2 READ &H51, 0, 2, RData()        ' zwei Register lesen
I2C2 CLOSE                           ' den I2C-Kanal schließen
PRINT "Zeit ist " hex$(RData(1),2) ":" hex$(RData(0),2) 'zeige die BCD-
codierten Daten an
```

Dies ist ein Beispiel für die Kommunikation zwischen zwei Raspberry Pi Picos. Der Master sendet die Zeichen „A“ bis „Z“ und sendet nach jeder Übertragung eine Abfrage an den Slave und gibt die Antwort aus. Der Slave liest das vom Master gesendete Byte und gibt es aus. Als Antwort auf die Abfrage des Masters sendet er die aktuelle Uhrzeit.

Zuerst der Master:

```
SETPIN GP2, GP3, I2C2
I2C2 OPEN 100, 1000
FOR i = 65 to 90
  a$ = CHR$(i)
```

```
  I2C2 WRITE &H50, 0, 1, a$
  PAUSE 500
  I2C2 READ &H50, 0, 8, a$
  PRINT a$
  PAUSE 500
NECT i
```

Dann der Slave:

```
SETPIN GP2, GP3, I2C2
I2C2-SLAVE EIN &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 über eine einzige Signalleitung mit Chips zu kommunizieren. Diese Implementierung wurde von Gerard Sexton für MMBasic geschrieben.

Es gibt drei Befehle, die du verwenden kannst:

| | |
|---|---|
| ONEWIRE RESET pin | Setzt den 1-Wire-Bus zurück |
| ONEWIRE WRITE pin, flag, length, data [, data…] | Sende eine Anzahl von Bytes |
| ONEWIRE READ pin, flag, length, data [, data…] | Eine Anzahl von Bytes abrufen |

Wobei:

pin – Der zu verwendende I/O-Pin. Es kann jeder Pin sein, der für digitale Ein- und Ausgänge geeignet ist.

flag – Eine Kombination der folgenden Optionen:

1 – Reset vor dem Befehl senden

2 – Reset nach dem Befehl senden

4 – Nur ein Bit statt eines Datenbytes senden/empfangen

8 – Rufe nach dem Befehl einen starken Pullup auf (der Pin wird auf High gesetzt und Open Drain deaktiviert)

length – Länge der zu sendenden oder zu empfangenden Daten

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

Nach Ausführung des Befehls wird der I/O-Pin auf den nicht konfigurierten Zustand gesetzt, sofern nicht die Flag-Option 8 verwendet wird.

Wenn ein Reset angefordert wird, gibt die automatische Variable MM.ONEWIRE den Wert „true“ zurück, sofern ein Gerät gefunden wurde. Dies geschieht beim Befehl ONEWIRE RESET sowie bei den Befehlen ONEWIRE READ und ONEWIRE WRITE, wenn ein Reset angefordert wurde (Flag = 1 oder 2).

Das 1-Wire-Protokoll wird häufig für die Kommunikation mit dem Temperaturmesssensor DS18B20 verwendet. Um dies zu erleichtern, enthält MMBasic die Funktion TEMPR(), die eine bequeme Methode bietet, die Temperatur eines DS18B20 direkt abzulesen, ohne diese Funktionen nutzen zu müssen.

# Anhang D
# SPI-Kommunikation

Das Kommunikationsprotokoll „Serial Peripheral Interface“ (SPI) wird zum Senden und Empfangen von Daten zwischen integrierten Schaltkreisen verwendet. Der Raspberry Pi Pico fungiert als Master (d. h. er erzeugt den Takt).

## I/O-Pins

Bevor eine SPI-Schnittstelle genutzt werden kann, müssen die I/O-Pins für den Kanal mit den folgenden Befehlen zugewiesen werden. Für den ersten Kanal (als SPI bezeichnet) lautet der Befehl:

SETPIN rx, tx, clk, SPI

| | | |
|---|---|---|
| Gültige Pins 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

| | | |
|---|---|---|
| Gültige Pins sind | RX: | GP8, GP12 oder GP28 |
| | TX: | GP11, GP15 oder GP27 |
| | CLK: | GP10, GP14 oder GP26 |

TX sind Daten vom Raspberry Pi Pico und RX sind Daten an ihn.

Beachte, dass bei der WebMite-Version SPI1 und SPI2 auf GP20 bis GP28 nicht verfügbar sind

## SPI öffnen

Um die SPI-Funktion zu nutzen, muss der SPI-Kanal zuerst geöffnet werden.

Die Syntax zum Öffnen des ersten SPI-Kanals lautet (verwende SPI2 für den zweiten Kanal):

```
SPI OPEN speed, mode, bits
```

Wobei:

- „speed“ die gewünschte Geschwindigkeit für den SPI-Takt ist. Es kann ein beliebiger Wert angegeben werden; die Firmware wählt die nächsthöhere Geschwindigkeit aus, die erreicht werden kann und die gleich oder langsamer als die gewünschte Geschwindigkeit ist. Die tatsächlich eingestellte Geschwindigkeit ist CPUspeed/2 geteilt durch 1 bis 256.
- „mode“ ist eine einzelne Ziffer, die den Übertragungsmodus angibt – siehe Übertragungsformat weiter unten.
- „bits“ ist die Anzahl der zu sendenden/empfangenden Bits. Dies kann eine beliebige Zahl im Bereich von 4 bis 16 Bits sein.
- Es liegt in der Verantwortung des Programms, den CS-Pin (Chip Select) bei Bedarf separat zu steuern.

## Übertragungsformat

Das höchstwertige Bit wird zuerst gesendet und empfangen. Das Format der Übertragung kann durch den 'mode' wie unten gezeigt festgelegt werden. Modus 0 ist das gängigste Format.

| mode | Beschreibung | CPOL | CPHA |
|---|---|---|---|
| 0 | Der Takt ist aktiv hoch, Daten werden an der steigenden Flanke erfasst und an der fallenden Flanke ausgegeben | 0 | 0 |
| 1 | Der Takt ist aktiv hoch, Daten werden an der fallenden Flanke erfasst und an der steigenden Flanke ausgegeben | 0 | 1 |
| 2 | Der Takt ist Low-aktiv, Daten werden an der fallenden Flanke erfasst und an der steigenden Flanke ausgegeben | 1 | 0 |
| 3 | Der Takt ist aktiv-niedrig, Daten werden an der steigenden Flanke erfasst und an der fallenden Flanke ausgegeben | 1 | 1 |

Eine ausführlichere Erklärung findest du unter: http://en.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 (verwende SPI2 für den zweiten Kanal). Die Syntax lautet:

```
received_data = SPI(data_to_send)
```

Beachte, dass bei einer einzelnen SPI-Transaktion Daten gesendet und gleichzeitig Daten vom Slave empfangen werden. „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, eine Gleitkommazahl oder eine Konstante sein.

Wenn du keine Daten senden möchtest (d. h. du möchtest nur empfangen), kann eine beliebige Zahl (z. B. Null) für die zu sendenden Daten verwendet werden. Ebenso kannst du die empfangenen Daten einer Variablen zuweisen und ignorieren, wenn du sie nicht verwenden möchtest.

## Massen-Senden/Empfangen

Daten können auch in großen Mengen gesendet werden (verwende SPI2 für den zweiten Kanal):

```
SPI WRITE nbr, data1, data2, data3, … usw.
SPI WRITE nbr, string$
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. „data1“, „data2“ usw.). Die Daten können eine Ganzzahl, eine Gleitkommavariable oder eine Konstante sein.

Bei der zweiten oder dritten der oben aufgeführten Methoden sind die zu sendenden Daten in „string$“ oder im Inhalt von „array()“ enthalten (wobei es sich um ein eindimensionales Array aus Ganzzahlen oder Gleitkommazahlen handeln muss). Die Länge der Zeichenkette bzw. die Größe des Arrays muss mindestens so groß sein wie „nbr“. Alle vom Slave zurückgegebenen Daten werden verworfen.

Daten können auch gebündelt empfangen werden (verwende SPI2 für den zweiten Kanal):

```
SPI READ nbr, array()
```

Dabei ist „nbr“ die Anzahl der zu empfangenden Datenelemente und „array()“ ein eindimensionales Ganzzahl-Array, in dem die empfangenen Datenelemente gespeichert werden. Dieser Befehl sendet Nullen, während die Daten vom Slave gelesen werden.

## SPI Close

Bei Bedarf kann der erste SPI-Kanal wie folgt geschlossen werden (die I/O-Pins werden auf inaktiv gesetzt):

```
SPI CLOSE
```

Verwende SPI2 für den zweiten Kanal.

## Beispiele

Das folgende Beispiel zeigt, wie du den SPI-Port für allgemeine E/A nutzen kannst. Es sendet den Befehl 80 (hex) und empfängt zwei Bytes vom SPI-Slave-Gerät mithilfe der Standard-Sende-/Empfangsfunktion:

```
PIN(10) = 1 : SETPIN 10, DOUT      ' Pin 10 wird als Freigabesignal verwendet
SETPIN GP20, GP3, GP2, SPI         ' weise die I/O-Pins zu
SPI OPEN 5000000, 3, 8             ' Die Geschwindigkeit beträgt 5 MHz und die Datengröße 8 Bit
PIN(10) = 0                        ' die Freigabeleitung setzen (aktiv niedrig)
junk = SPI(&H80)                   ' den Befehl senden und die Rückmeldung ignorieren
byte1 = SPI(0)                     ' das erste Byte vom Slave abrufen
byte2 = SPI(0)                     ' das zweite Byte vom Slave abrufen
PIN(10) = 1                        ' denSlave deaktivieren
SPI CLOSE                          ' und schließe den Kanal
```

Das Folgende ähnelt dem oben genannten Beispiel, doch diesmal erfolgt die Übertragung mithilfe der Befehle zum Massen-Senden/Empfangen:

```
OPTION BASE 1                      ' Unser Array beginnt mit dem Index 1
DIM data%(2)                       ' definiere das Array zum Empfangen der Daten
SETPIN GP20, GP3, GP2, SPI         ' Weise die I/O-Pins zu
PIN(10) = 1 : SETPIN 10, DOUT      ' Pin 10 wird als Freigabesignal verwendet
SPI OPEN 5000000, 3, 8             ' Geschwindigkeit ist 5 MHz, 8 Bit Daten
PIN(10) = 0                        ' die Freigabeleitung aktivieren (aktiv bei Low)
```

```
SPI WRITE 1, &H80                ' Befehl senden
SPI READ 2, data%()              ' zwei Bytes vom Slave abrufen
PIN(10) = 1                      ' den Slave deaktivieren
SPI CLOSE                        ' und den Kanal schließen
```

# Anhang E
# Regex-Syntax

Die alternativen Formen der Funktionen INSTR() und LINSTR() können einen regulären Ausdruck als Suchmuster verwenden.

Die alternative Form der Funktionen lautet:

**INSTR([start],text$, search$ [,size])**

**LINSTR(text%(),search$ [,start] [,size]**

In beiden Fällen führt die Angabe des Parameters „size“ dazu, dass die Firmware die Suchzeichenfolge als regulären Ausdruck interpretiert. Der Parameter „size“ ist eine Gleitkomma- oder Ganzzahlvariable, die von der Firmware verwendet wird, um die Länge einer übereinstimmenden Zeichenfolge zurückzugeben. Wie in MMBasic implementiert, musst du die zurückgegebenen Werte ***für „start“*** und ***„size“*** auf die Funktion **MID$** anwenden, um die übereinstimmende Zeichenfolge zu extrahieren. Z. B.

**IF start THEN match$=MID$(text$,start,size) ELSE match$=”” ENDIF**

Die Syntax regulärer Ausdrücke kann je nach Implementierung leicht variieren. Dieser Anhang ist eine Zusammenfassung der Syntax und der unterstützten Operationen, die in der MMBasic-Implementierung verwendet werden.

Hinweis: Die Verwendung der Syntax für reguläre Ausdrücke mit OPTION ESCAPE sollte möglichst vermieden werden, da es sonst sehr leicht zu Verwechslungen kommen kann!

Anker

- ^ Anfang der Zeichenfolge
- $ Ende der Zeichenkette
- \b Wortgrenze
- \B Keine Wortgrenze

Qualifizierer

- * 0 oder mehr (nicht maskiert)
- + 1 oder mehr
- ? 0 oder 1
- {3} Genau 3
- {3,} 3 oder mehr
- {,5} 5 oder weniger
- {3,5\} 3, 4 oder 5

Gruppen und Bereiche

- (a|b|c) a oder b oder c
- (…) Gruppe
- [abc] Bereich (a oder b oder c)
- [^abc] Nicht (a oder b oder c)
- [a-q] Kleinbuchstaben a bis q
- [A-Q] Großbuchstaben A bis Q
- [0-7] Ziffern von 0 bis 7

Zeichenklassen

- \w Ziffern, Buchstaben und _
- \W Nicht-Alphabet
- \s Leerzeichen \t \f \r \n \v Leerzeichen
- \S kein Leerzeichen
- \d Ziffern
- \D keine Ziffern
- \xXX hexadezimal codiertes Byte

Erforderliche Escape-Zeichen zur Übereinstimmung mit normalen Zeichen

- \^ um ^ (Caret) abzugleichen
- \. um . (Punkt) zu ersetzen
- * um * (Sternchen) zu ersetzen
- \$ um $ (Dollarzeichen) zu ersetzen
- \[ entspricht [ (linke Klammer)
- \\ entspricht \ (Backslash)
- \? passt auf ? (Fragezeichen)
- \{ passt auf { (linke geschweifte Klammer)
- \} passt zu } (rechte geschweifte Klammer)
- \| passt auf | (Pipe)
- \( passt auf ( (linke Klammer)
- \) passt zu ) (rechte Klammer)
- \ + entspricht + (Pluszeichen)

Übereinstimmungsregeln

Nicht-Sonderzeichen passen auf sich selbst

. passt auf jedes Zeichen

Eine Wortgrenze befindet sich am Anfang oder Ende

einer Zeichenkette oder dort, wo ein \w-Zeichen ein

\W-Zeichen benachbart ist.

Einschränkungen

Anker innerhalb einer Gruppe werden nicht unterstützt.

z. B. passen (^hello) oder (hello$) nicht wie erwartet auf „hello“ am Anfang oder Ende der Zeile.

Anker außerhalb der Gruppe sind jedoch zulässig. Z. B. passen ^(hello) oder (hello)$

Beispielausdruck zum Abgleichen einer IP-Adresse.

```
"[\d]+\.[\d]+\.[\d]+\.[\d]+"
```

## Verwendung regulärer Ausdrücke mit OPTION ESCAPE

Wenn ein regulärer Ausdruck wörtlich in die INSTR- oder LINSTR-Funktion eingebettet ist, wird jede Syntax, die das Escape-Zeichen „\“ verwendet, korrekt verarbeitet, ohne dass der Backslash zusätzlich maskiert werden muss, unabhängig davon, ob OPTION ESCAPE verwendet wird oder nicht. z. B.

```
? instr("test123","\d",size)
```

Die Funktionen INSTR / LINSTR deaktivieren OPTION ESCAPE vorübergehend, während der Regex-Ausdruck gelesen wird.

Wenn der reguläre Ausdruck jedoch als Zeichenfolgenvariable übergeben wird und OPTION ESCAPE aktiviert ist, wenn du den regulären Ausdruck der Variablen zuweist, musst du jeden Backslash im regulären Ausdruck escapen. z. B.

```
s$=”\\d”
? instr("test123",s$,size)
```

# Anhang F
# Das PIO-Programmierpaket

## Einführung in PIO

Der RP2040 und der RP2350 verfügen über viele integrierte Peripheriegeräte wie PWM, UART, ADC und SPI. Mit PIOs lassen sich spezielle Funktionen/Peripheriegeräte wie serielle Hochgeschwindigkeits-Datenschnittstellen und Bitströme hinzufügen.

PIOs kann man sich als abgespeckte, hochspezialisierte CPU-Kerne vorstellen. Der RP2040 enthält zwei PIO-Blöcke, während der RP2350 über drei Blöcke verfügt. MMBasic bezeichnet sie entsprechend der Raspberry-Pi-Dokumentation als PIO0, PIO1 und PIO2. Die PIOs laufen völlig unabhängig vom Hauptsystem und voneinander und sind extrem schnell, mit einem Durchsatz von bis zu 32 Bit pro Taktzyklus.

PIOs implementieren Zustandsmaschinen. Bevor eine Zustandsmaschine ihr Programm ausführen kann, muss das Programm in den PIO-Speicher geschrieben und die Zustandsmaschine konfiguriert werden.

Dieser Anhang beschreibt, welche Unterstützung MMBasic bei der Verwendung von PIOs bieten kann. Er enthält keine Erklärung dazu, wie man PIO-Zustandsmaschinenprogramme schreibt. Um besser zu verstehen, wie diese funktionieren, schau dir den folgenden Thread „PIO explained PICOMITE“ im Forum auf thebackshed.com an: https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=15385

## Verfügbarkeit von PIOs

| PicoMite-Plattform | I2S-Audio | PIO0 | PIO1 | PIO2 |
|---|---|---|---|---|
| 2040 | | ✓ | ✓ | |
| 2040 | ja | X | ✓ | |
| 2350 | | ✓ | ✓ | ✓ |
| 2350 | ja | ✓ | ✓ | X |
| 2040 VGA | | X | ✓ | |
| 2040 VGA | ja | X | ✓ | |
| 2350 VGA | | X | ✓ | ✓ |
| 2350 VGA | ja | X | ✓ | X |
| 2350 HDMI | | ✓ | ✓ | ✓ |
| 2350 HDMI | ja | ✓ | ✓ | X |
| WebMite2040 | | ✓ | X | |
| WebMite2040 | ja | X | X | |
| WebMite2350 | | ✓ | X | ✓ |
| WebMite2350 | ja | ✓ | X | X |

✓ = VERFÜGBAR X = NICHT VERFÜGBAR

## Übersicht über PIO

Ein einzelner PIO-Block verfügt über vier unabhängige Zustandsmaschinen. Alle vier Zustandsmaschinen teilen sich einen einzigen Programmbereich im Flash-Speicher, der Platz für 32 Befehle bietet. Dieser Speicher ist vom Hauptsystem aus schreibgeschützt, verfügt jedoch über vier Leseanschlüsse – einen für jede Zustandsmaschine –, sodass jede unabhängig und mit ihrer eigenen Geschwindigkeit darauf zugreifen kann. Jede Zustandsmaschine hat ihren eigenen Programmzähler.

Jede Zustandsmaschine verfügt außerdem über zwei 32-Bit-„Scratchpad“-Register, X und Y, die als temporäre Datenspeicher genutzt werden können.

Der Zugriff auf die I/O-Pins erfolgt über ein Input/Output-Mapping-Modul, das auf 32 Pins zugreifen kann (beim RP2040 jedoch auf 30 begrenzt). Alle Zustandsmaschinen können unabhängig voneinander und gleichzeitig auf alle Pins zugreifen.

MMBasic kann Daten in das Eingangsende eines 4-Wort-TX-FIFO-Puffers mit 32 Bit Breite schreiben. Die Zustandsmaschine kann dann mit PULL das Ausgangswort des FIFO in das OSR (Output Shift Register) verschieben. Sie kann auch mit OUT jeweils 1–32 Bits aus dem OSR in das Ausgangs-Mapping-Modul oder andere Ziele verschieben. Mit AUTOPULL kannst du Daten automatisch abrufen, bis der TX-FIFO leer ist oder einen voreingestellten Füllstand erreicht.

MMBasic kann Daten vom Ausgangsende eines 4-Wort-RX-FIFO-Puffers mit 32 Bit Breite lesen. Die

Zustandsmaschine kann dann mit IN jeweils 1–32 Bits Daten aus dem Eingangs-Mapping-Modul in das ISR (Input Shift Register) verschieben. Anschließend kann sie mit PUSH den Inhalt des ISR in den FIFO verschieben. Mit AUTOPUSH kannst du Daten automatisch einfügen, bis der RX-FIFO voll ist oder einen voreingestellten Füllstand erreicht.

Die FIFO-Puffer können so umkonfiguriert werden, dass sie einen unidirektionalen 8-Wort-32-Bit-FIFO bilden. Die Puffer ermöglichen den Datenaustausch zwischen den Zustandsmaschinen, ohne dass das System oder die Zustandsmaschine aufeinander warten müssen.

Jeder der vier Zustandsmaschinen im PIO sind vier Register zugeordnet:

- CLKDIV ist der Taktteiler, der über einen 16-Bit-Ganzzahlteiler und einen 8-Bit-Bruchteilsteiler verfügt. Dieser legt fest, wie schnell die Zustandsmaschine läuft. Er teilt den Hauptsystemtakt herunter.
- EXECCTRL enthält Informationen zur Steuerung der Übersetzung und Ausführung des Programmspeichers
- SHIFTCTRL steuert die Anordnung und Nutzung der Schieberegister
- PINCTRL steuert, welche GPIO-Pins wie verwendet werden.

Die vier Zustandsmaschinen eines PIO haben gemeinsamen Zugriff auf seinen Block mit 8 Interrupt-Flags. Jede Zustandsmaschine kann jedes Flag verwenden. Sie können diese setzen, zurücksetzen oder auf ihre Änderung warten. Auf diese Weise können sie bei Bedarf synchron laufen. Die unteren vier Flags sind auch vom Hauptsystem aus zugänglich, sodass der PIO von MMBasic aus gesteuert werden oder Interrupts zurückgeben kann.

DMA kann verwendet werden, um Informationen über den FIFO des RP2040-Speichers an den PIO-Block zu übergeben und von diesem zu empfangen

Ein PIO verfügt über neun mögliche Programmierbefehle, aber es gibt viele Variationen für jeden einzelnen. Zum Beispiel kann „Mov“ bis zu 8 Quellen, 8 Ziele und 3 Verarbeitungsoperationen während des Kopiervorgangs haben, mit optionalen Verzögerungen und/oder Nebenoperationen!

- Jmp Springe zu einer absoluten Adresse im Programmspeicher, wenn eine Bedingung wahr ist (oder sofort).
- Wait Hält den Betrieb der Zustandsmaschine an, bis eine Bedingung wahr ist.
- In Verschiebe eine bestimmte Anzahl von Bits aus einer Quelle in den ISR.
- Out Verschiebe eine Anzahl von Bits aus dem OSR an ein Ziel.
- Push Schiebe den Inhalt des ISR als einzelnes 32-Bit-Wort in den RX-FIFO.
- Pull Lade ein 32-Bit-Wort aus dem TX-FIFO in den OSR.
- Mov Kopiere Daten von einer Quelle zu einem Ziel.
- Irq Setzt oder löscht ein Interrupt-Flag.
- Set Schreibe Daten sofort an ein Ziel.

Alle Befehle sind 16-Bit-Befehle und enthalten sowohl den Befehl selbst als auch alle damit verbundenen Daten. Alle Befehle werden in einem Taktzyklus ausgeführt, es ist jedoch möglich, zwischen einem Befehl und dem nächsten eine Verzögerung von mehreren Leerlauf-Taktzyklen einzufügen.

Zusätzlich gibt es eine Funktion namens „Side-Set“, die es ermöglicht, einen Wert an vordefinierte Ausgangspins zu schreiben, während ein Befehl aus dem Speicher gelesen wird. Dies ist für das Programm transparent.

## PIO programmieren

Die PicoMite-Firmware programmiert den Speicher der PIO-Zustandsmaschine mit einer von drei Methoden. Jede Methode wird anhand eines Beispiels für genau dasselbe Programm erklärt, das einen der GPIO-Pins des Raspberry Pi Pico umschaltet. Welcher GPIO-Pin umgeschaltet wird, wird durch die Konfiguration bestimmt, nicht im Programm selbst.

PIO ASSEMBLE

Dieser Befehl wird verwendet, um mit dem integrierten Assembler das Programm aus Mnemonics zu generieren und es dann direkt in den PIO-Speicher zu schreiben.

```
PIO ASSEMBLE 1,".program test"          'a program has to have a name
PIO ASSEMBLE 1,".line 0"                'start the program at line 0
PIO ASSEMBLE 1,"SET PINDIRS,1"          'SET the GPIO line to output
PIO ASSEMBLE 1,"label:"                 'define a label called "label"
PIO ASSEMBLE 1,"SET PINS,1"             'SET the GPIO pin high
PIO ASSEMBLE 1,"SET PINS,0"             'SET the GPIO pin low
```

```
PIO ASSEMBLE 1,"JMP label"              'JuMP to "label"
PIO ASSEMBLE 1,".end program list"      'end program, list=show result
```

Der Assembler ermöglicht auch ein besser lesbares Format wie dieses

```
PIO ASSEMBLE 1,".program test"    'a program has to have a name
.line 0                           'start the program at line 0
   SET PINDIRS,1                  'SET the GPIO line to output
.LABEL label:                     'define a label called "label"
   SET PINS,1                     'SET the GPIO pin high
   SET PINS,0                     'SET the GPIO pin low
   JMP label                      'JuMP to "label"
.end program list                 'end program, list=show result
```

### PIO-PROGRAMMZEILE

Mit diesem Befehl kannst du 16-Bit-Werte in einzelne Zeilen (Speicherplätze) des PIO-Speichers programmieren.

```
PIO PROGRAM LINE 1,0,&hE081             'SET pin output
PIO PROGRAM LINE 1,1,&hE001             'SET pin high
PIO PROGRAM LINE 1,2,&hE000             'SET pin low
PIO PROGRAM LINE 1,3,&h0001             'JMP to line 1
```

### PIO-PROGRAMM

Dieser Befehl schreibt alle 32 Zeilen eines PIO aus einem Array. Das ist nützlich, sobald ein PIO-Programm debuggt wurde. Es ist extrem kompakt.

```
DIM a%(7)=(&h0001E0000E001E081,0,0,0,0,0,0,0)
PIO PROGRAM 1,a%()
```

## PIO konfigurieren

Die PicoMite-Firmware kann jede Zustandsmaschine individuell konfigurieren. Die Konfiguration ermöglicht es, dass zwei Zustandsmaschinen genau dieselben Programmzeilen ausführen (z. B. eine SPI-Schnittstelle), aber mit unterschiedlichen GPIO-Pins und bei unterschiedlichen Geschwindigkeiten arbeiten. Es gibt mehrere Konfigurationsfelder.

### FREQUENZ

Die PicoMite-Firmware enthält für jedes Konfigurationsfeld eine Standardkonfiguration, mit Ausnahme der Frequenz. Die Frequenz wird durch einen 16-Bit-CLKDIV-Teiler vom ARM-Takt festgelegt. Beispiel: Wenn OPTION CPUSPEED 126000 eingestellt ist, kann der PIO mit Geschwindigkeiten zwischen 126 MHz und 1,922 kHz laufen (126000000 / 65536). Beachte, dass höhere CPU-Geschwindigkeiten (Übertaktung) sich direkt auf die Frequenz der Zustandsmaschine auswirken.

### PIN-STEUERUNG

Die PicoMite-Firmware legt die GPIO-Pins standardmäßig für die Verwendung durch MMBasic fest. Damit der PIO die Kontrolle über einen GPIO-Pin übernimmt, muss MMBasic diesen wie unten beschrieben dem PIO zuweisen.

SETPIN GPxx,PIOx (z. B. SETPIN gp0,pio1)

Eine Zustandsmaschine kann den Zustand eines Pins setzen (SET ist ein Befehl der Zustandsmaschine), aber auch serielle Daten an einen oder mehrere GPIO-Pins ausgeben, indem sie den Befehl OUT verwendet. Oder sie kann serielle Daten mit dem Befehl IN lesen. Und GPIO-Pins können als Nebeneffekt eines beliebigen Befehls der Zustandsmaschine gesetzt werden (SIDE SET). Für jede Schnittstellenmethode können unterschiedliche Pins der Zustandsmaschine zugeordnet werden.

Wichtig zu verstehen ist, dass diese Befehle auf aufeinanderfolgende Pins wirken. Das bedeutet, dass es einen Bereich von Pins gibt, die gesteuert werden können, beginnend mit der niedrigsten GPx-Pin-Nummer (z. B. GP0), und dass benachbarte Pins einbezogen werden können (insgesamt bis zu 5 Pins). GP0, GP1, GP2 ist also ein gültiger Satz von IO-Pins. GP0, GP1, GP6 hingegen nicht. Berücksichtige dies beim Entwerfen einer PIO-Anwendung.

Die Zuweisung von GPIO-Pins zu einer Zustandsmaschine erfolgt über die PIO-Hilfsfunktion PINCTRL:

PIO(PINCTRL a,b,c,d,e,f,g)
a/ die Anzahl der SIDE-SET-Pins (0...5), SIDE-SET kann 5 Pins gleichzeitig beschreiben
b/ die Anzahl der SET-Pins (0...5), SET kann 5 Pins gleichzeitig beschreiben

c/ die Anzahl der OUT-Pins (0...31), OUT kann 32 Pins gleichzeitig beschreiben
d/ der niedrigste Pin für IN-Pins (GP0.....GP31) IN kann bis zu 32 Pins gleichzeitig lesen
e/ der niedrigste Pin für SIDE SET (GP0.....GP31)
f/ der niedrigste Pin für SET (GP0.....GP31)
g/ der niedrigste Pin für OUT (GP0.....GP31)

Die Bereiche für die verschiedenen Funktionen (SET/OUT/SIDESET/IN) können sich überschneiden, identisch sein oder nebeneinander liegen.

## AUSFÜHRUNGSSTEUERUNG

Das Ausführungssteuerungsregister EXECCTRL konfiguriert den Programmablauf. Es gibt ein Feld, das einen GPIO-Pin mit einem bedingten Sprung (JMP-Befehl) verbindet, sowie Felder, die die Zeilenadresse des Anfangs (.WRAP TARGET) und des Endes (.WRAP) der Hauptprogrammschleife enthalten.

Wenn wir wollen, dass sich der Programmablauf als Reaktion auf den Zustand eines GPIO-Pins ändert, wird ein JMP PIN verwendet. Der spezifische Pin wird in der Ausführungssteuerungskonfiguration zugewiesen (es kann nur 1 Pin pro Zustandsmaschine geben) und der JMP findet nur statt, wenn der Pin auf High ist.

Das Zustandsmaschinenprogramm beginnt am Anfang und läuft, bis es das Ende erreicht. Im obigen Demoprogramm durchläuft das Programm „ “ mithilfe eines (bedingungslosen) JMP-Befehls eine Schleife vom Ende zum Anfang. Eine Alternative zur Verwendung des JMP-Befehls besteht darin, den Anfang der Schleife (WRAP TARGET = Zeile 1) und das Ende der Schleife (WRAP = Zeile 2) zu definieren und die Zustandsmaschine so zu konfigurieren, dass sie nur die dazwischen liegenden Befehle ausführt. Die JMP-Anweisung in Zeile 3 ist überflüssig, wenn WRAP/WRAP TARGET verwendet wird.

PIO(EXECCTRL a,b,c)
a/ der GPIO-Pin für bedingtes JMP (z. B. GP0)
b/ die WRAP-TARGET-Zeilennummer (z. B. 1)
c/ die WRAP-Zeilennummer (z. B. 2)

## SHIFT CONTROL

Die Befehle IN und OUT verschieben Daten vom FIFO-Register zu den GPIO-Pins. Zwischen MMBasic und dem PIO können 32-Bit-Wörter ausgetauscht werden. Da sowohl die ARM-Kerne als auch die PIO-Mikrocontroller unabhängig voneinander arbeiten, werden die Daten über FIFOs ausgetauscht. Der ARM (MMBasic) legt Daten im FIFO ab, der PIO liest sie aus. Dabei wird das TX-FIFO verwendet. In umgekehrter Richtung wird das RX-FIFO verwendet. Die FIFOs sind normalerweise 4 Wörter tief, können aber auch als einzelnes 8-Wort-tiefes RX- oder TX-FIFO konfiguriert werden.

Der PIO kann Daten im RX-FIFO von der MSB-Seite oder von der LSB-Seite „verschieben“. Das wird mit dem IN SHIFTDIR-Bit eingestellt. Ähnlich verhält es sich mit dem OUT SHIFTDIR-Bit für OUT-Daten. Die Autopull- und Autopush-Flags bestimmen in Kombination mit den Pull- und Push-Schwellenwerten, wann das FIFO aufgefüllt wird.

Im RP2350 können die FIFOs auch als einzelne Register verwendet werden, was eine flexiblere Kommunikation zwischen MMBasic und den Zustandsmaschinen ermöglicht. Dies wird durch die Befehle FJOIN_RX_GET und FJOIN_RX_PUT im SHIFTCTRL-Register erreicht.

PIO(SHIFTCTRL a,b,c,d,e,f,g,h)
a/ push threshold (1..31 Bits, die im ISR vor autoPUSH eingeschoben werden)
b/ pull threshold (1..31 Bits werden vor autoPULL aus dem OSR verschoben)
c/ autopush (1 = automatisches Push zulassen)
d/ autopull (1 = automatisches Pull zulassen)
e/ IN-shiftdir (1 = MSB verschieben, 0 = LSB verschieben)
f/ OUT-shiftdir (1 = MSB verschieben, 0 = LSB verschieben)
g/ fjoin_tx (TX- und RX-FIFO zu 1 RX-FIFO zusammenführen)
h/ fjoin_rx (TX- und RX-FIFO zu 1 TX-FIFO zusammenführen)
i/ fjoin_rx_get (1=ARM schreibt einzelne RX-FIFO-Register, nur 2350)
j/ fjoin_rx_put (1=ARM liest einzelne RX-FIFO-Register, nur 2350)

## SCHREIBEN DER ZUSTANDSMASCHINENKONFIGURATION

Eine Zustandsmaschinenkonfiguration wird mit dem Befehl geschrieben:

PIO INIT MACHINE a,b,c,d,e,f,g
a/ der PIO (0, 1 oder 2 (nur 2350))
b/ die Nummer der Zustandsmaschine (0...3)
c/ Frequenz (CPUSPEED/65536...CPUSPEED in Hz)
d/ Pin-Steuerwert (PIO(PINCTRL .....))
e/ Ausführungssteuerungswert (PIO(EXECCTRL.....))

f/ Schiebesteuerungswert (PIO(SHIFCTRL.....))
g/ Startadresse (0....31, die Zeile, an der die Zustandsmaschine mit der Ausführung beginnt, kann ein Label sein)

## SCHREIBEN DER ZUSTANDSMASCHINENKONFIGURATION IN KOMPAKTER FORM

Eine Zustandsmaschinenkonfiguration wird mit diesem einzigen Befehl konfiguriert und geschrieben:

PIO CONFIGURE a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,aa,bb,cc

| | |
|---|---|
| a/ pio | (0, 1 oder 2 (nur 2350)) |
| b/ sm | (0...3) |
| c/ frequency | (CPUSPEED/65535 bis zu CPUSPEED in Hz) |
| d/ startaddress | (0..31) |
| e/ sidesetbase | (SIDE SET-Bereich beginnt bei GPx) |
| f/ sidesetno | (0..5...Auswirkung auf den DELAY-Bereich) |
| g/ sidesetout | (1 = die oben definierten Pins automatisch als Ausgang nutzen) |
| h/ setbase | (SET-Bereich beginnt bei GPy) |
| i/ setno | (0...5) |
| j/ setout | (1=die oben definierten Pins automatisch als Ausgangsanschlüsse festlegen) |
| k/ outbase | (OUT/MOV-Bereich beginnt bei GPz) |
| l/ outno | (0...5) |
| m/ outout | (1=die oben definierten Pins automatisch als Ausgangs-Pins festlegen) |
| n/ inbase | (IN-Bereich beginnt bei GPxx) |
| o/ jmppin | (Bedingter JMP-Pin ist GPyy) |
| p/ wraptarget | (0..31 = Zeilennummer für WRAP TARGET) |
| q/ wrap | (0..31 = Zeilennummer für WRAP) |
| r/ sideenable | (1 = Pins nur explizit aktualisieren, wenn „side set“ im Befehl enthalten ist) |
| s/ sidepindir | (1 = „side set“ ändert die Pin-Richtung, nicht den Pin-Zustand) |
| t/ pushthreshold | (1..31 Bits werden vor autoPUSH im ISR eingeschoben) |
| u/ pullthreshold | (1..31 Bits werden vor autoPULL aus dem OSR herausgeschoben) |
| v/ autopush | (1 = automatisches Push zulassen) |
| w/ autopull | (1 = automatisches Pull zulassen) |
| x/ inshiftdir | (1 = Verschiebung MSB→LSB, 0 = Verschiebung LSB→MSB) |
| y/ outshiftdir | (1 = Verschiebung MSB→LSB, 0 = Verschiebung LSB→MSB) |
| z/ joinrxfifo | (1 = RX-FIFO hat 8 Einträge, TX-FIFO = 0 |
| aa/ jointxfifo | (1 = TX-FIFO hat 8 Einträge, RX-FIFO = 0 |
| bb/ joinrxfifoget | (1 = RX-FIFO wechselt zu 4 lesbaren Registern (nur 2350) |
| cc/ joinrxfifoput | (1 = RX-FIFO wechselt zu 4 beschreibbaren Registern (nur 2350) |

## STARTEN UND STOPPEN EINER ZUSTANDSMASCHINE

Sobald der PIO konfiguriert ist, kannst du die Zustandsmaschine wie folgt starten und stoppen:

PIO START a,b
PIO STOP a,b

| | |
|---|---|
| a/ die PIO-Nummer | (0, 1 oder 2 (nur 2350)) |
| b/ die Zustandsmaschine | (0...3) |

Beachte, dass eine Zustandsmaschine beim Anhalten genau dort stoppt, wo sie gerade ist. Um die Zustandsmaschine neu zu starten, ist es ratsam, zuerst PIO INIT MACHINE auszuführen.

## BEISPIELPROGRAMM 1

Eine vollständige PIO-Implementierung, die einen GPIO-Pin umschaltet, lässt sich in MMBasic wie unten gezeigt umsetzen. Schließe einen Summer an GP0 an und höre dir den vom PIO erzeugten Ton an.

```
'disconnect ARM from GP0
setpin gp0,pio1                  'use GP0 as output pin for PIO 1

'pio program used
'0 E081   'SET pin output
'1 E001   'SET pin high
'2 E000   'SET pin low
'3 0001   'jmp 1

'program pio 1 using an array to write the program in PIO memory, and start
Dim a%(7)=(&h0001E000E001E081,0,0,0,0,0,0,0)
```

```
PIO program 1,a%()

'configure pio 1 statemachine 0
p=Pio(pinctrl 0,1,,,,gp0,)    'define SET uses 1 pin, and that is GP0
f=3100                        '3051 Hz is lowest frequency CPUSPEED 200000
PIO init machine 1,0,f,p      'use default for execctrl, shiftctrl, start
                                   'address(=0)

'start the PIO 1 state machine 0
PIO start 1,0
```

Beachte, dass das MMBasic-Programm endet, der Ton des Summers aber weiterläuft. PIO ist unabhängig von der ARM-CPU und läuft weiter, bis es gestoppt wird. Wenn du den MMBasic-Editor öffnest, wird PIO gestoppt.

## FIFOs

MMBasic und der PIO tauschen Informationen über FIFOs aus. Der PIO schiebt Daten in das RX-FIFO (MMBasic ist der Empfänger) oder zieht Daten aus dem TX-FIFO (MMBasic ist der Sender).

Wenn der PIO Daten aus dem FIFO abruft, werden diese an das OSR (Output Shift Register) übertragen, von wo aus sie verarbeitet werden können. Der PIO kann die Daten aus dem ISR (Input Shift Register) in das FIFO schieben. Außerdem verfügt der PIO über zwei Register, X und Y, die zum Speichern oder Zählen verwendet werden können. Der PIO kann nicht addieren, subtrahieren oder vergleichen.

Datenfluss:

MMBasic -> FIFO -> OSR -> PIO (oder pins)
PIO (oder pins) -> ISR -> FIFO -> MMBasic

MMBasic kann Daten in das TX-FIFO schreiben und Daten aus dem RX-FIFO lesen mit:

PIO READ a,b,c,d
PIO WRITE a,b,c,d

| | |
|---|---|
| a/ PIO-Nummer | (0, 1 oder 2 (nur 2350)) |
| b/ Zustandsmaschinennummer | (0...3) |
| c/ Anzahl der 32-Bit-Wörter | (1...4) |
| d/ Name der Integer-Variablen | (z. B. Variable% oder Array%()) |

PIO CLEAR löscht alle PIO-FIFOs, ebenso wie PIO START und PIO INIT MACHINE.

Das MMBasic-Programm muss nicht darauf warten, dass Daten im FIFO erscheinen, da dem RX-FIFO ein Interrupt zugewiesen werden kann. Die MMBasic-Interrupt-Routine kann die Daten aus dem FIFO abrufen.

Ähnlich verhält es sich mit dem TX-Interrupt: In diesem Fall erhält MMBasic einen Interrupt, wenn Daten für das TX-FIFO benötigt werden.

PIO INTERRUPT a,b,c,d

| | |
|---|---|
| a/ PIO | (0, 1 oder 2 (nur 2350)) |
| b/ Zustandsmaschine | (0...3) |
| c/ Name des RX-Interrupt-Handlers | (z. B. „myRX_Interrupt“ oder 0 zum Deaktivieren) |
| d/ Name des TX-Interrupt-Handlers | (z. B. „myTX_Interrupt“ oder 0 zum Deaktivieren) |

BEISPIELPROGRAMM 2

Das folgende Programm erläutert viele der oben vorgestellten MMbasic-Funktionen und -Befehle. Das Programm liest einen an den Raspberry Pi Pico angeschlossenen NES-Controller (SPI) aus. Der NES-Controller besteht aus einem HEF4021-Schieberegister, das mit 8 Drucktastern verbunden ist.

Das Programm verwendet: **wrap** und **wrap target**, IN, side set und delay, PUSH, PIO READ. GP0 und GP1 sind in SET für die Pin-Richtung und in side set für kompakten Code.

Die Verdrahtung ist wie im Code definiert:

```
'disconnect ARM from GP0/1/2
  setpin gp0,pio1                'clock out
  setpin gp1,pio1                'load out
  setpin gp2,pio1                'data in

'PIO program
```

```
PIO assemble 1,".program NES"     'a program needs a name
.side set 2                       'use 2 bits for side set, 3 for delay
.line 0                           'start code at line 0
   SET pindirs,&b11               'set GP0,GP1 output, side GP0,GP1 low
.wrap target                      'wrap target = top of the loop
   IN null,32 side 2              'set ISR to 0, GP1 high (load), GP0 low
   SET X,7 side 0                 'set X counter to 7, GP0,GP1 low
.label loop:                      'inner loop label
   IN pins,1 side 0               'shift 1 databit in, keep GP0,GP1 low
   JMP X-- loop side 1            'jmp to loop, dec. X, GP0 'high(clock)
   PUSH side 0 [7]                'now X=0, PUSH result into FIFO, delay 7
.wrap"                            'end outer loop, repeat
.end program list                 'end of program, list result

'configure pio1
  p=Pio(pinctrl 2,2,,gp2,gp0,gp0,)'GP0,GP1 out (SET and SIDESET), GP2 IN
  f=1e5                           '100kHz
  s=PIO(shiftctrl 0,0,0,0,0,0)    'shift in from LSB for IN (and OUT)
  e=PIO(execctrl gp0,PIO(.wrap target),PIO(.wrap))  'wrap and wrap target

'write the configuration
  PIO init machine 1,0,f,p,e,s,0  'start at line 0

'start the pio1 code
  PIO start 1,0


'Check the the read data in MMBasic and print
  dim d%
  do
    pio read 1,0,1,d%
    print bin$(d%)
    pause 200
  loop
  END
```

## DMA zu und von den FIFOs

So funktioniert DMA:

Beim Lesen aus dem FIFO wartet der DMA-Controller darauf, dass Daten im FIFO vorhanden sind, und sobald diese erscheinen, überträgt er sie in den Prozessorspeicher. Bei jedem Lesevorgang erhöht er den Zeiger im Prozessorspeicher, sodass er beispielsweise ein Array schrittweise füllen kann, sobald jedes einzelne Datenelement verfügbar wird.

Beim Schreiben in den FIFO schreibt der DMA-Controller Daten automatisch aus dem Prozessorspeicher in den FIFO und wartet, wenn der FIFO voll ist. So können Daten in einem Array vorbereitet werden, und der DMA-Controller überträgt diese Daten so schnell an den PIO-FIFO, wie es das PIO-Programm benötigt.

DMA kann ein 32-Bit-Wort, ein 16-Bit-Short oder ein 8-Bit-Byte übertragen, und beim Einrichten von DMA musst du die Größe der Übertragung und die Anzahl der durchzuführenden Übertragungen angeben. Da jeder Transfer den Speicherzeiger um 1, 2 oder 4 Bytes erhöht, muss MMBasic mit den im Speicher gepackten Daten arbeiten und nicht mit den 64 Bits, die für MMbasic-Ganzzahlen und Gleitkommazahlen verwendet werden. Glücklicherweise implementiert MMBasic zwei Befehle, MEMORY PACK und MEMORY UNPACK, um dies sehr effizient zu erledigen, aber es könnte genauso gut mit Standard-BASIC-Arithmetik gemacht werden.

Der DMA kann so konfiguriert werden, dass er Daten wiederholt in einen Speicherbereich (einen Ringpuffer) hinein- oder herausschleift

Die Befehle lauten:

PIO DMA RX a, b, c, d, e, f, g
PIO DMA TX a, b, c, d, e, f, g

| | | |
|---|---|---|
| Wobei: | a = pio | (0, 1 oder 2 (nur 2350)) |
| | b = Zustandsmaschine | (0...3) |
| | c = nbr | (Anzahl der zu übertragenden Wörter) |
| | d = data%() | (Name des Integer-Arrays) |

e = completioninterrupt (Wohin nach Abschluss, optional)
f = Übertragungsgröße (8 = 16 = 32, optional)
g = loopbackcount (verwendet data%() als Ringpuffer, optional, loopbackcount = 2^n)

Der DMA startet die Zustandsmaschine automatisch, sodass kein PIO-START-Befehl erforderlich ist. Bevor du jedoch mit der Übertragung beginnst, solltest du sicherstellen, dass ein neuer PIO-INIT-MACHINE-Befehl ausgeführt wurde, damit die Zustandsmaschine an der gewünschten Startadresse beginnt.

Wenn ein Ringpuffer verwendet wird, sind besondere Vorbereitungen erforderlich:

PIO MAKE RING BUFFER a, b
Wobei: a = Name des Integer-Puffers
b = Größe des Arrays in Byte

Beispiel:

```
DIM packed%                                 'Achtung: KEINE Klammern()
PIO MAKE RING BUFFER packed%,4096
```

packed% ist dann ein Integer-Array, das 4096/8 = 512 Integer enthält

Dies kann dann vom DMA für einen Loopback-Zähler mit 1024 32-Bit-Wörtern, 2048 16-Bit-Shorts oder 4096 8-Bit-Bytes verwendet werden.

## BEISPIELPROGRAMM 3

Dieses Programm führt alles zusammen und nutzt DMA, um 128 Samples aus dem PIO-RX-FIFO zu lesen. Zur Demonstration sind GP0 bis GP5 Ausgänge von 3 PWMs und werden gleichzeitig vom PIO als 6-Kanal-Logikanalysator oder Oszilloskop abgetastet. Die 128 Samples werden als Wellenformen an den seriellen Port gesendet.

Dieses Logikanalysator-Programm demonstriert außerdem PIO-DMA-RX, MEMORY-UNPACK und die Verwendung von Puffern. Es nutzt PWMs, um zu Demonstrationszwecken ein Testsignal an gp0..gp5 zu erzeugen. Dieselben Pins werden vom Logikanalysator ausgelesen und an die Konsole ausgegeben.

Um diesen Logikanalysator normal zu verwenden, kommentiere die ersten 14 Zeilen aus.

```
'generate a 50Hz 3 phase test signal to demonstrate the DMA on 6 GPIO pins.
SetPin gp0,pwm  'CH 0a
SetPin gp1,pwm  'CH 0b
SetPin gp2,pwm  'CH 1a
SetPin gp3,pwm  'CH 1b
SetPin gp4,pwm  'CH 2a
SetPin gp5,pwm  'CH 2b

Fpwm = 50: PW = 100 / 3
PWM 0, Fpwm, PW, PW - 100, 1, 1
PWM 1, Fpwm, PW, PW - 100, 1, 1
PWM 2, Fpwm, PW, PW - 100, 1, 1
PWM sync 0, 100/3, 200/3

'------------------------------ LA code PIO -------------------------
'PIO code to sample GP0..GP6 as elementary logic analyser
PIO clear 1

'in this program the PIO reads GP0..GP5 brute force
'and pushes data into FIFO. The clock speed determines the
'sampling rate. There are 2 instructions per cycle
'taking 10000/2 / 50 = 100 samples per 50Hz cycle.

PIO assemble 1,".program push"
   .line 0
   .wrap target
   IN pins,6                'get 6 bits from GP0..GP5
   PUSH block               'push data when FIFO has room
   .wrap
.end program list

'configuration
f=1e4                                   'PIO run at 10kHz
```

```
p=Pio(pinctrl 0,0,0,gp0,,,)             'IN base = GP0
e=Pio(execctrl gp0,PIO(.wrap target),PIO(.wrap)) 'wrap 1 to 0, gp0 is
                                                 'default
s=Pio(shiftctrl 0,0,0,0,0,0)      'shift in through LSB, out is not used

'write the configuration, speed 10kHz (data in FIFO 10us after edge GP0)
PIO init machine 1,0,f,p,e,s,0          'start address = 0

'------------------------- LA code MMBasic ----------------------------
'define memory buffers
Dim a$(1)=("_","-")                     'characters for the printout
length%=64                              'size of the packed array
Dim data%(2*length%-1)                  'array to put the 32 bit samples
                                        'FIFO format
Dim packed%(length%-1)                  'DMA array to pack 32 bit samples 'in 64
                                       bit integers

'let the DMA machine run, and repeat at will
Do
  PIO DMA RX 1,0,2*length%,packed%(),ReadyInt
  print "press any key to restart sampling"
  do:loop while inkey$=""
Loop
End

'--------------------------------SUBS MMBasic -------------------------
Sub ReadyInt
'stop the PIO and re-init for next run
PIO stop 1,0
PIO init machine 1,0,f,p,e,s,0          'start address = 0

'get the data from the packed DMA buffer and unpack to original 32 bit 'format
Memory unpack packed%(),data%(),2*length%,32

'Serial output as if logic analyzer traces
For j=0 To 5
  mask%=2^j
  For i=0 To 2*length%-1
    If i<106 Then Print a$(((data%(i) And mask%)=mask%));
  Next i
  Print : Print
Next j
End Sub
```

## BEISPIELPROGRAMM 4

Dieses Programm läuft nur auf dem RP2350 und demonstriert die Verwendung der FIFOs als einzelne Register. Der PIO des RP2350 verfügt über spezielle Befehle zur Unterstützung dieser Funktion: MOV RXFIFO[y],ISR und MOV OSR,RXFIFO[y].

MMBasic kann die 4 einzelnen FIFO-Register mit folgenden Befehlen lesen/schreiben:

PIO WRITEFIFO a, b, c, d
PIO READFIFO( a, b, c)

| | |
|---|---|
| a/ pio | (0, 1 oder 2 (nur 2350)) |
| b/ state machine | (0...3) |
| c/ nbr. | (FIFO-Register 0…3) |
| d/ data% | (32-Bit-Ganzzahlwert) |

Das Programm konfiguriert das FIFO für einzelne Lesevorgänge und schreibt dann einige Werte in diese Register. Es startet den PIO, der 2 der 4 einzelnen FIFO-Register aktualisiert. Dann liest MMBasic die Werte aus, um zu zeigen, dass sich nur 2 Register geändert haben und keine Daten verschoben wurden (wie es beim RP2040-FIFO der Fall wäre).

```
'only for RP2350 assembler. this does not work on RP2040

pio clear 1

pio assemble 1,".program test"
.line 0
set y,4                 'just a value 4 in Y
mov isr,y               'copy Y to isr
jmp y--,next            'y=y-1 always to next
.label next             'label
mov rxfifo[y],isr       'should program 4 in fifo [3]
mov isr,y               'copy Y to isr
mov rxfifo[2],isr       'should program 3 in fifo [2]
jmp 0                   'repeat
.end program

f=1e6 '1MHz

'PIO(EXECCTRL a,b,c)
e=pio(execctrl gp0,0,31)            'default value, not actually changed

'PIO(SHIFTCTRL a,b,c,d,e,f,g,h,i,j)
sr=pio(shiftctrl 0,0,0,0,0,0,0,0,0,1) 'read individual RX, RX=4 deep
sw=pio(shiftctrl 0,0,0,0,0,0,0,0,1,0) 'write individual RX, RX=4 deep

'PIO(PINCTRL a,b,c,d,e,f,g)
p=pio(pinctrl 0,0,0,gp0,gp0,gp0,gp0) 'defaultvalue , not actually changed

'test the use of FIFO as individual registers
'fill the fifo with pre-determined values
pio init machine 1,0,f,p,e,sw,0   'init machine for writing to RX fifo
for i=0 to &h3                    'write the 4 RXFIFO registers
 pio writefifo 1,0,i,&h100*(i+1)  'write values &h100, &h200, &h300, &h400
next

'check the values are written correctly
print "3 RXFIFO registers before running the program"
pio init machine 1,0,f,p,e,sr,0   'init machine for reading the RX fifo
for i=0 to 3                      'read the 4 RXFIFO registers
   print i,hex$(pio(readfifo 1,0,i))  'verify they are correctly written
next
print

'run the PIO program. That should (continuously) write to reg 2 and 3 in the FIFO,
but not alter register 0 and 1
pio start 1,0

'show the updated FIFO registers
print "3 RXFIFO registers after running the program"
for i=0 to 3      'read the 4 FIFO registers to see if the program works
   print i,hex$(pio(readfifo 1,0,i))
next
```

The expected output is:

```
3 RXFIFO registers before running the program
0      100
1      200
2      300
3      400

3 RXFIFO registers after running the program
0      100
1      200
2      3
3      4
```

BEISPIELPROGRAMM 5

Endlich ein Beispielprogramm, das zeigt, wie mehrere Zustandsmaschinen zusammenarbeiten können, wobei jede durch einen Interrupt der anderen ausgelöst wird. Beachte auch die Verwendung von .LINE NEXT als Startpunkt des zweiten Programms und die Verwendung von x=PIO(NEXT LINE), um den Startpunkt des zweiten Programms zu bestimmen.

IRQ 1 setzt und IRQ. Wenn die andere Zustandsmaschine auf WAIT 1 IRQ 1 wartet, fährt sie fort, sobald IRQ 1 gesetzt ist, und löscht gleichzeitig IRQ 1.

```
'test for PIO IRQ's on PIO 0 state machines 0 and 1

  'uses GP0 and GP1
  setpin gp0,pio0
  setpin gp1,pio0
  pio clear 0

  'target frequency for PIO
  f=1e5 '100kHz

pio assemble 0
  .program sm0
  .line 0
    set pindirs,1                   'set GP0 out, see pinctrl
  .wrap target
    set pins,1 [31]                 'set GP0 high, wait 31 cycles
    set pins,0 [31]                 'set GP0 low, wait 31 cycles
    irq 0                           'set IRQ 0
    wait 1 irq 1                    'wait until sm1 sets IRQ 1
  .wrap
.end program list

  ln=pio(next line)                 'remember the line in the program
  p0=pio(pinctrl 0,1,,,,gp0)        'GP0 is pin for SET
  e0=pio(execctrl gp0,pio(.wrap target),pio(.wrap))

pio assemble 0
  .program sm1
  .line next
    set pindirs,1                   'set GP1 output
  .wrap target
    wait 1 irq 0                    'wait for IRQ 0
    set pins,1 [31]                 'set GP1 high and wait 31 cycles
    set pins,0 [31]                 'set GP1 low and wait 31 cycles
    irq 1                           'set IRQ 1
  .wrap
.end program list

  p1=pio(pinctrl 0,1,,,,gp1)
  e1=pio(execctrl gp0,pio(.wrap target),pio(.wrap))

  pio init machine 0,1,f,p1,e1,,ln  'start sm1 at line ln
  pio init machine 0,0,f,p0,e0,,0   'start sm0 at line 0

  pio start 0,1
  pio start 0,0

do:loop 'do nothing, let PIO generate the signals

* end *
```

# Anhang G
# Sprites
## VGA-, HDMI- und LCD-FRAMEBUFFER

Du kannst ein Sprite auf verschiedene Arten erstellen, aber im Grunde speicherst du nur ein Bild in einem Puffer. Der Unterschied zeigt sich, wenn du das Sprite ANZEIGST. In diesem Fall speichert die Firmware beim ersten Mal den Speicherbereich (oder die Bildschirmfläche), der durch das Sprite ersetzt wird, und zeichnet dann das Sprite an dieser Stelle.

Nachfolgende SHOW-Befehle ersetzen das Sprite durch den gespeicherten Hintergrund, speichern den Hintergrund für die neue Position und zeichnen schließlich das Sprite. Auf diese Weise kannst du das Sprite ohne zusätzlichen Code über den Hintergrund bewegen.

Die Kollisionserkennung sitzt dann darüber und sucht nach den rechteckigen Begrenzungen von Sprites, die sich berühren, um einen Interrupt auszulösen, oder nach einem Sprite, das den Rand des Frames berührt.

Sprites werden so angeordnet, dass die Zeichenreihenfolge in einem LIFO gespeichert wird. Angenommen, Sprite 1 wird von Sprite 2 und dann von Sprite 3 überlagert. Wenn du Sprite 1 einfach verschieben würdest, würde sein Hintergrund Teile von 2 und 3 überschreiben – was wir nicht wollen. SPRITE SHOW SAFE löst das LIFO auf, indem es jedes Sprite in umgekehrter Reihenfolge entfernt, Sprite 1 verschiebt und dann zuerst 2 und dann 3 wieder darüberlegt. Schließlich gibt es noch das Konzept der Ebenen (das ist der 4. Parameter in SPRITE SHOW).

Das Konzept der Sprite-Implementierung ist wie folgt:

- Sprites sind vollfarbig und können jede beliebige Größe haben. Die Kollisionsgrenze ist das umschließende Rechteck.
- Sprites werden unter einer bestimmten Nummer (1 bis 64) geladen.
- Sprites werden mit dem Befehl SPRITE SHOW angezeigt.
- Für jeden SHOW-Befehl muss der Benutzer eine „Ebene“ auswählen. Diese kann zwischen 0 und 10 liegen.
- Sprites kollidieren mit Sprites auf derselben Ebene, auf Ebene 0 oder am Bildschirmrand.
- Ebene 0 ist ein Sonderfall, und Sprites auf allen anderen Ebenen kollidieren mit ihr.
- Die SCROLL-Befehle lassen Sprites auf allen Ebenen außer Ebene 0 unberührt.
- Sprites auf Ebene 0 scrollen mit dem Hintergrund mit, was zu Kollisionen führen kann.
- Es gibt keine praktische Begrenzung für die Anzahl der Kollisionen, die durch SHOW- oder SCROLL-Befehle verursacht werden.
- Mit der Funktion SPRITE() kannst du alle Details einer Kollision abrufen.
- Ein SHOW-Befehl überschreibt die Details aller vorherigen Kollisionen für dieses Sprite.
- Ein SCROLL-Befehl überschreibt die Details früherer Kollisionen für ALLE Sprites.
- Um einen Bildschirm in einen früheren Zustand zurückzusetzen, sollten Sprites in umgekehrter Reihenfolge zu ihrer Erstellung entfernt werden (d. h. „Last in, first out“).

Da das Verschieben eines Sprites oder insbesondere das Scrollen des Hintergrunds zu mehreren Sprite-Kollisionen führen kann, ist es wichtig zu verstehen, wie diese abgefragt werden können.

Der beste Weg, mit einer Sprite-Kollision umzugehen, ist die Verwendung der Interrupt-Funktion. Eine Kollisions-Interrupt-Routine wird mit dem Befehl SPRITE INTERRUPT eingerichtet. Beispiel:

```
SPRITE INTERRUPT collision
```

Im Folgenden findest du ein Beispielprogramm zur Erkennung aller Kollisionen, die entweder durch einen SPRITE SHOW-Befehl oder einen SCROLL-Befehl verursacht wurden

```
'
' This routine demonstrates a complete interrogation of collisions
'
SUB collision
  LOCAL INTEGER i
```

```
' First use the SPRITE(S) function to see what caused the interrupt
  IF SPRITE(S) <> 0 THEN 'collision of specific individual sprite
    'SPRITE(S) returns the sprite that moved to cause the collision
    PRINT "Collision on sprite ", SPRITE(S)
    process_collision(SPRITE(S))
    PRINT
  ELSE     '0 means collision of one or more sprites caused by background move
           ' SPRITE(C, 0) will tell us how many sprites had a collision
    PRINT "Scroll caused a total of ", SPRITE(C,0)," sprites to have collisions"
    FOR I = 1 TO SPRITE(C, 0)
      ' SPRITE(C, 0, i) will tell us the sprite number of the "I"th sprite
      PRINT "Sprite ", SPRITE(C, 0, i)
      process_collision(SPRITE(C, 0, i))
    NEXT i
    PRINT
  ENDIF
END SUB

' get details of the specific collisions for a given sprite
SUB process_collision(S AS INTEGER)
  LOCAL INTEGER i, j
  ' SPRITE(C, #n) returns the number of current collisions for sprite n
  PRINT "Total of " SPRITE(C, S) " collisions"
   FOR I = 1 TO SPRITE(C, S)
    ' SPRITE(C, S, i) will tell us the sprite number of the "I"th sprite
    j = SPRITE(C, S, i)
    IF j = &HF1 THEN
      PRINT "collision with left of screen"
    ELSE IF j = &HF2 THEN
      PRINT "collision with top of screen"
    ELSE IF j = &HF4 THEN
      PRINT "collision with right of screen"
    ELSE IF j = &HF8 THEN
      PRINT "collision with bottom of screen"
    ELSE
      ' SPRITE(C, #n, #m) returns details of the mth collision
      PRINT "Collision with sprite ", SPRITE(C, S, i)
    ENDIF
  NEXT i
END SUB
```

# Anhang H
# Turtle-Grafik

Diese Version enthält eine sehr umfassende Implementierung von Turtle-Grafiken für alle Versionen außer WebMite RP2040 und WebMite RP2350.

Die unterstützten Befehle, denen „TURTLE“ vorangestellt ist, lauten:

Bewegungsbefehle

| | |
|---|---|
| FORWARD distance | (FD) – Vorwärts bewegen um distance Pixel |
| BACK distance | (BK) – Um distance Pixel rückwärts bewegen |
| LEFT [angle] | (LT) – Nach links um Winkel Grad drehen, 90 Grad, wenn nicht anders angegeben |
| RIGHT [angle] | (RT) - Um Winkel Grad nach rechts drehen, 90 Grad, wenn nicht anders angegeben |

Positionsbefehle

| | |
|---|---|
| SET XY x, y | - Bewege dich zur absoluten Position (x,y) |
| SET X x | - X-Koordinate setzen, Y beibehalten |
| SET Y y | - Y-Koordinate festlegen, X beibehalten |
| SET HEADING angle | (SETH) - Absolutkurs einstellen (0=nach oben, 90=nach rechts) |
| HOME | - Zurück zur Mitte (MM.HRES\2,MM.VRES\2) Kurs 0 |

Stiftsteuerungsbefehle

| | |
|---|---|
| PEN UP | (PU) – Stift anheben (Zeichnen beenden) |
| PEN DOWN | (PD) - Stift absetzen (mit dem Zeichnen beginnen) |
| PEN COLOUR Farbe | (PC) - Stiftfarbe einstellen |
| PEN WIDTH Breite | (PW) - Stiftbreite einstellen |

Befehle für Bögen und Kurven

| | |
|---|---|
| ARC radius angle | - Bogen mit gegebenem Radius und Winkel zeichnen |
| ARCLEFT radius,angle | (ARCL) – Bogen nach links zeichnen |
| ARCRIGHT radius,angle | (ARCR) – Zeichne einen nach rechts gekrümmten Bogen |
| BEZIER cp1, cp1angle, cp2, cp2angle, end, endangle | - Zeichne eine Bézier-Kurve mit Kontrollpunkten |

Grundlegende Formbefehle

| | |
|---|---|
| CIRCLE radius | - Zeichne einen Kreis an der aktuellen Position |
| DOT size | - Gefüllten Punkt zeichnen (Standardgröße=5) |
| FCIRCLE radius | - Gefüllten Kreis zeichnen |
| FRECTANGLE width,heigh | (FRECT) - Gefülltes Rechteck zeichnen |
| WEDGE radius start end | - Gefüllten Keil/Kuchenscheibe zeichnen |

Füllbefehle

| | |
|---|---|
| FILL COLOUR color | (FC) - Füllfarbe festlegen und Füllung aktivieren |
| FILL PATTERN pattern | (FP) - Füllmuster festlegen (0-31) |
| NO FILL | - Füllung deaktivieren |
| FILL | - Füllung an der aktuellen Position |
| BEGIN FILL | (BF) - Aufzeichnung des Polygons für die Füllung starten |
| END FILL | (EF) - Aufzeichnung beenden und Polygon füllen |

Cursor-Befehle

| | |
|---|---|
| SHOW TURTLE | (ST) - Turtle-Cursor anzeigen |
| HIDE TURTLE | (HT) – Schildkröten-Cursor ausblenden |
| CURSOR SIZE size | (CS) - Cursorgröße einstellen |
| CURSOR COLOUR color | CC) - Cursor-Farbe festlegen |
| STAMP | - Zeichne eine Schildkröte an der aktuellen x,y-Position |

Befehle zur Zustandsverwaltung

| | |
|---|---|
| RESET [show] | - Bildschirm löschen und alles zurücksetzen, die Schildkröte anzeigen, wenn show = 1 |
| PUSH | - Aktuelle Position und Richtung auf dem Stapel speichern |
| POP | - Stelle Position und Richtung vom Stapel wieder her |

Der Code ermöglicht es, Kreise, Rechtecke und Polygone mit einer strukturierten Füllung zu füllen. Führe ihn im Modus 2 auf einem RP2040-VGA-System oder im Modus 3 auf einem RP2350-VGA- oder HDMI-System aus.

Füllmuster

0: Einfarbige Füllung
1: Schachbrettmuster
2: Vertikale Linien
3: Horizontale Linien
4: Diagonales Kreuz
5: Diagonale Streifen
6: Kreuzschraffur
7: Feine Diagonale
8: Dichtes Schachbrettmuster
9: Diagonal rechts mittel
10: Diagonal links mittel
11: Vertikale Linien mittel
12: Horizontale Linien, mittel
13: Großes Schachbrettmuster
14: Vertikale Punkte
15: Enge horizontale Streifen
16: Gitter
17: Webmuster
18: Rautenmuster
19: Diagonaler Farbverlauf
20: Diagonaler Farbverlauf umgekehrt
21: Rand/Rahmen
22: Vertikaler Schnitt
23: Gewebt
24: Vereinzelte Punkte
25: Diagonal sehr fein
26: Pfeil nach oben
27: Dichte Punkte
28: Chevron
29: Hohlraute
30: Kreis
31: Gefüllter Kreis

# Anhang I
## Sondertasten

MMBasic generiert für die Funktionstasten und andere Sondertasten auf der Tastatur jeweils ein einziges, eindeutiges Zeichen. Hinweis: Die DE-USB-Tastatur gibt bei normaler Eingabe (input$/inkey$) die Codes 200–209 für Zeichen mit Akzent zurück
bei normaler Eingabe (input$/inkey$)

Die Codes sind in dieser Tabelle als Hexadezimal- und Dezimalzahlen aufgeführt:

| Tastaturtaste | Tastencode (Hex) | Tastencode (dezimal) |
|---|---|---|
| DEL | 7F | 127 |
| Pfeil nach oben | 80 | 128 |
| Abwärtspfeil | 81 | 129 |
| Pfeil nach links | 82 | 130 |
| Pfeil nach rechts | 83 | 131 |
| Einfügen | 84 | 132 |
| Startseite | 86 | 134 |
| Ende | 87 | 135 |
| Seite nach oben | 88 | 136 |
| Seite nach unten | 89 | 137 |
| Alt | 8B | 139 |
| F1/Umschalt-F1 | 91/B1 | 145/177 |
| F2/Umschalt F2 | 92/B2 | 146/178 |
| F3/Umschalt-F3 ** | 93/B3 | 147/179 |
| F4/Umschalt-F4 ** | 94/B4 | 148/180 |
| F5/Umschalt-F5 ** | 95/B5 | 149/181 |
| F6/Umschalt-F6 ** | 96/B6 | 150/182 |
| F7/Umschalt-F7 ** | 97/B7 | 151/183 |
| F8/Umschalt-F8 ** | 98/B8 | 152/184 |
| F9/Umschalt-F9 | 99/B9 | 153/185 |
| F10/Umschalt-F10 | 9A/BA | 154/186 |
| F11/Umschalt-F11 | 9B/BB | 155/187 |
| F12/Umschalt-F12 | 9C/BC | 156/188 |
| PrtScr/SysRq | 9D | 157 |
| PAUSE/BREAK | 9E | 158 |
| UMSCHALT_TAB | 9F | 159 |
| Umschalt-Entf | A0 | 160 |
| Umschalt-Pfeil-nach-unten | A1 | 161 |
| UMSCHALT_PFEIL_RECHTS | A3 | 163 |

** bedeutet, dass es auch für VT100-Emulatoren funktioniert

Bei angeschlossenen PS2- und USB-Tastaturen wird, wenn die Umschalttaste gleichzeitig mit den Funktionstasten F1 bis F12 gedrückt wird, 20 (hex) zum Code addiert (das entspricht dem Setzen von Bit 5). Zum Beispiel erzeugt Shift-F10 den Wert BA (hex).

Der Shift-Modifikator funktioniert mit den Funktionstasten F1 bis F12; bei den anderen Tasten wird er ignoriert, außer bei TAB, DEL, DOWN_ARROW und RIGHT_ARROW, wie oben beschrieben. MMBasic übersetzt die meisten VT100-Escape-Codes, die von Terminalemulatoren wie TeraTerm und Putty generiert werden, in diese Codes (der Shift-Modifikator funktioniert nur für F3–F8). Das bedeutet, dass ein Terminalemulator, der über einen als Konsole geöffneten USB- oder seriellen Anschluss läuft, dieselben Tastencodes generiert wie eine direkt angeschlossene Tastatur.

# Anhang J
# Programmieren in BASIC – Ein Tutorial

Die Sprache BASIC wurde 1964 vom Dartmouth College in den USA als Programmiersprache für den Unterricht eingeführt und ist dementsprechend einfach zu verwenden und zu erlernen. Gleichzeitig hat sie sich als kompetente und leistungsstarke Programmiersprache bewährt und wurde daher in den späten 70er und frühen 80er Jahren sehr beliebt. Auch heute noch werden einige große kommerzielle Datensysteme in der Sprache BASIC geschrieben (hauptsächlich Pick Basic).

Der in der PicoMite-Firmware verwendete BASIC-Interpreter heißt MMBasic und ist eine moderne Version der Programmiersprache BASIC, die den vor Jahren beliebten Microsoft-BASIC-Interpreter lose nachahmt.

Für einen Programmierer ist der größte Vorteil von BASIC seine Benutzerfreundlichkeit. Einige modernere Sprachen wie C und C++ können einem regelrecht den Kopf zerbrechen, aber mit BASIC kannst du mit einem einzeiligen Programm beginnen und schon etwas Sinnvolles damit erreichen. MMBasic ist zudem leistungsstark, da du damit anspruchsvolle Grafiken zeichnen, die externen I/O-Pins zur Steuerung anderer Geräte manipulieren und über eine Reihe integrierter Kommunikationsprotokolle mit anderen Geräten kommunizieren kannst.

## Befehlszeile

Interaktion mit Die Interaktion mit MMBasic erfolgt über die Konsole an der Befehlszeile (d. h. das Größer-als-Zeichen (>) auf der Konsole). Beim Start zeigt MMBasic die Befehlszeile an und wartet darauf, dass ein Befehl eingegeben wird. Es kehrt auch zur Befehlszeile zurück, wenn dein Programm endet oder eine Fehlermeldung ausgibt.

Wenn die Eingabeaufforderung angezeigt wird, steht dir eine große Auswahl an Befehlen zur Verfügung, die du eingeben und ausführen kannst. Typischerweise listest du damit das im Speicher befindliche Programm auf (LIST), bearbeitest es (EDIT) oder legst vielleicht einige Optionen fest (der Befehl OPTION). Meistens lautet der Befehl einfach RUN, was MMBasic anweist, das im Programmspeicher befindliche Programm auszuführen.

An der Eingabeaufforderung kann fast jeder Befehl eingegeben werden, und dies wird oft genutzt, um einen Befehl zu testen und zu sehen, wie er funktioniert. Ein einfaches Beispiel ist der Befehl PRINT (mehr dazu später), den du testen kannst, indem du Folgendes an der Eingabeaufforderung eingibst:

```
PRINT 2 + 2
```

und es überrascht nicht, dass MMBasic die Zahl 4 ausgibt, bevor es zur Eingabeaufforderung zurückkehrt.

Diese Möglichkeit, einen Befehl an der Befehlszeile zu testen, ist nützlich, wenn du lernst, in BASIC zu programmieren. Es lohnt sich also, einen Raspberry Pi Pico mit der PicoMite-Firmware griffbereit zu haben, um während der Arbeit an diesem Tutorial gelegentlich Tests durchzuführen.

## Aufbau eines BASIC-Programms

Ein BASIC-Programm beginnt in der ersten Zeile und läuft so lange, bis es das Ende erreicht oder auf einen END-Befehl stößt – 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, von denen jede den BASIC-Interpreter dazu veranlasst, etwas auszuführen (die Begriffe *„Anweisung“* und *„Befehl“* bedeuten im Allgemeinen dasselbe und werden in diesem Tutorial synonym verwendet).

Normalerweise steht jede Anweisung in einer eigenen Zeile, aber du kannst auch mehrere Anweisungen in einer Zeile haben, die durch das Doppelpunktzeichen (:) getrennt sind.

Zum Beispiel:

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

Jede Zeile kann mit einer Zeilennummer beginnen. In den frühen BASIC-Interpretern waren Zeilennummern obligatorisch, moderne Implementierungen (wie MMBasic) benötigen sie jedoch nicht. Du kannst sie weiterhin verwenden, wenn du möchtest, aber sie haben keinen Vorteil und machen deine Programme im Allgemeinen nur unübersichtlich.

Hier 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 als Ziel für einen Programmsprung mit dem GOTO-Befehl verwendet werden kann. Das wird genauer erklärt, wenn wir den GOTO-Befehl behandeln, aber hier ist ein Beispiel (der Label-Name lautet `JmpBack`):

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

### Kommentare

Ein Kommentar ist jeder Text, der auf das einfache Anführungszeichen (') folgt. Ein Kommentar kann an beliebiger Stelle platziert werden und erstreckt sich bis zum Zeilenende. Wenn MMBasic auf einen Kommentar stößt, springt es einfach zum Ende des Kommentars (d. h., es führt keine Aktion in Bezug auf den Kommentar aus).

Kommentare sollten verwendet werden, um nicht offensichtliche Teile des Programms zu erklären und generell jemanden, der mit dem Programm nicht vertraut ist, darüber zu informieren, wie es funktioniert und was es zu tun versucht. Denk daran, dass dir ein Programm, das du geschrieben hast, schon nach wenigen Monaten aus dem Gedächtnis verschwunden sein wird und dir seltsam vorkommen wird, wenn du es wieder in die Hand nimmst. Aus diesem Grund wirst du dir später dankbar sein, wenn du reichlich Kommentare verwendest.

Hier sind einige Beispiele für Kommentare:

```
' Berechne die Hypotenuse
PRINT SQR(a * a + b * b)
INPUT var    ' Temperatur abfragen
```

Ältere BASIC-Programme verwendeten den Befehl REM, um einen Kommentar zu beginnen, und du kannst diesen auch verwenden, wenn du möchtest, aber das einfache Anführungszeichen ist einfacher zu verwenden und praktischer.

### Der PRINT-Befehl

Es gibt eine Reihe gängiger Befehle, die grundlegend sind, und wir werden sie in diesem Tutorial behandeln, aber der wohl nützlichste ist der PRINT-Befehl. Seine Aufgabe ist einfach: etwas auf der Konsole auszugeben. Dies wird meist verwendet, um Daten für dich sichtbar auszugeben (wie das Ergebnis von Berechnungen) oder informative Meldungen anzuzeigen.

PRINT ist auch nützlich, wenn du einen Fehler in deinem Programm aufspürst; du kannst damit die Werte von Variablen ausgeben und Meldungen an wichtigen Stellen während der Programmausführung anzeigen.

In seiner einfachsten Form gibt der Befehl einfach alles aus, was in seiner Befehlszeile steht. Zum Beispiel:

```
PRINT 54
```

zeigt auf der Konsole die Zahl 54 gefolgt von einer neuen Zeile an.

Die auszugebenden Daten können etwas Einfaches wie dies sein oder ein Ausdruck, also etwas, das berechnet werden muss. Wir werden Ausdrücke später noch genauer behandeln, aber als Beispiel hier:

```
> PRINT 3/21
 0.1428571429
>
```

würde das Ergebnis von drei geteilt durch einundzwanzig berechnen und anzeigen. Beachte, dass das Größer-als-Zeichen (>) die von MMBasic erzeugte Eingabeaufforderung ist – das gibst du nicht ein.

Weitere Beispiele für den PRINT-Befehl sind:

```
> PRINT „Wonderful World“
Wonderful World
> PRINT (999 + 1) / 5
 200
>
```

Du kannst das an der Eingabeaufforderung ausprobieren.

Der Befehl PRINT funktioniert auch mit mehreren Werten gleichzeitig, zum Beispiel:

```
> PRINT "The first number is" 20+25 " and the second is" 18/3
The first number is 45 and the second is 6
>
```

Normalerweise werden die Werte durch ein Leerzeichen getrennt, wie im vorherigen Beispiel gezeigt, aber du kannst Werte auch durch ein Komma (,) trennen. Das Komma bewirkt, dass zwischen den beiden Werten ein Tabulator eingefügt wird. In MMBasic liegen die Tabulatoren im PRINT-Befehl acht Zeichen auseinander.

Um die Tabulatoren zu veranschaulichen, gibt der folgende Befehl eine tabulatorgetrennte Liste von Zahlen aus:

```
> PRINT 12, 34, 9.4, 1000
 12       34       9.4      1000
>
```

Beachte, dass vor jeder Zahl ein Leerzeichen gedruckt wird. Dieses Leerzeichen ist ein Platzhalter für das Minuszeichen (-) für den Fall, dass der Wert negativ ist. Du kannst den Unterschied bei den Zahlen 12 und 9,4 in diesem Beispiel sehen:

```
> PRINT -12, 34, -9.4, 1000
-12       34      -9.4      1000
>
```

Die PRINT-Anweisung kann mit einem Semikolon (;) beendet werden. Dadurch wird verhindert, dass der PRINT-Befehl in eine neue Zeile wechselt, wenn er den gesamten Text ausgegeben hat. Zum Beispiel:

```
PRINT "Das wird";
PRINT " in einer einzigen Zeile gedruckt."
```

Das ergibt folgende Ausgabe:

```
Dies wird in einer einzigen Zeile gedruckt.
```

Ohne das Semikolon am Ende der ersten Zeile würde die Meldung so aussehen:

```
Das wird
 in einer einzigen Zeile gedruckt.
```

### Variablen

Bevor wir weitermachen, müssen wir definieren, was eine „Variable“ ist, da sie für die Funktionsweise der Sprache BASIC (und eigentlich der meisten Programmiersprachen) grundlegend ist. Eine Variable ist einfach ein Ort, an dem ein Datenelement (d. h. sein „Wert“) gespeichert wird.

Dieser Wert kann während der Programmausführung geändert werden, weshalb er als „Variable“ bezeichnet wird.

Variablen in MMBasic können drei verschiedene Typen haben. Der häufigste ist der Gleitkomma-Typ, der automatisch angenommen wird, wenn der Typ der Variablen nicht angegeben wird. Die beiden anderen Typen sind Ganzzahl und Zeichenkette, auf die wir später eingehen werden. Eine Gleitkommazahl ist eine gewöhnliche Zahl, die einen Dezimalpunkt enthalten kann. Zum Beispiel sind 3,45, -0,023 oder 100,00 allesamt Gleitkommazahlen.

Eine Variable kann zum Speichern einer Zahl verwendet werden und lässt sich dann genauso wie die Zahl selbst verwenden; in diesem Fall repräsentiert sie den Wert der zuletzt zugewiesenen Zahl.

Ein einfaches Beispiel:

```
A = 3
B = 4
PRINT A + B
```

gibt die Zahl 7 aus. In diesem Fall sind sowohl A als auch B Variablen, und MMBasic hat ihre aktuellen Werte in der PRINT-Anweisung verwendet. MMBasic erstellt automatisch eine Variable, wenn es zum ersten Mal auf sie stößt. Die Anweisung `A = 3` hat also eine Gleitkommavariable (der Standardtyp) mit dem Namen `A` erstellt und ihr dann den Wert 3 zugewiesen.

Der Name einer Variablen muss mit einem Buchstaben beginnen, während der Rest des Namens aus Buchstaben, Zahlen, dem Unterstrich oder dem Punkt bestehen kann. Der Name darf bis zu 31 Zeichen lang sein, und die Groß-/Kleinschreibung spielt keine Rolle. Hier sind einige Beispiele:

```
Total_Count
ForeColour
temp3
count
x
DasIstEinSehrLangerVariablenname
increment.value
```

Du kannst den Wert einer Variablen an jeder beliebigen Stelle in deinem Programm ändern, indem du den Zuweisungsbefehl verwendest,
z. B.:

```
variable = expression
```

Zum Beispiel:

```
temp3 = 24.6
count = 5
CTemp = (FTemp - 32) * 0.5556
```

Im letzten Beispiel sind sowohl CTemp als auch FTemp Variablen, und diese Zeile wandelt den Wert von FTemp (in Grad Fahrenheit) in Grad Celsius um und speichert das Ergebnis in der Variablen CTemp.

## Ausdrücke

Wir sind in diesem Tutorial bereits auf den Begriff „Ausdruck“ gestoßen, und in der Programmierung hat er eine bestimmte Bedeutung. Es handelt sich um eine Formel, die vom BASIC-Interpreter zu einer einzelnen Zahl oder einem Wert aufgelöst werden kann.

MMBasic wertet numerische Ausdrücke nach denselben Regeln aus, die wir in der Schule gelernt haben. Beispielsweise werden Multiplikation und Division zuerst durchgeführt, gefolgt von Addition und Subtraktion. Diese Regeln werden als Prioritätsregeln bezeichnet und sind weiter oben in diesem Handbuch ausführlich beschrieben (siehe das Kapitel *„Ausdrücke und Operatoren*“).

Das bedeutet, dass MMBasic 2 + 3 * 6 so auswertet, dass zuerst 3 mit 6 multipliziert wird, was 18 ergibt, und dann 2 addiert wird, was zu einem Endwert von 20 führt. Ebenso ergeben sowohl 5 * 4 als auch 10 + 4 * 3 – 2 ebenfalls 20.

Wenn du den Interpreter zwingen willst, Teile des Ausdrucks zuerst auszuwerten, kannst du diesen Teil des Ausdrucks in Klammern setzen. Zum Beispiel ergibt (10 + 4) * (3 – 2) das Ergebnis 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 nennenswert, daher solltest du sie großzügig einsetzen, wenn die Möglichkeit besteht, dass MMBasic deine Absicht falsch interpretiert.

Wie bereits erwähnt, kannst du Variablen in einem Ausdruck genauso verwenden wie einfache Zahlen. Das hier erhöht zum Beispiel den Wert der Variablen `temp` um eins:

```
temp = temp + 1
```

Du kannst in Ausdrücken auch Funktionen verwenden. Das sind spezielle Operationen, die MMBasic bereitstellt, zum Beispiel zur Berechnung trigonometrischer Werte.

Ein Beispiel für die Verwendung einer Funktion ist das folgende, das die Länge der Hypotenuse eines rechtwinkligen Dreiecks ausgibt. Hier wird die Funktion SQR() verwendet, die die Quadratwurzel einer Zahl zurückgibt (`a` und `b` sind Variablen, die die Längen der anderen Seiten enthalten):

```
PRINT SQR(a * a + b * b)
```

MMBasic wertet diesen Ausdruck zunächst aus, indem es `a` mit `a` und `b` mit `b` multipliziert und dann die Ergebnisse addiert. Die resultierende Zahl wird dann an die Funktion `SQR()` übergeben, die die Quadratwurzel dieser Zahl (d. h. die Hypotenuse) berechnet und sie an den Befehl PRINT zur Anzeige zurückgibt.

Zu den weiteren mathematischen Funktionen, die MMBasic bietet, gehören:

SIN(r) – den Sinus von r
COS(r) – der Kosinus von r
TAN(r) – der Tangens von r

Es stehen dir noch viele weitere Funktionen zur Verfügung, die alle weiter oben in diesem Handbuch aufgeführt sind.

Beachte, dass bei den oben genannten trigonometrischen Funktionen der an die Funktion übergebene Wert (d. h. „r“) der Winkel in Radianten ist. In MMBasic kannst du die Funktion RAD(d) verwenden, um einen Winkel von Grad in Radianten umzurechnen („d“ ist der Winkel in Grad).

Ein weiteres Merkmal der meisten Programmiersprachen (einschließlich BASIC) ist, dass du Funktionsaufrufe ineinander verschachteln kannst. Wenn du beispielsweise den Winkel in Grad (d. h. „d“) hast, kannst du den Sinus dieses Winkels mit folgendem Ausdruck berechnen:

```
PRINT SIN(RAD(d))
```

In diesem Fall nimmt MMBasic zunächst den Wert von d und wandelt ihn mit der Funktion RAD() in Radiant um. Die Ausgabe dieser Funktion wird dann zur Eingabe für die Funktion SIN().

### Die IF-Anweisung

Entscheidungen zu treffen ist das Herzstück der meisten Computerprogramme, und in BASIC geschieht dies in der Regel mit der IF-Anweisung. Diese wird fast wie ein englischer Satz geschrieben:

```
IF Bedingung THEN Aktion
```

Die *Bedingung* ist meist ein Vergleich wie „gleich“, „kleiner als“, „größer als“ usw.

Zum Beispiel:

```
IF Temp < 25 THEN PRINT "Kalt"
```

`Temp` wäre eine Variable, die die aktuelle Temperatur (in ºC) enthält, und `PRINT „Kalt“` die auszuführende Aktion.

Es gibt eine Reihe von Prüfungen, die du durchführen kannst:

| | | | |
|---|---|---|---|
| = | gleich | <> | ungleich |
| < | kleiner als | <= | kleiner oder gleich |
| > | größer als | >= | größer als oder gleich |

Du kannst auch eine ELSE-Klausel hinzufügen, die ausgeführt wird, wenn die ursprüngliche Bedingung falsch ist:

```
IF Bedingung THEN wahre-Aktion ELSE unwahre-Aktion
```

Das führt zum Beispiel unterschiedliche Aktionen aus, wenn die Temperatur unter 25 oder 25 oder mehr liegt:

```
IF Temp < 25 THEN PRINT "Kalt" ELSE PRINT "Heiss"
```

In den vorherigen Beispielen wurden alle einzeilige IF-Anweisungen verwendet, aber du kannst auch mehrzeilige IF-Anweisungen verwenden. Diese sehen so aus:

```
IF Bedingung THEN
    wahre-Aktion
    wahre-Aktion
ENDIF
IF Bedingung THEN
    wahre-Aktion
    wahre-Aktion
SONST
    unwahre-Aktion
    unwahre-Aktion
ENDIF
```

Im Gegensatz zur einzeiligen IF-Anweisung kannst du viele True-Aktionen haben, wobei jede in einer eigenen Zeile steht, und ebenso viele False-Aktionen. Im Allgemeinen ist die einzeilige IF-Anweisung praktisch, wenn du eine einfache Aktion hast, die ausgeführt werden muss, während die mehrzeilige Version viel leichter zu verstehen ist, wenn die Aktionen zahlreich und komplizierter sind.

Ein Beispiel für eine mehrzeilige IF-Anweisung mit mehr als einer Aktion ist:

```
IF Amount < 100 THEN
    PRINT "Too low"
    PRINT “Minimum value is 100”
ELSE
    PRINT "Input accepted"
    SaveToSDCard
    PRINT "Enter second amount"
ENDIF
```

Beachte, dass im obigen Beispiel jede Aktion eingerückt ist, um zu zeigen, zu welchem Teil der IF-Struktur sie gehört. Einrückungen sind nicht zwingend erforderlich, machen ein Programm aber für jemanden, der damit nicht vertraut ist, viel verständlicher und sind daher sehr zu empfehlen.

In einer mehrzeiligen IF-Anweisung kannst du mit dem Befehl ELSE IF zusätzliche Prüfungen durchführen. Das lässt sich am besten anhand eines Beispiels erklären (die Temperaturen sind alle in ºC angegeben):

```
IF Temp < 0 THEN
    PRINT "Freezing"
ELSE IF Temp < 20 THEN
    PRINT "Cold"
ELSE IF Temp < 35 THEN
    PRINT "Warm"
ELSE
    PRINT "Hot"
ENDIF
```

Das ELSE IF verwendet dieselben Bedingungen wie ein gewöhnliches IF (d. h. <, <= usw.), aber diese Bedingung wird nur geprüft, wenn die vorhergehende Bedingung falsch war. So erhältst du beispielsweise nur die Meldung *„Warm"*, wenn `Temp < 0` falsch war, `Temp < 20` falsch war, aber `Temp < 35` wahr war. Das abschließende ELSE erfasst den Fall, in dem alle Bedingungen falsch waren.

Ein Ausdruck wie „`Temp < 20`" wird von MMBasic entweder als „true" oder als „false" ausgewertet, wobei „true" den Wert 1 und „false" den Wert 0 hat. Du kannst das sehen, wenn du Folgendes in die Konsole eingibst:

```
PRINT 30 > 20
```

MMBasic gibt `1` aus, was bedeutet, dass der Wert des Ausdrucks wahr ist.

Ebenso gibt das Folgende `0` aus, was bedeutet, dass der Ausdruck als falsch ausgewertet wurde.

```
PRINT 30 < 20
```

Die IF-Anweisung kümmert sich nicht wirklich darum, wie die Bedingung tatsächlich lautet; sie wertet die Bedingung einfach aus und wenn das Ergebnis Null ist, wird dies als „false" gewertet, und wenn es ungleich Null ist, wird es als „true" gewertet.

Das ermöglicht einige praktische Abkürzungen. Wenn zum Beispiel „`BalanceCorrect`" eine Variable ist, die wahr (ungleich Null) ist, wenn eine bestimmte Funktion des Programms korrekt ist, dann kannst du Folgendes verwenden, um auf Basis dieses Werts eine Entscheidung zu treffen:

```
IF BalanceCorrect THEN …etwas tun…
```

### FOR-Schleifen

Eine weitere häufige Anforderung beim Programmieren ist die Wiederholung einer Reihe von Aktionen. Beispielsweise möchtest du vielleicht alle sieben Tage der Woche durchlaufen und für jeden Tag dieselbe Funktion ausführen. BASIC bietet für diese Art von Aufgabe die FOR-Schleife, die wie folgt funktioniert:

```
FOR day = 1 TO 7
  Führe etwas basierend auf dem Wert von „day" aus
NEXT day
```

Zunächst wird die Variable „`day`" erstellt und ihr der Wert 1 zugewiesen. Das Programm führt dann die folgenden Anweisungen aus, bis es zur NEXT-Anweisung gelangt. Diese weist den BASIC-Interpreter an, den Wert von „`day`" zu erhöhen, zur vorherigen FOR-Anweisung zurückzukehren und die folgenden Anweisungen ein zweites Mal auszuführen. Dies wird so lange wiederholt, bis der Wert von „`day`" 7 überschreitet; dann verlässt das Programm die Schleife und fährt mit den Anweisungen fort, die auf die NEXT-Anweisung folgen.

Als einfaches Beispiel kannst du die Zahlen von eins bis zehn wie folgt ausgeben:

```
FOR nbr = 1 TO 10
  PRINT nbr,;
NEXT nbr
```

Das Komma am Ende der PRINT-Anweisung weist den Interpreter an, nach dem Ausgeben der Zahl zur nächsten Tabulator-Spalte zu springen, und das Semikolon sorgt dafür, dass der Cursor auf dieser Zeile bleibt, anstatt automatisch zur nächsten Zeile zu springen. Dadurch werden die Zahlen in ordentlichen Spalten über die Seite verteilt ausgegeben.

Das würde dann so aussehen:

```
1       2       3       4       5       6       7       8       9       10
```

Die FOR-Schleife hat noch ein paar zusätzliche Tricks auf Lager. Du kannst den Schritt, um den die Variable erhöht wird, mit dem Schlüsselwort STEP ändern. So werden zum Beispiel mit dem folgenden Code nur die ungeraden Zahlen ausgegeben:

```
FOR nbr = 1 TO 10 STEP 2
  PRINT nbr,;
NEXT nbr
```

Der Wert des Schritts (oder Inkrementwerts) ist standardmäßig eins, wenn das STEP-Schlüsselwort nicht verwendet wird, aber du kannst ihn auf eine beliebige Zahl setzen.

Wenn MMBasic die Variable erhöht, prüft es, ob der Wert der Variablen den TO-Wert überschritten hat, und wenn ja, verlässt es die Schleife. Im obigen Beispiel erreicht der Wert von `nbr` also neun und wird ausgegeben, aber in der nächsten Schleife ist `nbr` gleich elf, und an diesem Punkt verlässt die Ausführung die Schleife. Diese Prüfung wird auch zu Beginn der Schleife durchgeführt. Wenn zum Beispiel der Wert der Variablen zu Beginn den TO-Wert überschreitet, wird die Schleife nie ausgeführt, nicht einmal.

Indem du den STEP-Wert auf eine negative Zahl setzt, kannst du die FOR-Schleife nutzen, um von einer hohen Zahl nach unten zu zählen. In diesem Fall muss die Startzahl größer sein als die TO-Zahl.

Beispielsweise gibt der folgende Code die Zahlen von 1 bis 10 in umgekehrter Reihenfolge aus:

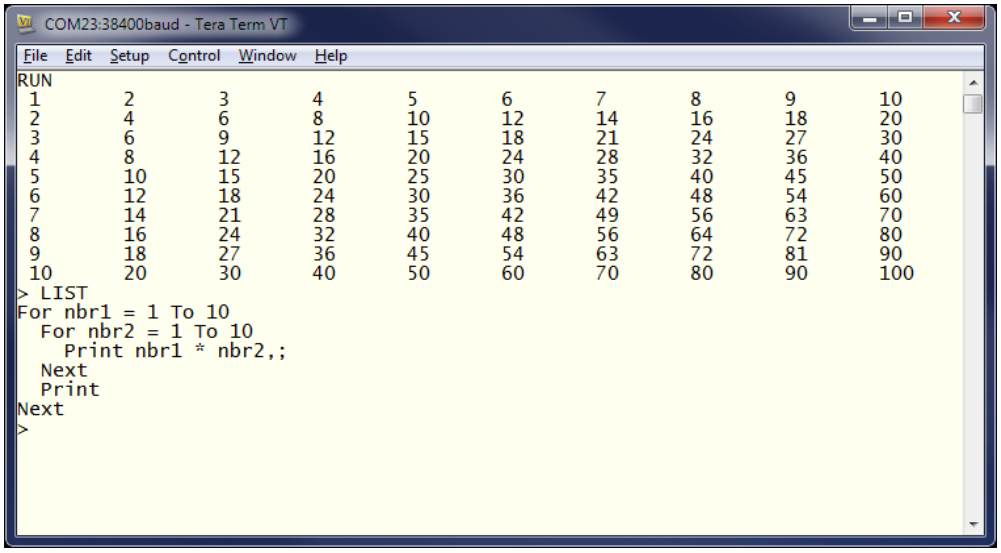
```
FOR nbr = 10 TO 1 STEP -1
  PRINT nbr,;
NEXT nbr
```

### Multiplikationstabelle

Um noch besser zu veranschaulichen, wie Schleifen funktionieren und wie nützlich sie sein können, verwendet das folgende kurze Programm zwei FOR-Schleifen, um das Einmaleins auszugeben, das wir alle in der Schule gelernt haben. Das Programm dafür ist nicht kompliziert:

```
FOR nbr1 = 1 bis 10
  FOR nbr2 = 1 bis 10
    PRINT nbr1 * nbr2,;
  NEXT nbr2
  PRINT
NEXT nbr1
```

Die Ausgabe siehst du im folgenden Screenshot, der auch eine Auflistung des Programms zeigt.




Du musst die Logik dieses Beispiels Zeile für Zeile durchgehen, um zu verstehen, was es tut. Im Wesentlichen besteht es aus einer Schleife innerhalb einer anderen. Die innere Schleife, die die Variable `nbr2` erhöht, gibt eine horizontale Zeile der Tabelle aus. Wenn diese Schleife beendet ist, führt sie den folgenden PRINT-Befehl aus, der nichts zu drucken hat – daher gibt er einfach eine neue Zeile aus (d. h., er beendet die von der inneren Schleife gedruckte Zeile).

Das Programm führt dann eine weitere Iteration der äußeren Schleife durch, indem es `nbr1` erhöht und die innere Schleife erneut ausführt. Schließlich, wenn die äußere Schleife beendet ist (wenn `nbr1` größer als 10 wird), erreicht das Programm das Ende und wird beendet.

Noch ein letzter Punkt: Du kannst den Variablennamen in der NEXT-Anweisung weglassen, und MMBasic wird erraten, auf welche Variable du dich beziehst. Es ist jedoch empfehlenswert, den Namen anzugeben, damit andere, die das Programm lesen, es leichter verstehen können. Du kannst auch mehrere Schleifen mit einer durch Kommas getrennten Liste von Variablen in der NEXT-Anweisung beenden. Zum Beispiel:

```
FOR var1 = 1 TO 5
  FOR var2 = 10 to 13
    PRINT var1 * var2
NEXT var1, var2
```

## DO-Schleifen

Eine weitere Methode für Schleifen ist die DO…LOOP-Struktur, die so aussieht:

```
DO WHILE Bedingung
   <Anweisung>
   <Anweisung>
LOOP
```

Zunächst wird die *Bedingung* geprüft. Ist sie wahr, werden die Anweisungen ausgeführt, bis der LOOP-Befehl erreicht wird. An diesem Punkt kehrt das Programm zur DO-Anweisung zurück und die *Bedingung* wird erneut geprüft. Ist sie weiterhin wahr, wird die Schleife erneut ausgeführt. Die Bedingung ist dieselbe wie beim IF-Befehl (z. B. `X < Y`).

Beispielsweise gibt der folgende Code 4 Sekunden lang das Wort „Hello“ auf der Konsole aus und stoppt dann:

```
Timer = 0
DO WHILE Timer < 4000
  PRINT „Hello“
LOOP
```

Beachte, dass `Timer` eine Funktion in MMBasic ist, die die Zeit in Millisekunden seit dem Zurücksetzen des Timers zurückgibt. Ein Zurücksetzen erfolgt durch Zuweisen von Null an `Timer` (wie oben gezeigt) oder beim Einschalten des PicoMite.

Eine Variante der DO-LOOP-Struktur ist die folgende:

```
DO
   <Anweisung>
   <Anweisung>
LOOP UNTIL Bedingung
```

Bei dieser Anordnung wird die Schleife zunächst einmal ausgeführt, dann wird die *Bedingung* geprüft, und wenn die Bedingung falsch ist, wird die Schleife so lange wiederholt, bis die *Bedingung* wahr wird. Beachte, dass die Prüfung in LOOP UNTIL das Gegenteil von DO WHILE ist.

Ähnlich wie im vorherigen Beispiel gibt auch der folgende Code vier Sekunden lang „Hallo“ aus:

```
Timer = 0
DO
  PRINT „Hallo“
LOOP UNTIL Timer >= 4000
```

Beide Formen der DO-LOOP-Schleife bewirken im Grunde dasselbe, du kannst also die Struktur verwenden, die am besten zu der Logik passt, die du umsetzen möchtest.

Schließlich ist es möglich, eine DO-Schleife zu haben, die überhaupt keine Bedingungen enthält – d. h.

```
DO
   <Anweisung>
   <Anweisung>
LOOP
```

Diese Konstruktion läuft endlos weiter, und du als Programmierer musst eine Möglichkeit vorsehen, die Schleife explizit zu verlassen (der Befehl EXIT DO erledigt das). Zum Beispiel:

```
Timer = 0
DO
  PRINT „Hallo“
  IF Timer >= 4000 THEN EXIT DO
LOOP
```

## Konsoleneingabe

Deine Programme sollen nicht nur Daten ausgeben, damit der Benutzer sie sehen kann, sondern auch Eingaben vom Benutzer entgegennehmen. Dafür musst du Tastatureingaben von der Konsole erfassen, was mit dem Befehl INPUT möglich ist. In seiner einfachsten Form lautet der Befehl:

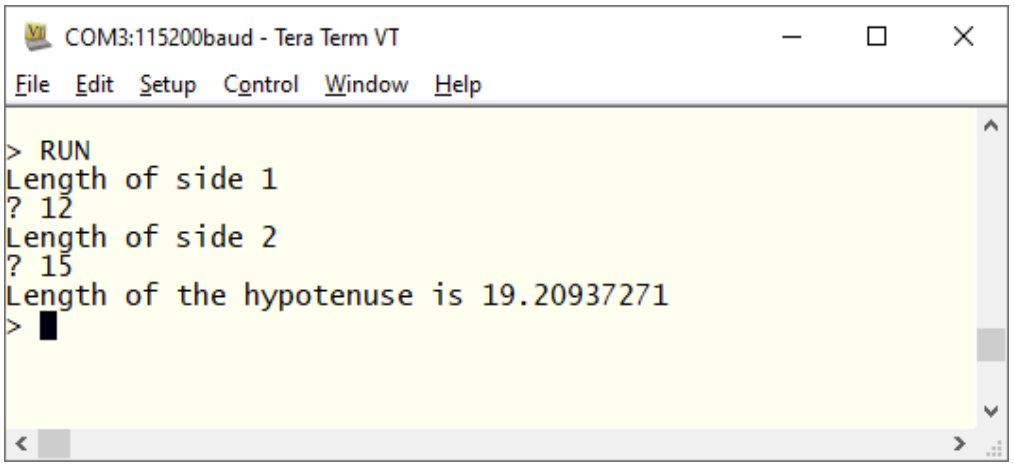
```
INPUT var
```

Dieser Befehl zeigt ein Fragezeichen auf dem Bildschirm der Konsole an und wartet darauf, dass eine Zahl eingegeben und anschließend die Eingabetaste gedrückt wird. Diese Zahl wird dann der Variablen `var` zugewiesen.

Das folgende Programm erweitert beispielsweise den Ausdruck zur Berechnung der Hypotenuse eines Dreiecks, indem es dem Benutzer ermöglicht, die Längen der anderen Seiten über die Konsole einzugeben.

```
PRINT "Length of side 1"
INPUT a
PRINT "Length of side 2"
INPUT b
PRINT "Length of the hypotenuse is" SQR(a * a + b * b)
```

Dies ist ein Screenshot einer typischen Sitzung:




Der Befehl INPUT kann auch deine Eingabeaufforderung ausgeben, sodass du keinen separaten PRINT-Befehl benötigst. Das funktioniert zum Beispiel genauso wie das obige Programm:

```
INPUT "Length of side 1"; a
INPUT "Length of side 2"; b
PRINT "Length of the hypotenuse is" SQR(a * a + b * b)
```

Schließlich kannst du mit dem Befehl INPUT eine Reihe von durch Kommas getrennten Zahlen eingeben, wobei jede Zahl in einer eigenen Variablen gespeichert wird.

Zum Beispiel:

```
INPUT "Enter the length of the two sides: ", a, b
PRINT "Length of the hypotenuse is" SQR(a * a + b * b)
```

Wenn der Benutzer `12,15` eingegeben hat, wird die Zahl `12` in der Variablen `a` und `15` in `b` gespeichert.

Eine weitere Methode, um Eingaben über die Konsole zu erhalten, ist der Befehl LINE INPUT. Dieser erfasst die gesamte vom Benutzer eingegebene Zeile und weist sie einer Zeichenfolgenvariablen zu. Wie beim Befehl INPUT kannst du auch hier eine Eingabeaufforderung festlegen. Hier ein einfaches Beispiel:

```
LINE INPUT "What is your name? ", s$
PRINT "Hello " s$
```

Wir werden uns später in diesem Tutorial mit Zeichenfolgenvariablen befassen, aber vorerst kannst du sie dir als eine Variable vorstellen, die eine Folge von Zeichen enthält. Wenn du das obige Programm ausführst und bei der Eingabeaufforderung `John` eingibst, würde das Programm mit `Hello John` antworten.

Manchmal möchtest du nicht warten, bis der Benutzer die Eingabetaste drückt, sondern jedes Zeichen sofort nach der Eingabe erhalten. Dies ist mit der Funktion INKEY$ möglich, die den Wert des Zeichens als Zeichenkette zurückgibt, die aus nur einem Zeichen besteht, oder eine leere Zeichenkette (d. h. ohne Zeichen), wenn nichts eingegeben wurde.

### GOTO und Labels

Eine Methode zur Steuerung des Programmablaufs ist der GOTO-Befehl. Dieser weist MMBasic im Wesentlichen an, zu einem anderen Teil des Programms zu springen und die Ausführung von dort fortzusetzen. Das Ziel des GOTO ist ein Label, und das muss zunächst erklärt werden.

Ein Label ist ein Bezeichner, der einen Teil des Programms markiert. Es muss das erste Element in der Zeile sein und mit dem Doppelpunkt (:) enden. Der Name, den du verwendest, kann bis zu 31 Zeichen lang sein und muss denselben Regeln folgen wie der Name einer Variablen. In der folgenden Programmzeile ist beispielsweise LoopBack ein Label:

```
LoopBack:  a = a + 1
```

Wenn du den GOTO-Befehl verwendest, um zu diesem bestimmten Teil des Programms zu springen, würdest du den Befehl wie folgt verwenden:

```
GOTO LoopBack
```

Um das Ganze in einen Zusammenhang zu bringen: Das folgende Programm gibt 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 in der nächsten Zeile auf 1 zu erhöhen. Der Wert von `z` wird ausgegeben und anschließend geprüft, ob er kleiner als 10 ist. Ist er kleiner als 10, springt die Programmausführung zurück zum Label `LoopBack`, wo sich der Vorgang wiederholt. Schließlich erreicht der Wert von `z` 10, das Programm läuft bis zum Ende und wird beendet.

Beachte, dass eine FOR-Schleife dasselbe leisten kann (und einfacher ist), daher dient dieses Beispiel lediglich dazu, zu veranschaulichen, was der GOTO-Befehl leisten kann.

In der Vergangenheit hat der GOTO-Befehl einen schlechten Ruf bekommen. Das liegt daran, dass man mit GOTOS ein Programm erstellen kann, das ständig von einem Punkt zum anderen springt (oft als „Spaghetti-Code“ bezeichnet), und diese Art von Programm ist für einen anderen Programmierer fast unmöglich zu verstehen. Durch Konstrukte wie mehrzeilige IF-Anweisungen ist der Bedarf an GOTO-Anweisungen zurückgegangen, und sie sollten nur verwendet werden, wenn es keine andere Möglichkeit gibt, den Programmablauf zu ändern.

### Prüfen auf Primzahlen

Im Folgenden findest du ein einfaches Programm, das viele der zuvor besprochenen Programmierfunktionen vereint.

```
DO
  InpErr:
  PRINT
  INPUT "Enter a number: "; a
  IF a < 2 THEN
    PRINT "Number must be equal or greater than 2"
    GOTO InpErr
  ENDIF

  Divs = 0
  FOR x = 2 TO SQR(a)
    r = a/x
    IF r = FIX(r) THEN Divs = Divs + 1
  NEXT x

  PRINT a " is ";
  IF Divs > 0 THEN PRINT "not ";
  PRINT "a prime number."
LOOP
```

Das fordert dich zuerst (auf der Konsole) zur Eingabe einer Zahl auf und prüft, sobald du sie eingegeben hast, ob es sich um eine Primzahl handelt oder nicht, und zeigt eine entsprechende Meldung an.

Es beginnt mit einer DO-Schleife, die keine Bedingung hat – sie läuft also endlos weiter. Genau das wollen wir. Das bedeutet: Sobald der Benutzer eine Zahl eingegeben hat, wird gemeldet, ob es sich um eine Primzahl handelt oder nicht, und dann wird die Schleife fortgesetzt und um eine weitere Zahl gebeten. Der Benutzer kann das Programm (falls gewünscht) beenden, indem er das Break-Zeichen (normalerweise STRG-C) eingibt.

Das Programm gibt dann eine Eingabeaufforderung für den Benutzer aus, die mit einem Semikolon endet. Das bedeutet, dass der Cursor am Ende der Eingabeaufforderung für den Befehl INPUT stehen bleibt, der die Zahl abruft und in der Variablen a speichert.

Anschließend wird die Zahl geprüft. Ist sie kleiner als 2, wird eine Fehlermeldung ausgegeben und das Programm springt zurück und fragt erneut nach der Zahl.

Jetzt können wir prüfen, ob die Zahl eine Primzahl ist. Das Programm verwendet eine FOR-Schleife, um die möglichen Teiler durchzugehen und zu prüfen, ob jeder einzelne die eingegebene Zahl ohne Rest teilen kann. Bei jedem Durchlauf erhöht das Programm die Variable `Divs`.

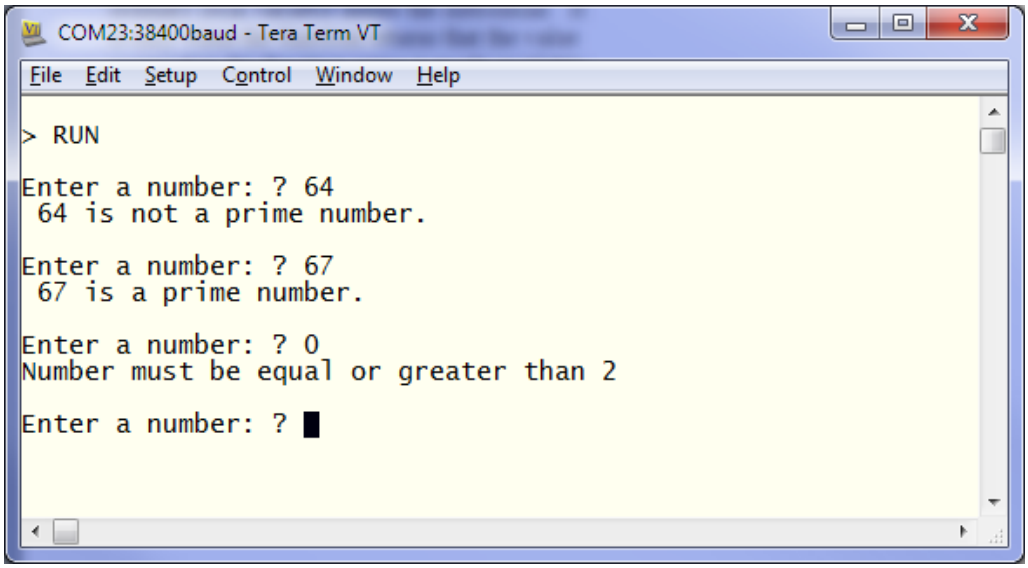
Beachte, dass der Test mit der Funktion FIX(r) durchgeführt wird, die einfach alle Ziffern nach dem Komma entfernt. Die Bedingung `r = FIX(r)` ist also wahr, wenn `r` eine ganze Zahl ist (d. h. keine Ziffern nach dem Komma hat).

Schließlich erstellt das Programm die Meldung für den Benutzer. Der entscheidende Punkt ist: Wenn die Variable `Divs` größer als Null ist, bedeutet das, dass eine oder mehrere Zahlen gefunden wurden, die die Testzahl ohne Rest teilen. In diesem Fall fügt die IF-Anweisung das Wort „nicht“ in die Ausgabemeldung ein.

Wenn die eingegebene Zahl beispielsweise 21 war, erhält der Benutzer folgende Antwort:

```
21 is not a prime number.
```

Das ist das Ergebnis der Programmausführung und ein Teil der Ausgabe:




Du kannst dieses Programm testen, indem du es über den Editor (den Befehl EDIT) eingibst.

Mit deinen neu erworbenen Fähigkeiten könntest du dann versuchen, es effizienter zu gestalten. Da das Programm beispielsweise zählt, wie oft eine Zahl durch die Testzahl teilbar ist, dauert es viel länger als nötig, eine Nicht-Primzahl zu erkennen. Das Programm würde viel effizienter laufen, wenn es bei der ersten Zahl, die sich gleichmäßig teilen lässt, aus der FOR-Schleife springen würde. Du könntest dafür den Befehl GOTO verwenden oder den Befehl EXIT FOR – das würde dazu führen, dass die FOR-Schleife sofort beendet wird.

Weitere Möglichkeiten zur Optimierung wären, die Division nur mit ungeraden Zahlen zu testen (indem du zunächst prüfst, ob die Zahl gerade ist, dann die FOR-Schleife bei 3 startest und STEP 2 verwendest) oder nur Primzahlen für den Test zu verwenden (was allerdings viel komplizierter wäre).

## Arrays


Arrays sind etwas, das du auf den ersten Blick wahrscheinlich nicht als nützlich empfinden wirst, aber wenn du sie tatsächlich brauchst, wirst du sie als sehr praktisch empfinden.

Ein Array lässt sich am besten als eine Reihe von Briefkästen für einen Wohnblock oder eine Wohnanlage vorstellen, wie rechts dargestellt. Die Briefkästen befinden sich alle an derselben Adresse, und jeder Kasten steht für eine Wohnung oder eine Eigentumswohnung an dieser Adresse. Du kannst einen Brief in den Kasten für Wohnung eins, Wohnung zwei usw. legen.

Ähnlich verhält es sich mit einem Array in BASIC: Es handelt sich um eine einzelne Variable mit mehreren Untereinheiten (in BASIC als *Elemente* bezeichnet), die nummeriert sind. Du kannst Daten in Element eins, Element zwei usw. ablegen. In BASIC wird ein Array mit dem Befehl DIM erstellt, zum Beispiel:

```
DIM numarr(300)
```

Dadurch wird ein Array mit dem Namen `numarr` erstellt, das 301 Elemente (stell dir diese als Briefkästen vor) im Bereich von 0 bis 300 enthält. Standardmäßig beginnt ein Array bei Null, daher gibt es ein zusätzliches Element, sodass die Gesamtzahl 301 beträgt. Um ein bestimmtes Element im Array (d. h. einen bestimmten Briefkasten) anzugeben, verwendest du einen Index, der einfach die Nummer des Array-Elements ist, auf das du zugreifen möchtest. Wenn du beispielsweise das Element Nummer 100 in diesem Array auf (sagen wir) die Zahl 876 setzen möchtest, würdest du das so machen:

```
numarr(100) = 876
```

Normalerweise ist der Index eines Arrays keine konstante Zahl wie in diesem Beispiel (d. h. 100), sondern eine Variable, die geändert werden kann, um auf verschiedene Array-Elemente zuzugreifen.

Als Beispiel dafür, wie du ein Array verwenden könntest, stell dir den Fall vor, dass du die Höchsttemperatur für jeden Tag des Jahres aufzeichnen und am Ende des Jahres den Gesamtdurchschnitt berechnen möchtest. Du könntest gewöhnliche Variablen verwenden, um die Temperatur für jeden Tag zu speichern, aber du bräuchtest 365 davon, und das würde dein Programm ziemlich unhandlich machen. Stattdessen könntest du ein Array definieren, um die Werte wie folgt zu speichern:

```
DIM days(365)
```

Jeden Tag müsstest du die Temperatur an der richtigen Stelle im Array speichern. Wenn die Tagesnummer im Jahr in der Variablen `` `doy` `` und die Höchsttemperatur in der Variablen `` `maxtemp` `` gespeichert wäre, würdest du den Messwert wie folgt speichern:

```
days(doy) = maxtemp
```

Am Ende des Jahres wäre es ganz einfach, den Jahresdurchschnitt zu berechnen.
Zum Beispiel:

```
total = 0
FÜR i = 1 bis 365
  total = total + days(i)
```

```
NEXT i
PRINT "Average is:" total / 365
```

Das ist viel einfacher, als 365 einzelne Variablen zu addieren und zu mitteln.

Das obige Array war eindimensional, aber du kannst auch mehrdimensionale Arrays verwenden. Um auf unsere Analogie mit den Briefkästen zurückzukommen: Ein zweidimensionales Array könnte man sich als ein Mehrfamilienhaus mit mehreren Stockwerken vorstellen. Ein solches Haus könnte eine Reihe von vier Briefkästen im ersten Stock haben, eine weitere Reihe von vier Briefkästen im zweiten Stock und so weiter. Um einen Brief in einen Briefkasten zu stecken, musst du die Stockwerksnummer und die Wohnungsnummer in diesem Stockwerk angeben.

In BASIC wird ein solches Array mit zwei durch ein Komma getrennten Indizes angegeben. Zum Beispiel:

```
LetterBox(floor, unit)
```

Nehmen wir als praktisches Beispiel an, du möchtest die Höchsttemperatur für jeden Tag über einen Zeitraum von fünf Jahren aufzeichnen. Dazu könntest du 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 angibt. Wenn du für Tag 100 im Jahr 3 eine Temperatur von 24 Grad festlegen möchtest, würdest du das so machen:

```
days(100, 3) = 24
```

In MMBasic für die PicoMite-Firmware kannst du mit dem RP2040-Prozessor bis zu sechs Dimensionen oder mit dem RP2350-Prozessor bis zu fünf Dimensionen verwenden. Die Größe eines Arrays ist nur durch die Menge des verfügbaren freien RAMs begrenzt.

### Ganzzahlen / Integers

Bisher waren alle Zahlen und Variablen, die wir verwendet haben, Gleitkommazahlen. Wie bereits erklärt, sind Gleitkommazahlen praktisch, da sie Stellen nach dem Komma berücksichtigen und bei Divisionen ein sinnvolles Ergebnis liefern. Wenn du also einfach nur deine Aufgaben erledigen willst und dich nicht um die Details kümmerst, solltest du bei Gleitkommazahlen bleiben.

Die Einschränkung von Gleitkommazahlen besteht jedoch darin, dass sie in der PicoMite-Firmware als Annäherungswerte mit einer Genauigkeit von 14 Stellen gespeichert werden. Meistens ist diese Eigenschaft von Gleitkommazahlen kein Problem, aber es gibt Fälle, in denen du größere Zahlen genau speichern musst.

Nehmen wir zum Beispiel an, du möchtest die Zeit auf die Mikrosekunde genau bearbeiten , damit du zwei verschiedene Datums-/Zeitangaben vergleichen kannst, um herauszufinden, welche früher liegt. Der einfache Weg, dies zu tun, besteht darin, das Datum/die Uhrzeit in die Anzahl der Mikrosekunden seit einem bestimmten Datum (z. B. dem 1. Januar im Jahr Null) umzuwandeln – dann ist das Ermitteln der früheren der beiden nur eine Frage der Verwendung eines arithmetischen Vergleichs in einer IF-Anweisung.

Das Problem ist, dass die Anzahl der Mikrosekunden seit diesem Datum den Genauigkeitsbereich von Gleitkommavariablen überschreitet, und hier kommen ganzzahlige Variablen ins Spiel. Eine ganzzahlige Variable kann sehr große Zahlen bis zu neun Millionen Millionen Millionen (oder ±9223372036854775807, um genau zu sein) genau speichern.

Der Nachteil bei der Verwendung einer Ganzzahl ist, dass sie keine Bruchteile (d. h. Zahlen hinter dem Komma) speichern kann. Jede Berechnung, die ein Bruchteil als Ergebnis liefert, wird auf- oder abgerundet auf die nächste ganze Zahl, wenn sie einer Ganzzahl zugewiesen wird. Diese Eigenschaft kann jedoch praktisch sein, wenn es um Geld geht – du möchtest zum Beispiel niemandem eine Rechnung über 100,13333333333 $ schicken.

Es ist einfach, eine Ganzzahlvariable zu erstellen: Füge einfach das Prozentzeichen (%) als Suffix an den Variablennamen an. Zum Beispiel ist `sec%` eine Ganzzahlvariable. Innerhalb eines Programms kannst du Ganzzahlen und Gleitkommazahlen mischen, und MMBasic führt die notwendigen Umrechnungen durch; wenn du jedoch die volle Genauigkeit von Ganzzahlen beibehalten möchtest, solltest du es vermeiden, beide zu mischen.

Genau wie bei Gleitkommazahlen kannst du Arrays aus Ganzzahlen haben; du musst lediglich das Prozentzeichen als Suffix an den Array-Namen anhängen. Zum Beispiel: `days%(365, 5)`.

Anfänger sind oft verwirrt, wann sie Gleitkommazahlen oder Ganzzahlen verwenden sollen, und die Antwort ist einfach… verwende immer Gleitkommazahlen, es sei denn, du benötigst eine extrem hohe Genauigkeit. Das kommt nicht oft vor, aber wenn du sie brauchst, wirst du feststellen, dass Ganzzahlen sehr nützlich sind.

### Zeichenketten / Strings

Zeichenketten sind ein weiterer Variablentyp (wie Gleitkommazahlen und Ganzzahlen). Zeichenketten dienen dazu, eine Folge von Zeichen zu speichern. Zum Beispiel im Befehl:

```
PRINT "Hello"
```

ist die Zeichenkette „Hello“ eine Zeichenkettenkonstante. Beachte, dass eine Konstante etwas ist, das sich nicht ändert (im Gegensatz zu einer Variablen, die sich ändern kann), und dass Zeichenkettenkonstanten immer in doppelte Anführungszeichen gesetzt werden.

Die Namen von Zeichenfolgenvariablen verwenden das Dollarzeichen ($) als Suffix, um sie als Zeichenfolge und nicht als normale Gleitkommavariable zu kennzeichnen, und du kannst ihren Wert mit einer gewöhnlichen Zuweisung festlegen. Hier sind einige Beispiele (beachte, dass das zweite Beispiel ein Array von Zeichenfolgen verwendet):

```
Car$ = "Holden"
Country$(12) = "India"
Name$ = "Fred"
```

Du kannst Zeichenfolgen auch mit dem Plus-Operator verknüpfen:

```
Word1$ = "Hello"
Word2$ = "World"
Greeting$ = Word1$ + " " + Word2$
```

In diesem Fall lautet der Wert von `Greeting$ "Hello World"`.

Zeichenfolgen können auch mit Operatoren wie = (gleich), <> (ungleich), < (kleiner als) usw. verglichen werden. Zum Beispiel:

```
IF Car$ = "Holden" THEN PRINT "Was an Aussie made car"
```

Der Vergleich erfolgt unter Verwendung des vollständigen ASCII-Zeichensatzes, sodass vor einem druckbaren Zeichen ein Leerzeichen steht. Außerdem wird beim Vergleich die Groß-/Kleinschreibung beachtet, sodass „holden“ nicht mit „Holden“ übereinstimmt. Mit der Funktion UCASE(), die die Zeichenfolge in Großbuchstaben umwandelt, kannst du einen Vergleich durchführen, bei dem die Groß-/Kleinschreibung keine Rolle spielt. Zum Beispiel:

```
IF UCASE$(Car$) = "HOLDEN" THEN PRINT "Was an Aussie made car"
```

Du kannst Arrays aus Zeichenfolgen verwenden, musst aber bei der Deklaration vorsichtig sein, da dir schnell der RAM (der allgemeine Speicher, der zum Speichern von Variablen usw. verwendet wird) ausgehen kann. Das liegt daran, dass MMBasic standardmäßig 255 Byte RAM für jedes Element des Arrays zuweist. Ein Zeichenfolgen-Array mit 100 Elementen verbraucht beispielsweise standardmäßig 25 KB RAM.

Um das zu vermeiden, kannst du den LENGTH-Qualifizierer verwenden, um die maximale Größe jedes Elements zu begrenzen. Wenn du zum Beispiel weißt, dass die maximale Länge jeder

Zeichenkette, die im Array gespeichert wird, weniger als 20 Zeichen beträgt, kannst du die folgende Deklaration verwenden, um für jedes Element nur 20 Bytes zuzuweisen:

```
DIM MyArray$(100) LENGTH 20
```

Das resultierende Array belegt dann nur 2 KB RAM.

Du kannst den LENGTH-Qualifier bei der Deklaration einer normalen (nicht als Array definierten) Zeichenfolgenvariablen verwenden und sparst damit 256 Byte RAM, wenn die Länge 9 oder weniger (RP2040) bzw. 15 oder weniger (RP2350) beträgt.

### Bearbeitung von Zeichenketten

Die Zeichenfolgenverarbeitung ist eine der Stärken von MMBasic, und mit ein paar einfachen Funktionen kannst du Zeichenfolgen zerlegen und allgemein bearbeiten. Die grundlegenden Zeichenfolgenfunktionen sind:

| | |
|---|---|
| LEFT$(string$, nbr ) | Gibt eine Teilzeichenfolge von *string$* mit *nbr* Zeichen ab dem linken Anfang (Anfang) der Zeichenkette zurück. |
| RIGHT$(string$, nbr ) | Wie oben, gibt jedoch *nbr* Zeichen vom rechten Ende (Ende) der Zeichenkette zurück. |
| MID$(string$, pos, nbr ) | Gibt eine Teilzeichenfolge von *string$* mit *nbr* Zeichen zurück, beginnend mit dem Zeichen *pos* in der Zeichenfolge (d. h. in der Mitte der Zeichenfolge). |

Zum Beispiel, wenn `S$ = "This is a string"`

| | | |
|---|---|---|
| dann: | `R$ = LEFT$(S$, 7)` | würde dazu führen, dass der Wert von R$ auf „This is“ gesetzt wird |
| und: | `R$ = RIGHT$(S$, 8)` | würde dazu führen, dass der Wert von R$ auf „a string“ gesetzt wird |
| final: | `R$ = MID$(S$, 6, 2)` | würde dazu führen, dass der Wert von R$ auf „is“ gesetzt wird |

Beachte, 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 Anfang des Wortes „is“.

Eine weitere nützliche Funktion ist:

| | |
|---|---|
| INSTR(string$, pattern$ ) | Gibt eine Zahl zurück, die die Position angibt, an der *pattern$* in string$ vorkommt. |

Dies kann verwendet werden, um innerhalb einer Zeichenfolge nach einer anderen Zeichenfolge zu suchen. Die zurückgegebene Zahl ist die Position der Teilzeichenfolge innerhalb der Hauptzeichenfolge. Wie bei MID$() ist der Anfang der Zeichenfolge die Position 1.

Zum Beispiel, wenn `S$ = "This is a string"`

Dann: `pos = INSTR(S$, " ")`

würde dazu führen, dass `pos` auf die Position des ersten Leerzeichens in S$ gesetzt wird (d. h. 5).

INSTR() lässt sich mit anderen Funktionen kombinieren, 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 angibt, an der *pattern$* in *string$* vorkommt, wenn die Suche an der Zeichenposition *pos* beginnt. |

So können wir das zweite Wort in S$ mit folgendem Befehl finden:

```
pos1 = INSTR(S$, " ")
pos2 = INSTR(pos1 + 1, S$, " ")
r$ = MID$(S$, pos1 + 1, pos2 - pos1)
```

Dieses letzte Beispiel ist ziemlich kompliziert, daher lohnt es sich vielleicht, es im Detail durchzugehen, damit du verstehst, wie es funktioniert.

Beachte, dass INSTR() die Zahl Null zurückgibt, wenn die Teilzeichenfolge nicht gefunden wird, und dass jede Zeichenfolgenfunktion einen Fehler auslöst (und das Programm abbricht), wenn sie als Zeichenposition verwendet wird. In einem praktischen Programm würdest du also zuerst prüfen, ob INSTR() Null zurückgibt, bevor du diesen Wert verwendest.

Zum Beispiel:

```
r$ = ""
pos1 = INSTR(S$, " ")
if pos1 > 0 THEN
  pos2 = INSTR(pos1 + 1, S$, " ")
  if pos2 > 0 THEN r$ = MID$(S$, pos1 + 1, pos2 - pos1)
ENDIF
```

### Wissenschaftliche Notation

Bevor wir das Thema Datentypen abschließen, müssen wir noch auf Gleitkommazahlen und die wissenschaftliche Schreibweise eingehen.

Die meisten Zahlen lassen sich ganz normal schreiben, zum Beispiel 11 oder 24,5, aber sehr große oder sehr kleine Zahlen sind schwieriger. Schätzungen zufolge gibt es auf der Erde beispielsweise 7500000000000000000 Sandkörner. Das Problem bei dieser Zahl ist, dass man leicht den Überblick darüber verliert, wie viele Nullen darin enthalten sind, und es daher schwierig ist, sie mit einer ähnlich großen Zahl zu vergleichen.

Ein Wissenschaftler würde diese Zahl als $7{,}5 \times 10^{18}$ schreiben, was als wissenschaftliche Notation bezeichnet wird und viel leichter zu verstehen ist.

Bei Verwendung des Befehls PRINT verwendet MMBasic automatisch die wissenschaftliche Schreibweise, wenn sehr große oder sehr kleine Gleitkommazahlen ausgegeben werden. Wenn beispielsweise die oben genannte Zahl in einer Gleitkommavariablen gespeichert wäre, würde der Befehl PRINT sie als `7,5E+18` anzeigen (so stellt BASIC 7,5 $x^{10^{18}}$ dar). Ein weiteres Beispiel: Die Zahl 0,0000000456 würde als `4,56E-8` angezeigt werden, was dem Wert 4,56 x $10^{-8}$ entspricht.

Du kannst die wissenschaftliche Notation auch verwenden, wenn du konstante Zahlen in MMBasic eingibst. Zum Beispiel:

```
SandGrains = 7,5E+18
```

MMBasic verwendet die wissenschaftliche Notation nur für Gleitkommazahlen (nicht für Ganzzahlen). Wenn du beispielsweise die Anzahl der Sandkörner einer Ganzzahlvariablen zuweisen würdest, würde der PRINT-Befehl sie als normale Zahl anzeigen (mit vielen Nullen).

### DIM-Befehl

Wir haben den DIM-Befehl bereits zur Definition von Arrays verwendet, aber er kann auch zum Erstellen gewöhnlicher Variablen genutzt werden. Beispielsweise kannst du auf diese Weise gleichzeitig vier Zeichenfolgenvariablen erstellen:

```
DIM STRING Car, Name, Street, City
```

Beachte, dass wir, da diese Variablen mit dem DIM-Befehl als Zeichenfolgen definiert wurden, das Suffix $ nicht benötigen; die Definition allein reicht aus, damit MMBasic ihren Typ erkennt. Ebenso brauchst du das Typ-Suffix nicht, wenn du diese Variablen in einem Ausdruck verwendest: Zum Beispiel:

```
City = "Sydney"
```

Du kannst auch das Schlüsselwort INTEGER verwenden, um eine Reihe von Ganzzahlvariablen zu definieren, und FLOAT, um dasselbe für Gleitkommavariablen zu tun. Diese Art der Notation kann in ähnlicher Weise zur Definition von Arrays verwendet werden.

Zum Beispiel:

```
DIM INTEGER seconds(200)
```

Eine weitere Methode zur Definition des Variablentyps ist die Verwendung des Schlüsselworts AS. Zum Beispiel:

```
DIM Car AS STRING, Name AS STRING, Street AS STRING
```

Dies ist die von Microsoft verwendete Methode (MMBasic versucht, die Kompatibilität mit Microsoft zu gewährleisten) und ist nützlich, wenn die Variablen unterschiedliche Typen haben. Zum Beispiel:

```
DIM Car AS STRING, Age AS INTEGER, Value AS FLOAT
```

Du kannst jede dieser Methoden zur Definition des Variablentyps verwenden, sie funktionieren alle gleich.

Der Vorteil der Variablendefinition mit dem Befehl DIM besteht darin, dass die Variablen klar definiert sind (vorzugsweise am Anfang des Programms) und ihr Typ (Float, Integer oder String) nicht missverstanden werden kann.

Du kannst dies noch verstärken, indem du ganz am Anfang deines Programms die folgenden Befehle verwendest:

```
OPTION EXPLICIT
OPTION DEFAULT NONE
```

Der erste weist MMBasic an, dass alle Variablen vor ihrer Verwendung explizit mit DIM definiert werden müssen. Der zweite legt fest, dass der Typ aller Variablen bei ihrer Erstellung angegeben werden muss.

Warum sind diese beiden Befehle wichtig?

Der erste kann helfen, einen häufigen Programmierfehler zu vermeiden, bei dem du versehentlich den Namen einer Variablen falsch schreibst. Beispielsweise könnte dein Programm die aktuelle Temperatur in einer Variablen namens `Temp` speichern, aber an einer Stelle schreibst du sie versehentlich als `Tmp`. Dies führt dazu, dass MMBasic automatisch eine Variable namens `Tmp` erstellt und deren Wert auf Null setzt.

Das ist natürlich nicht das, was du willst, und es führt zu einem subtilen Fehler, der schwer zu finden sein könnte, selbst wenn dir bewusst wäre, dass etwas nicht stimmt. Wenn du hingegen den Befehl OPTION EXPLICIT am Anfang deines Programms verwendest, würde MMBasic die automatische Erstellung der Variablen verweigern und stattdessen eine Fehlermeldung anzeigen, was dir wahrscheinlich viel Kopfzerbrechen ersparen würde.

Der Befehl OPTION DEFAULT NONE ist ebenfalls hilfreich, da er MMBasic mitteilt, dass der Programmierer den Typ jeder Variablen bei der Deklaration ausdrücklich angeben muss. Es passiert leicht, die Angabe des Typs zu vergessen, und wenn man MMBasic den Typ automatisch annehmen lässt, kann das zu unerwarteten Folgen führen.

Bei kleinen, schnell geschriebenen Programmen ist es in Ordnung, MMBasic die automatische Erstellung von Variablen zu überlassen, aber bei größeren Programmen solltest du diese Funktion immer mit OPTION EXPLICIT deaktivieren und durch OPTION DEFAULT NONE verstärken.

Wenn eine Variable angelegt wird, wird sie bei Float- und Integer-Typen auf Null gesetzt und bei String-Variablen auf eine leere Zeichenkette (d. h. ohne Zeichen) . Du kannst ihren Anfangswert bei der Anlage mit DIM auf einen anderen Wert setzen.

Zum Beispiel:

```
DIM FLOAT nbr = 12.56
DIM STRING Car = "Ford", City = "Perth"
```

Du kannst Arrays auch initialisieren, indem du die Initialisierungswerte wie folgt in Klammern setzt:

```
DIM s$(2) = ("zero", "one", "two")
```

Beachte, dass Arrays standardmäßig bei Null beginnen; dieses Array hat also tatsächlich drei Elemente mit den Indexnummern 0, 1 und 2. Deshalb brauchten wir drei String-Konstanten, um es zu initialisieren.

### Konstanten

Eine häufige Anforderung beim Programmieren ist es, einen Bezeichner zu definieren, der einen Wert repräsentiert, ohne dass die Gefahr besteht, dass dieser Wert versehentlich geändert wird – was passieren kann, wenn Variablen für diesen Zweck verwendet werden. Diese werden als Konstanten bezeichnet und können I/O-Pin-Nummern, Signalgrenzen, mathematische Konstanten und so weiter darstellen.

Du kannst eine Konstante mit dem Befehl CONST erstellen. Dadurch wird ein Bezeichner definiert, der sich wie eine Variable verhält, aber auf einen Wert gesetzt ist, der nicht geändert werden kann.

Wenn du beispielsweise die Spannung einer an Pin 31 angeschlossenen Batterie überprüfen möchtest, könntest du die entsprechenden Werte wie folgt definieren:

```
CONST BatteryVoltagePin = 31
CONST BatteryMinimum = 1.5
```

Diese Konstanten können dann im Programm verwendet werden, wo sie für den Laienleser verständlicher sind als einfache Zahlen. Zum Beispiel:

```
SETPIN BatteryVoltagePin, AIN
IF PIN(BatteryVoltagePin) < BatteryMinimum THEN SoundAlarm
```

Es ist eine gute Programmierpraxis, Konstanten für alle festen Zahlen zu verwenden, die einen wichtigen Wert darstellen. Normalerweise werden sie am Anfang eines Programms definiert, wo sie gut sichtbar und für einen anderen Programmierer bequem zu finden sind, um sie (falls nötig) anzupassen.

### Unterprogramme

Eine Subroutine ist ein Block von Programmcode, der in sich geschlossen ist (wie ein Modul) und von überall in deinem Programm aufgerufen werden kann. Für dein Programm sieht sie wie ein integrierter MMBasic-Befehl aus und kann genauso verwendet werden. Angenommen, du benötigst einen Befehl, der einen Fehler signalisiert, indem er eine Meldung auf der Konsole ausgibt. Du könntest die Subroutine wie folgt definieren:

```
SUB ErrMsg
  PRINT „Fehler erkannt“
END SUB
```

Wenn diese Subroutine in dein Programm eingebettet ist, musst du nur den Befehl ErrMsg verwenden, wann immer du die Meldung anzeigen möchtest. Zum Beispiel:

```
IF A < B THEN ErrMsg
```

Die Definition einer Subroutine kann an beliebiger Stelle im Programm stehen, befindet sich jedoch in der Regel am Ende. Wenn MMBasic während der Ausführung deines Programms auf die Definition stößt, überspringt es diese einfach.

Das obige Beispiel ist zwar ausreichend, aber es wäre besser, wenn eine nützlichere Meldung angezeigt werden könnte, die bei jedem Aufruf der Subroutine individuell angepasst werden kann. Dies lässt sich erreichen, indem man der Subroutine eine Zeichenkette als Argument (manchmal auch Parameter genannt) übergibt.

In diesem Fall würde die Definition der Subroutine wie folgt aussehen:

```
SUB ErrMsg  Msg$
  PRINT "Fehler: " + Msg$
END SUB
```

Wenn du dann die Subroutine aufrufst, kannst du die Zeichenkette, die ausgegeben werden soll, in der Befehlszeile der Subroutine angeben.

Zum Beispiel:

```
IF A < B THEN ErrMsg "Zahl zu klein"
```

Wenn die Subroutine so aufgerufen wird, wird die Meldung „`Fehler: Zahl zu` klein“ auf der Konsole ausgegeben. Innerhalb der Subroutine hat `Msg$` bei einem solchen Aufruf den Wert „Zahl zu klein“ und wird in der PRINT-Anweisung an die vollständige Fehlermeldung angehängt.

Eine Subroutine kann beliebig viele Argumente haben, die vom Typ Float, Integer oder String sein können, wobei jedes Argument durch ein Komma getrennt wird.

Innerhalb der Subroutine verhalten sich die Argumente wie gewöhnliche Variablen, existieren jedoch nur innerhalb der Subroutine und verschwinden, sobald die Subroutine endet. Du kannst im Hauptprogramm Variablen mit demselben Namen haben; diese bleiben innerhalb der Subroutine verborgen und unterscheiden sich von den für die Subroutine definierten Argumenten.

Der Typ des zu übergebenden Arguments kann mit einem Typ-Suffix angegeben werden (z. B. $, % oder ! für Zeichenkette, Ganzzahl und Gleitkomma). Im folgenden Beispiel muss das erste Argument eine Zeichenkette und das zweite eine Ganzzahl sein:

```
SUB MySub Msg$, Nbr%
  …
END SUB
```

MMBasic konvertiert die übergebenen Werte, sofern dies möglich ist. Wenn dein Programm also einen Gleitkommawert als zweites Argument übergibt, wandelt MMBasic diesen in eine Ganzzahl um. Wenn MMBasic den Wert nicht konvertieren kann, wird eine Fehlermeldung angezeigt und die Eingabeaufforderung wieder aufgerufen. Wenn du beispielsweise eine Zeichenkette als zweites Argument übergeben hast, bricht dein Programm mit einer Fehlermeldung ab.

Du musst die Typ-Suffixe nicht verwenden; stattdessen kannst du den Typ der Argumente mit dem Schlüsselwort AS definieren, ähnlich wie es im DIM-Befehl verwendet wird.

Das folgende Beispiel ist beispielsweise identisch mit dem obigen Beispiel:

```
SUB MySub Msg AS STRING, Nbr AS INTEGER
  …
END SUB
```

Wenn du im gesamten Programm nur einen Variablentyp verwendest und diesen mit OPTION DEFAULT festlegst, kannst du die Frage nach den Variablentypen natürlich komplett ignorieren.

Wenn eine Subroutine mit einem Argument aufgerufen wird, das eine Variable ist (d. h. keine Konstante oder kein Ausdruck), erstellt MMBasic innerhalb der Subroutine eine entsprechende Variable, *die auf diese Variable verweist.* Jede Änderung an der Variable, die das Argument innerhalb der Subroutine repräsentiert, ändert auch die im Aufruf verwendete Variable. Dies wird als Übergabe von Argumenten per Referenz bezeichnet.

Das lässt sich am besten anhand eines Beispiels erklären:

```
DIM MyNumber = 5       ' set the variable to 5
CalcSquare MyNumber    ' the subroutine will square its value
PRINT MyNumber         ' this will print the number 25
END

SUB CalcSquare nbr
  nbr = nbr * nbr      ' square the argument and pass it back
END SUB
```

Die Subroutine „CalcSquare“ nimmt ihr Argument, quadriert es und schreibt das Ergebnis zurück in die Variable, die das Argument repräsentiert (`nbr`). Da die Subroutine mit einer Variablen

(MyNumber) aufgerufen wurde, verweist die Variable nbr auf MyNumber, und jede Änderung an nbr wirkt sich entsprechend auch auf MyNumber aus. Daher gibt die PRINT-Anweisung 25 aus.

Die Übergabe von Argumenten per Referenz ist praktisch, da sie es einer Subroutine ermöglicht, Werte an den aufrufenden Code zurückzugeben. Es könnte jedoch zu Problemen führen, wenn eine Subroutine die Variable, die ein Argument repräsentiert, als allgemeine Variable verwendet und ihren Wert ändert. Wenn sie dann mit einer Variablen als Argument aufgerufen würde, würde diese Variable unbeabsichtigt geändert werden. Um dies zu vermeiden, solltest du ihrer Definition das Schlüsselwort BYVAL voranstellen. Dies weist MMBasic an, immer den Wert des Arguments zu verwenden, selbst wenn es sich um eine Variable handelt, und niemals auf die im Aufruf verwendete Variable zurückzuverweisen.

Wenn du eine Subroutine aufrufst, kannst du einige (oder alle) Parameter weglassen, und sie nehmen den Wert Null (bei Fließkommazahlen oder Ganzzahlen) oder eine leere Zeichenkette an. Das ist praktisch, da deine Subroutine erkennen kann, ob ein Parameter fehlt, und entsprechend reagieren kann.

Hier ist zum Beispiel unsere Subroutine zum Generieren einer Fehlermeldung, aber diese Version kann auch verwendet werden, ohne eine Fehlermeldung als Parameter anzugeben:

```
SUB ErrMsg  Msg$
  IF Msg$ = "" THEN
    PRINT „Fehler erkannt“
  ELSE
    PRINT "Fehler: " + Msg$
  ENDIF
END SUB
```

Innerhalb einer Subroutine kannst du die meisten Funktionen von MMBasic nutzen, darunter das Aufrufen anderer Subroutinen, IF…THEN-Befehle, FOR…NEXT-Schleifen und so weiter. Eine Sache, die du jedoch nicht tun kannst, ist, mit GOTO aus einer Subroutine herauszuspringen (wenn du das tust, ist das Ergebnis undefiniert und könnte dir graue Haare bescheren).

Normalerweise wird die Subroutine beendet, wenn der Befehl END SUB erreicht wird, aber du kannst die Subroutine auch vorzeitig mit dem Befehl EXIT SUB beenden.

## Funktionen

Funktionen ähneln Unterprogrammen, wobei der Hauptunterschied darin besteht, dass eine Funktion dazu dient, einen Wert in einem Ausdruck zurückzugeben. Wenn du beispielsweise eine Funktion haben möchtest, die eine Temperatur von Grad Celsius in Fahrenheit umrechnet, könntest du Folgendes definieren:

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

Dann könntest du sie in einem Ausdruck verwenden:

```
Input "Enter a temperature in Celsius: ", t
PRINT "That is the same as" Fahrenheit(t) "F"
```

Oder als weiteres Beispiel:

```
IF Fahrenheit(temp) <= 32 THEN PRINT "Freezing"
```

Du könntest auch das Gegenteil definieren:

```
FUNCTION Celsius(F)
  Celsius = (F - 32) * 0.5556
END FUNCTION
```

Wie du siehst, wird der Funktionsname innerhalb der Subroutine als gewöhnliche lokale Variable verwendet. Erst wenn die Funktion zurückkehrt, wird der Wert für den Ausdruck verfügbar, der sie aufgerufen hat.

Die Regeln für die Argumentliste in einer Funktion ähneln denen für Unterprogramme. Der einzige Unterschied besteht darin, dass beim Aufruf einer Funktion die Argumentliste immer in Klammern gesetzt werden muss, auch wenn keine Argumente vorhanden sind (beim Aufruf eines Unterprogramms sind Klammern optional).

Um einen Wert aus der Funktion zurückzugeben, weist du dem Funktionsnamen innerhalb der Funktion einen Wert zu. Wenn der Funktionsname mit einem Typ-Suffix endet (d. h. $, %, oder !), gibt die Funktion diesen Typ (Zeichenkette, Ganzzahl oder Gleitkomma) zurück; andernfalls gibt sie den Wert zurück, auf den OPTION DEFAULT gesetzt ist. Die folgende Funktion gibt beispielsweise eine Zeichenkette zurück:

```
FUNCTION LVal$(nbr)
  IF nbr = 0 THEN LVal$ = "False" ELSE LVal$ = "True"
END FUNCTION
```

Du kannst den Typ der Funktion explizit mit dem Schlüsselwort AS angeben; dann musst du kein Typ-Suffix verwenden (ähnlich wie beim Definieren einer Variablen mit DIM).

Hier ist das obige Beispiel, umgeschrieben, um diese Funktion zu nutzen:

```
FUNKTION LVal(nbr) AS STRING
  IF nbr = 0 THEN LVal = "False" ELSE LVal = "True"
END FUNCTION
```

In diesem Fall ist der von der Funktion `LVal` zurückgegebene Typ eine Zeichenkette.

Wie bei Unterprogrammen kannst du die meisten Funktionen von MMBasic innerhalb von Funktionen nutzen. Dazu gehören FOR…NEXT-Schleifen, der Aufruf anderer Funktionen und Unterprogramme usw. Außerdem kehrt die Funktion zu dem Ausdruck zurück, der sie aufgerufen hat, wenn der Befehl END FUNCTION erreicht wird, aber du kannst auch vorzeitig zurückkehren, indem du den Befehl EXIT FUNCTION verwendest.

### Lokale Variablen

Variablen, die mit DIM erstellt oder einfach automatisch angelegt werden, heißen *globale* Variablen. Das bedeutet, dass sie überall im Programm sichtbar sind und verwendet werden können, auch innerhalb von Unterprogrammen und Funktionen. Innerhalb eines Unterprogramms oder einer Funktion musst du jedoch oft Variablen für verschiedene Aufgaben verwenden, die nur innerhalb des Unterprogramms/der Funktion relevant sind. In portierbarem Code möchtest du nicht, dass der Name, den du für eine solche Variable wählst, mit einer globalen Variable gleichen Namens kollidiert. Zu diesem Zweck kannst du eine Variable mit dem Befehl LOCAL innerhalb der Unterroutine/Funktion definieren.

Die Syntax für LOCAL ist identisch mit dem Befehl DIM, das heißt, die Variable kann ein Array sein, du kannst den Typ der Variable festlegen und sie mit einem Wert initialisieren.

Hier ist zum Beispiel unsere ErrMsg-Subroutine, die diesmal jedoch erweitert wurde, um eine lokale Variable zum Verknüpfen der Fehlermeldungs-Strings zu verwenden.

```
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 sie (genau wie die Argumentliste) nur innerhalb der Subroutine existiert und verschwindet, sobald die Subroutine beendet wird. Du kannst in deinem Hauptprogramm eine globale Variable namens `tstr`

haben, die sich von der Variablen `tstr` in der Subroutine unterscheidet (in diesem Fall wird die globale `tstr` innerhalb der Subroutine ausgeblendet).

Du solltest für Operationen innerhalb deiner Subroutine oder Funktion immer lokale Variablen verwenden, da sie dazu beitragen, das Modul wesentlich eigenständiger und portabler zu machen.

### Statische Variablen

Lokale Variablen werden bei jedem Start der Subroutine oder Funktion auf ihre Anfangswerte (normalerweise Null oder eine leere Zeichenkette) zurückgesetzt. Es gibt jedoch Fälle, in denen du möchtest, dass die Variable ihren Wert zwischen den Aufrufen beibehält. Diese Art von Variable wird mit dem Befehl STATIC definiert.

Wir können zeigen, wie nützlich STATIC-Variablen sind, indem wir die ErrMsg-Subroutine erweitern, um zu verhindern, dass bei wiederholten Aufrufen der Subroutine immer wieder dieselbe Meldung angezeigt wird. Unser Programm könnte diese Subroutine beispielsweise von mehreren Stellen aus aufrufen, aber wenn die Meldung bei mehreren aufeinanderfolgenden Aufrufen dieselbe ist, möchten wir die Meldung nur einmal sehen.

Das 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 den Überblick über die zuletzt angezeigte Meldung zu behalten, verwenden wir eine statische Variable namens `lastmsg`. Diese speichert den Text der letzten Meldung, und wir können ihn mit dem aktuellen Meldungstext vergleichen, um festzustellen, ob er sich unterscheidet und daher ausgegeben werden sollte. So wird bei jedem Aufruf mit doppeltem Meldungstext nur eine einzige Meldung ausgegeben.

Der Befehl STATIC verwendet genau dieselbe Syntax wie DIM. Das bedeutet, dass du verschiedene Arten von statischen Variablen definieren kannst, einschließlich Arrays, und dass du sie auch mit einem bestimmten Wert initialisieren kannst.

Die statische Variable wird beim ersten Auftreten des Befehls STATIC erstellt und automatisch auf Null (bei einem Float oder Integer) oder eine leere Zeichenkette gesetzt. Bei nachfolgenden Aufrufen der Subroutine oder Funktion erkennt MMBasic, dass die Variable bereits erstellt wurde, und lässt ihren Wert unverändert (d. h. so, wie er beim vorherigen Aufruf war). Wie bei DIM kannst du eine statische Variable auch mit einem bestimmten Wert initialisieren. Zum Beispiel:

```
STATIC INTEGER var = 123
```

Beim ersten Aufruf (wenn die Variable erstellt wird) wird sie auf 123 initialisiert, bei nachfolgenden Aufrufen behält sie jedoch den Wert bei, der zuvor festgelegt wurde.

Meistens werden statische Variablen verwendet, um den *Zustand* von etwas innerhalb einer Subroutine oder Funktion zu verfolgen. Ein *Zustand* ist eine Aufzeichnung von etwas, das zuvor geschehen ist.

Beispiele hierfür sind:

- Wurde der COM-Port bereits geöffnet?
- Welche Schritte in einer Sequenz haben wir bereits abgeschlossen?
- Welcher Text wurde bereits angezeigt?

Normalerweise verwendest du globale Variablen (die mit DIM erstellt werden), um einen *Zustand* zu verfolgen, aber manchmal möchtest du, dass dieser innerhalb eines Moduls enthalten ist, und genau hier sind statische Variablen nützlich. Genau wie LOCAL trägt die Verwendung von STATIC dazu bei, deine Unterprogramme und Funktionen eigenständiger und portabler zu machen.

### Tage berechnen

Wir haben in diesem Tutorial bisher viele Programmierbefehle und Techniken behandelt, und bevor wir zum Ende kommen, lohnt es sich, ein Beispiel dafür zu geben, wie sie zusammenwirken. Das folgende Beispiel nutzt viele Funktionen der Sprache BASIC, um die Anzahl der Tage zwischen zwei Daten zu berechnen:

```
' Example program to calculate the number of days between two dates

OPTION EXPLICIT
OPTION DEFAULT NONE

DIM STRING s
DIM FLOAT d1, d2

DO
  ' main program loop
  PRINT : PRINT "Enter the date as  dd mmm yyyy"
  PRINT " First date";
  INPUT s
  d1 = GetDays(s)
  IF d1 = 0 THEN PRINT "Invalid date!" : CONTINUE DO
  PRINT "Second date";
  INPUT s
  d2 = GetDays(s)
  IF d2 = 0 THEN PRINT "Invalid date!" : CONTINUE DO
  PRINT "Difference is" ABS(d2 - d1) " days"
LOOP

' Calculate the number of days since 1/1/1900
FUNCTION GetDays(d$) AS FLOAT
  LOCAL STRING Month(11) =
("jan","feb","mar","apr","may","jun","jul","aug","sep","oct","nov","dec")
  LOCAL FLOAT Days(11) = (0,31,59,90,120,151,181,212,243,273,304,334)
  LOCAL FLOAT day, mth, yr, s1, s2

  ' Find the separating space character within a date
  s1 = INSTR(d$, " ")
  IF s1 = 0 THEN EXIT FUNCTION
  s2 = INSTR(s1 + 1, d$, " ")
  IF s2 = 0 THEN EXIT FUNCTION

  ' Get the day, month and year as numbers
  day = VAL(MID$(d$, 1, s2 - 1)) - 1
  IF day < 0 OR day > 30 THEN EXIT FUNCTION
  FOR mth = 0 TO 11
    IF LCASE$(MID$(d$, s1 + 1, 3)) = Month(mth) THEN EXIT FOR
  NEXT mth
  IF mth > 11 THEN EXIT FUNCTION
  yr = VAL(MID$(d$, s2 + 1)) - 1900
  IF yr < 1 OR yr >= 200 THEN EXIT FUNCTION

  ' Calculate the number of days including adjustment for leap years
  GetDays = (yr * 365) + FIX((yr - 1) / 4)
  IF yr MOD 4 = 0 AND mth >= 2 THEN GetDays = GetDays + 1
  GetDays = GetDays + Days(mth) + day
END FUNCTION
```

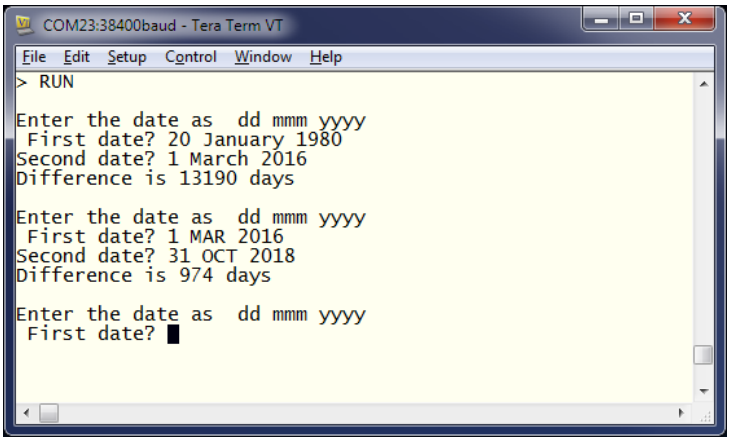
Beachte, dass die Zeile, die mit `LOCAL STRING Month(11)` beginnt, aufgrund der begrenzten Seitenbreite umgebrochen wurde – sie besteht aus einer einzigen Zeile wie folgt:

```
LOCAL STRING Month(11) = ("jan","feb","mar","apr","may","jun","jul","aug","sep","oct","nov","dec")
```

Dieses Programm funktioniert so, dass es zwei Datumsangaben vom Benutzer an der Konsole abfragt und diese dann in die Anzahl der Tage seit 1900 umrechnet. Mit diesen beiden Zahlen ergibt eine einfache Subtraktion die Anzahl der Tage zwischen ihnen.

Wenn dieses Programm ausgeführt wird, wirst du aufgefordert, die beiden Datumsangaben einzugeben, und zwar im Format: `dd mmm yyyy`.

Der Screenshot rechts zeigt, wie das laufende Programm aussieht.




Das Hauptmerkmal des Programms ist die definierte Funktion `GetDays()`, die eine an der Konsole eingegebene Zeichenkette entgegennimmt, sie in ihre Komponenten Tag, Monat und Jahr aufteilt und dann die Anzahl der Tage seit dem 1. Januar 1900 berechnet.

Diese Funktion wird zweimal aufgerufen, einmal für das erste Datum und dann noch einmal für das zweite Datum. Anschließend muss nur noch das eine Datum (in Tagen) vom anderen abgezogen werden, um die Differenz in Tagen zu erhalten.

Wir werden nicht im Detail darauf eingehen, wie die Berechnungen durchgeführt werden (z. B. die Behandlung von Schaltjahren), da dies als Übung für den Leser überlassen bleiben kann. Es ist jedoch angebracht, auf einige Funktionen von MMBasic hinzuweisen, die vom Programm verwendet werden.

Es zeigt, wie lokale Variablen verwendet und initialisiert werden können. In der Funktion `GetDays()` werden zwei Arrays gleichzeitig deklariert und initialisiert. Diese dienen lediglich als praktische Methode, um die Namen der Monate und die Gesamtzahl der Tage für jeden Monat abzurufen. Später in der Funktion (in der FOR-Schleife) kannst du sehen, wie sie den Umgang mit zwölf verschiedenen Monaten recht effizient gestalten.

Ein weiteres Merkmal, das dieses Programm hervorhebt, sind die Funktionen zur Zeichenfolgenverarbeitung von MMBasic. Die Funktion INSTR() wird verwendet, um die beiden Leerzeichen in der Datumszeichenfolge zu finden, und später nutzt die Funktion MID$() diese, um die Komponenten Tag, Monat und Jahr des Datums zu extrahieren. Die Funktion VAL() wird verwendet, um eine Zeichenfolge aus Ziffern (wie das Jahr) in eine Zahl umzuwandeln, die in einer numerischen Variablen gespeichert werden kann.

Beachte, dass der Wert einer Funktion bei jedem Aufruf auf Null initialisiert wird und, sofern er nicht auf einen bestimmten Wert gesetzt wird, einen Wert von Null zurückgibt. Das erleichtert die Fehlerbehandlung, da wir die Funktion einfach verlassen können, wenn ein Fehler entdeckt wird. Es liegt dann in der Verantwortung des aufrufenden Programmcodes, auf einen Rückgabewert von Null zu prüfen, der einen Fehler signalisiert.

Dieses Programm veranschaulicht einen der Vorteile der Verwendung von Unterprogrammen und Funktionen: Wenn sie geschrieben und vollständig getestet sind, können sie als vertrauenswürdige „Black Box“ behandelt werden, die nicht geöffnet werden muss. Aus diesem Grund sollten Funktionen wie diese ordnungsgemäß getestet und dann, wenn möglich, unverändert gelassen werden (für den Fall, dass du einen Fehler hinzufügst).

Es gibt ein paar Funktionen in diesem Programm, die wir bisher noch nicht behandelt haben. Die erste ist der MOD-Operator, der den Rest der Division einer Zahl durch eine andere berechnet. Wenn du beispielsweise 15 durch 4 teilst, ergibt sich ein Rest von 3 – genau das gibt der Ausdruck 15 MOD 4

zurück. Die Funktion ABS() ist ebenfalls neu und gibt ihr Argument als positive Zahl zurück (z. B. gibt ABS(-15) +15 zurück, ebenso wie ABS(15)).

Der Befehl EXIT FOR beendet eine FOR-Schleife, auch wenn sie noch nicht am Ende angelangt ist, EXIT FUNCTION beendet eine Funktion sofort, auch wenn die Ausführung noch nicht am Ende der Funktion angelangt ist, und CONTINUE DO bewirkt, dass das Programm sofort zum Ende einer DO-Schleife springt und diese erneut ausführt.

Wozu ist dieses Programm gut?  Nun, manche Leute zählen ihr Alter gerne in Tagen – so ist jeder Tag ein Geburtstag!  Du kannst dein Alter in Tagen berechnen, gib einfach dein Geburtsdatum und das heutige Datum ein. Das ist zwar nicht besonders nützlich, aber das Programm selbst ist wertvoll, da es viele der Eigenschaften der Programmierung in MMBasic demonstriert. Arbeite dich also durch das Programm und gehe jeden Abschnitt durch, bis du ihn verstehst – es sollte eine lohnende Erfahrung sein.