PicoMite

Benutzerhandbuch

MMBasic- -BASIC-Interpreter
Version 6.02.00

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

Revision 2
(, 26. Januar 2026,)

Aktualisierungen dieses Handbuchs und weitere Infos zu MMBasic findest du unter

http://geoffg.net/picomite.html

und http://mmbasic.com

Über

Peter Mather (matherp im Back Shed Forum) hat das Projekt geleitet, MMBasic auf den Raspberry Pi Pico portiert und die Treiber für dessen Hardwarefunktionen geschrieben. Der MMBasic-Interpreter und dieses Handbuch wurden von Geoff Graham (http://geoffg.net) geschrieben. Darüber hinaus haben viele andere das Projekt mit speziellem 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.

Urheberrecht und Danksagungen

Die PicoMite-Firmware und MMBasic unterliegen dem Copyright 2011-2025 von Geoff Graham und Peter Mather 2016-2025.

Das Copyright für die 1-Wire-Unterstützung liegt bei Dallas Semiconductor Corporation (1999–2006) und Gerard Sexton (2012).

Der FatFs-Treiber (SD-Karte) unterliegt dem Copyright 2014 von ChaN.

Die Unterstützung für WAV-, MP3- und FLAC-Dateien unterliegt dem Copyright 2019 von David Reid.

Die JPG-Unterstützung verdanken wir Rich Geldreich.

Das Copyright für pico-sdk liegt bei Raspberry Pi (Trading) Ltd. 2021.

TinyUSB ist urheberrechtlich geschützt durch tinyusb.org

LittleFS ist urheberrechtlich geschützt, Copyright Christopher Haster

Thomas Williams und Gerry Allardice für die Verbesserungen an MMBasic

Der VGA-Treibercode basiert auf der Arbeit von Miroslav Nemecek

Die CRC-Berechnungen sind urheberrechtlich geschützt durch Rob Tillaart

Der kompilierte Objektcode (die .uf2-Datei) für die PicoMite-Firmware ist freie Software: Du kannst ihn nach Belieben verwenden oder weitergeben. Der Quellcode ist auf GitHub (https://github.com/UKTailwind/PicoMiteAllVersions) und kann unter bestimmten Bedingungen frei verwendet werden (siehe Kopfzeile in den Quelldateien).

Dieses Programm wird in der Hoffnung verteilt, dass es nützlich ist, aber OHNE JEGLICHE GARANTIE, auch ohne die implizite Garantie der MARKTGÄNGIGKEIT oder EIGNUNG FÜR EINEN BESTIMMTEN ZWECK.

Dieses Handbuch

Copyright2026 Geoff Graham und Peter Mather

Der Autor dieses Handbuchs ist Geoff Graham, mit umfangreichen Beiträgen von Peter Mather, Harm de Leeuw, Mick Ames und vielen anderen aus dem Back Shed-Forum. Es wird unter einer Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Australia-Lizenz (CC BY-NC-SA 3.0) verbreitet.

Inhalt

Einführung

PicoMite ist eine Betriebsfirmware für alle Versionen des Raspberry Pi Pico , einschließlich Pico, Pico 2, Pico W und Pico 2 W.

Es enthält einen BASIC-Interpreter (MMBasic), eine mit Microsoft BASIC kompatible Implementierung der Programmiersprache BASIC 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, Einbruchmelder usw.) geeignet sind, sowie Versionen mit VGA/HDMI- und Tastaturunterstützung für den Bau eines eigenständigen Computers.

Mit MMBasic kannst du die I/O-Pins steuern und Kommunikationsprotokolle wie I^2C oder SPI nutzen, um Daten von verschiedenen Sensoren zu bekommen. Du kannst Daten auf günstigen 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 WEB-Server auf diesem günstigen Modul aufbauen.

Die PicoMite-Firmware kann komplett kostenlos runtergeladen und genutzt werden.

Zusammengefasst sind die Funktionen der PicoMite-Firmware:

- **Der BASIC-Interpreter ist voll ausgestattet** mit doppelter Genauigkeit bei Gleitkommazahlen, 64-Bit-Ganzzahlen und String-Variablen, langen Variablennamen, Arrays von Gleitkommazahlen, Ganzzahlen oder Strings mit mehreren Dimensionen, umfangreicher String-Verarbeitung und benutzerdefinierten Strukturen, Unterprogrammen und Funktionen. Außerdem ermöglicht MMBasic die Einbettung von kompilierten C-Programmen für Hochleistungsfunktionen. Der Schwerpunkt liegt auf Benutzerfreundlichkeit und schneller Entwicklung.
- **Unterstützung für alle Raspberry Pi Pico-Eingangs-/Ausgangspins**. Diese können unabhängig voneinander als digitaler Eingang oder Ausgang, analoger Eingang, Frequenz- oder Periodemessung und Zählung konfiguriert werden. Interrupts können verwendet werden, um zu benachrichtigen, wenn ein Eingangspin seinen Zustand geändert hat. PWM-Ausgänge können verwendet werden, um verschiedene Töne zu erzeugen, Servos zu steuern oder computergesteuerte Spannungen zu erzeugen.
- **Unterstützung für LCD-/OLED-Anzeigetafeln** mit parallelen, SPI- und I^2C-Schnittstellen, sodass das BASIC-Programm Text anzeigen und Linien, Kreise, Kästchen usw. in bis zu 16 Millionen Farben zeichnen kann. Resistive Touch-Controller auf diesen Tafeln werden auch unterstützt, sodass sie als anspruchsvolle Eingabegeräte genutzt werden können.
- **Unterstützung für Internet- und WEB-Protokolle mit dem Raspberry Pi Pico W und Pico 2 W**. Dazu gehört ein WEB-Server mit TCP und HTML, der über TCP und HTTP auf andere Ressourcen zugreifen kann. MQTT-Protokoll für die Verbindung über einen Message Broker. 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 eine PS2- oder USB-Tastatur und HDMI- oder VGA-Videoausgabe**. Dazu gehört die vollständige Unterstützung für Grafiken, Audio (Soundeffekte und Musik), internen Programmspeicher, Gamecontroller und mehr. Dadurch wird der Raspberry Pi Pico oder zu einem eigenständigen Computer wie der Apple II oder Tandy TRS-80 von früher. Ideal zum Schreiben von Spielen, zum Lernen von BASIC oder einfach zum Ausbalancieren deines Scheckbuchs.
- **Flexible Programm- und Datenspeicherung**. Programme und Daten können aus einem internen Dateisystem, 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, gelesen/geschrieben werden. Dazu gehört das Öffnen von Dateien zum Lesen, Schreiben oder für den Direktzugriff sowie das Laden und Speichern von Programmen.
- **Ein Vollbild-Editor** ist in die Firmware integriert und kann das gesamte Programm in einer Sitzung bearbeiten. Er hat erweiterte Funktionen wie farbcodierte Syntax, Suchen und Kopieren, Ausschneiden und Einfügen in die und aus der Zwischenablage.

- **Programme können ganz einfach** von einem Desktop- oder Laptop-Computer (Windows, Mac oder Linux) über die serielle Konsole oder eine SD-Karte **übertragen werden**.
- Es gibt **eine ganze Reihe von Kommunikationsprotokollen**, wie I^2C, asynchrones serielles Protokoll, RS232, SPI und 1-Wire. Diese können für die Kommunikation mit vielen Sensoren (Temperatur, Feuchtigkeit, Beschleunigung usw.) sowie für das Senden von Daten an Testgeräte genutzt werden.
- **Integrierte Befehle** für die direkte Anbindung an Infrarot-Fernbedienungen, den Temperatursensor DS18B20, LCD-Anzeigemodule, batteriegepufferte Uhren, numerische Tastaturen und mehr.

Firmware-Versionen und Dateien

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

Eigenständiger Computer

Versionen mit VGA- oder HDMI-Videoausgang sind für die Verwendung als eigenständiger Computer gedacht. Diese starten und zeigen die Ausgabe des BASIC-Interpreters auf dem an den Videoausgang angeschlossenen Monitor an. Sie sind mit einer PS2- oder USB-Tastatur gekoppelt, und mithilfe der Tastatur und des Videoausgangs kannst du ein Programm eingeben und bearbeiten, es ausführen, Optionen festlegen 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 sehr beliebt, zum Beispiel der Apple II, Tandy TRS-80, Commodore 64 und andere.

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

Eingebetteter Controller

Versionen der Firmware ohne Videoausgabe 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. Beispiele hierfür sind Einbruchmeldeanlagen, Heizungssteuerungen, Wetterstationen usw. Oft sind diese Geräte mit einem berührungsempfindlichen LCD-Panel ausgestattet, 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 und der auf das Internet zugreifen kann, um die Uhrzeit abzurufen, E-Mails zu versenden usw.

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

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 auch für die Verwendung mit Modulen anderer Hersteller ausgelegt, die die gleichen Chips verwenden.

Während es vom RP2040 nur eine Version gibt, gibt's vom RP2350 vier Unterversionen: RP2350A, RP2350B, RP2354A und RP2354B. Der RP2350B ist wie der RP2350A, hat aber 18 zusätzliche I/O-Pins (Pins GP30 bis GP47), die automatisch in MMBasic verfügbar sind. Beide Chips werden von derselben PicoMite-Firmware unterstützt und funktionieren gleich. 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 aber in Zukunft unterstützt werden).

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

Dateinamen

Die ZIP-Datei mit der Firmware 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** die unterstützten Funktionen sind (VGA und USB).
- **V6.02.00** die Versionsnummer ist. Diese wird in zukünftigen Versionen erhöht.
- **.uf2** ist die Erweiterung, die ein ladbares Raspberry Pi Pico-Firmware-Image kennzeichnet.

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

Firmware-Dateiname 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	✓	✓				✓

Firmware laden

Der Raspberry Pi Pico und der Pico 2 haben einen eigenen eingebauten Firmware-Loader, der echt einfach zu bedienen ist.

Um die PicoMite-Firmware zu laden, machst du Folgendes:

- Lade die PicoMite-Firmware von http://geoffg.net/picomite.html herunter, entpacke die Datei und suche die Firmware, die zu deiner Verwendung passt (siehe vorherige Überschriften).
- Schließ 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 (genauso, als hättest du einen USB-Stick angeschlossen). Du kannst alle Dateien, die sich auf diesem „Laufwerk" befinden, ignorieren.
- Kopier die Firmware-Datei (mit der Erweiterung .uf2) auf dieses virtuelle Laufwerk.
- Wenn der Kopiervorgang abgeschlossen ist, startet der Raspberry Pi Pico neu und erstellt einen virtuellen seriellen Anschluss über USB auf deinem Computer. Details zur Verwendung findest du im Kapitel *„Serielle Konsole"* weiter unten.
- Die LED am Raspberry Pi Pico blinkt langsam und zeigt damit an, dass die PicoMite-Firmware mit MMBasic jetzt läuft.

Das vom Raspberry Pi Pico erstellte virtuelle Laufwerk sieht zwar wie ein USB-Speicherstick aus, ist aber keiner. Die Firmware-Datei verschwindet nach dem Kopieren, und wenn du versuchst, andere Dateitypen zu kopieren, werden 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. Stell also sicher, dass du diese Daten vor dem Upgrade der Firmware sicherst.

Es kann passieren, dass der Flash-Speicher beschädigt wird, was zu ungewöhnlichem und unvorhersehbarem Verhalten führen kann. In diesem Fall solltest du die passende Firmware-Datei, die unten aufgeführt ist, runterladen und wie oben beschrieben auf den Pico laden. Dadurch wird der Raspberry Pi Pico auf den Werkszustand zurückgesetzt, und du kannst die PicoMite-Firmware neu 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 dem Zugriff auf die Konsole kannst du Programme eingeben, bearbeiten, ausführen usw. Die meisten Firmware-Versionen erstellen die serielle Konsole automatisch als virtuellen seriellen Anschluss über USB. In diesem Kapitel wird beschrieben, wie das funktioniert und wie du es nutzen kannst.

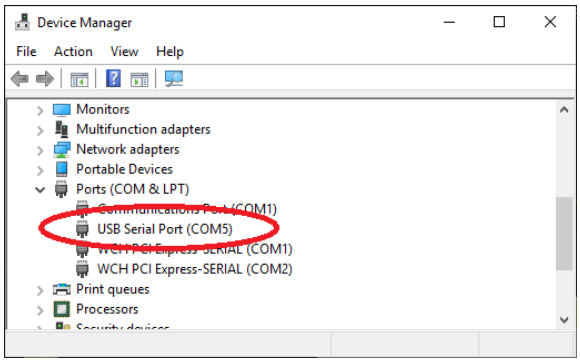
Bei einem eigenständigen Computer (wie oben beschrieben) ist die serielle Konsole nur eine zusätzliche Kommunikationsmethode, aber wenn du den Raspberry Pi Pico als eingebetteten Controller benutzt, ist sie die einzige Kommunikationsmethode, die du hast. Deshalb ist es wichtig, dass du eine Verbindung herstellen kannst.

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

Virtueller serieller Anschluss

Der von der PicoMite-Firmware erstellte virtuelle serielle Anschluss über USB nutzt das CDC-Protokoll (Communication Device Class) und funktioniert wie ein normaler serieller Anschluss, nur über USB. Windows 10 und 11 haben einen Treiber dafür, aber bei anderen Betriebssystemen musst du vielleicht 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.

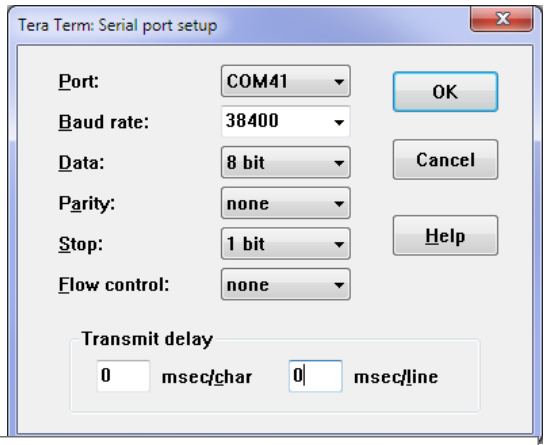


Du solltest dir dann die Portnummer notieren, die dein Computer für die virtuelle serielle Verbindung erstellt hat. Unter Windows kannst du dazu den Geräte-Manager starten und unter „Anschlüsse (COM & LPT)" nach einem neuen COM-Port suchen, wie rechts gezeigt.

Terminalemulator

Du brauchst außerdem einen Terminalemulator, der auf deinem Desktop- oder Laptop-Computer läuft. Dabei handelt es sich um ein Programm, das wie ein altmodischer Computerterminal funktioniert, auf dem Text von einem Remote-Computer angezeigt wird und alle Tastendrücke über die serielle Verbindung an den Remote-Computer gesendet werden. Der von dir verwendete Terminalemulator sollte die VT100-Emulation unterstützen, da dies für den in die PicoMite-Firmware integrierten Editor erforderlich ist.

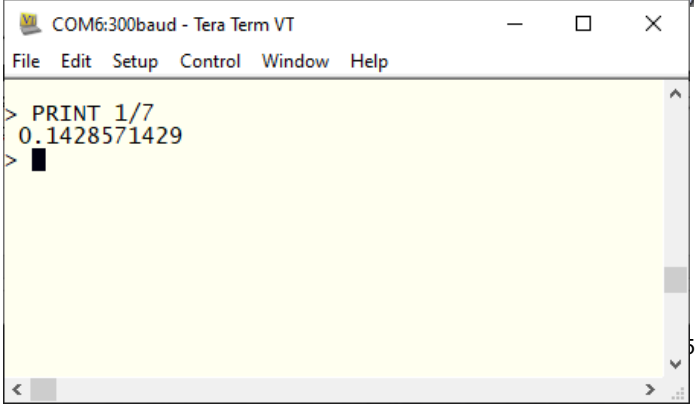
Für Windows-Benutzer wird die Verwendung von Tera Term empfohlen, da dieses Programm über einen guten VT100-Emulator verfügt und bekanntermaßen mit dem XModem-Protokoll funktioniert, mit dem Sie Programme zum und vom PicoMite übertragen können. Tera Term kann unter folgender Adresse heruntergeladen werden: http://tera-term.en.lo4d.com .



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

Die PicoMite-Firmware ignoriert die Baudrateneinstellung, sodass du jede beliebige Geschwindigkeit einstellen kannst (außer 1200 Baud, da dies den Pico in den Firmware-Upgrade-Modus versetzt).

Wenn du Tera Term benutzt, stell keine Verzögerung zwischen den Zeichen ein, und wenn du Putty benutzt, stell 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 zum Konfigurieren von MMBasic, zum Laden des BASIC-Programms, zum Bearbeiten und Ausführen des Programms eingeben kannst. MMBasic verwendet die Konsole auch zur Anzeige von Fehlermeldungen.

Windows 7 und 8.1

Der USB-Seriell-Port nutzt das CDC-Protokoll, und die Treiber dafür sind in Windows 10 und 11 Standard und werden automatisch geladen.

Die Raspberry Pi Foundation listet Windows 7 oder 8.1 als „nicht unterstützt“ auf, aber du kannst ein Tool wie Zadig (https://zadig.akeo.ie) verwenden, um einen generischen Treiber für ein „usbser“-Gerät zu installieren, der die Verbindung dieser Computer ermöglichen sollte. 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. Starten Sie zunächst die Anwendung „Terminal“ und listen Sie die angeschlossenen seriellen Geräte auf, indem Sie Folgendes eingeben:

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

Der USB-zu-Seriell-Konverter wird als etwas wie `/dev/tty.usbmodem12345` aufgelistet. Während du noch an der Terminal-Eingabeaufforderung bist, kannst du den Terminalemulator mit 115200 Baud ausführen, indem du den folgenden Befehl verwendest:

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

Standardmäßig sind die Funktionstasten für die Verwendung im integrierten Programmeditor des PicoMite nicht richtig definiert, sodass du die im Kapitel *„Vollbild-Editor”* dieses Handbuchs definierten Steuerungssequenzen verwenden musst. Um das 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 sieh dir diese Beiträge an:

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 check diesen Beitrag:

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 machen, um zu zeigen, dass dein Computer funktioniert. Alle diese Befehle solltest du in die Eingabeaufforderung (d. h. „>“) eingeben.

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

Probier mal eine einfache Berechnung aus:

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

Schau nach, wie viel Speicher du hast:

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

Gespeicherte Variablen:
  16 KB (100 %) frei

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

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

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

Stell die Uhr auf die aktuelle Zeit ein:

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

Überprüfe die Uhrzeit noch mal:

```
> 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 musst du aber nur wissen, dass alles, was du eingibst, an der Cursorposition eingefügt wird, dass du den Cursor mit den Pfeiltasten bewegen kannst und dass du mit der Rücktaste das Zeichen vor dem Cursor löschen kannst.

Um einen schnellen Eindruck von der Funktionsweise von MMBasic zu bekommen, probier mal diese Sequenz aus:

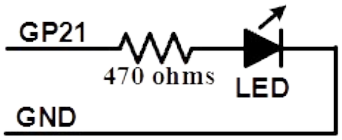
- Gib an der Eingabeaufforderung `EDIT` ein und drück dann die ENTER-Taste.
- Der Editor sollte starten und du kannst diese Zeile eingeben: `PRINT „Hallo Welt“`
- Drück die Taste F1 in deinem Terminalemulator (oder STRG-Q, was dasselbe bewirkt). Dadurch wird der Editor angewiesen, dein Programm zu speichern und zur Eingabeaufforderung zurückzukehren.
- Gib an der Eingabeaufforderung `RUN` ein und drück dann die ENTER-Taste.
- 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 zum Editor, wo du dein Programm ändern oder ergänzen kannst.

Eine LED blinken lassen

Schließ eine LED an den Pin GP21 (auf der Unterseite der Platine markiert) und einen Massepin 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 zeigt, wie die PicoMite-Firmware über deine Programmierung mit der physischen Welt interagieren kann.

Das Programm selbst ist einfach. In der ersten Zeile wird Pin GP21 als Ausgang festgelegt. Dann geht das Programm in eine Endlosschleife, in der der Ausgang dieses Pins auf „hoch" gesetzt wird, um die LED einzuschalten, gefolgt von einer kurzen Pause (300 Millisekunden). Der Ausgang wird dann auf „niedrig" gesetzt, gefolgt von einer weiteren Pause. Das Programm wiederholt dann die Schleife.

Wenn du es so lässt, bleibt der Raspberry Pi Pico einfach stehen und die LED blinkt weiter. Wenn du was ändern willst (z. B. die Blinkgeschwindigkeit), kannst du das Programm mit STRG-C auf der Konsole unterbrechen und dann nach Bedarf anpassen. Das ist der große Vorteil von MMBasic: Es ist super einfach, ein Programm zu schreiben und zu ändern.

Wenn du möchtest, dass dieses Programm jedes Mal automatisch startet, wenn die Stromversorgung eingeschaltet wird, kannst du den folgenden Befehl an der Eingabeaufforderung oder im Programm verwenden:

```
OPTION AUTORUN ON
```

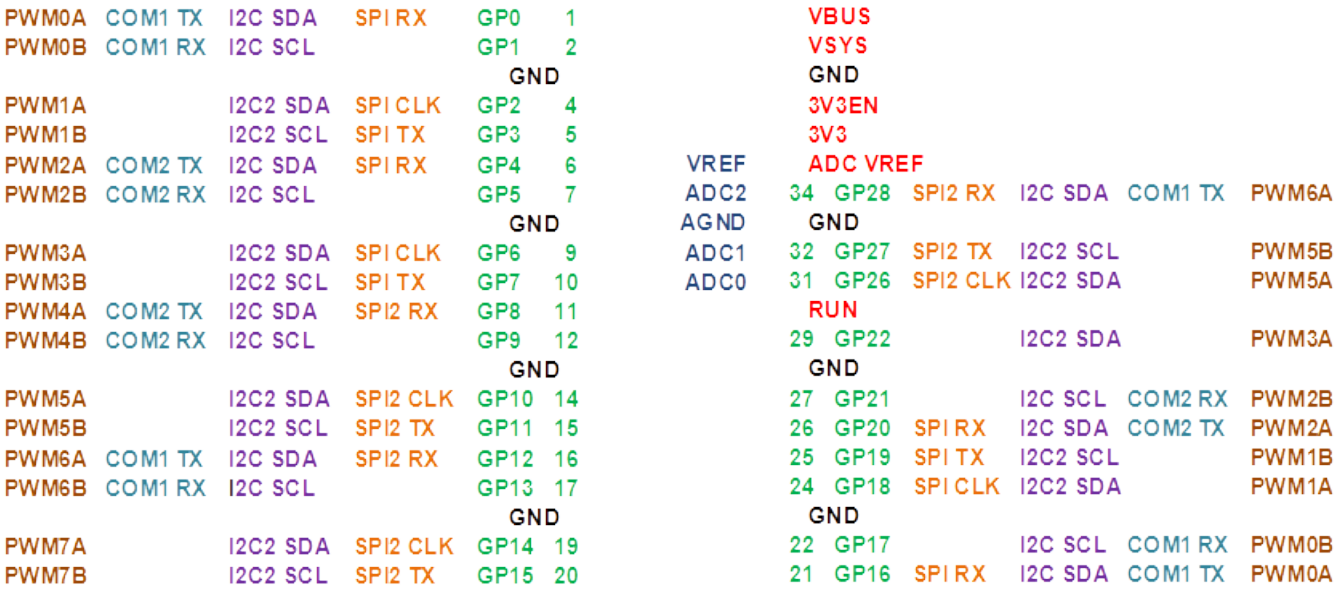
Um das zu testen, kannst du die Stromversorgung unterbrechen und dann wieder einschalten. Das Gerät sollte dann mit dem Blinken der LED starten.

Tutorial zur Programmierung in der Sprache BASIC

Wenn du noch keine Erfahrung mit der Programmiersprache BASIC hast, solltest du dir jetzt den Anhang J (*Programmieren in BASIC – Ein Tutorial*) am Ende dieses Handbuchs ansehen. Das ist ein umfassendes Tutorial zur Sprache, das dir die Grundlagen in einem leicht verständlichen Format mit vielen Beispielen vermittelt.

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 haben HDMI-Versionen acht Pins (GP12 bis GP19), die für diese Funktion reserviert sind. Weitere Infos findest du im Kapitel *„Videoausgang"*.

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

Die Notation ist wie folgt:

GP0 bis GP28	Können für digitale Ein- oder Ausgänge genutzt werden.
COM1, COM2	Kann für asynchrone serielle E/A verwendet werden (Pins UART0 und UART1 im Pico-Datenblatt).
I2C, I2C2	Kann für I²C-Kommunikation verwendet werden (I2C0- und I2C1-Pins im Pico-Datenblatt).
SPI, SPI2	Kann für SPI-E/A verwendet werden (siehe Anhang D). (SPI0- und SPI1-Pins im Pico-Datenblatt).
PWMnx	Kann für PWM-Ausgabe verwendet werden (siehe die Befehle PWM und SERVO).
GND	Gemeinsame Masse.
VBUS	5-V-Versorgung direkt über den USB-Anschluss.
VSYS	5-V-Versorgung, die vom SMPS genutzt wird, um 3,3 V zu liefern. Kann als 5-V-Ausgang oder -Eingang genutzt werden.
3V3EN	3,3-V-Regler einschalten (niedrig = aus, hoch = eingeschaltet).
RUN	Reset-Pin, niedrig hält das Gerät im Reset-Zustand.
ADCn	Diese Pins können zum Messen der Spannung (Analogeingang) verwendet werden.
ADC VREF	Referenzspannung für die Spannungsmessung.
AGND	Analoge Masse.

Im MMBasic-Programm kannst du auf I/O-Pins mit der physischen Pin-Nummer (also 1 bis 40) oder der GP-Nummer (also GP0 bis GP28) zugreifen. Zum Beispiel beziehen sich die folgenden Befehle auf denselben Pin und funktionieren gleich:

```
SETPIN 32, DOUT
```

und `SETPIN GP27, DOUT`

Die Pin-Nummern (d. h. 1 bis 40) gelten nur für die Standardformate Pico und Pico 2. Andere Module von Drittanbietern können ein anderes Pin-Nummerierungsschema haben. Aus diesem Grund wird dringend empfohlen, bei der Referenzierung eines I/O-Pins nur die GP-Nummer zu verwenden.

In der PicoMite-Firmware sind On-Chip-Funktionen wie die SPI- und I2C-Schnittstellen nicht wie beispielsweise beim Micromite festen Pins zugeordnet. Die PicoMite-Firmware nutzt den Befehl SETPIN ausgiebig, nicht nur zur Konfiguration von I/O-Pins, sondern auch zur Konfiguration der Pins, die für Schnittstellen wie Seriell, SPI, I^2C usw. verwendet werden.

Die Pins müssen gemäß dieser Zeichnung zugewiesen werden. Beispielsweise kann der SPI TX den Pins GP3, GP7 oder GP19 zugewiesen werden, aber nicht dem Pin GP11, der nur dem SPI2-Kanal zugewiesen werden kann. Die Zuweisungen müssen nicht im selben „Block" erfolgen, sodass Sie beispielsweise SPI2 TX dem Pin GP11 und SPI2 RX dem Pin GP28 zuweisen können.

Module von Drittanbietern

Auf Pins, die auf dem Raspberry Pi Pico nicht freigelegt sind, kann über MMBasic trotzdem über ihre GPn-Nummer zugegriffen werden. Dadurch kann MMBasic auf anderen Modulen verwendet werden, die die Prozessoren RP2040 oder RP2350 nutzen.

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

- GP23 ist ein digitaler Ausgang, der auf den Wert von OPTION POWER gesetzt ist. (ON=PWM, OFF=PFM).
- GP24 ist ein digitaler Eingang, der bei vorhandener VBUS-Spannung hoch ist.
- 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 liest.

Auf Modulen von Drittanbietern, die diese Pins zur Verfügung stellen, können sie aber wie folgt genutzt 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 hat auch ein paar GPIO-Pins, die für interne Board-Funktionen genutzt werden:

- GP29 (Eingang/Ausgang) drahtloser SPI CLK/ADC-Modus (ADC3) misst VSYS/3
- GP25 SPI CS (Ausgang) aktiviert bei hohem Pegel auch den GPIO29-ADC-Pin zum Lesen von VSYS
- GP24 (Eingang/Ausgang) kabellose SPI-Daten / IRQ
- GP23 (Ausgang) drahtloses Einschaltsignal

Die WebMite-Firmware lässt keine Neuzuweisung dieser Pins zu.

Anders als beim normalen Raspberry Pi Pico ist die LED auf dem Pico W nicht mit einem Pin auf dem RP2040 verbunden, sondern mit einem GPIO-Pin auf dem Wireless-Chip und kann nicht über ein BASIC-Programm angesprochen werden.

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

CPU-Varianten

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

RP2040

Dieser ist in einem 60-Pin-Gehäuse untergebracht und wird im originalen Raspberry Pi Pico und vielen anderen Modulen von Drittanbietern verwendet. Die Pinbelegung und Funktionen der I/O-Pins sind wie oben beschrieben.

RP2350A

Auch dieser Chip ist in einem 60-Pin-Gehäuse verpackt und wird im Raspberry Pi Pico 2 und in Modulen von Drittanbietern verwendet. Die Pinbelegung und Funktionen der I/O-Pins sind die gleichen wie beim RP2040 und sind oben beschrieben.

RP2350B

Der RP2350B kommt in einem 80-Pin-Gehäuse mit zusätzlichen 18 GPIO-Pins. Die PicoMite-Firmware unterstützt diese zusätzlichen Pins, einschließlich der PWM-Kanäle 8-11 (was maximal 24 gleichzeitige PWM-Ausgänge ermöglicht). Die Konfiguration für den RP2350B läuft automatisch ab, sodass alle zusätzlichen Pins und drei PIO-Kanäle verfügbar sind (die VGA-Version hat 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 Pinbelegung für die Pins GP0 bis GP29 auf dem RP2350B ist die gleiche wie beim RP2040 und RP2350A. Die Pins GP30 bis GP47 können wie folgt genutzt 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

Die werden gerade nicht von der PicoMite-Firmware unterstützt.

PSRAM

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

Der Zugriff auf den PSRAM erfolgt über denselben Quad-SPI-Bus, den auch der Flash-Speicher nutzt, sodass er vergleichsweise langsam ist, obwohl er über einen Cache gepuffert wird, der dieses Problem etwas abmildert. Wenn ein PSRAM vorhanden und konfiguriert ist, fügt MMBasic ihn dem allgemeinen RAM-Pool hinzu, sodass Programme eine riesige Menge an allgemeinem RAM zur Verfügung haben, auch wenn dieser etwas langsamer sein kann.

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

Nach dem Einschalten ist der Inhalt des PSRAM unbestimmt – er ist nicht Null. Du musst RAM ERASE verwenden, um ihn vor der Verwendung zu löschen. Dieses Verhalten ist beabsichtigt, damit der Inhalt nach einem Reset erhalten bleibt.

I/O-Pin-Beschränkungen

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 aufnehmen, während der Chip mit Strom versorgt wird.

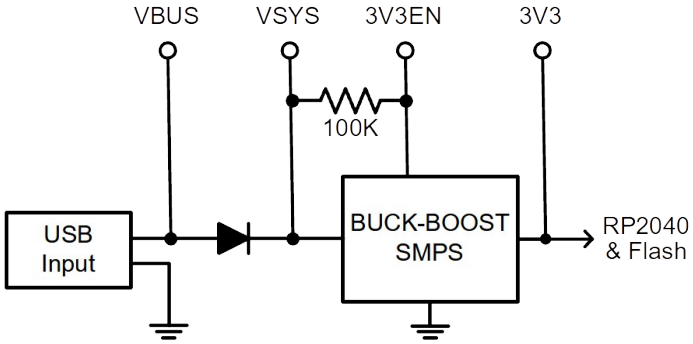
Als Ausgänge können alle I/O-Pins einzeln bis zu 8 mA liefern oder aufnehmen. Bei dieser Last sinkt die Ausgangsspannung auf etwa 2,3 V. Eine praktischere Last ist 5 mA, bei der die Ausgangsspannung typischerweise 3 V beträgt. Um eine rote LED mit 5 mA zu betreiben, wird ein Widerstand von 220 Ω empfohlen.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

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

Stromversorgung

Der Raspberry Pi Pico und Pico 2 hat ein flexibles Stromversorgungssystem .

Die Eingangsspannung von den USB- oder VBUS-Eingängen wird über eine Schottky-Diode mit dem Buck-Boost-SMPS (Switch Mode Power Supply) verbunden, das eine Ausgangsspannung von 3,3 V hat. Das SMPS kann Eingangsspannungen von 1,8 V bis 5,5 V verarbeiten, 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 eingebettete Controller-Anwendungen wird in der Regel eine externe Stromquelle (außer USB) benötigt, die über eine Schottky- -Diode ( ) an VSYS angeschlossen werden kann. Dadurch kann der Raspberry Pi Pico mit der Stromquelle betrieben werden, die die höchste Spannung liefert (USB oder VSYS). Die Dioden verhindern eine Rückkopplung in die Stromquelle mit niedrigerer Spannung.

Um das Rauschen der Stromversorgung zu minimieren, kann man 3V3EN erden, um das SMPS auszuschalten. Beim Herunterfahren hört der Wandler auf zu schalten, die interne Steuerschaltung wird ausgeschaltet und die Last wird getrennt. Dann kann man die Platine über einen linearen 3,3-V-Regler versorgen, der in den 3V3-Pin eingespeist wird.

Taktrate

Standardmäßig ist die Taktrate für den RP2040 im Raspberry Pi Pico auf 200 MHz und für den RP2350 im Raspberry Pi Pico 2 auf 150 MHz eingestellt. Das sind die empfohlenen Höchstwerte.

Mit dem Befehl OPTION CPUSPEED können die meisten RP2040-CPUs auf bis zu 420 MHz und der RP2350 (nur PicoMite und WebMite) auf bis zu 396 MHz übertaktet werden. Sie können auch langsamer mit einer Mindestgeschwindigkeit von 48 MHz laufen. Diese Option wird gespeichert und beim Einschalten wieder angewendet. Wenn du die Taktrate änderst, wird die PicoMite-Firmware zurückgesetzt und neu gestartet, sodass die USB-Verbindung unterbrochen wird.

Bei VGA/HDMI-Versionen mit einer Videoauflösung von 640 x 400 wird die Taktrate auf 252 MHz eingestellt, das kann aber mit OPTION RESOLUTION auf 252 MHz, 315 MHz oder 378 MHz geändert werden. Bei anderen Videoauflösungen ist die Taktrate je nach gewählter Videoauflösung auf 283,2 MHz, 324 MHz, 360 MHz, 372 MHz oder 375 MHz festgelegt und kann nicht geändert werden.

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

Stromverbrauch

Der Stromverbrauch von „ “ hängt von der Taktrate ab, aber bei der Standardtaktrate (200 MHz für den RP2040 und 150 MHz für den RP2350) liegt der typische Stromverbrauch bei 25 mA. Dabei ist der Strom, der von den I/O-Pins oder dem 3V3-Pin bezogen oder abgegeben wird, nicht mit eingerechnet:

Der Stromverbrauch der WebMite-Version für den Raspberry Pi Pico W ist bei deaktiviertem WLAN mit 20 mA gleich, aber wenn das WLAN aktiviert ist, steigt der Stromverbrauch auf 40 bis 70 mA.

Verwendung von MMBasic

Befehle und Programmeingabe

An der Eingabeaufforderung kannst du einen Befehl eingeben, der dann sofort ausgeführt wird. Meistens machst du das, um der PicoMite-Firmware zu sagen, dass sie etwas tun soll, wie zum Beispiel ein Programm ausführen oder eine Option einstellen. Mit dieser Funktion kannst du aber auch Befehle an der Eingabeaufforderung ausprobieren.

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 hat erweiterte Funktionen wie Suchen und Kopieren, Ausschneiden und Einfügen in die Zwischenablage.

Du kannst das Programm auch auf deinem Desktop-Computer mit einem Programm wie Notepad erstellen und es dann über die Protokolle XModem oder YModem (siehe Befehl XMODEM oder YMODEM) oder per Streaming über die serielle Konsolenverbindung (siehe 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 mit dem du dein Programm auf deinem Computer bearbeiten und dann mit einem einzigen Mausklick auf den PicoMite übertragen kannst. MMEdit wurde von Jim Hiley geschrieben und kann kostenlos heruntergeladen und verwendet werden. Weitere Informationen findest du unter: https://geoffg.net/mmedit.html

Eine Sache, die du nicht machen kannst, ist die alte BASIC-Methode zum Eingeben eines Programms, 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 bis zum Ende des Programms oder bis es auf einen END-Befehl trifft. 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 jeder den BASIC-Interpreter dazu veranlasst, etwas zu tun (die Begriffe „Anweisung" und „Befehl" haben im Allgemeinen dieselbe Bedeutung 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 enthält ein umfassendes Tutorial zur Sprache, das dir die Grundlagen in einem leicht verständlichen Format mit vielen Beispielen vermittelt.

Bearbeiten der Befehlszeile

Wenn du eine Zeile an der Eingabeaufforderung eingibst, kannst du sie mit den Pfeiltasten nach links und rechts bearbeiten, indem du dich entlang der Zeile bewegst, 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 kannst du zum Anfang/Ende der Zeile springen, und wenn du zweimal Home drückst, wird die Bearbeitung beendet. Mit der Eingabetaste kannst du die Zeile jederzeit an MMBasic senden, das sie dann ausführt.

Mit den Aufwärts- und Abwärtspfeiltasten kannst du durch den Verlauf zuvor eingegebener Befehlszeilen navigieren, die du bearbeiten und wiederverwenden kannst.

Tastenkombinationen

Die Funktionstasten auf der Tastatur, die für die Konsole verwendet werden, kannst du an der Eingabeaufforderung verwenden, um häufig verwendete 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 Schickt eine ESC-Sequenz, um den VT100-Bildschirm zu löschen. Löscht auch die Konsole.
F10 AUTOSAVE
F11 XMODEM EMPFANGEN
F12 XMODEM SENDEN

Die Funktionstasten F1 und F5 bis F9 kannst du mit eigenem Text programmieren. Schau dir dazu den Befehl OPTION FNKey an.

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 auf der Konsole eingibst. MMBasic kehrt dann zur Befehlszeile zurück.

Optionen einstellen

Viele Optionen kannst du mit Befehlen einstellen, die mit dem Schlüsselwort OPTION anfangen. Die sind in einem eigenen Abschnitt dieses Handbuchs aufgelistet. Bei manchen Firmware-Versionen kannst du zum Beispiel die CPU-Taktrate mit dem Befehl ändern:

```
OPTION CPUSPEED Geschwindigkeit
```

Gespeicherte Variablen

Oft musst du Daten speichern, die nach einem Stromausfall wiederhergestellt werden können, z. B. Programmoptionen, Kalibrierungseinstellungen usw. Das kannst du mit dem Befehl VAR SAVE machen, 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 vom Programm verwendet werden können.

Diese Funktion ist zum Speichern von Kalibrierungsdaten, vom Benutzer ausgewählten Optionen und anderen Elementen gedacht, die sich selten ändern. Sie sollte nicht für Hochgeschwindigkeitsspeicherungen verwendet werden, da dies zu einer Abnutzung des Flash-Speichers führen kann. Der für den Raspberry Pi Pico verwendete Flash-Speicher hat eine hohe Lebensdauer, die jedoch durch ein Programm, das wiederholt Variablen speichert, überschritten werden kann. Wenn du Daten häufig speichern möchtest, solltest du einen Echtzeituhr-Chip hinzufügen. Mit den RTC-Befehlen kannst du dann Daten im batteriegepufferten Speicher der Echtzeituhr speichern und abrufen. Weitere Informationen findest du unter dem Befehl RTC.

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

Watchdog-Timer

Eine der Anwendungen 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, wenn es mit Strom versorgt wird, und fungiert als benutzerdefinierte Schaltung, die eine bestimmte Aufgabe ausführt. Der Benutzer muss nichts darüber wissen, was darin abläuft.

Es besteht jedoch die Möglichkeit, dass ein Fehler im Programm dazu führt, dass MMBasic einen Fehler generiert und zur Eingabeaufforderung zurückkehrt. In einer eingebetteten Situation wäre dies wenig hilfreich, da das Gerät nicht mit der Konsole verbunden wäre. Eine weitere Möglichkeit besteht darin, dass das BASIC-Programm aus irgendeinem Grund in einer Endlosschleife hängen bleibt. In beiden Fällen wäre der sichtbare Effekt derselbe: Das Programm würde so lange nicht mehr laufen, bis die Stromversorgung unterbrochen und wiederhergestellt wird.

Um das zu verhindern, kann man den Watchdog-Timer nutzen. Das ist ein Timer, der bis Null runterzählt und wenn er Null erreicht, werden die Prozessoren automatisch neu gestartet (genau wie beim ersten Einschalten), auch wenn MMBasic gerade an der Eingabeaufforderung hängt. Nach dem Neustart wird die automatische Variable MM.WATCHDOG auf „true” gesetzt, um zu zeigen, dass der Neustart wegen eines Watchdog-Timeouts passiert ist.

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. Wenn dann ein Fehler auftritt, wird der Timer nicht zurückgesetzt, sondern zählt bis Null herunter und das Programm wird neu gestartet (vorausgesetzt, die Option AUTORUN ist aktiviert).

PIN-Sicherheit

Manchmal ist es wichtig, die Daten und das Programm in einem eingebetteten Controller vertraulich zu behandeln. In der PicoMite-Firmware kann das mit dem Befehl OPTION PIN gemacht werden. Dieser Befehl legt eine PIN-Nummer fest (die im Flash-Speicher gespeichert wird), und wenn die PicoMite-Firmware (aus welchem Grund auch immer) zur Eingabeaufforderung zurückkehrt, wird der Benutzer an der Konsole aufgefordert, die PIN-Nummer einzugeben. Ohne die richtige PIN kann der Benutzer nicht zur Befehlszeile gelangen 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 zugreifen zu können.

Da ein Eindringling die Befehlszeile nicht erreichen kann, kann er kein Programm auflisten oder kopieren, das Programm nicht ändern und auch keine Änderungen an MMBasic oder der PicoMite-Firmware vornehmen. 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 Verwendung eines Programmiergeräts zur Überprüfung des Flash-Speichers), daher sollte dies nicht als ultimative Sicherheitsmaßnahme angesehen werden, aber es wirkt als erhebliche Abschreckung.

Die Bibliothek-

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 zum Beispiel vielleicht 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 wie normale BASIC-Routinen schreiben und testen. Wenn sie richtig funktionieren, kannst du den Befehl LIBRARY SAVE verwenden. Dadurch werden die Routinen (so viele 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 zu beachten sind:

- Bibliotheksroutinen funktionieren genau wie normaler BASIC-Code und können aus beliebig vielen Unterprogrammen, Funktionen, eingebetteten C-Routinen und Schriftarten bestehen. Der einzige Unterschied besteht darin, dass sie bei der Auflistung eines Programms nicht angezeigt 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 es gesetzt ist.
- Wenn die Routinen in die Bibliothek übertragen werden, komprimiert MMBasic sie, indem Kommentare, zusätzliche Leerzeichen, Leerzeilen und die Hex-Codes in eingebetteten C-Routinen und Schriftarten entfernt werden. Dadurch wird der Bibliotheksspeicherplatz 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 einen GOTO-Befehl usw. verwenden. Das liegt daran, dass der Befehl LIBRARY SAVE alle Leerzeilen entfernt.
- Du kannst READ-Befehle in der Bibliothek verwenden, aber sie lesen standardmäßig DATA-Anweisungen im Hauptprogrammspeicher. Wenn du aus DATA-Anweisungen in der Bibliothek lesen möchtest, musst du vor dem ersten READ-Befehl den Befehl RESTORE verwenden. Dadurch wird der Zeiger auf den Bibliotheksbereich zurückgesetzt.

- Die Bibliothek wird im Programm-Flash-Speicher Slot 3 gespeichert, der dann nicht mehr für die Speicherung eines Programms verfügbar ist, wenn LIBRARY SAVE verwendet wird.
- Mit dem Befehl LIBRARY LIST kannst du den Inhalt der Bibliothek anzeigen, da dieser Befehl den Inhalt des Bibliotheksbereichs auflistet.
- Der Inhalt der LIBRARY kann mit LIBRARY DISK SAVE fname$ auf der Festplatte gespeichert und mit LIBRARY DISK LOAD fname$ wiederhergestellt werden.

Um die Routinen im Bibliotheksbereich zu löschen, benutzt du den Befehl LIBRARY DELETE. Dadurch wird der Speicherplatz freigegeben und der von der Bibliothek belegte Flash-Speicherplatz Slot 3 steht wieder für die Speicherung normaler Programme zur Verfügung. 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 bestimmte Weise einzurichten. Wenn Sie beispielsweise einige Konstanten festlegen möchten, können Sie die folgenden Zeilen in den Bibliothekscode einfügen:

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

Die Bezeichner TRUE und FALSE wurden der Sprache hinzugefügt und stehen jedem Programm, das ausgeführt wird, zur Verfügung.

MM.STARTUP

Manchmal musst du beim ersten Einschalten Code ausführen, zum Beispiel um Hardware zu initialisieren, Optionen einzustellen oder ein benutzerdefiniertes Startbanner zu drucken. Dazu kannst du eine Subroutine mit dem Namen MM.STARTUP erstellen. Wenn die PicoMite-Firmware zum ersten Mal eingeschaltet oder zurückgesetzt wird, sucht sie nach dieser Subroutine und führt sie einmal aus, wenn sie gefunden wird.

Wenn zum Beispiel eine Echtzeituhr an den Raspberry Pi Pico 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 bei Verwendung des Befehls RUN zunächst den Status des Interpreters löscht, um einen Neuanfang zu ermöglichen.

MM.PROMPT

Wenn eine Subroutine mit diesem Namen vorhanden ist, wird sie automatisch von MMBasic ausgeführt, anstatt die Befehlszeile anzuzeigen. Dies kann verwendet 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, sodass alle in dieser Subroutine festgelegten Einstellungen nur für Befehle gelten, die an der Befehlseingabe (d. h. im Sofortmodus) eingegeben werden.

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, weil eine in einer Subroutine definierte Konstante nur für diese Subroutine gilt. DIM erstellt aber globale Variablen, die stattdessen verwendet werden sollten.

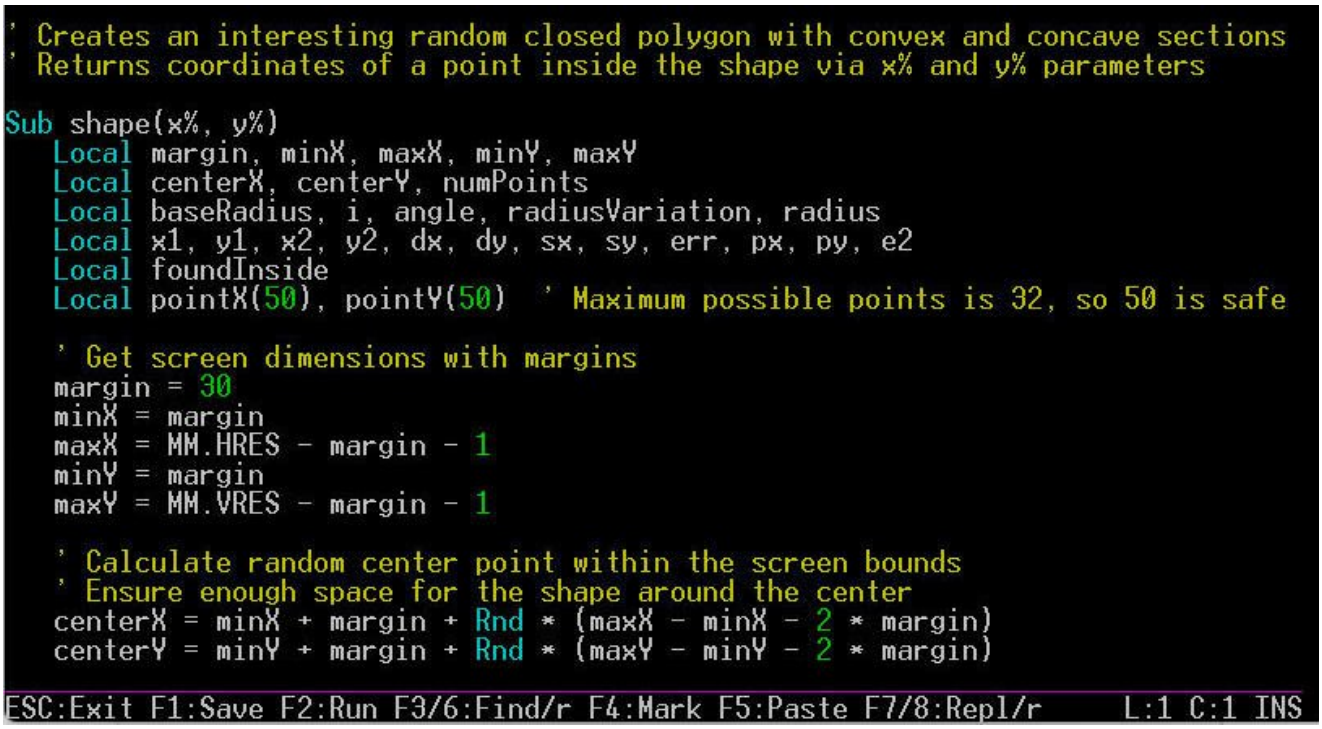
MM.END

Wenn im Programm eine Unterroutine namens MM.END vorhanden ist, wird sie immer dann ausgeführt, wenn das Programm mit einem tatsächlichen oder implizierten 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 Unterroutine MM.END bei Bedarf zu blockieren (weitere Informationen findest du unter dem END-Befehl).

Vollbild-Editor-

Eine wichtige Funktion zur Steigerung der Produktivität ist der integrierte Vollbild-Editor. Wenn er läuft, sieht er so aus:



Der Editor funktioniert mit:

- Der seriellen Konsole mit einem VT100-unterstützten Terminalemulator (wie Tera Term) in allen Versionen.
- Der VGA- oder HDMI-Videoausgang (bei Versionen mit dieser Funktion).
- Einem angeschlossenen LCD-Panel, das mit OPTION LCDPANEL CONSOLE konfiguriert wurde.

Der Editor ist eine super produktive Methode zum Schreiben von Programmen. Mit dem Befehl EDIT kannst du dein Programm eingeben und dann mit der Taste F2 speichern und ausführen. Wenn dein Programm mit einem Fehler stoppt, drückst du die Funktionstaste F4 an der Eingabeaufforderung, um den Editor zu laden und den Cursor an die Zeile zu setzen, die den Fehler verursacht hat. Dieser Zyklus aus Bearbeiten/Ausführen/Bearbeiten geht echt schnell.

Wenn du schon mal einen Editor wie Windows Notepad benutzt hast, wirst du feststellen, dass die Bedienung dieses Editors vertraut ist. Mit den Pfeiltasten kannst du den Cursor im Text bewegen, mit Home und End gelangst du zum Anfang oder Ende der Zeile. Mit „Bild auf“ und „Bild ab“ machst du genau das, was die Namen sagen. Mit der Entf-Taste löschst du das Zeichen am Cursor und mit der Rücktaste das Zeichen davor. Mit der Einfügen-Taste wechselst du zwischen Einfügen und Überschreiben. Die einzige ungewöhnliche Tastenkombination ist, dass du mit zweimaligem Drücken der Home-Taste zum Anfang des Programms und mit zweimaligem Drücken der Ende-Taste zum Ende gelangst.

Am besten lernst du die Verwendung des Editors, indem du ihn einfach startest und damit experimentierst.

Bearbeitungsfunktionen

Wenn der Editor startet, steht der Cursor an der letzten Bearbeitungsposition oder, falls dein Programm wegen eines Fehlers abgebrochen wurde, an der Zeile, die den Fehler verursacht hat. Unten auf dem Bildschirm zeigt die Statuszeile Details wie die aktuelle Cursorposition und die vom Editor unterstützten Standardfunktionen an.

Im Einzelnen sind dies folgende Funktionen:

ESC	Damit verwirft der Editor alle Änderungen und kehrt zur Eingabeaufforderung zurück, ohne dass sich der Programmspeicher verändert hat. Wenn du den Text

geändert hast, wirst du gefragt, ob du deine Änderungen wirklich verwerfen möchtest.

F1: SPEICHERN — Damit wird der Programm-Speicher gespeichert und du kommst zurück zur Eingabeaufforderung.

F2: AUSFÜHREN — Damit wird der Speicherstand gespeichert und das Programm sofort ausgeführt.

F3: SUCHEN — Hier kannst du den Text eingeben, den du suchen willst. Wenn du die Eingabetaste drückst, springt der Cursor an den Anfang des ersten gefundenen Eintrags.

F4: MARK — Das wird weiter unten genauer erklärt.

F5: EINFÜGEN — Damit wird der zuvor ausgeschnittene oder kopierte Text (siehe unten) an der aktuellen Cursorposition eingefügt.

F6: ERNEUT SUCHEN — Wenn du die Suchfunktion benutzt hast, kannst du die Suche mit F6 wiederholen (Shift-F3 geht auch).

F7: ERSETZEN — Wenn der Cursor nach F3 oder F6 steht, öffnet sich ein Dialogfeld, in dem du direkt eine Zeichenfolge in die Zwischenablage eingeben und durch Drücken der Eingabetaste die gesuchte Zeichenfolge ersetzen kannst.

F8: NOCHMAL ERSETZEN — Wenn der Cursor nach F3 oder F6 steht, ersetzt das System die gesuchte Zeichenfolge durch den Inhalt der Zwischenablage.

F9: EINFÜGEN — Du wirst nach einem Dateinamen gefragt und wenn du die Eingabetaste drückst, wird die Datei an der aktuellen Cursorposition eingefügt.

F10: CLIPBOARD SPEICHERN — Du wirst nach einem Dateinamen gefragt und wenn du die Eingabetaste drückst, wird der Inhalt der Zwischenablage (falls vorhanden) in dieser Datei gespeichert. Sieh dir den Markierungsmodus unten an.

Markierungsmodus

Wenn du die Markierungstaste (F4) drückst, geht der Editor in den Markierungsmodus, in dem du Text ausschneiden oder in die Zwischenablage kopieren oder Text löschen oder speichern kannst.

Im Markierungsmodus kannst du mit den Pfeiltasten Text markieren, der dann invers hervorgehoben wird. Wenn du den Anfang oder das Ende des Bildschirms erreichst, markieren die Pfeiltasten weiterhin Text, während der Bildschirm scrollt. Das 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 — Verlasse den Markierungsmodus, ohne etwas zu ändern.

F4: AUSSCHNEIDEN — Kopiere den markierten Text in die Zwischenablage und entferne ihn aus dem Programm.

F5: COPY — Kopiere den markierten Text einfach in die Zwischenablage.

F10: SPEICHERN — Frag nach einem Dateinamen und speicher den markierten Text in der Datei, wenn die Eingabetaste gedrückt wird.

LÖSCHEN — Lösch den markierten Text, ohne die Zwischenablage zu verändern.

Das Ausschneiden oder Kopieren ist auf maximal 2048 Zeichen begrenzt.

Alternative Tasten

Du kannst auch Steuertasten anstelle der oben aufgeführten Funktionstasten verwenden. Diese Steuertasten sind:

LINKS	Strg-S	RECHTS	Strg-D	OBEN	Strg-E	DOWN	Strg-X
HOME	Strg-U Strg-L	END	Strg-K	Bild auf	Strg-P	Bild nach unten	
Entf	Strg-]	EINFÜGEN		Strg-N	F1	Strg-Q	F2 Strg-W
F3	Strg-R	F4	Strg-T	F5	Strg-Y	F6	Strg-G
F7	Strg-F	F8	Strg-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 da, wird aber nicht angezeigt und kann nicht bearbeitet werden. Wenn du eine sehr lange Zeile bearbeiten willst, 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, kannst du mit der Entf- oder Rücktaste den zuvor eingefügten Zeilenumbruch entfernen.

Alternativ kannst du vor dem Aufrufen des Editors die Fortsetzungszeilen aktivieren (OPTION CONTINUATION LINES ON). Damit kannst du am Ende einer Zeile ein Leerzeichen gefolgt von einem Unterstrich verwenden, um anzuzeigen, dass die nächste Zeile eine Fortsetzung ist und für die Ausführung des Programms zusammengefügt werden soll. Die Zusammenfügung erfolgt beim Speichern der Datei, und beim erneuten Bearbeiten werden lange Zeilen automatisch geteilt, wenn die Datei in den Editor eingelesen wird. Die Zeilenumbrüche sind möglicherweise nicht an derselben Stelle, aber der Editor versucht, sie an sinnvollen Stellen (am Ende von Wörtern usw.) zu platzieren. Es gibt keine Einschränkungen hinsichtlich der Platzierung von Fortsetzungszeichen. Sie können sich beispielsweise in der Mitte einer Zeichenfolge in Anführungszeichen befinden. Die Begrenzung auf maximal 255 Zeichen in einer verketteten Zeile gilt weiterhin, und der Editor lässt dich nicht aussteigen, 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. Details zum Anschließen einer Maus findest du unter der Überschrift *„Tastatur/Maus/Gamepad“* weiter unten in diesem Handbuch.

Wenn du den Editor mit angeschlossener Maus startest und dich im Videomodus 1 mit aktivierter Farbcodierung befindest, siehst du ein Zeichen, das auf weißem Hintergrund rot hervorgehoben ist. Diese Markierung kann mit der Maus verschoben werden. Durch Klicken mit der linken Maustaste wird der Bearbeitungscursor an diese Position verschoben (d. h. wie bei Verwendung der Cursortasten). Ein Klick mit der rechten Maustaste entspricht dem Drücken der Taste F4 auf der Tastatur, und ein Klick auf das Scrollrad entspricht der Taste F5.

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

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

Farbcodierte Editoranzeige

Der Editor kann das bearbeitete Programm mit Schlüsselwörtern, Zahlen und Kommentaren in verschiedenen Farben farblich kennzeichnen. Diese Funktion kann über die Eingabeaufforderung 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, Bezeichnungen, Variablennamen, Dateinamen usw. nicht zwischen Groß- und Kleinschreibung unterschieden, sodass „Run" und „RUN" gleich sind und „dOO" und „Doo" sich auf dieselbe Variable beziehen.

Variablen

Variablen können mit einem Buchstaben oder einem Unterstrich beginnen und beliebige Buchstaben, Zahlen, Punkte (.) und Unterstriche (_) enthalten. Sie können bis zu 31 Zeichen lang sein.

Ein Variablenname oder eine Bezeichnung 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.
Beispielsweise ist step = 5 nicht zulässig, da STEP ein Schlüsselwort ist.

MMBasic kann drei Arten von Variablen:

1. Doppelte Genauigkeit mit Gleitkomma.
   Damit kannst du Zahlen mit Dezimalstellen und Bruchteilen speichern (z. B. 45,386), aber wenn du mehr als 14 Stellen nimmst, wird es ungenauer. 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 ohne Genauigkeitsverlust speichern, aber sie können keine Bruchteile (d. h. den Teil nach dem Dezimalpunkt) speichern. Sie werden angegeben, indem man das Suffix „%" an den Namen einer Variablen anhängt. Zum Beispiel i%, nbr% usw.
3. Eine Zeichenfolge.
   Eine Zeichenkette speichert eine Folge von Zeichen (z. B. „Tom"). Jedes Zeichen in der Zeichenkette wird als 8-Bit-Zahl gespeichert und kann daher einen Dezimalwert von 0 bis 255 haben. Die Namen von Zeichenkettenvariablen enden mit dem Symbol „$" (z. B. name$, s% usw.). Zeichenketten können bis zu 255 Zeichen lang sein.

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

Die meisten Programme verwenden Fließkommavariablen für arithmetische Operationen, da diese mit den in typischen Situationen verwendeten Zahlen umgehen können und bei Divisionen und Brüchen intuitiver sind als Ganzzahlen. Wenn Sie sich also nicht um die Details kümmern möchten, verwenden Sie immer Fließkommazahlen.

Konstanten

Numerische Konstanten können mit einer Ziffer (0-9) für eine Dezimalkonstante, mit &H für eine Hexadezimalzahl, mit &O für eine Oktalzahl oder mit &B für eine Binärzahl beginnen. Zum Beispiel entspricht &B1000 der Dezimalkonstante 8. Konstanten, die mit &H, &O oder &B beginnen, werden immer als 64-Bit-Ganzzahlkonstanten ohne Vorzeichen behandelt.

Dezimalen können mit einem Minuszeichen (-) oder Pluszeichen (+) beginnen und mit „E" gefolgt von einer Exponentialzahl enden, um die Exponentialdarstellung anzuzeigen. Zum Beispiel entspricht 1,6E+4 dem Wert 16000.

Wenn eine konstante Zahl verwendet wird, wird angenommen, dass es sich um eine Ganzzahl handelt, wenn kein Dezimalpunkt oder Exponent verwendet wird. Zum Beispiel wird 1234 als Ganzzahl interpretiert, während 1234.0 als Gleitkommazahl interpretiert wird.

Zeichenfolgenkonstanten müssen in doppelte Anführungszeichen („") gesetzt werden. Beispiel: „Hello World".

OPTION DEFAULT

Eine Variable kann ohne Suffix (d. h. !, % oder $) verwendet werden. In diesem Fall verwendet MMBasic den Standardtyp „Gleitkomma". So wird zum Beispiel die folgende Zeile eine Gleitkommavariable erstellen:

```
Nbr = 1234
```

Der Standardtyp kann aber mit dem Befehl OPTION DEFAULT geändert werden. Mit OPTION DEFAULT INTEGER wird zum Beispiel festgelegt, dass alle Variablen ohne bestimmten Typ Ganzzahlvariablen sind. So wird mit dem folgenden Befehl eine Ganzzahlvariable erstellt:

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

Der Standardwert kann auf FLOAT (das ist der Standardwert, wenn ein Programm ausgeführt wird), INTEGER, STRING oder NONE gesetzt werden. Im letzteren Fall müssen alle Variablen explizit typisiert werden, sonst kommt es zu einem Fehler.

Der Befehl OPTION DEFAULT kann an beliebiger Stelle im Programm platziert und jederzeit geändert werden. Es empfiehlt sich jedoch, 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 wird mit `Nbr = 1234` die Variable erstellt und gleichzeitig auf den Wert 1234 gesetzt. Dies ist für kurze und schnelle Programme praktisch, kann jedoch in großen Programmen zu subtilen und schwer zu findenden Fehlern führen. In der dritten Zeile dieses Fragments wurde die Variable `Nbr` beispielsweise 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 eine gute Programmierpraxis zu unterstützen. Wenn er verwendet wird, sollte er am Anfang des Programms platziert werden, bevor Variablen verwendet werden.

DIM und LOCAL

Die Befehle DIM und LOCAL können zum Definieren einer Variablen und zum Festlegen ihres Typs verwendet werden und sind obligatorisch, wenn der Befehl OPTION EXPLICIT verwendet wird.

Der Befehl DIM erstellt eine globale Variable, die im gesamten Programm, einschließlich innerhalb von Unterprogrammen und Funktionen, sichtbar ist und verwendet werden kann. Wenn die Definition jedoch nur innerhalb eines Unterprogramms oder einer Funktion sichtbar sein soll, sollte der Befehl LOCAL am Anfang des Unterprogramms oder der Funktion verwendet werden. LOCAL hat genau die gleiche Syntax wie DIM.

Wenn LOCAL verwendet wird, um eine Variable mit dem gleichen Namen wie eine globale Variable anzugeben, 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. Alle mit LOCAL erstellten Variablen verschwinden, wenn das Programm die Unterroutine verlässt.

Auf der einfachsten Ebene können DIM und LOCAL verwendet werden, um eine oder mehrere Variablen basierend auf ihrem Typ-Suffix oder der zu diesem Zeitpunkt gültigen 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 usw.
```

In diesem Fall werden nbr, nbr2, nbr3 usw. alle als Ganzzahlen erstellt. Wenn du die Variable in einem Programm benutzt, musst du das Typ-Suffix nicht angeben. Zum Beispiel funktioniert `MyStr` im folgenden Beispiel perfekt als String-Variable:

```
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 ihrer 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 auch benutzerdefinierte Funktionen enthalten kann.

Die Befehle DIM oder LOCAL werden auch zum Definieren eines Arrays verwendet, und alle oben aufgeführten Regeln gelten auch für die Definition eines Arrays. Beispiel:

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

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

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

oder

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

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 den 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 den folgenden Aufrufen nicht initialisiert).

Wenn du zum Beispiel die folgende Subroutine hast und sie wiederholt aufrufst, würde der erste Aufruf 5, der zweite 6, der dritte 7 usw. ausgeben.

```
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 wirkt. Bei späteren Aufrufen wird die Initialisierung ignoriert, da die Variable schon beim ersten Aufruf erstellt wurde.

Wie bei DIM und LOCAL können die mit STATIC erstellten Variablen Float-Werte, Ganzzahlen oder Zeichenfolgen und Arrays davon mit oder ohne Initialisierung sein. Die Länge des von STATIC erstellten Variablennamens und die Länge des Unterprogramm- oder Funktionsnamens dürfen zusammen nicht mehr als 31 Zeichen betragen.

CONST

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

Mit dem Befehl CONST kannst du einen Bezeichner erstellen, der wie eine Variable funktioniert, 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 gelegentlichen Leser sinnvoller sind als einfache Zahlen. Zum Beispiel:

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

In einer Zeile kannst du mehrere Konstanten erstellen:

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

Der Wert, der zum Initialisieren der Konstante verwendet wird, wird beim Erstellen der Konstante ausgewertet und kann ein Ausdruck sein, der benutzerdefinierte Funktionen enthält.

Der Typ der Konstante wird aus dem ihr zugewiesenen Wert abgeleitet; so ist beispielsweise `MaxValue` oben 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 jedoch mit dem ihr zugewiesenen Wert übereinstimmen.

Sonderzeichen in Zeichenfolgen

Spezielle, nicht druckbare Zeichen können mit dem Backslash (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:

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

Zum Beispiel wird das Folgende die Wörter „Hello“ und „World“ in separaten Zeilen ausgeben:

```
OPTION ESCAPE
PRINT „Hallo\r\nWelt”
```

Ausdrücke und Operatoren

MMBasic wertet einen mathematischen Ausdruck nach den üblichen mathematischen Regeln aus. Zum Beispiel werden Multiplikation und Division zuerst gemacht, dann kommen Addition und Subtraktion. Das sind die Vorrangregeln, die weiter unten genauer erklärt werden.

Das heißt, dass 2 + 3 * 6 zu 20 führt, genauso wie 5 * 4 und auch 10 + 4 * 3 – 2.

Wenn du den Interpreter dazu bringen willst, Teile des Ausdrucks zuerst zu berechnen, kannst du diesen Teil des Ausdrucks in Klammern setzen. Zum Beispiel ergibt (10 + 4) * (3 – 2) 14 und nicht 20, wie es ohne Klammern der Fall gewesen wäre. 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 in der Reihenfolge ihrer Priorität implementiert. Operatoren, die auf derselben Ebene stehen (z. B. + und -), werden in der Reihenfolge ihrer Erscheinung in der Programmzeile von links nach rechts verarbeitet.

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	Die funktionieren auf eine besondere Art. << heißt, dass der Wert, den du zurückbekommst, der Wert von x ist, der um y Bits nach links verschoben wurde, während >> dasselbe bedeutet, nur nach rechts verschoben. Das sind ganzzahlige Funktionen, und alle Bits, die rausgeschoben werden, werden weggeworfen, und alle Bits, die reingeschoben werden, werden auf Null gesetzt.

Logische Operatoren:

NOT INV	Kehren den logischen Wert auf der rechten Seite um (z. B. `NOT(a=b)` ist `a<>b`) oder bitweise Umkehrung des Werts auf der rechten Seite (z. B. `a = INV b`)
<> < > <= =< >= =>	Ungleichheit, kleiner als, größer als, kleiner oder gleich, kleiner oder gleich (alternative Version), größer oder gleich, größer oder gleich (alternative Version)
=	Gleichheit
UND ODER XOR	Konjunktion, Disjunktion, exklusives Oder

Aus Gründen der Kompatibilität mit Microsoft sind die Operatoren AND, OR und XOR ganzzahlige Bitoperatoren. Zum Beispiel gibt PRINT (3 AND 6) die Zahl 2 aus. Weil 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) funktionieren können, wird der Interpreter verwirrt, wenn sie im selben Ausdruck gemischt werden. Also, bewerte logische und bitweise Ausdrücke immer in separaten Ausdrücken.

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

Der NOT-Operator kehrt den logischen Wert auf seiner rechten Seite 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 mit dem nächsten Wert verbunden sind. Für die normale Verwendung von NOT oder INV sollte der zu verarbeitende Ausdruck in Klammern gesetzt werden. Beispiel:

```
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 des ASCII-Werts.
=	Gleichheit

Bei der Vergleichung von Zeichenfolgen wird die Groß-/Kleinschreibung beachtet. Zum Beispiel ist „A" größer als „a".

Mischen von Gleitkommazahlen und Ganzzahlen

MMBasic kümmert sich automatisch um die Umwandlung von Zahlen zwischen Gleitkomma- und Ganzzahlen. Wenn eine Operation sowohl Gleitkomma- als auch Ganzzahlen mischt (z. B. `PRINT A% + B!`), wird die Ganzzahl zuerst in eine Gleitkommazahl umgewandelt, dann wird die Operation durchgeführt und eine Gleitkommazahl zurückgegeben. Wenn beide Seiten des Operators Ganzzahlen sind, wird eine Ganzzahloperation durchgeführt und eine Ganzzahl zurückgegeben.

Die einzige Ausnahme ist die normale Division („/“), bei der immer beide Seiten des Ausdrucks in eine Gleitkommazahl umgewandelt werden und dann eine Gleitkommazahl zurückgegeben wird. Für die ganzzahlige Division solltest du den ganzzahligen Divisionsoperator „\“ verwenden.

MMBasic-Funktionen geben je nach ihren Eigenschaften einen Float- oder einen Integer-Wert zurück. PIN() gibt zum Beispiel einen Integer-Wert zurück, wenn der Pin als digitaler Eingang konfiguriert ist, aber einen Float-Wert, wenn er als analoger Eingang konfiguriert ist.

Bei Bedarf kannst du eine Gleitkommazahl mit der Funktion INT() in eine Ganzzahl umwandeln. Es ist nicht nötig, eine Ganzzahl speziell in eine Gleitkommazahl umzuwandeln, aber wenn es nötig wäre, könnte der Ganzzahlwert einer Gleitkommavariablen zugewiesen werden und würde bei der Zuweisung automatisch umgewandelt werden.

Strukturen

Mit Strukturen (auch als benutzerdefinierte Typen bekannt) kannst du zusammengehörige Variablen unterschiedlicher Typen unter einem einzigen Namen zusammenfassen. Das ist praktisch, um komplexe Daten wie Koordinaten, Datensätze oder jede Sammlung zusammengehöriger Werte zu organisieren. Dieser Abschnitt gibt einen kurzen Überblick, aber Strukturen werden ausführlich in der Datei *MMBasic_Structures_Manual.pdf* beschrieben, die im ZIP-Archiv zum Firmware-Download enthalten ist.

Als Beispiel für eine Struktur nehmen wir an, dass du mehrere Alarme verfolgen musst. Jeder Alarm hat mehrere Eigenschaften: den Zeitpunkt des Alarms, seine Stufe (1, 2, 3 usw.) und die ergriffene Maßnahme. Mit Strukturen kannst du die Daten für einen Alarm als eine einzige „Sache” 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 alle normalen Datentypen in MMBasic sein (FLOAT, INTEGER oder STRING). Der neue Typ, den du definiert hast, kann dann zum Definieren einer Variablen verwendet werden. Du könntest zum Beispiel eine neue Variable namens „alarm” mit folgendem Befehl definieren:

```
DIM alarm AS StructAlarm
```

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

Auf die Elemente der Struktur kannst du mit einem Punkt (.) zwischen dem Namen der Variablen und dem Namen des Elements zugreifen. Zum Beispiel: `alarm.time`.

Du kannst den Elementen der Struktur Werte mit einer normalen Zuweisung zuweisen. Zum Beispiel:

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

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

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

Du kannst eine Struktur einer anderen Struktur desselben Typs 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 wie jedes normale Array definieren:

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

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

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

MMBasic hat viele Befehle für die spezielle Behandlung von Strukturen, zum Beispiel zum Speichern und Laden aus einer Datei, zum Sortieren eines Struktur-Arrays, zum Zurücksetzen der Elemente einer Struktur und so weiter. Alle Details findest du in der Datei *MMBasic_Structures_Manual.pdf,* die im ZIP-Archiv mit der Firmware enthalten ist.

64-Bit-Ganzzahlen ohne Vorzeichen

MMBasic unterstützt 64-Bit-Ganzzahlen mit Vorzeichen. Das heißt, es gibt 63 Bits für die Zahl und ein Bit (das höchstwertige Bit), das das Vorzeichen (positiv oder negativ) angibt. Man kann aber auch 64-Bit-Ganzzahlen ohne Vorzeichen verwenden, solange man keine Rechenoperationen mit den Zahlen macht.

64-Bit-Ganzzahlen ohne Vorzeichen können mit den Präfixen &H, &O oder &B vor einer Zahl erstellt werden, und diese Zahlen können in einer Ganzzahlvariablen gespeichert werden. Dann gibt es nur eine begrenzte Anzahl von Operationen, die du mit diesen Zahlen durchführen kannst. Das sind << (nach links verschieben), >> (nach rechts verschieben), AND (bitweise UND), OR (bitweise ODER), XOR (bitweise exklusives 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$() benutzen.

Die folgende 64-Bit-Operation ohne Vorzeichen liefert zum Beispiel die erwarteten Ergebnisse:

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

Zeigt „800F0000FFFF0044" an

Unterprogramme und Funktionen

Eine in einem Programm definierte Unterroutine oder Funktion ist einfach ein Block von Programmcode, der in einem Modul enthalten ist und von überall in deinem Programm aufgerufen werden kann. Das ist so, als hättest du der Sprache deinen eigenen Befehl oder deine eigene Funktion hinzugefügt.

Unterprogramme

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

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

Wenn du die Unterroutine aufrufst, kannst du den Argumenten Werte zuweisen. Zum Beispiel:

```
MYSUB  23, "Katze", 55
```

Innerhalb der Unterprogramm hat `arg1` den Wert `23`, `arg2$` den Wert „`Cat`" und so weiter. Die Argumente verhalten sich wie normale Variablen, existieren aber nur innerhalb der Unterprogramm und verschwinden, wenn die Unterprogramm endet. Du kannst Variablen mit dem gleichen Namen im Hauptprogramm haben, die dann von den für die Unterprogramm definierten Argumenten verdeckt werden.

Wenn du eine Subroutine aufrufst, kannst du weniger als die erforderliche Anzahl von Werten angeben. In diesem Fall werden die fehlenden Werte entweder als Null oder als leere Zeichenfolge angenommen. Du kannst auch einen Wert in der Mitte der Liste weglassen, dann passiert dasselbe. Zum Beispiel:

```
MYSUB  23, , 55
```

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

Anstatt das Typ-Suffix (z. B. das $ in arg2$) zu verwenden, kannst du das Suffix AS <Typ> in der Definition des Unterprogrammarguments verwenden, und dann wird das Argument als der angegebene Typ erkannt, auch wenn das Suffix nicht verwendet wird. 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 die gleiche Syntax wie DIM). Diese Variable existiert nur innerhalb der Subroutine und verschwindet, wenn die Subroutine beendet wird.

Funktionen

Funktionen sind ähnlich wie Unterprogramme, mit dem Hauptunterschied, 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 man dem Namen der Funktion innerhalb der Funktion einen Wert zu. Wenn der Name der Funktion mit einem $, einem % oder einem ! endet, gibt die Funktion diesen Typ zurück, andernfalls gibt sie den Wert zurück, auf den OPTION DEFAULT gesetzt ist. Man kann den Typ der Funktion auch angeben, indem man AS <Typ> an das Ende der Funktionsdefinition anhängt.

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 Funktion eine normale Variable (also keinen Ausdruck) als Wert verwendest, zeigt das Argument innerhalb der Subroutine/Funktion zurück auf die im Aufruf verwendete Variable, und alle Änderungen am Argument werden auch an der übergebenen Variable vorgenommen. Dies wird als Übergabe von Argumenten per Referenz bezeichnet.

Du kannst zum Beispiel 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 das klappt, müssen der Typ der übergebenen Variablen (z. B. `nbr1`) und das definierte Argument (z. B. `a`) gleich sein (im obigen Beispiel sind beide standardmäßig float).

Wenn du ein Argument als allgemeine Variable in einer Subroutine oder Funktion verwenden willst (also seinen Wert ändern willst), solltest du vor seiner Definition das Schlüsselwort BYVAL setzen. Das sagt MMBasic, dass es immer den Wert des Arguments nehmen soll, auch wenn es eine Variable ist, und nie auf die Variable zurückgreifen soll, die im Aufruf benutzt wird. Der Grund dafür ist, dass ein anderer Nutzer deiner Routine unwissentlich eine Variable in seinem Aufruf verwenden könnte, die dann von deiner Routine „auf magische Weise" geändert würde, wenn du BYVAL nicht verwendet hättest.

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

Arrays übergeben

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

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

Diese Art von Konstruktion wird oft zum Sortieren von Arrays verwendet.

Du kannst auch ein oder mehrere komplette Arrays an eine Subroutine oder Funktion übergeben, indem du das Array mit leeren Klammern anstelle der normalen Dimensionen angibst, z. B. `a()`. In der Subroutine- oder Funktionsdefinition 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, die bei der Indizierung des Arrays beachtet werden müssen. Bei Bedarf können die Dimensionen des Arrays als zusätzliche Argumente an die Subroutine oder Funktion übergeben werden oder 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 zum Beispiel das Folgende ausgeführt wird, werden die Wörter „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
```

Vorzeitiges Beenden

Für jede Definition einer Unterroutine oder Funktion kann es nur ein END SUB oder END FUNCTION geben. Um eine Unterroutine vorzeitig zu verlassen (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. Auf ähnliche Weise kannst du EXIT FUNCTION verwenden, um eine Funktion vorzeitig zu verlassen.

Rekursion

Rekursion ist, wenn eine Unterprogramm oder Funktion sich selbst aufruft. Du kannst Rekursion in MMBasic machen, aber es gibt ein paar Probleme (die sind eine direkte Folge der Einschränkungen von Mikrocontrollern und der Sprache BASIC):

- Die Tiefe der Rekursion ist auf einen festen Wert begrenzt. In der PicoMite-Firmware sind das 50 Ebenen.
- Wenn du viele Argumente für die Unterroutine oder Funktion und viele LOCAL-Variablen (insbesondere Zeichenfolgen) hast, kann der Speicherplatz schnell knapp werden, bevor die Grenze von 50 Ebenen erreicht ist.
- 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, aber in vielen Fällen können diese mit einer gewöhnlichen Unterprogramm oder Funktion erstellt werden, die dann genau wie ein integrierter Befehl oder eine integrierte Funktion funktioniert.

Manchmal wird zum Beispiel eine TRIM-Funktion benötigt, die bestimmte Zeichen am Anfang und am Ende einer Zeichenfolge entfernt. Im Folgenden findest du ein Beispiel dafür, wie man eine solche einfache Funktion in MMBasic erstellen kann.

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

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

' Alle Zeichen in c$ vom Ende von s$ entfernen
Funktion RTrim$(s$, c$)
  RTrim$ = s$
  Solange Instr(c$, Right$(RTrim$, 1))
    RTrim$ = Mid$(RTrim$, 1, Len(RTrim$) - 1)
  Schleife
Ende Funktion

' Alle Zeichen in c$ vom Anfang von s$ entfernen
Funktion LTrim$(s$, c$)
  LTrim$ = s$
  Do While Instr(c$, Left$(LTrim$, 1))
    LTrim$ = Mid$(LTrim$, 2)
  Schleife
Ende Funktion
```

Hier ein Beispiel, wie man diese Funktionen benutzt:

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

Das Ergebnis ist „****23.56700”.

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

Ergibt „23,567”

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

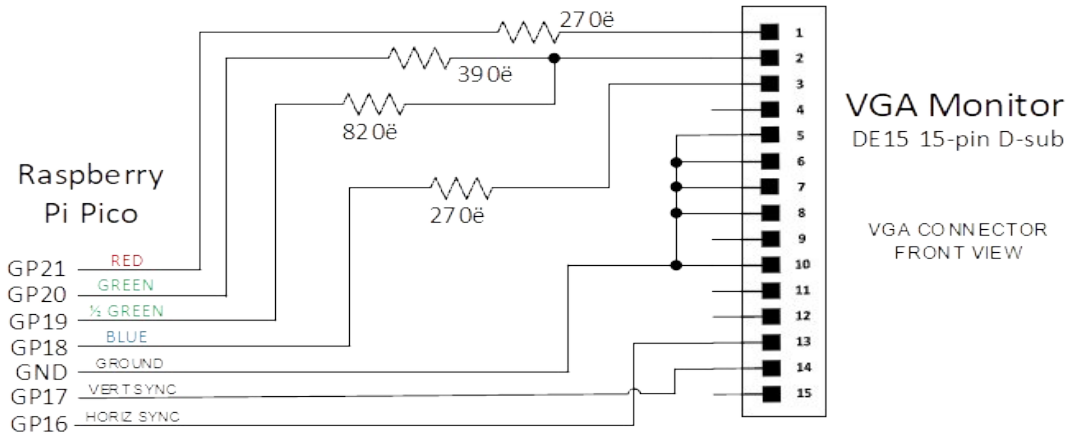
Ergibt „23,56700”

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 anschließen kannst.



Die Ausgabe erfolgt im Standard-VGA-Format mit einer Pixelfrequenz von 25,175 MHz und einer Bildfrequenz 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 bei 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 MODUS 2 und 3 sind es 16 Farben im 4-Bit-RGB121-Format (also 1 Bit für Rot, 2 Bits für Grün und 1 Bit für Blau). In 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 immer noch ein 640 x 480-Signal sieht.

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 senden und so die Anzeige zu erzeugen. Da dies unabhängig vom Hauptprozessor läuft, hat die Erzeugung der VGA-Ausgabe kaum oder gar keinen Einfluss auf die Geschwindigkeit von MMBasic.

HDMI-Video

Für Versionen der Firmware, die HDMI-Video unterstützen, sind in der folgenden Tabelle die Anschlüsse an die Standard-HDMI-Buchse vom Typ A aufgeführt. Der HDMI-Ausgang wird beim Start automatisch aktiviert – es müssen keine Optionen eingestellt werden.

HDMI-Pin 1: Pin 21 (GP16) über einen 220-Ω-Widerstand

HDMI-Pin 2: Masse

HDMI-Pin 3: Pin 22 (GP17) über einen 220-Ω-Widerstand

HDMI-Pin 4: Pin 24 (GP18) über einen 220-Ω-Widerstand

HDMI-Pin 5: Masse

HDMI-Pin 6: Pin 25 (GP19 über einen 220-Ω-Widerstand

HDMI-Pin 7: Pin 16 (GP12) über einen 220-Ω-Widerstand

HDMI-Pin 8: Masse

HDMI-Pin 9: Pin 17 (GP13) über einen 220-Ω-Widerstand

HDMI-Pin 13: Keine Verbindung

HDMI-Pin 14: Keine Verbindung

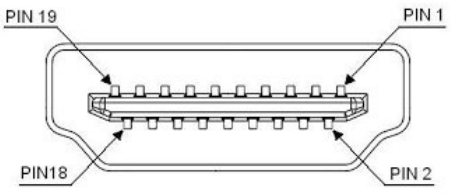
HDMI-Pin 15: Keine Verbindung

HDMI-Pin 16: Keine Verbindung

HDMI-Pin 17: Masse

HDMI-Pin 18: +5 V über Schottky-Barrierendiode

HDMI-Pin 19: Keine Verbindung



HDMI-Pin 10: Pin 19 (GP14) über einen 220-Ω-Widerstand HDMI
HDMI-Pin 11: Masse Vorderseite
HDMI-Pin 12: Pin 20 (GP15) über einen 220-Ω-Widerstand Ansicht

Die HDMI-Signal-Pins werden mit hoher Frequenz angesteuert, deshalb solltest du auf Folgendes achten:

- Halte die Signalleitungen so kurz wie möglich.
- Stell sicher, dass alle Signalleitungen gleich lang sind.
- Die 220-Ω-Widerstände sollten vorzugsweise oberflächenmontiert sein.

Um das DVI/HDMI-Signal zu erzeugen, muss die Firmware den RP2350 auf bis zu 372 MHz übertakten, was für die meisten Raspberry Pi Pico 2-Module bei diesen Geschwindigkeiten kein Problem darstellt. Dies kann jedoch nicht garantiert werden, insbesondere bei Modulen von Drittanbietern. Ein Beispiel hierfür ist das Pimoroni Pico Plus 2, das bei den erforderlichen Geschwindigkeiten an seine Grenzen stößt und daher nicht für die HDMI-Versionen der PicoMite-Firmware empfohlen werden kann.

Ähnlich wie bei der Erzeugung von VGA wird die Ausgabe von MMBasic in einen Framebuffer geschrieben, der mithilfe der zweiten CPU und DMA an das HSTX-Peripheriegerät weitergeleitet wird, das 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 anspruchsvolle HDMI-Funktionen wie High Definition Content Protection (HDCP) und Ethernet nicht unterstützt werden.

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

```
OPTION RESOLUTION nn
```

Dabei ist „nn" einer der folgenden Werte:

640x480 oder 640
1280x720 oder 1280
1024x768 oder 1024
800x600 oder 800
720x400 oder 720
848 x 480 oder 848

Jede HDMI-Auflösung kann in verschiedenen Modi betrieben werden, die mit dem 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, aber der Monitor sieht immer die mit dem Befehl OPTION RESOLUTION eingestellte Auflösung (d. h. Pixeldichte). Die Standardeinstellung ist RESOLUTION 640x480 und MODE 1

VGA/PS2-Referenzdesign (Raspberry Pi Pico)

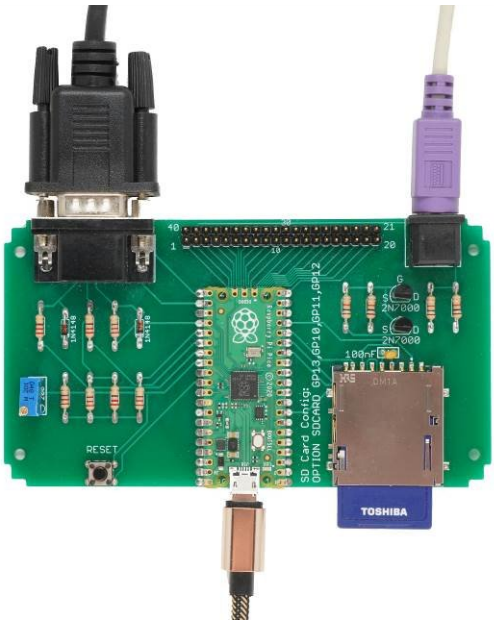
Dies ist ein einfach zu montierendes Design, das den VGA-Ausgang, die PS2-Tastaturschnittstelle und den SD-Kartensteckplatz implementiert (dieses Design wurde im Magazin *„Silicon Chip"* vorgestellt).

Es verwendet gängige Durchsteckkomponenten und kann in weniger als einer Stunde zusammengebaut werden.

Alle 40 Pins auf dem Raspberry Pi Pico sind mit einem 40-poligen Stecker auf der Rückseite der Leiterplatte verbunden, genau wie beim Pico. So kannst du ganz einfach externe Geräte anschließen, indem du dir das Pinbelegungsdiagramm in diesem Handbuch anschaust und dann die passenden Pins auf dem 40-poligen Stecker auswählst.

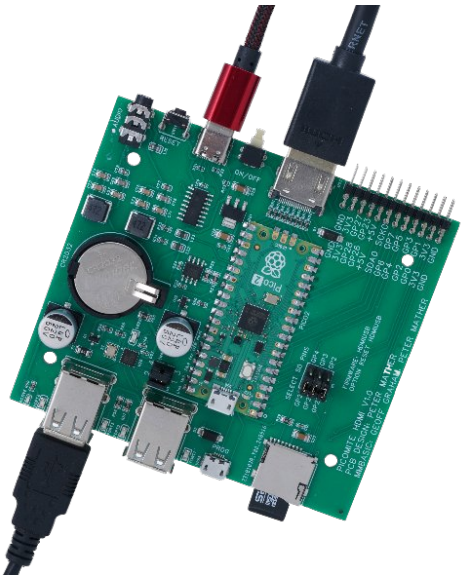
Unter Berücksichtigung der für den VGA-Ausgang, die Tastatur und die SD-Karte reservierten I/O-Pins stehen 14 I/O-Pins für externe Schaltungen zur Verfügung.

Die Platine passt in ein Altronics-Steckgehäuse mit den Maßen 130 x 75 x 28 mm (Teilenummer H0376).



Das Konstruktionspaket für dieses Design kannst du hier runterladen: https://geoffg.net/picomitevga.html (unten auf der Seite).

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 verstärkte Lautsprecher.
- USB-Schnittstelle zur seriellen Konsole.
- Batteriegepufferte Echtzeituhr.
- Micro-SD-Kartensteckplatz.
- 14 I/O-Pins auf der Rückseite.
- Passend für ein Multicomp MCRM2015S- oder Hammond RM2015S-Gehäuse.

Das Konstruktionspaket für dieses Design kannst du hier runterladen: https://geoffg.net/picomitevga.html (unten auf der Seite).

Tastatur/Maus/Gamepad

Die PicoMite-Firmware unterstützt Tastaturen und Mäuse, die entweder über eine PS2- oder eine USB-Schnittstelle angeschlossen werden. Ob PS2 oder USB, hängt von der Version der Firmware ab, die geladen ist. Schau dir dazu das Kapitel *„Firmware-Versionen und Dateien"* am Anfang dieses Handbuchs an.

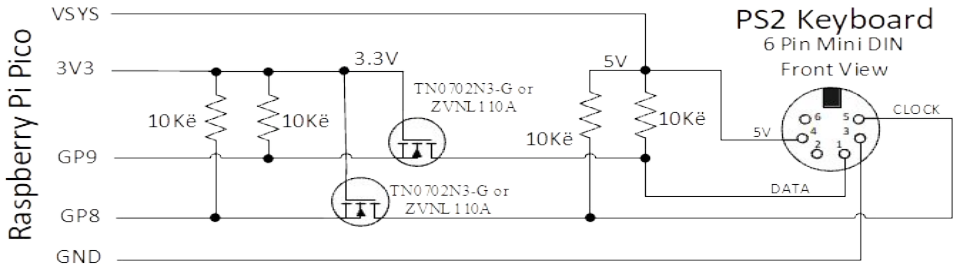
USB-Versionen der Firmware unterstützen auch ein PS3- oder PS4-Gamepad mit USB-Schnittstelle. Bei Versionen ohne USB-Unterstützung können die Befehle WII (CLASSIC) und WII NUNCHUCK verwendet werden, um ein über I^2C angeschlossenes Gamepad anzugeben.

Eine Tastatur kann zur Eingabe von Daten in das BASIC-Programm verwendet werden oder, mit einem VGA/HDMI-Videoausgang, zum Erstellen eines eigenständigen Computers mit Tastatur und Display. Anstelle eines Videoausgangs kannst du bei Versionen, die dies unterstützen, auch ein LCD-Panel anschließen und die Ausgabe der MMBasic-Konsole auf dem LCD-Panel anzeigen, wodurch eine kompaktere Version eines eigenständigen Computers entsteht. Weitere Infos dazu findest du im Abschnitt *„LCD-Display als Konsolenausgabe*".

PS2-Tastatur auf dem 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 mehr als 3,6 V ausgesetzt werden. Deshalb 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.

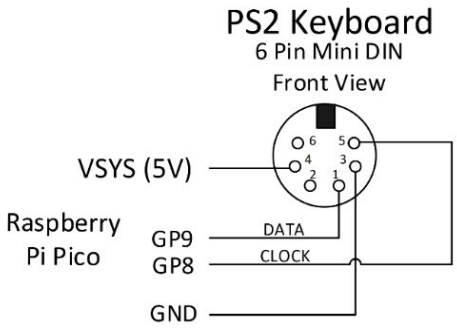
Es gibt viele Möglichkeiten, dies zu erreichen, aber die folgende Schaltung ist eine einfache und kostengünstige Lösung:



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

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

PS2-Tastatur auf dem Raspberry Pi Pico 2 (RP2350)



Die I/O-Pins der Mikrocontroller der RP2350-Serie können 5 V aushalten (wenn sie eingeschaltet sind), sodass die Tastatur direkt angeschlossen werden kann, wie gezeigt, mit der 5-V-Spannung vom VSYS-Pin des Raspberry Pi Pico 2.

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 Eingabeaufforderung) oder MOUSE OPEN (innerhalb eines Programms) konfiguriert werden.

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

USB-Schnittstelle

Versionen der Firmware (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 zum normalen Modus als USB-Client). Das geht, weil der Anschluss und die Elektronik des Pico USB-OTG-kompatibel (On The Go) sind, ähnlich wie der Anschluss vieler Handys.

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 willst.

Um ein USB-Gerät anzuschließen, brauchst du ein Konverterkabel mit einem Micro-USB-Stecker an einem Ende (für den Pico) und einer USB-Buchse vom Typ A am anderen Ende (für das Gerät). Ein typisches Beispiel ist das Jaycar Cat Nbr WC7725.

Da die USB-Schnittstelle des Pico zu einem USB-Host umgewandelt wurde, hast du keinen Zugriff mehr auf die serielle MMBasic-Konsole. Die Firmware gleicht das aus, indem sie automatisch Pin 11 (GP8) für die serielle Konsole Tx und Pin 12 (GP9) für Rx nutzt und die Baudrate auf 115200 Baud einstellt. Um auf diese Konsole zuzugreifen, brauchst du eine USB-zu-Seriell-Brücke, die auf der einen Seite eine TTL-Seriellschnittstelle und auf der anderen Seite eine USB-Schnittstelle hat (such nach Modulen mit dem CP2102- oder CH340-Chip). Bei Bedarf kannst du mit OPTION SERIAL CONSOLE die für die Konsole verwendeten Pins ändern.

USB-Hub

Die Firmware unterstützt über diese Schnittstelle auch einen USB-Hub, sodass mehrere Tastaturen oder eine Tastatur plus Maus plus Gamepad usw. angeschlossen werden können. 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 zurückzugeben. Standardmäßig wird eine Tastatur dem Kanal 1 zugewiesen. Eine Maus wird dem Kanal 2 zugewiesen. Das erste Gamepad wird dem Kanal 3 und ein zweites Gamepad dem Kanal 4 zugewiesen.

Wenn du einen USB-Hub verwendest, ist es besser, einen Hub ohne eigene Stromversorgung zu verwenden (d. h. einen, der vom Raspberry Pi Pico mit Strom versorgt wird). Das liegt daran, dass der USB-Protokollstack den Hub nicht zurücksetzen kann und es zu Verwirrung kommen kann, wenn der Pico aus- und wieder eingeschaltet wird, ohne dass dies auch für den Hub geschieht. 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 unbedingt nötig ist. Wenn du nur ein Gerät (z. B. eine Tastatur) anschließen willst, kannst du das Gerät einfach (mit einem Adapterkabel) 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 sind keine weiteren Schritte erforderlich.

USB-Maus

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

USB-Gamepad

Ein oder mehrere PS3- oder PS4-Controller oder Gamepads wie zum Beispiel 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 das zweite Gamepad dem Kanal 4 zugewiesen. Innerhalb eines Programms können die Daten vom Gamepad mit der Funktion DEVICE(GAMEPAD) gelesen werden.

Konfigurieren der Tastatur

Standardmäßig wird die Tastaturkonfiguration als Standard-US-Layout angenommen. Mit dem Befehl OPTION KEYBOARD kannst du aber auch Layouts für andere Länder einrichten.

Die Syntax des Befehls lautet:

```
OPTION KEYBOARD Sprache
```

Dabei ist „language" ein zweistelliger Code, z. B. US für die Standardtastatur, die in den USA, Australien und Neuseeland verwendet wird. Andere Tastaturlayouts sind Großbritannien (UK), Frankreich (FR), Deutschland (GR), Belgien (BE), Italien (IT), Brasilien (BR) oder Spanien (ES).

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

Verwendung einer Maus

Die Maus ist besonders nützlich im MMBasic-Programmeditor, wo sie viele der Funktionen von GUI-Editoren wie Notepad in Windows nachbilden kann (siehe den Abschnitt *„Vollbild-Editor"* weiter oben in diesem Handbuch). Dazu gehören das Positionieren der Einfügemarke sowie das Kopieren und Einfügen mithilfe der Zwischenablage.

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

Beachte, dass die Maus immer dem 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 mx, my ausgeben
  tx = mx : ty = my
Schleife
```

Programm- und Datenspeicherung

Das BASIC-Programm wird im Flash-Speicher gehalten 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 bei einem Stromausfall oder einem Neustart des Prozessors nicht verloren geht.

Neben diesem Programmspeicher gibt es drei weitere Speicherorte, an denen Programme gespeichert werden können. Diese werden im Folgenden unter „ “ (Speicherorte) ausführlich beschrieben: Flash-Slots, das Flash-Dateisystem und eine angeschlossene SD-Karte.

Flash-Steckplätze

Es gibt drei davon, die zum Speichern völlig unterschiedlicher Programme oder früherer Versionen des Programms, an dem Sie gerade arbeiten, verwendet werden können (für den Fall, dass Sie zu einer früheren Version zurückkehren müssen). Außerdem kannst du mit MMBasic ein BASIC-Programm laden und ein anderes Programm ausführen, das an einem nummerierten Flash-Speicherort gespeichert ist, wobei alle Variablen und Einstellungen des ursprünglichen Programms beibehalten werden – das nennt man Verkettung und ermöglicht die Ausführung eines viel größeren Programms, als es die Programmspeicherkapazität normalerweise zulassen würde.

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

FLASH SAVE *n*	Speichert das Programm im Programmspeicher an der Flash-Speicherstelle *n*.
FLASH LOAD *n*	Lade ein Programm aus dem Flash-Speicherplatz *n* in den Programmspeicher.
FLASH RUN *n*	Führ ein Programm vom Flash-Speicherplatz *n* aus, löscht alle Variablen, aber nicht das Programm im Hauptprogrammspeicher.
FLASH LIST	Zeigt eine Liste aller Flash-Speicherplätze einschließlich der ersten Zeile des Programms an.
FLASH LIST n [,ALL]	Zeigt das Programm an Speicherplatz n an. Mit FLASH LIST n,ALL kannst du die Liste ohne Seitenumbrüche anzeigen.
FLASH ERASE *n*	Löscht den Flash-Speicherplatz *n*.
FLASH ALLE LÖSCHEN	Löscht alle Flash-Speicherplätze.
FLASH CHAIN *n*	Starte das Programm an Speicherplatz *n* und lass alle Variablen so, wie sie sind. So kannst du echt große Programme ausführen, die in zwei oder drei Teile aufgeteilt sind. Das Programm im Hauptspeicher wird dabei nicht gelöscht oder geändert.
FLASH ÜBERSCHREIBEN *n*	Löscht den Flash-Speicherplatz *n* und speichert 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 ist.

Außerdem kann mit dem Befehl OPTION AUTORUN ein Flash-Programmspeicherplatz angegeben werden, der beim Einschalten oder Neustart der CPU ausgeführt werden soll. Diese Option kann auch ohne Angabe eines Flash-Speicherplatzes verwendet werden. In diesem Fall führt MMBasic das Programm im Programmspeicher automatisch aus.

Hinweise:

- 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-Speicher gespeicherten BASIC-Programme können gelöscht werden, wenn du die PicoMite-Firmware aktualisierst (oder downgradest). Mach also vorher ein Backup davon.
- Der Befehl LIBRARY nutzt Slot 3 zum Speichern von Bibliotheksdaten, sodass bei Verwendung der Bibliotheksfunktion nur 2 Slots verfügbar sind.

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

Um zum Beispiel ein Programm auszuführen:

```
RUN „A:/MyProgram.bas”
```

Öffnen einer Textdatei für den wahlfreien Zugriff:

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

Dieses Laufwerk wird automatisch erstellt, wenn die PicoMite-Firmware geladen wird, sodass es für das BASIC-Programm immer verfügbar ist. 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-Speicher sind es 200 bis 500 KB, auf einem Raspberry Pi Pico 2 mit 4 MB Flash-Speicher etwas mehr als 2 MB und auf einem Modul mit 16 MB kann das Flash-Dateisystem bis zu 14 MB groß sein.

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

SD-Karten

Ein SD-Kartensteckplatz kann an den Raspberry Pi Pico angeschlossen und als Laufwerk B: aufgerufen werden. Wie beim Flash-Dateisystem können die normalen BASIC-Dateibefehle zum Speichern/Laden von Programmen sowie zum Öffnen von Datendateien zum Lesen/Schreiben verwendet werden.

Es werden Karten mit bis zu 32 GB unterstützt, die mit FAT16 oder FAT32 formatiert sind. Die erstellten Dateien können auch auf PCs mit Windows, Linux oder Mac-Betriebssystem gelesen/geschrieben werden. Die PicoMite-Firmware nutzt das SPI-Protokoll, um mit der Karte zu kommunizieren. Das ist unabhängig vom Kartentyp, 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 geschehen – entweder über den „System”-SPI-Port oder durch direkte Angabe der zu verwendenden I/O-Pins:

System-SPI-Port

Dieser Port wird für Systemzwecke verwendet (SD-Karte, LCD-Display und Touch-Controller auf einem LCD-Panel). Es gibt eine Reihe von Ports und Pins, die verwendet werden können (siehe Kapitel *„PicoMite-Hardware*”), aber in diesem Beispiel wird SPI auf den Pins GP18, GP19 und GP16 für Clock, MOSI und MISO verwendet.

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

Dann musst du MMBasic sagen, dass eine SD-Karte eingesteckt ist und welcher Pin für das Chip-Select-Signal (CS) benutzt wird:

```
OPTION SDCARD GP22
```

Spezielle I/O-Pins

Wenn keine anderen Geräte den SPI-Bus mit der SD-Karte teilen, kann man sie auch so einrichten:

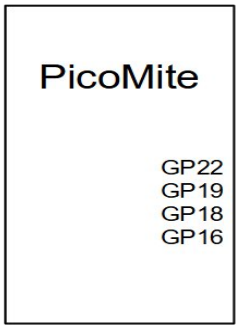
```
OPTION SDCARD  CS_pin, CLK_pin, MOSI_pin, MISO_pin
```

In diesem Fall können die Pins ganz flexibel zugewiesen werden und müssen nicht SPI-fähig sein, aber die Leistung der SD-Karte ist besser, wenn gültige SPI-Pins ausgewählt werden.

Diese Befehle müssen an der Eingabeaufforderung (nicht in einem Programm) eingegeben werden und führen zu einem Neustart der PicoMite-Firmware. Dies hat den Nebeneffekt, dass die USB-Konsolenschnittstelle getrennt wird und erneut 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 mehr zur Verfügung (d. h. er ist reserviert). Um die Konfiguration zu überprüfen, kannst du den Befehl OPTION LIST verwenden, um alle eingestellten Optionen einschließlich der Konfiguration der SD-Karte aufzulisten.

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

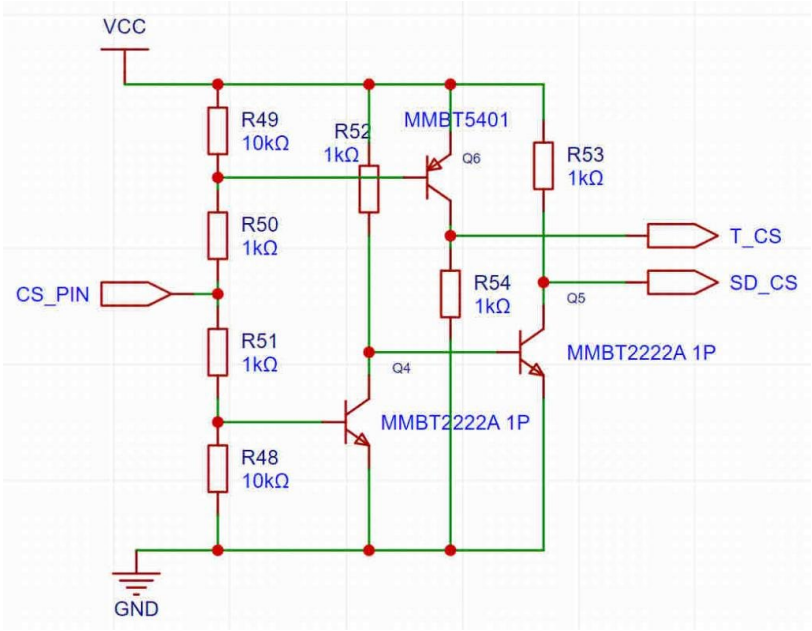


Beachte, dass du viele verschiedene Konfigurationen mit unterschiedlichen Pin-Zuweisungen verwenden kannst – dies ist nur ein Beispiel, das auf den oben aufgeführten Konfigurationsbefehlen basiert.

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 das nicht gemacht wird, können alle schwebenden Chip-Select-Signalleitungen dazu führen, dass der falsche Controller auf Befehle auf dem SPI-Bus reagiert.

Kombinierte Chip-Auswahl

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 das angegeben wird, ist die folgende Schaltung nötig, um den SD-Chip-Select zu machen:



Die Firmware nutzt den Touch-Pin wie folgt:

TOUCH_CS niedrig: TOUCH_CS niedrig, SD_CS hoch

TOUCH CS hoch: SD_CS niedrig: TOUCH_CS hoch

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

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

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

Beachte dabei Folgendes:

- Groß-/Kleinschreibung und Leerzeichen sind erlaubt. Das Dateisystem auf der SD-Karte unterscheidet **NICHT** zwischen Groß- und Kleinschreibung, das Flash-Dateisystem hingegen **schon** (dies ist der einzige Unterschied zwischen den beiden).
- Neben dem alten 8.3-Format werden auch lange Datei-/Verzeichnisnamen unterstützt.
- Die maximale Datei-/Pfadlänge beträgt 63 Zeichen (127 Zeichen für RP2350-Versionen).
- Jeder Dateipfad, der den Laufwerksbuchstaben verwendet, muss ein vollständiger Pfad vom Stammverzeichnis sein (z. B. „A:/mypath/myfile.txt“).
- Verzeichnispfade sind in Datei-/Verzeichniszeichenfolgen erlaubt. (z. B. OPEN „A:\dir1\dir2\file.txt“ FOR …).
- In Pfaden können sowohl Vorwärts- als auch Rückwärtsschrägstriche verwendet werden. Beispielsweise entspricht \dir\file.txt dem Pfad /dir/file.txt.
- Beim Start ist das aktive Laufwerk (also das Standardlaufwerk) A: (das Flash-Dateisystem).
- Die aktuelle PicoMite-Firmware-Zeit wird für die Erstellungs- und letzte Zugriffszeit von Dateien verwendet.
- Es können bis zu zehn Dateien gleichzeitig geöffnet sein, gemischt zwischen den Laufwerken A: und B:.
- Außer bei INPUT, LINE INPUT und PRINT ist das # in #fnbr optional und kann weggelassen werden.

Mit diesen Befehlen kannst du Programme aus dem Flash-Dateisystem und von SD-Karten laden oder dort speichern.

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

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

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

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

□ Öffne fname$ im Modus 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.
Schreibt Text in die als #fnbr geöffnete Datei.

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

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

□ FLUSH #fnbr
Sorgt dafür, dass alle zwischengespeicherten Schreibvorgänge ins Flash-Dateisystem oder auf die SD-Karte geschrieben werden. Es wird empfohlen, diesen Befehl regelmäßig zu verwenden, wenn bei einem Stromausfall Daten verloren gehen könnten.

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

Grundlegende Datei- und Verzeichnisbearbeitung. Die meisten Befehle können über die Befehlszeile oder aus einem BASIC-Programm heraus ausgeführt werden.

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

- DRIVE „A:/FORMAT“
  Formatiert das Flash-Dateisystem (Laufwerk A:) wieder auf seinen ursprünglichen Zustand.

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

- LIST fname$
  Zeigt den Inhalt eines Programms oder einer Textdatei auf der Konsole an.

- KILL fname$
  Löscht eine Datei im aktuellen Verzeichnis auf dem aktuellen Laufwerk.
  Weitere Infos 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 „..“ (Punkt Punkt) für das übergeordnete Verzeichnis oder „\“ für das Stammverzeichnis sein. Der Startpunkt ist das aktuelle Verzeichnis auf dem aktuellen Laufwerk.

- RMDIR dir$
  Löscht das Verzeichnis „dir$” im aktuellen Verzeichnis auf dem aktuellen Laufwerk.

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

- RENAME fromname$ AS toname$
  Benennt die Datei fromname$ um, sodass sie im aktuellen Verzeichnis auf dem aktuellen Laufwerk den Namen toname$ hat.

- COPY [Modus] fromname$ TO toname$
  Kopiert die Datei fromname$, sodass sie den Namen toname$ ~~erhält~~.
  Weitere Infos zu den optionalen Kopiermodi und Platzhaltern findest du in der Befehlsreferenz.

Es gibt auch eine Reihe von Funktionen, die die oben genannten Befehle unterstützen.

- INPUT$(nbr, #fnbr)
  Gibt eine Zeichenfolge 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 die vorhandenen Zeichen zurück (einschließlich einer leeren Zeichenfolge, 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 das 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, also „A:“ oder „B:“.
- MM.INFO(free space)
  Zeigt an, wie viel Speicherplatz auf dem aktiven Laufwerk noch verfügbar ist.
- MM.INFO(disk size)
  Zeigt die Größe des aktiven Laufwerks an
- MM.INFO(Datei fname$ vorhanden)
  Zeigt „true“ an, wenn die Datei da ist
- MM.INFO(exists dir dirname$)
  Gibt „true“ zurück, wenn das Verzeichnis da ist
- MM.INFO(Pfad)
  Gibt den Dateipfad des ausgeführten Programms zurück. Damit kann der Benutzer 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 oder aus einer Datei auf dem Flash-Dateisystem oder der SD-Karte kopieren. Die Syntax lautet:

```
XMODEM SEND Dateiname$
```

oder

```
XMODEM RECEIVE Dateiname$
```

Wobei „filename$“ die Datei ist, die gespeichert oder gesendet werden soll. „filename$“ kann ein String-Ausdruck, eine Variable oder eine Konstante sein. Wenn es sich um eine Konstante handelt, muss der String in Anführungszeichen gesetzt werden (z. B. XMODEM SEND „prbas.bas“). Beim Empfang einer Datei wird jede Datei mit dem gleichen 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. Das kann genutzt werden, um ein Logo zu zeichnen oder einen Hintergrund auf dem Display hinzuzufügen.

Die Syntax lautet:

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

Dabei ist „Dateiname$” 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 Displays gesetzt, wenn sie nicht angegeben werden).

Das Bild muss im richtigen Format sein (BMP, JPG oder PNG) und MMBasic fügt die Dateiendung zum Dateinamen hinzu, wenn sie nicht angegeben ist. Alle Bildtypen werden unterstützt, einschließlich Schwarzweiß- und Echtfarben-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 Dateiname$ [,StartX, StartY, Breite, Höhe]
```

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

Beispiel für sequentielle E/A

Im folgenden Beispiel wird eine Datei erstellt und zwei Zeilen in die Datei geschrieben (mit dem Befehl PRINT). Die Datei wird dann 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” und b$ den Text „jumps over the lazy dog” enthält.

Eine andere Möglichkeit, aus einer Datei zu lesen, ist die Funktion INPUT$(). Damit wird eine bestimmte Anzahl von Zeichen gelesen. Zum Beispiel:

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

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

Dateien haben normalerweise nur Text und der Befehl „print” macht aus Zahlen Text. Im folgenden Beispiel steht also in der ersten Zeile „123” und in der zweiten „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 dann aber den Text 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 hat 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 einen Datensatz in der Datei zu suchen, benutzt du den Befehl SEEK, der den Lese-/Schreibzeiger auf ein bestimmtes Byte setzt. Das erste Byte in einer Datei ist mit eins nummeriert, sodass zum Beispiel der fünfte Datensatz in einer Datei, die 64-Byte-Datensätze verwendet, bei Byte 257 anfängt. In diesem Fall würdest du Folgendes verwenden, um darauf zu zeigen:

```
SEEK #1, 257
```

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

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

Beim Schreiben in die Datei sollte eine feste Datensatzgröße verwendet werden. Dies lässt sich leicht erreichen, indem den zu schreibenden Daten ausreichend Füllzeichen (normalerweise Leerzeichen) hinzugefügt werden. 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 Bytes). Das Semikolon am Ende des Druckbefehls verhindert, dass die Zeichen für Wagenrücklauf und Zeilenvorschub hinzugefügt werden, die den Datensatz länger als beabsichtigt machen würden.

Zwei weitere Funktionen können bei der Verwendung des zufälligen Dateizugriffs 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 zeigt den zufälligen Dateizugriff. Mit ihm kannst du an die Datei anhängen (um zunächst einige Daten hinzuzufügen) und dann Datensätze mit zufälligen Datensatznummern lesen/schreiben. Der erste Datensatz in der Datei ist Datensatznummer 1, der zweite ist 2 usw.

```
RecLen = 64
OPEN "test.dat" FOR RANDOM AS #1

DO
 abort: PRINT
 PRINT "Anzahl der Datensätze in der Datei =" LOF(#1)/RecLen
 INPUT "Befehl (r = lesen, w = schreiben, a = anhängen, q = beenden): ", cmd$
 WENN cmd$ = "q" DANN SCHLIESSEN #1 : ENDE
 WENN cmd$ = "a" DANN
    SUCHEN #1, LOF(#1) + 1
 SONST
    INPUT "Datensatznummer: ", nbr
    WENN nbr < 1 oder nbr > LOF(#1)/RecLen DANN DRUCKE „Ungültiger Datensatz” :
GOTO Abbruch
    SUCHEN #1, RecLen * (nbr - 1) + 1
 ENDIF
 WENN cmd$ = "r" DANN
    DRUCKE „Der Datensatz = “ INPUT$(RecLen, #1)
 SONST
    LINE INPUT „Gib die Daten ein, die geschrieben werden sollen: ", dat$
    DRUCKE #1,dat$ + SPACE$(RecLen - LEN(dat$));
 ENDIF
LOOP
```

Der Direktzugriff kann auch bei einer normalen Textdatei genutzt 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
#1 schließen
```

Tonausgabe

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

Beachte, dass der Schaltregler auf dem Raspberry Pi Pico zu Störgeräuschen bei der Audioausgabe führen kann. Diese lassen sich reduzieren, indem man den Regler deaktiviert und das Modul über einen externen linearen Regler mit Strom versorgt.

Pulsweitenmoduliertes (PWM) Signal

Der Ton wird über PWM-Ausgänge erzeugt. Bevor die PLAY-Befehle verwendet werden können, 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 muss 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:

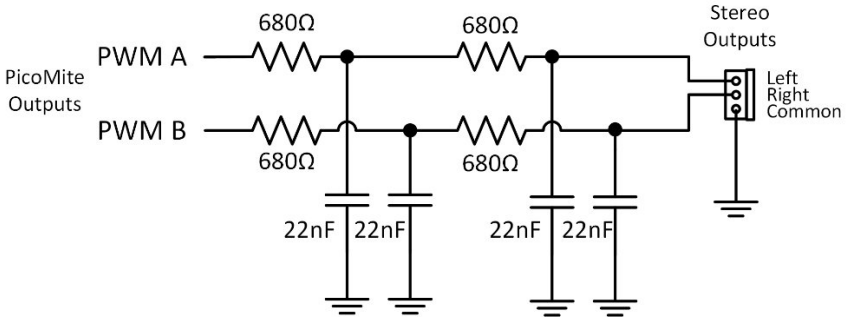
```
OPTION AUDIO GP0, GP1
```

Das Audiosignal wird als pulsweitenmoduliertes (PWM) Signal einer 44-kHz-Rechteckwelle (der sogenannten Trägerwelle) überlagert. Das heißt, dass ein Tiefpassfilter nötig ist, um die Trägerwelle zu entfernen und das Audiosignal wiederherzustellen.

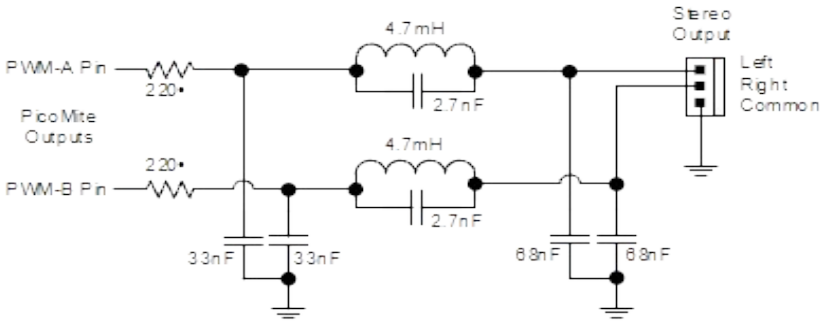
Filterschaltungen

Die meisten preisgünstigen Verstärkerlautsprecher (für einen PC) reagieren nicht auf die Trägerfrequenz, sodass sie selbst als Tiefpassfilter wirken. Wenn du es also einfach halten möchtest, kannst du den PWM-Ausgang direkt an den Eingang eines Verstärkerlautsprechers anschließen, um eine angemessene Tonausgabe zu erzielen.

Allerdings kann die hohe Frequenz der 44-kHz-Trägerwelle Probleme für den Verstärker verursachen (z. B. Überhitzung oder Verzerrung), daher ist der folgende Filter empfehlenswert. Dieser filtert den größten Teil der Trägerwelle heraus und liefert etwa 2 V Spitze-Spitze (0,6 V RMS) mit einer angemessenen Klangtreue bis zu 8 kHz (mehr als genug für die meisten verstärkten Lautsprecher):



Unten ist eine bessere Schaltung, die hochwertigen Klang mit nur noch einer unbedeutenden Menge an Träger liefert. Die ist für eine anspruchsvollere HiFi-Verstärker-/Lautsprecherkonfiguration geeignet. Der Ausgang ist gut für 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 (nicht direkt einen Kopfhörer oder Lautsprecher anzusteuern) und basieren auf einer Kondensatorkopplung in den nachfolgenden Verstärker (die meisten haben das).

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. Außerdem werden zusätzliche Wiedergabefunktionen unterstützt.

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

```
PLAY MP3 file$, interrupt
PLAY MIDIFILE file$, interrupt
MIDI abspielen
PLAY MIDI CMD cmd%, data1% [,data2%]
PLAY NOTE ON Kanal, Note, Anschlagstärke
NOTE OFF spielen Kanal, Note [, Anschlagstärke]
PLAY HALT
PLAY CONTINUE track$
PLAY STREAM buffer%(), readpointer%, writepointer%
```

Diese Befehle werden im Abschnitt „Befehlsliste” genauer erklärt.

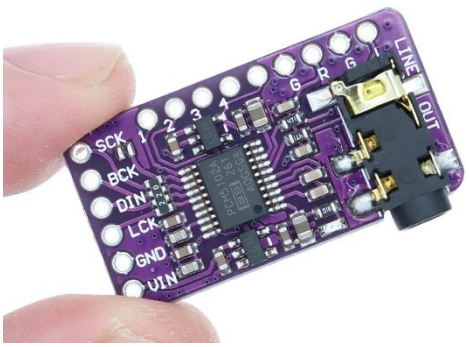
MCP48n2 DAC-

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 braucht keinen komplizierten Tiefpassfilter, und ein 120-Ω-Widerstand, der mit dem DAC-Ausgang verbunden ist, wobei das andere Ende des Widerstands über einen 100-nF-Kondensator mit GND verbunden ist, reicht aus. Wenn ein MCP4822 verwendet wird, 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 Erstellung eines eigenen Master-Clocks unterstützen, da dieser nicht von der Firmware erstellt wird. Der I2S-DAC auf dem Pico2 (RP2350A/B) nutzt PIO2, um den Ausgang zu erzeugen, 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 folgenden Befehl konfiguriert:

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

Der I2S-Worttakt (LRCK) liegt dann am nächsten Pin zum BCLK an. Wenn zum Beispiel BCLK auf GP0 gesetzt ist, liegt LRCK an GP1 an. Beide Pins und der DIN-Pin müssen unbenutzt sein, wenn der Befehl gegeben wird. Normalerweise muss beim DAC-Modul auch ein GND- und ein Stromversorgungs-Pin angeschlossen werden (normalerweise 5 V).

Der I2S-DAC erzeugt Audio in CD-Qualität aus FLAC-Dateien und gibt MP3-Dateien aus, die nur durch die inhärente MP3-Komprimierung begrenzt sind. FLAC-Dateien mit bis zu 96000 Hz und 24 Bit wurden getestet.

Wiedergabe von WAV-, FLAC-, MP3- und MOD- -Dateien

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

Die Befehle lauten:

```
WAV-Datei abspielen, Unterbrechung
PLAY FLAC Datei$, Unterbrechung
PLAY MODFILE file$, Unterbrechung
MP3-Datei $ abspielen, Unterbrechung „Nur RP2350
```

„file$" ist der Name der Audiodatei, die abgespielt werden soll. Sie muss sich im Flash-Dateisystem oder auf der SD-Karte befinden, und die entsprechende Erweiterung (z. B. .WAV) wird angehängt, wenn sie fehlt. Die Audiodatei wird im Hintergrund abgespielt (d. h., das Programm läuft ohne Unterbrechung weiter). „interrupt" ist optional und ist der Name einer Subroutine, die aufgerufen wird, wenn die Datei fertig abgespielt ist.

Sinuswellen erzeugen

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, um Aufmerksamkeit erregende Töne zu erzeugen, aber weil die Frequenz sehr genau ist, kann sie auch 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 links, rechts, Dauer, Unterbrechung
```

„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 „duration" gibt an, wie viele Millisekunden der Ton erklingen soll. „duration" ist optional. Wenn es nicht angegeben wird, läuft der Ton so lange, bis er explizit gestoppt wird oder das Programm beendet wird. „interrupt" (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 Befehl PLAY TONE geändert werden.

Verwendung von PLAY

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

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

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

Man könnte erwarten, dass der 500-Hz-Ton 2 Sekunden lang zu hören ist, aber in der Praxis wird überhaupt kein Ton zu hören sein. Das liegt daran, dass MMBasic den Befehl PLAY TONE ausführt (der die Tonwiedergabe im Hintergrund startet) und dann sofort den Befehl END ausführt, der das Programm und den Hintergrundton beendet. Dies geschieht so schnell, dass nichts zu hören ist.

Das folgende Programm funktioniert auch nicht:

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

Das liegt daran, dass der erste Befehl einen Ton mit 500 Hz einstellt, der zweite PLAY-Befehl diesen aber sofort durch einen Ton mit 300 Hz ersetzt, woraufhin das Programm zu Ende läuft und beendet wird (und der Hintergrundton ebenfalls), sodass nichts zu hören ist.

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

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

Das gilt für alle Versionen des Befehls PLAY, einschließlich PLAY WAV.

Dienstprogramme

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

`PLAY PAUSE`	Hält die aktuell abgespielte Datei oder den Ton vorübergehend an (Pause).
`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.
`VORHERIGE WIEDERGABE`	Spiel die vorherige WAV-, MP3- oder FLAC-Datei in einem Verzeichnis ab.
`STOPP`	Beende die Wiedergabe der Datei oder des Tons, genau wie wenn das Programm beendet wird.
`LAUTSTÄRKE L, R`	Stell die Lautstärke zwischen 0 und 100 ein, wobei 100 die maximale Lautstärke ist. Die Lautstärke wird auf den maximalen Pegel zurückgesetzt, wenn ein Programm ausgeführt wird. Es wird eine logarithmische Skala verwendet, sodass PLAY VOLUME 50,50 halb so laut wie 100,100 klingen sollte.

Spezielle Audioausgabe

Der Befehl PLAY SOUND erzeugt eine Ausgabe, die 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 vom BASIC-Programm aus gesteuert werden können, wobei 3 davon einen Hochgeschwindigkeits-ADC (Analog-Digital-Wandler) unterstützen.

Ein I/O-Pin wird anhand seiner Pin-Nummer bezeichnet, die entweder eine Zahl (z. B. 2) oder eine GP-Nummer (z. B. GP1) sein kann.

Digitale Eingänge

Ein digitaler Eingang ist die einfachste Art der Eingangskonfiguration. Wenn die Eingangsspannung höher als 2,5 V ist, ist der Logikpegel wahr (numerischer Wert 1), und alles unter 0,65 V ist falsch (numerischer Wert 0). Die Eingänge nutzen einen Schmitt-Trigger-Eingang, sodass alles zwischen diesen Pegeln den vorherigen Logikpegel beibehält.

Beachte, dass die maximale Spannung an den RP2040-I/O-Pins (d. h. dem Raspberry Pi Pico) 3,3 V beträgt. Wenn ein Gerät 5-V-Pegel für die Signalübertragung nutzt, ist eine Pegelumsetzung nötig. Der Raspberry Pi Pico 2 mit dem RP2350 kann 5 V vertragen (wenn er mit Strom versorgt wird), sodass in diesem Fall keine Pegelumsetzung für Signale bis zu 5 V nötig ist.

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

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

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

Der Befehl SETPIN erkennt auch ein paar Optionen, die einen internen Widerstand vom Eingang entweder mit der Versorgung oder mit Masse verbinden. Dies wird als „Pullup”- oder „Pulldown”-Widerstand bezeichnet und ist praktisch, wenn eine Verbindung zu einem Schalter hergestellt wird, da dadurch die Installation eines externen Widerstands zur Anlegung einer Spannung an die Kontakte entfällt. Wegen eines Hardwareproblems mit dem RP2350-Prozessor wird empfohlen, einen externen Widerstand von 8,2 K oder weniger zu verwenden, wenn ein Pulldown für diesen Prozessor nötig ist.

Analoge Eingänge

Pins, die als 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. Beispiel:

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

Wenn du Spannungen über 3,3 V messen willst, brauchst du einen Spannungsteiler. Bei kleinen Spannungen musst du vielleicht einen Verstärker verwenden, um die Eingangsspannung in einen für die Messung geeigneten Bereich zu bringen.

Die Messung nutzt die 3,3-V-Stromversorgung 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 beiden höchsten und die beiden niedrigsten Werte verworfen und die verbleibenden 6 Werte gemittelt.

Wenn du den direkten Messwert vom ADC haben willst, kannst du den Raw-Modus verwenden, indem du die Option ARAW zum Befehl SETPIN hinzufügst:

```
SETPIN pinno,ARAW
```

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

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

Zähleingänge

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

Als Beispiel gibt der folgende Befehl die Frequenz des Signals an 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. Dies ist die Zeitspanne, die MMBasic zum Zählen der Zyklen am Eingang benötigt. Das bedeutet, dass der Messwert einmal pro Sekunde mit einer Auflösung von 1 Hz aktualisiert wird. Durch Angabe eines dritten Arguments für den Befehl SETPIN kann eine alternative Gate-Zeit zwischen 10 ms und 100000 ms festgelegt werden. Kürzere Zeiten führen dazu, dass die Messwerte häufiger aktualisiert werden, aber der zurückgegebene Wert hat eine geringere Auflösung. Die Funktion PIN() skaliert die zurückgegebene Zahl immer als Frequenz in Hz, unabhängig von der verwendeten Gate-Zeit.

Im Folgenden wird beispielsweise die Gate-Zeit auf 10 ms eingestellt, was mit einem entsprechenden Verlust an Auflösung einhergeht:

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

Für genaue Messungen von Signalen unter 10 Hz ist es normalerweise besser, die Periode des Signals zu messen. Wenn dieser Modus eingestellt ist, misst die PicoMite-Firmware die Anzahl der Millisekunden zwischen aufeinanderfolgenden steigenden Flanken des Eingangssignals. Der Wert wird beim Übergang von niedrig nach hoch aktualisiert. Wenn dein Signal also eine Periode von (sagen wir) 100 Sekunden hat, solltest du bereit sein, diese Zeit abzuwarten, bevor die Funktion PIN() einen aktualisierten Wert zurückgibt.

Die Zählpins können auch die Anzahl der Impulse an ihrem Eingang zählen. Wenn ein Pin als Zähler konfiguriert ist (z. B. `SETPIN 7,CIN`), wird der Zähler auf Null zurückgesetzt und 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.

Die Zähleingänge sind bei der Standardfrequenz des Prozessors bis zu etwa 200 kHz genau. Zum Auslösen des Zählers ist eine minimale Impulsbreite von etwa 40 nS erforderlich. Der RP2350 bietet außerdem die Möglichkeit, GP1 als extrem schnellen Frequenzzähl-Pin zu konfigurieren (siehe Befehl SETPIN GP1, FFIN).

Digitale Ausgänge

Alle I/O-Pins können mit dem Parameter DOUT des Befehls SETPIN als digitaler Ausgang konfiguriert werden. Beispiel:

```
SETPIN GP15, DOUT
```

Das heißt, wenn ein Ausgangspin auf logisch niedrig gesetzt wird, zieht er seinen Ausgang auf Null, und wenn er auf hoch gesetzt wird, zieht er seinen Ausgang auf 3,3 V. In MMBasic machst du das mit dem Befehl PIN. Zum Beispiel setzt `PIN(GP15) = 0` den Pin GP15 auf niedrig, während `PIN(GP15) = 1` ihn auf hoch setzt.

Pulsweitenmodulation

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

Durch Variieren des Tastverhältnisses kannst du eine programmgesteuerte Ausgangsspannung erzeugen, die zur Steuerung externer Geräte verwendet werden kann, die einen analogen Eingang benötigen (Netzteile, Motorsteuerungen usw.). Die PWM-Ausgänge sind auch nützlich zum Ansteuern von Servos und zum Erzeugen einer Tonausgabe über einen kleinen Wandler.

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 Befehl SETPIN 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 Befehl SETPIN als PWM-Ausgänge konfiguriert werden.

Kommunikationsschnittstellen (seriell, SPI und I²C)

Diese sind 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-Bildschirme, Touchscreens usw. nutzen auch 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, um mit Ereignissen umzugehen, die zu einem unvorhersehbaren Zeitpunkt auftreten können. Ein Beispiel hierfür ist das Drücken einer Taste durch den Benutzer. In Ihrem Programm könnten Sie nach jeder Anweisung einen Code einfügen, um zu überprüfen, 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 normal weiter.

Jeder I/O-Pin, der als digitaler Eingang genutzt werden kann, kann mit dem Befehl SETPIN so konfiguriert werden, dass er einen Interrupt erzeugt, wobei bis zu zehn Interrupts gleichzeitig aktiv sein können. Interrupts können so eingerichtet werden, dass sie bei einem steigenden oder fallenden digitalen Eingangssignal (oder beidem) auftreten und einen sofortigen Sprung zu der angegebenen benutzerdefinierten Subroutine verursachen. 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 Unterroutine übergeben werden können, jedoch sind innerhalb des Interrupts Aufrufe anderer Unterroutinen 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 aber während INPUT-Anweisungen deaktiviert. Außerdem werden Interrupts während einiger langer hardwarebezogener Vorgänge (z. B. der Funktion TEMPR(), LCD-Zeichenbefehlen und SD-Zugriffsbefehlen) nicht erkannt, obwohl sie erkannt werden, wenn sie nach Beendigung des Vorgangs noch vorhanden sind. Bei der Verwendung von Interrupts wird 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 laufen, können sie schwer zu diagnostizierende Fehler verursachen. Beachte bei der Verwendung von Interrupts die folgenden Faktoren:

- Interrupts werden von MMBasic nur nach Abschluss jedes Befehls überprüft und nicht von der Hardware zwischengespeichert. Das bedeutet, dass ein Interrupt, der nur kurz dauert, 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, aber einige Befehle, wie z. B. die Funktion TEMPR(), können bis zu 200 ms dauern, sodass ein Interrupt innerhalb dieses Zeitfensters auftreten und wieder verschwinden kann und somit nicht erkannt wird.
- 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 Verarbeitungsvorgänge 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 Subroutinen oder Funktionen, die von ihr aufgerufen werden), 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 die Subroutine beendet hast).

- Denk daran, einen Interrupt zu deaktivieren, wenn du ihn nicht mehr brauchst – Hintergrund-Interrupts können seltsame und nicht intuitive Fehler verursachen.

Zusätzlich zu den Interrupts, die durch die Zustandsänderung eines E/A-Pins erzeugt werden, kann ein Interrupt auch durch andere Bereiche von MMBasic erzeugt werden, darunter Timer und Kommunikationsports, und die obigen Hinweise gelten auch für diese.

Die Liste aller dieser Interrupts (in der Reihenfolge ihrer Priorität von hoch nach niedrig) 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 Int Down
10. GUI Int Up
11. Sprite-Kollision
12. WebMite: TCP-Empfang
13. WebMite: MQTT-Abschluss
14. WebMite: UDP-Empfang
15. USB-Gamecontroller/USB- oder PS2-Maus/Wii-Controller
16. ADC-Abschluss
17. I2C-Slave-Empfang
18. I2C-Slave-Tx
19. I2C2-Slave-Empfang
20. I2C2-Slave-Tx
21. WAV fertig
22. COM1: Serielle Schnittstelle
23. COM2: Serieller Anschluss
24. IR-Empfang
25. Tastatur
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. Sobald die Interrupt-Subroutine fertig 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.

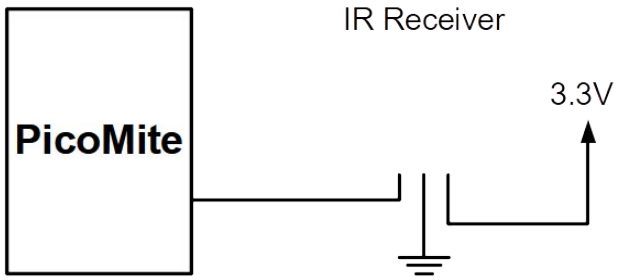
Das sind:

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

Infrarot-Fernbedienungsdecoder

Mit dem IR-Befehl kannst du ganz einfach eine Fernbedienung zu deinem Projekt hinzufügen. 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, auch mit solchen, die erweiterte Nachrichten generieren. Die meisten günstigen programmierbaren Fernbedienungen generieren eines dieser Protokolle, und mit einer davon kannst du deinem Pico-basierten Projekt eine raffinierte Note verleihen.



Das NEC-Protokoll wird auch von vielen anderen Herstellern wie Apple, Pioneer, Sanyo, Akai und Toshiba verwendet, sodass deren Markenfernbedienungen verwendet werden können.

Um das IR-Signal zu erkennen, brauchst du einen IR-Empfänger. NEC-Fernbedienungen verwenden eine 38-kHz-Modulation des IR-Signals. Geeignete Empfänger, die auf diese Frequenz abgestimmt sind, sind beispielsweise 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 kann 5 V vertragen.

Wichtiger Hinweis: Wegen eines Hardwarefehlers im RP2350 vor der Version A4 ist es echt wichtig, einen 4K7-Pullup-Widerstand zur Datenleitung des IR-Empfängers hinzuzufügen, wenn du den RP2350 benutzt.

Sony-Fernbedienungen nutzen eine 40-kHz-Modulation, aber Empfänger für diese Frequenz sind manchmal schwer zu finden. Normalerweise funktionieren auch 38-kHz-Empfänger, aber die maximale Empfindlichkeit wird mit einem 40-kHz-Empfänger erreicht.

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

```
SETPIN n, IR
```

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

Um den Decoder einzurichten, benutzt man 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 Schlüsselcode aktualisiert wird. *Interrupt* ist die Interrupt-Subroutine, die aufgerufen wird, wenn ein neuer Tastendruck erkannt wird. Die IR-Decodierung läuft im Hintergrund und das Programm läuft nach diesem Befehl ohne Unterbrechung weiter.

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

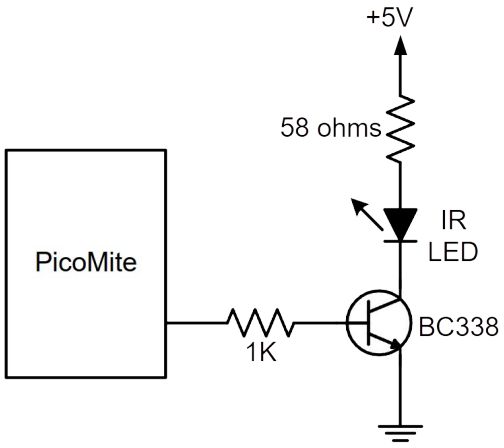
```
SETPIN GP6, IR                    ' Pin festlegen, der benutzt
werden soll
DIM INTEGER DevCode, KeyCode      ' vom Decoder verwendete Variablen
IR DevCode, KeyCode, IRInt        ' IR-Decoder starten
```

```
DO
  ' < Hauptteil des Programms >
LOOP

SUB IRInt                              ' Eine Taste wurde gedrückt
  PRINT „Empfangenes Gerät = “ DevCode „ Taste = “ KeyCode
END SUB
```

IR-Fernbedienungen können viele verschiedene Geräte (Videorekorder, Fernseher usw.) ansteuern, daher überprüft das Programm normalerweise zuerst den Gerätecode, um festzustellen, ob das Signal für das Programm bestimmt ist, und führt dann, falls dies der Fall ist, die entsprechende Aktion basierend auf der gedrückten Taste aus. Es gibt viele verschiedene Geräte und Tastencodes, daher ist die beste Methode, um festzustellen, welche Codes Ihre Fernbedienung generiert, die Verwendung des oben genannten Programms, um die Codes zu ermitteln.

Infrarot-Fernbedienungssender



Mit dem Befehl IR SEND kannst du ein 12-Bit-Infrarot-Fernbedienungssignal von Sony senden. Das 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 zeigt, was dafür nötig ist. Der Transistor wird zum Ansteuern der Infrarot-LED verwendet, weil die Ausgangsleistung des Raspberry Pi Pico begrenzt ist. Diese Schaltung liefert etwa 50 mA an die LED.

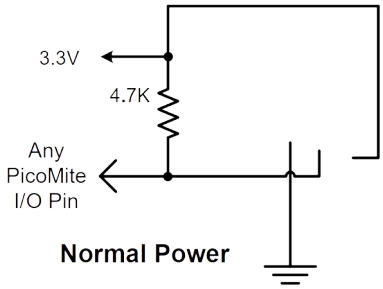
Um ein Signal zu senden, benutzt man den Befehl:

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

Dabei ist pin der verwendete I/O-Pin, dev der zu sendende Gerätecode und key der Schlüsselcode. Du kannst jeden beliebigen I/O-Pin des Raspberry Pi Pico verwenden und musst ihn nicht vorher einrichten (das macht IR SEND automatisch).

Die verwendete Modulationsfrequenz beträgt 38 kHz und entspricht den gängigen IR-Empfängern (siehe vorherige Seite), um eine maximale Empfindlichkeit bei der Kommunikation zwischen zwei Raspberry Pi Picos oder mit einem Micromite zu erreichen.

Temperatur messen



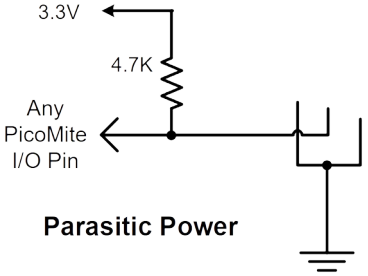
Die Funktion TEMPR() misst die Temperatur mit einem DS18B20-Temperatursensor. Dieses Gerät kann bei eBay für etwa 5 US-Dollar in verschiedenen Ausführungen gekauft werden, darunter auch eine wasserdichte Sonde.

Der DS18B20 kann separat mit einer 3,3-V-Stromversorgung betrieben werden oder, wie rechts gezeigt, mit der parasitären Stromversorgung des Raspberry Pi Pico. Es können mehrere Sensoren verwendet werden, aber für jeden Sensor ist ein separater I/O-Pin und ein 4,7-K-Pullup-Widerstand erforderlich.

Um die aktuelle Temperatur zu ermitteln, verwendest du einfach die Funktion TEMPR() in einem Ausdruck. 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 zurückgegebene Wert ist in Grad Celsius mit einer Auflösung von 0,25 ºC und hat eine Genauigkeit von ±0,5 ºC. Wenn während der Messung ein Fehler auftritt, ist der zurückgegebene Wert 1000.

Die gesamte Messung dauert 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 werden.

Wenn du das nicht willst, kannst du die Umwandlung 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.

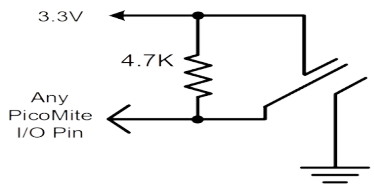
Beispiel:

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

Mit dem Befehl TEMPR START kannst du auch die Auflösung der Messung (von standardmäßig 0,25 ºC) und die dazugehörige Umwandlungszeit ändern.

Messung von Luftfeuchtigkeit und Temperatur

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



Der DHT22 muss mit 3,3 V (oder bis zu 5 V mit dem Raspberry Pi Pico 2) betrieben werden und sollte wie abgebildet über einen Pullup-Widerstand auf der Datenleitung verfügen. Dies ist für lange Kabelwege (bis zu 20 Meter) geeignet, bei kurzen Kabelwegen kann der Widerstand jedoch weggelassen werden, da die PicoMite-Firmware auch einen internen schwachen Pullup bereitstellt.

Um die Temperatur oder Luftfeuchtigkeit zu ermitteln, verwenden Sie 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 Auflösung 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 Gerätetimings. 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 anzeigen kann:

```
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 Befehl RTC GETTIME kannst du ganz einfach die aktuelle Uhrzeit 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, zeigen auch bei ausgeschaltetem Gerät die genaue Uhrzeit an und sind bei eBay für 2 bis 8 US-Dollar erhältlich. Komplette Module inklusive Batterie kann man bei eBay für etwas mehr Geld kaufen.

Der PCF8563 und der DS1307 halten die Zeit über einen Monat hinweg auf eine oder zwei Minuten genau, 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 pro Jahr) bietet.

Diese Chips sind I^2C-Geräte und sollten an die I^2C-I/O-Pins des Raspberry Pi Pico angeschlossen werden.

An den I^2C-I/O-Pins sind interne Pullup-Widerstände (100 kΩ) angebracht, sodass in den meisten Fällen keine externen Widerstände nötig sind.

Um die RTC zu aktivieren, musst du zuerst die zu verwendenden I^2C-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 Stunde im 24-Stunden-Format angegeben werden muss.

Mit dem folgenden Befehl stellst du die Echtzeituhr zum Beispiel auf 16 Uhr am 10. November 2025 ein:

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

Um die Uhrzeit abzurufen, benutzt man den Befehl RTC GETTIME, der die Uhrzeit vom Echtzeituhr-Chip liest 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 Einschalten eingestellt wird.

Mit dem Befehl OPTION RTC AUTO ENABLE kannst du auch eine automatische Aktualisierung der schreibgeschützten Variablen TIME$ und DATE$ aus dem Echtzeituhr-Chip beim Booten und jede Stunde einstellen.

Entfernungsmessung

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 sendet einen Ultraschallimpuls aus und misst die Zeit, 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, die laut Berichten gut funktionieren, verwenden den CS100-Chip – wie beispielsweise der HC-SR04 und US-025.

In der PicoMite-Firmware benutzt man 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 Eingang „trig" des Sensors verbunden ist, und echo der Pin, der mit dem Ausgang „echo" 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 Pegelverschiebung erforderlich, da er für seine Echoausgabe 5-V-Pegel verwendet. Der Raspberry Pi Pico 2 verträgt 5 V (bei eingeschaltetem Gerät), sodass in diesem Fall keine Pegelverschiebung erforderlich ist.

LCD-Anzeige

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

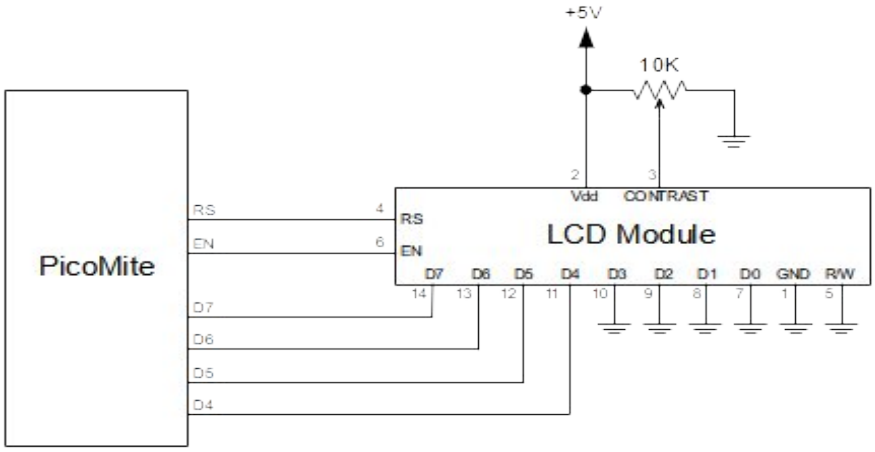
Dieser Befehl funktioniert mit LCD-Modulen, die den Controller-Chip KS0066, HD44780 oder SPLC780 verwenden und über 1, 2 oder 4 Zeilen verfügen. Typische Displays sind das LCD16X2 (futurlec.com), das Z7001 (altronics.com.au) und das QP5512 (jaycar.com.au). eBay ist eine weitere gute Quelle, wo die Preise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

Zum Einrichten des Displays benutzt man 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 auf dem LCD-Modul verbunden sind (die Eingänge D0 bis D3 und R/W auf dem Modul sollten mit Masse verbunden sein). „rs" ist der Pin, der mit dem Registerauswahleingang des Moduls verbunden ist (manchmal auch als CMD oder DAT bezeichnet). „en" ist der Pin, der mit dem Freigabe- oder Chipauswahleingang des Moduls verbunden ist.

Du kannst jeden beliebigen I/O-Pin auf dem Raspberry Pi Pico verwenden und musst ihn nicht vorher einrichten (das macht der LCD-Befehl automatisch für dich). Hier siehst du eine typische Konfiguration.



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

```
LCD Zeile, Position, Daten$
```

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

Hier ist ein typisches Beispiel, wo 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, „Temperature”
LCD 2, 6, STR$(TEMPR(GP15))    ' DS18B20 mit Pin GP15 verbunden
```

Beachte, dass dieses Beispiel auch die Funktion TEMPR() verwendet, um die Temperatur zu ermitteln (wie oben beschrieben).

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 der Tastendrücke erfolgt im Hintergrund. Wenn ein Tastendruck erkannt wird, wird ein Interrupt ausgelöst, den das Programm verarbeiten kann.

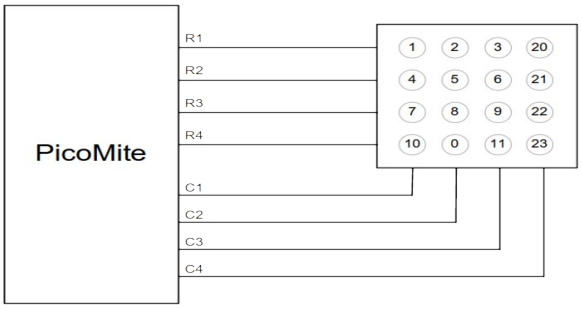
Beispiele für ein 4x3- und ein 4x4-Tastenfeld sind die Modelle Altronics S5381 und S5383 (siehe www.altronics.com).

Um die Tastaturfunktion zu aktivieren, benutzt man den Befehl:

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

aufgerufen, wenn ein neuer Tastendruck erkannt wird. „r1”, „r2”, „r3” und „r4” sind die Pin-Nummern für die vier Reihenverbindungen zum Tastenfeld (siehe Abbildung unten) und „c1”, „c2”, „c3” und „c4” sind die Spaltenverbindungen. „c4“ wird nur bei 4x4-Tastaturen verwendet und sollte weggelassen werden, wenn du eine 4x3-Tastatur benutzt.

Du kannst alle I/O-Pins auf dem Raspberry Pi Pico verwenden und musst sie nicht vorher einrichten, das macht der Befehl KEYPAD automatisch für dich.



Die Erkennung und Dekodierung von Tastendrücken läuft im Hintergrund ab, 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 innerhalb der Kreise in der Abbildung oben). 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                              ' Eine Taste wurde gedrückt
  PRINT „Tastendruck = “ KeyCode
END SUB
```

Schau dir auch den erweiterten Befehl KEYPAD in der Befehlsbeschreibung an, der beliebig viele Zeilen und Spalten mit vom Benutzer festgelegten Rückgabecodes erlaubt.

WS2812-Unterstützung

Die PicoMite-Firmware hat eine eingebaute Unterstützung für den mehrfarbigen LED-Chip WS2812. Dieser Chip braucht ein ganz bestimmtes Timing, um richtig zu funktionieren, und mit dem Befehl DEVICE WS2812 lassen sich diese Geräte ganz einfach und mit minimalem Aufwand steuern.

Dieser Befehl gibt die erforderlichen Signale aus, die zum Ansteuern einer Kette von WS2812-LED-Chips benötigt werden, 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, Anzahl%, Farben%[()]
```

Beachte, dass der Pin auf einen digitalen Ausgang eingestellt sein muss, bevor dieser Befehl verwendet werden kann. Das Array colours%() sollte mindestens so viele Elemente haben 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(rot), Rgb(grün), RGB(blau), RGB(gelb), rgb(cyan))
SETPIN GP5, DOUT
WS2812 O, GP5, 5, b%()
```

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

Es kann sein, dass ein WS2812 mit dem 3,3-V-Ausgang des Raspberry Pi Pico nicht richtig funktioniert. In diesem Fall gibt's ein paar Lösungen:

- Verwende den WS2812B, der mit einer 3,3-V-Versorgung und -Eingängen funktioniert.
- Nimm den Raspberry Pi Pico 2, der 5 V verträgt (wenn er eingeschaltet ist), sodass in diesem Fall keine Pegelverschiebung nötig 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 Mindestversorgung für den WS2812 beträgt 4 V, aber in vielen Fällen funktioniert er auch mit 3,3 V.

OV7670-Kameramodul

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

Anzeigepanels

NICHT VERFÜGBAR BEI HDMI- ODER VGA-VERSIONEN

Die PicoMite-Firmware unterstützt viele LCD-Anzeigetafeln, die eine SPI-, I^2C- oder parallele Schnittstelle verwenden.

Diese Befehle müssen an der Eingabeaufforderung (nicht in einem Programm) eingegeben werden und führen zu einem Neustart der PicoMite-Firmware. Das hat zur Folge, dass die USB-Konsolenschnittstelle getrennt wird und wieder angeschlossen 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 für die Signalübertragung verwenden, ist eine Pegelumsetzung erforderlich. Der Raspberry Pi Pico 2 verträgt 5 V (bei eingeschaltetem Gerät), sodass in diesem Fall keine Pegelumsetzung erforderlich ist.

System-SPI-Bus

Der System-SPI-Bus des Raspberry Pi Pico 2 ( ) ist ein dedizierter SPI-Kanal, der von vielen LCD-Bildschirmen, allen Touch-Controllern und auch für die Kommunikation mit einer SD-Karte genutzt wird. Wenn eines dieser Geräte angeschlossen ist, musst du zuerst die für den System-SPI-Bus verwendeten I/O-Pins definieren.

Dies geschieht mit dem folgenden Befehl:

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

Dieser Befehl muss an der Eingabeaufforderung eingegeben werden und führt dazu, dass die Firmware neu gestartet und die USB-Konsolenschnittstelle getrennt wird, die dann wieder angeschlossen 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 Anzeigefelder

Diese Anzeigetafeln werden mit den folgenden Befehlen konfiguriert. Für alle muss zuerst der System-SPI-Bus (siehe oben) definiert werden.

In allen Befehlen sind die Parameter:

- `ODER` Dies ist die Ausrichtung des Displays und kann LANDSCAPE, PORTRAIT, RLANDSCAPE oder RPORTRAIT sein. Diese können mit L, P, RL oder RP abgekürzt werden. Das Präfix R steht für die umgekehrte oder „auf den Kopf gestellte" Ausrichtung.
- `DC` Anzeigedaten-/Befehlssteuerungs-Pin.
- `RESET` Pin zum Zurücksetzen des Displays (wenn auf Low gezogen).
- `CS` Anzeige-Chipauswahl-Pin (aktiv niedrig).
- `BL` Optionaler Pin, der die Helligkeit der Hintergrundbeleuchtung über Pulsweitenmodulation (PWM) steuert.
- `INVERTIEREN` Diese Option sorgt dafür, dass die Farben umgekehrt werden, 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, die diesen Controller verwenden, 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. Unterstützt die Auflösungen 480 * 320 und 480 * 320 ( ).

OPTION LCDPANEL ILI9481IPS, OR, DC, RESET,CS [,BL][,INVERT]

Initialisiert ein IPS-Display mit dem ILI9481-Controller. Unterstützt eine Auflösung von 480 * 320.

OPTION LCDPANEL ILI9488, OR, DC, RESET, CS [,BL] [,INVERT]

Initialisiert ein TFT-Display mit dem ILI9488-Controller ( ). 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 sein.

OPTION LCDPANEL ILI9488P, OR, DC, RESET, CS [,BL] [,INVERT]

Initialisiert ein TFT-Display mit dem ILI9488-Controller. Dieser unterstützt eine Auflösung von 320 * 320.
Beachte, dass dieser Controller ein Problem mit dem LCD_SDO (MISO)-Pin hat, der den Touch-Controller stört. Damit dieses Display funktioniert, darf der LCD_SDO-Pin nicht direkt mit dem System-SPI-MISO verbunden sein. 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 3,5-Zoll-Display von Waveshare, wie es auf deren Pico-Eval-Board verwendet wird, sowie den normalen 3,5-Zoll-Display-Adapter.

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

Initialisiert ein LCD-Display mit dem Nokia 5110-Controller. Unterstützt eine Auflösung von 84 * 48. Ein zusätzlicher Parameter „contrast" kann angegeben werden, um den Kontrast „ " des Displays zu steuern. Probieren Sie Kontrastwerte zwischen &HA8 und &HD0 aus, um das Display anzupassen. Der Standardwert bei Auslassung ist &HB1.

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

Initialisiert ein OLED-Display mit dem SSD1306-Controller und einer SPI-Schnittstelle. Unterstützt eine Auflösung von 128 * 64. Ein zusätzlicher Parameter „offset" kann angegeben werden, um die Position des Displays zu steuern. 0,96-Zoll-Displays brauchen normalerweise den Wert 0. 1,3-Zoll-Displays brauchen normalerweise 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. 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. Unterstützt eine Auflösung von 160 × 80.

OPTION LCDPANEL ST7735S_W, OR, DC, RESET, CS [,BL][,INVERT]

Initialisiert ein Waveshare 128x128 ST7735S-Display. Unterstützt eine Auflösung von 128 * 128.

OPTION LCDPANEL ST7789, OR, DC, RESET, CS [,BL] [,INVERT]

Initialisiert ein IPS-Display mit dem 7789-Controller. 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. Das unterstützt eine Auflösung von 240 * 135.
HINWEIS: Display-Karten 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 Auflösung 320 * 240 von Waveshare (https://www.waveshare.com/wiki/Pico-ResTouch-LCD-2.8).
Diese sind in der Lage, transparenten Text anzuzeigen und funktionieren mit den Befehlen BLIT und BLIT READ.
HINWEIS: Display-Karten 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. Dieser unterstützt eine Auflösung von 480 * 320.
HINWEIS: Damit transparenter Text und Blit richtig funktionieren, sollte die Diode D1 auf der Rückseite des Displays überbrückt werden, und es wird empfohlen, J1 ebenfalls 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. Unterstützt eine Auflösung von 240 × 240.

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

Initialisiert ein LCD-Display mit dem ST7920-Controller. Unterstützt eine Auflösung von 128 × 64. Beachte, dass dieses Display keine Chipauswahl unterstützt, sodass der SPI-Bus bei Verwendung dieses Displays nicht gemeinsam genutzt werden kann.

I²C-basierte LCD-Panels

Alle I²C-basierten Display-Controller nutzen die System-I²C-Pins gemäß der Pinbelegung für das jeweilige Gerät. Andere I²C-Geräte können den Bus gemeinsam nutzen, sofern ihre Adressen eindeutig sind.

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

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

Wenn ein I²C-Display konfiguriert ist, musst du den I²C-Port nicht für ein zusätzliches Gerät „öffnen” (I2C OPEN), I2C CLOSE ist gesperrt und alle I²C-Geräte müssen für den 100-kHz-Betrieb geeignet sein. Die Geschwindigkeit des I²C-Busses wird durch Änderungen der CPU-Taktrate 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 sein. Diese können mit L, P, RL oder RP abgekürzt werden. Das Präfix R zeigt die umgekehrte oder „auf dem Kopf stehende” Ausrichtung an.

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

Initialisiert ein OLED-Display mit dem SSD1306-Controller ( ) mit einer 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 in der Regel den Wert 0. 1,3-Zoll-Displays benötigen in der Regel den Wert 2. Der Standardwert bei Auslassung ist 0.

Bitte beachten Sie, dass viele günstige I²C-Versionen von SSD1306-Displays aufgrund eines Verdrahtungsfehlers I²C nicht richtig implementieren. 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, OR
```

Initialisiert ein OLED-Display mit dem SSD1306-Controller mit einer I²C-Schnittstelle. Das unterstützt eine Auflösung von 128 * 32.

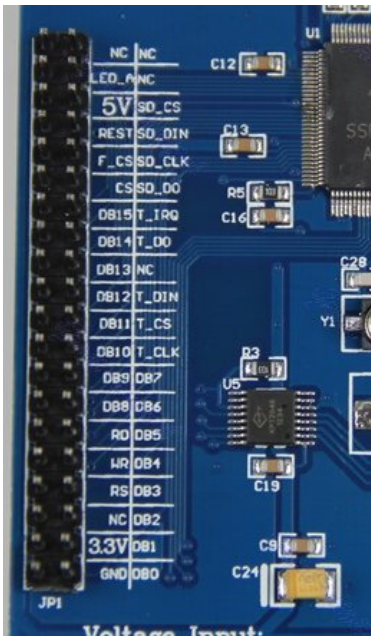
8-Bit-Parallel-LCD- -Panels

Neben den SPI- und I²C-basierten Controllern unterstützt die PicoMite-Firmware auch 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 haben einen SD-Kartensteckplatz, der von MMBasic voll 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 auch viel größer, haben mehr Pixel und sind heller. MMBasic kann einige von ihnen mit 24-Bit-True-Color für eine Vollfarbwiedergabe (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 einer parallelen 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 zum Glück haben sich die meisten Anbieter auf einen einzigen Stecker geeinigt, wie rechts abgebildet.

Es wird dringend empfohlen, ein Display mit einem passenden Stecker zu kaufen – das gibt dir die Gewissheit, dass der Hersteller den Standard befolgt hat, für den die PicoMite-Firmware entwickelt wurde.

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-Kartenfunktionen sind optional, aber wenn sie genutzt werden, verwenden sie den zweiten SPI-Port (SPI2).

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

8-Bit-Parallelanzeige	Beschreibung	Raspberry Pi Pico
DB0	Paralleler Datenbus Bit 0	Pin 1/GP0
DB1	Paralleler Datenbus Bit 1	Pin 2/GP1
DB2	Paralleler Datenbus Bit 2	Pin 4/GP2
DB3	Paralleler Datenbus Bit 3	Pin 5/GP3
DB4	Paralleler Datenbus Bit 4	Pin 6/GP4
DB5	Paralleler Datenbus Bit 5	Pin 7/GP5
DB6	Paralleler Datenbus Bit 6	Pin 9/GP6

DB7	Paralleler Datenbus Bit 7	Pin 10/GP7
CS	Chipauswahl (aktiv niedrig)	Masse (d. h. immer ausgewählt)
WR	Schreiben (aktiv niedrig)	Pin 19/GP14
RD	Lesen (aktiv niedrig)	Pin 20/GP15
DC	Befehl/Daten	Pin 17/GP13
RESET	SSD1963 zurücksetzen	Pin 21/GP16
LED_A	Hintergrundbeleuchtungssteuerung für ein nicht modifiziertes Display-Panel	Konfigurierbar, siehe OPTION LCDPANEL
5	5-V-Stromversorgung für die Hintergrundbeleuchtung bei einigen Displays (die meisten Displays nutzen dafür die 3,3-V-Versorgung).	
3,3	Stromversorgung.	
GND	Masse	

Die Pins DC, WR, RD und RESET können mit dem optionalen Parameter *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 festzulegen, die vom Display verwendet werden.

Die folgende Tabelle listet die Anschlüsse auf, die zur Unterstützung der Touch-Controller-Schnittstelle erforderlich sind:

8-Bit-Parallelanzeige	**Beschreibung**	**Raspberry Pi Pico**
T_CS	Touch-Chip-Auswahl	Empfohlener Pin 24/GP18
T_IRQ	Touch-Interrupt	Empfohlener Pin 25/GP19
T_DIN	Touch-Daten-Eingang (MOSI)	Empfohlener Pin 15/GP11
T_CLK	Touch-SPI-Takt	Empfohlener Pin 14/GP10
T_DO	Touch-Datenausgang (MISO)	Empfohlener Pin 16/GP12

Die folgende Tabelle zeigt die Anschlüsse, die du für den SD-Kartensteckverbinder brauchst:

8-Bit-Parallelanzeige	**Beschreibung**	**Raspberry Pi Pico**
SD_CS	SD-Karten-Chipauswahl	Empfohlener Pin 29/GP22
SD_DIN	SD-Karte Daten-Eingang (MOSI)	Empfohlener Pin 15/GP11
SD_CLK	SD-Karte SPI-Takt	Empfohlener Pin 14/GP10
SD_DO	SD-Karte Datenausgang (MISO)	Empfohlener Pin 16/GP12

Wenn ein Anschluss als „empfohlen” aufgeführt ist, ist das nur ein Vorschlag und je nach Hardware-Konfiguration können auch andere Pins verwendet werden. Trotzdem sollte der spezifische Pin im entsprechenden OPTION-Befehl angegeben werden (siehe unten).

Im Allgemeinen haben 7-Zoll- und größere Displays einen separaten Pin am Stecker (mit 5V gekennzeichnet) für die Stromversorgung der Hintergrundbeleuchtung aus einer 5-V-Quelle. Wenn dieser Pin nicht vorhanden ist, wird die Stromversorgung für die Hintergrundbeleuchtung über den 3,3-V-Pin erfolgen. Beachte, dass der Stromverbrauch der Hintergrundbeleuchtung beträchtlich sein kann. Beispielsweise verbraucht ein 7-Zoll-Display in der Regel 330 mA aus dem 5-V-Pin.

Wenn der 3,3-V-Ausgang des Pico zur Stromversorgung eines Panels und seiner Hintergrundbeleuchtung verwendet wird, kann der Strombedarf leicht höher sein, als der Pico liefern kann. Anzeichen für eine unzureichende Stromversorgung können Fehler bei der TOUCH-Kalibrierung oder beim SD-Zugriff sein. In diesem Fall sollte eine externe 3,3-V-Stromversorgung verwendet werden.

Der von der Hintergrundbeleuchtung verbrauchte Strom kann auch einen Spannungsabfall am Massepin des LCD-Displays verursachen, was wiederum die vom Display-Controller wahrgenommenen Logikpegel verschieben und zu verfälschten Farben oder Texten führen kann. Eine einfache Möglichkeit, diesen Effekt zu diagnostizieren, besteht darin, die CPU-Geschwindigkeit auf (z. B.) 48 MHz zu reduzieren. Wenn das Problem dadurch behoben wird, ist das ein starker Hinweis darauf, dass dies die Ursache ist. Eine Möglichkeit, das Problem zu beheben, besteht darin, die Strom- und Erdungskabel direkt an die Leiterplatte des LCD-Bildschirms anzulöten.

Bei Displaypanels, die den SPI-Port mit mehreren Geräten (SD-Karte, Touchscreen usw.) teilen, 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. Wenn das nicht gemacht wird, schwankt der Pin, was dazu führt, dass der falsche Controller auf Befehle auf dem SPI-Bus reagiert.

In der PicoMite-Firmware kann jeder SPI-Kanal für die Kommunikation mit dem Touch-Controller und der SD-Kartenschnittstelle verwendet werden, wie durch die Einstellung OPTION SYSTEM SPI definiert. Wenn diese Einstellung aktiviert ist, steht dieser SPI-Kanal für BASIC-Programme nicht zur Verfügung (die den anderen SPI-Kanal verwenden können).

Konfigurieren eines 8-Bit-Parallel-LCD-Panels

Um das Display zu nutzen, musst du MMBasic mit dem Befehl OPTION LCDPANEL einrichten, den du normalerweise über die Befehlszeile eingibst. Jedes Mal, wenn die PicoMite-Firmware neu gestartet wird, initialisiert MMBasic das Display automatisch.

Die Syntax lautet:

OPTION LCDPANEL Controller, Ausrichtung [,Backlightpin] [,DCpin] [,NORESET] [,INVERT] [,DB0pin]

Dabei kann „controller" entweder sein:

- SSD1963_4 Für ein 4,3-Zoll-Display
- SSD1963_5 Für ein 5-Zoll-Display
- SSD1963_5A Für eine andere Version des 5-Zoll-Displays, wenn SSD1963_5 nicht passt
- SSD1963_7 Für ein 7-Zoll-Display
- SSD1963_7A Für eine andere Version des 7-Zoll-Displays, falls SSD1963_7 nicht passt.
- 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 zeigt die umgekehrte oder „auf den Kopf gestellte" Ausrichtung an.

„DCpin" ist optional und steht für den Daten-/Befehlspin (früher als RS-Pin bezeichnet). Wenn dieser Parameter weggelassen wird, gilt die Pinbelegung wie oben in der Tabelle angegeben. Wenn er angegeben wird, werden die Pins DC, WR, RD und RESET sequenziell vom DC-Pin aus zugewiesen.

Der optionale Parameter „NORESET" kann zum Speichern eines Pins verwendet werden. In diesem Fall sollte der Reset-Pin auf High gesetzt werden.

Der optionale Parameter „INVERT" gibt an, 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. Danach initialisiert MMBasic das Display automatisch beim Start oder beim Zurücksetzen. 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). In

diesem Fall kann der GUI-Befehl RESET LCDPANEL verwendet werden, um das Display neu zu initialisieren.

Wenn das LCD-Panel nicht mehr gebraucht wird, kann der Befehl OPTION LCDPANEL DISABLE verwendet werden, der die I/O-Pins für die allgemeine Verwendung zurückgibt.

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

Um das Display zu testen, kannst du den Befehl GUI TEST LCDPANEL eingeben. Du solltest eine animierte Anzeige mit schnell übereinander gezeichneten Farbcirkeln sehen. 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 (erhältlich unter www.buydisplay.com) getestet. Diese müssen als TFT-LCD-Panel mit 8080-Schnittstelle, 800x480-Pixel-LCD, SSD1963-Display-Controller und XPT2046-Touch-Controller gekauft werden. Beachte, dass die EastRising-Panels eine nicht standardmäßige Pinbelegung des Schnittstellenanschlusses verwenden, sodass du beim Anschluss an den Raspberry Pi Pico die Datenblätter zu Rate ziehen musst. Ein passender Adapter zur Umwandlung in den standardmäßigen 40-Pin-Anschluss kann unter https://www.rictech.nz/micromite-products gekauft werden.

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 nur mit dem optionalen Parameter DB0pin im Konfigurationsbefehl ausgewählt werden. Um den 16-Bit-Betrieb zu aktivieren, füge „_16“ an den Controller an. Beispiel: SSD1963_4_16. Die Firmware unterstützt auch 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 andere Version des 5-Zoll-Displays, wenn SSD1963_5 nicht passt
- SSD1963_7_16 Für ein 7-Zoll-Display
- SSD1963_7A_16 Für eine andere Version des 7-Zoll-Displays, falls SSD1963_7 nicht passt.
- 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

RP2350 Erweiterte Display-Unterstützung

Der RP2350 hat 320 KB Speicher, den der MMBasic-Programmierer nutzen kann. Damit lassen sich verschiedene zusätzliche Funktionen mit gepufferten Displaytreibern umsetzen.

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

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 nutzen, musst du zuerst 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 SD-Karte benötigt wird (Hinweis: Bei den Waveshare-Boards führt die Verwendung dieses Treibers zu Konflikten mit der Verwendung von Touch und SD-Karte, da sie 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 kann.

Das Format für den Befehl zum Aktivieren eines SPI-gepufferten Treibers ist das gleiche 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 beansprucht die Aktivierung eines dieser Treiber einen 400x240 RGB332-Framebuffer aus dem Heap-Speicher. Wenn du diese Treiber benutzt, 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 das gleiche wie bei jedem SSD1963-Treiber

```
OPTION LCDPANEL Controller, Ausrichtung [,Backlightpin]  [,DCpin]
[,NORESET] [,INVERT] [,DB0pin]
```

Der Vorteil der gepufferten Treiber besteht darin, dass die Grafikbefehle in MMBasic einfach in einen Speicherpuffer geschrieben werden und die Aktualisierung der physischen Anzeige dann im Hintergrund auf dem zweiten Prozessor stattfindet. Dadurch kann das Basic-Programm die Verarbeitung fortsetzen, ohne auf die Anzeige warten zu müssen. Natürlich ändert die Verwendung eines gepufferten Treibers nichts Wesentliches am Gesamtdurchsatz zum physischen Display. Kontinuierliche Schreibvorgänge müssen also wie bei einem Standardtreiber auf das Display warten. Der große Vorteil kommt zum Tragen, wenn ein Programm eine rechenintensive Schleife hat, bei der Bildschirmschreibvorgänge die Verarbeitung verzögern. In diesem Fall wird der Overhead der Verarbeitungsschleife drastisch reduziert, solange die Gesamtbildschirmaktualisierungsrate überschaubar ist.

VGA222-Treiber

Der PICOMITE RP2350 und der PICOMITEUSB RP2350 können auch einen verbesserten VGA-Treiber nutzen, der im RGB222-Modus läuft und 64 Farben anzeigt. Das wird mit dem 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, ob hysyncpin und hysyncpin+1 (vsync) bluelpin - bluelpin+5 (blue_l, blue_h, green_l, green_h, red_l, red_h) verfügbar sind. Wenn das alles frei ist, wird die VGA222-Ausgabe auf den angegebenen Pins eingerichtet.

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

Hintergrundbeleuchtungssteuerung

Für die Displays 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 zur Steuerung der Helligkeit der Hintergrundbeleuchtung (LED_A) angibt. Dadurch wird ein PWM-Ausgang an diesem Pin 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-Verwendung gesperrt und darf nicht mit dem PWM-Kanal in Konflikt stehen, der möglicherweise für Audio eingerichtet ist.

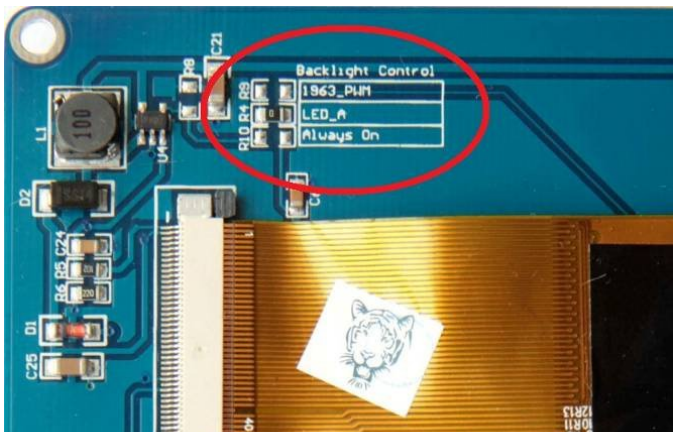
Zum Beispiel:

```
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 haben drei Paare von Lötpads auf der Leiterplatte, die unter der Überschrift „Backlight Control" (Hintergrundbeleuchtungssteuerung) zusammengefasst sind, wie rechts dargestellt.



Normalerweise ist das mit „LED-A" markierte Paar mit einem Null-Ohm-Widerstand kurzgeschlossen, wodurch die Helligkeit der Hintergrundbeleuchtung mit einem PWM-Signal (Pulsweitenmodulation) am LED-A-Pin des Hauptanschlusses des Displaypanels gesteuert werden kann.

Es ist aber besser, den SSD1963-Controller zu nutzen, um dieses Signal zu erzeugen, weil dadurch ein I/O-Pin frei wird und eine feinere Steuerung möglich ist.

Um die SSD1963-Steuerung zu nutzen, sollte der Null-Ohm-Widerstand vom mit „LED-A" gekennzeichneten Paar entfernt und zum Kurzschließen des nahe gelegenen Paares von Lötpads mit der Kennzeichnung „1963-PWM" verwendet werden. Die Helligkeit kann dann mit dem Befehl BACKLIGHT über den SSD1963-Controller 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, sodass viele der in einem Projekt verwendeten physischen Knöpfe und Schalter als durch Berührung aktivierte Bildschirmsteuerungen implementiert werden können. Als Alternative zum resistiven Touchscreen unterstützt MMBasic auch kapazitive Touchscreens auf Basis des FT6336-Controllers.

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

Bei Displaypanels, die I^2C oder eine parallele Schnittstelle nutzen, 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 am Anfang dieses Kapitels besprochen und im Kapitel *„Optionen*" dieses Handbuchs ausführlich beschrieben.

Wenn du zum Beispiel 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 zu nutzen, muss MMBasic mit dem Befehl OPTION TOUCH mitgeteilt werden, dass der Touch-Controller auf dem System-SPI-Bus verfügbar ist. Dadurch wird MMBasic mitgeteilt, welche Pins für die Chip-Select- und Interrupt-Signale verwendet werden.

Bei einem typischen ILI9341-Display wird damit zum Beispiel Chip Select auf den GP12-Pin und Interrupt auf GP11 gesetzt:

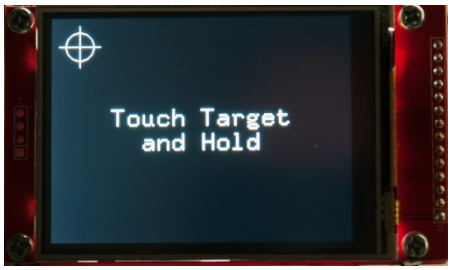
```
OPTION TOUCH GP12, GP11
```

Das musst du an der Eingabeaufforderung eingeben, damit die Firmware neu startet und die USB-Konsolenschnittstelle trennt, die dann 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 Display-Panels 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 mit dem Befehl CALIBRATE der GUI „ ” kalibriert werden: .

Dieser Befehl zeigt ein Ziel in der oberen linken Ecke des Bildschirms an (wie abgebildet). Drück 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 dann die Kalibrierung fort, indem es das Ziel nacheinander in den anderen drei Ecken des Bildschirms zur Berührung und Kalibrierung anzeigt.

Die Kalibrierungsroutine kann eine Warnung ausgeben, dass die Kalibrierung nicht genau war. Dies ist nur eine Warnung, und du kannst die Touch-Funktion trotzdem verwenden, wenn du möchtest, aber es wäre besser, die Kalibrierung mit mehr 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 war, sollte der Punkt genau unter der Position des Stylus auf dem Bildschirm angezeigt werden. Um den Test zu beenden, kannst du die Leertaste auf der Tastatur der Konsole drücken.

Touch-Funktionen

Um zu erkennen, 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 hoch nach niedrig) konfiguriert werden. Der Interrupt kann mit dem Befehl SETPIN pin, OFF abgebrochen werden.

Das folgende Programm zeigt, wie der Touch-Interrupt verwendet werden kann. Immer wenn der Bildschirm berührt wird, werden die Koordinaten dieser Berührung auf der Konsole angezeigt. Es wird davon ausgegangen, dass der Befehl OPTION TOUCH 7, 15 verwendet wurde, um die Touch-Funktion zunächst zu konfigurieren:

```
SETPIN 15, INTL, MyInt ' geht davon aus, 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

Mit einer PS2- oder USB-Tastatur und einem LCD-Display kannst du einen komplett eigenständigen Computer bauen, der direkt mit der MMBasic-Eingabeaufforderung startet. Das funktioniert besonders gut mit 5- und 7-Zoll-LCD-Displays, die den SSD1963-Controller nutzen. Die können viel Text anzeigen und schnell

aktualisieren, was ein Benutzererlebnis bietet, das mit einem PicoMite-basierten Computer mit VGA- oder HDMI-Videoausgang vergleichbar ist.

Das LCD muss zuerst 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 Standardfarbe für den Vordergrund, „bc“ ist die Standardfarbe für den Hintergrund und „blight“ ist die Standardhelligkeit der Hintergrundbeleuchtung (2 bis 100). Diese Einstellungen werden im Flash-Speicher gespeichert und beim Einschalten zum Konfigurieren von MMBasic verwendet. 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 oben fortgesetzt. 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 eingestellt werden kann, um die Ausgabe- und Bearbeitungsleistung zu verbessern.

Beachte, dass das nicht nötig ist, wenn das Display im Hochformat verwendet wird, da dann H/W-Scrollen verwendet wird.

Die Farbcodierung im Editor wird ebenfalls durch diesen Befehl aktiviert (siehe 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 wird zusammengefasst, wie ein typisches LCD-Panel mit einem ILI9341-Controller angeschlossen werden kann. Dieses Beispiel unterstützt den SD-Kartensteckplatz, das LCD-Display und die Touch-Schnittstelle.

Typische Panels findest du auf ebay.com und ähnlichen Websites, wenn du nach dem Stichwort „ILI9341” suchst. Achte darauf, dass die Anschlüsse auf der Rückseite des Panels denen unten entsprechen:

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

Um die oben genannten Anschlüsse anzupassen, 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 Einschalten automatisch angewendet. Beachte, dass nach der Eingabe jedes Befehls die Firmware neu gestartet wird und die USB-Verbindung unterbrochen wird und neu hergestellt werden muss.

Als Nächstes sollte der Touchscreen kalibriert werden mit:

```
GUI CALIBRATE
```

Anschließend kannst du die verschiedenen Komponenten testen.

Mit dem folgenden Befehl werden mehrere bunte, sich überlappende Kreise auf dem LCD-Bildschirm angezeigt, um zu überprüfen, ob das LCD richtig angeschlossen ist:

```
GUI TEST LCDPANEL
```

Mit dem folgenden Befehl kannst du die Touch-Oberfläche testen. Wenn du den LCD-Bildschirm berührst, sollte genau an der Berührungsstelle ein Punkt auf dem Bildschirm erscheinen.

```
GUI TEST TOUCH
```

Wenn das nicht genau ist, musst du den Befehl GUI CALIBRATE vielleicht noch mal ausführen und dabei genauer aufpassen.

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-Kartenschnittstelle in Ordnung ist.

```
DATEIEN
```

Wenn du Probleme mit der Anzeige hast, solltest du alles trennen und die Optionen mit dem Befehl OPTION RESET zurücksetzen, damit du neu anfangen kannst. Schließe dann alles Schritt für Schritt wieder an und konfiguriere und teste jeden neuen Schritt, während du fortfährst. Beachte dabei, dass jedes an den SPI-Bus angeschlossene Gerät in MMBasic konfiguriert werden muss oder dass seine Chip-Auswahlleitung auf logisch hoch gehalten werden muss. Dies ist wichtig, da die Spannung an einer nicht angeschlossenen Chip-Auswahlleitung schwankt und möglicherweise dazu führt, dass das falsche Gerät auf Signale reagiert, die für ein anderes Gerät bestimmt sind.

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

Grafikfunktionen

Diese Befehle und Funktionen arbeiten mit angeschlossenen LCD-Panels und VGA/HDMI-Videoausgängen. Eine Anleitung 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 – mehr dazu findest du im Kapitel *„Anzeigepanels*".

VGA-Video

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

OPTION AUFLÖSUNG 640x480

MODE 1 640x480 monochrom mit RGB121-Kacheln, optionaler Layer-Puffer
MODUS 2 320x240 4-Bit-Farbe, optionaler Ebenenpuffer (nur RP2350) zweiter optionaler Ebenenpuffer
MODUS 3 640 x 480 4-Bit-Farbe, optionaler Layer-Puffer (nur RP2350)

OPTIONALE AUFLÖSUNG 720 x 400

MODUS 1 720 x 400 monochrom mit RGB121-Kacheln, optionaler Ebenenpuffer
MODUS 2 360 x 200 4-Bit-Farbe, optionaler Layer-Puffer (nur RP2350) zweiter optionaler Layer-Puffer
MODUS 3 720 x 400 4-Bit-Farbe, optionaler Layer-Puffer (nur RP2350)

OPTIONALE AUFLÖSUNG 800 x 600 (nur RP2350)

MODUS 1 800 x 600 monochrom mit RGB121-Kacheln, optionaler Ebenenpuffer
MODUS 2 400 x 300 4-Bit-Farbe, zwei optionale Ebenen
MODUS 3 800 x 600 4-Bit-Farbe, optionaler Ebenenpuffer

OPTIONALE AUFLÖSUNG 848 x 480 (nur RP2350)

MODUS 1 848 x 480 monochrom mit RGB121-Kacheln, optionaler Ebenenpuffer
MODUS 2 424 x 240 4-Bit-Farbe, zwei optionale Ebenen
MODUS 3 848 x 480 4-Bit-Farbe, optionaler Ebenenpuffer

HDMI-Video (nur RP2350)

Jede HDMI-Auflösung kann in verschiedenen Modi betrieben werden, die mit dem Befehl MODE eingestellt werden:

OPTION AUFLÖSUNG 640 x 480

MODUS 1 640 x 480 x 2 Farben mit RGB555, optionaler Layer-Puffer
MODUS 2 320x240x16 Farben und Farbabbildung auf RGB555-Palette, zwei optionale Ebenen
MODUS 3 640 x 480 x 16 Farben und Farbabbildung auf RGB555-Palette, optionaler Ebenenpuffer
MODUS 4 320 x 240 x 32768 Farben, optionaler Ebenenpuffer
MODUS 5 320 x 240 x 256 Farben und Farbabbildung auf RGB555-Palette, optionaler Ebenenpuffer

OPTIONALE AUFLÖSUNG 720 x 400

MODUS 1 720 x 400 monochrom mit RGB555-Kacheln, optionaler Ebenenpuffer
MODUS 2 360 x 200 4-Bit-Farben und Farbabbildung auf RGB555-Palette, zwei optionale Ebenen
MODUS 3 720 x 400 4-Bit-Farbe und Farbabbildung auf RGB555-Palette, optionaler Ebenenpuffer
MODUS 4 360 x 200 x 32768 Farben, optionaler Ebenenpuffer
MODUS 5 360 x 200 x 256 Farben und Farbabbildung auf RGB555-Palette, optionaler Ebenenpuffer

OPTIONALE AUFLÖSUNG 800 x 600 (nur RP2350)

MODUS 1 800 x 600 monochrom mit RGB332-Kacheln, optionaler Ebenenpuffer

MODUS 2 400 x 300 4-Bit-Farben und Farbabbildung auf RGB332-Palette, optionaler Ebenenpuffer
MODUS 3 800 x 600 4-Bit-Farbe und Farbabbildung auf RGB332-Palette, optionaler Ebenenpuffer
MODUS 5 400 x 300 x 256 Farben, optionaler Ebenenpuffer

OPTIONALE AUFLÖSUNG 848 x 480 (nur RP2350)
MODUS 1 848 x 480 monochrom mit RGB332-Kacheln, optionaler Ebenenpuffer
MODUS 2 424 x 240 4-Bit-Farbe und Farbabbildung auf RGB332-Palette, optionaler Ebenenpuffer
MODUS 3 848 x 480 4-Bit-Farbe und Farbabbildung auf RGB332-Palette, optionaler Ebenenpuffer
MODUS 5 424 x 240 x 256 Farben, optionaler Ebenenpuffer

OPTIONALE AUFLÖSUNG 1280 x 720
MODUS 1 1280 x 720 x 2 Farben mit RGB332, optionaler Layer-Puffer
MODUS 2 320 x 180 x 16 Farben und Farbabbildung auf RGB332-Palette, optionaler Ebenenpuffer
MODUS 3 640 x 360 x 16 Farben und Farbabbildung auf RGB332-Palette, optionaler Layer-Puffer
MODUS 5 320 x 180 x 256 Farben, optionaler Layer-Puffer

OPTIONALE AUFLÖSUNG 1024 x 768
MODUS 1 1024 x 768 x 2 Farben mit RGB332-Kacheln, optionaler Layer-Puffer
MODUS 2 256 x 192 x 16 Farben und Farbabbildung auf RGB332-Palette, optionaler Ebenenpuffer
MODUS 3 512 x 384 x 16 Farben und Farbabbildung auf RGB332-Palette, optionaler Ebenenpuffer
MODUS 5 256 x 192 x 256 Farben, optionaler Layer-Puffer

OPTION AUFLÖSUNG 1024 x 600
MODUS 1 1024 x 600 Mono mit Kacheln
MODUS 2 256 x 150 RGB121
MODUS 3 512x300 RGB121
MODUS 5 256x150 RGB332

OPTIONALE AUFLÖSUNG 800x480 (Anmerkung: verringert den verfügbaren Heap-Speicher)
MODUS 1 800x480 Mono mit Kacheln
MODUS 2 400x240 RGB121
MODUS 3 800x480 RGB121
MODUS 5 400 x 240 RGB332

Farben

Die Farbe wird als Echtfarben-24-Bit-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(rot, grün, blau)
```

Die Funktion RGB() unterstützt auch eine Abkürzung, mit der du gängige Farben durch ihre Bezeichnung angeben kannst, z. B. RGB(red) oder RGB(cyan). Die Farben, die mit der Abkürzungsform benannt werden können, sind Weiß, Schwarz, Blau, Grün, Cyan, Rot, Magenta, Gelb, Braun, Weiß, Orange, Rosa, Gold, Lachs, Beige, Hellgrau und Grau (oder in der US-amerikanischen Schreibweise Gray/Lightgray).

MMBasic wandelt alle Farben automatisch in das vom jeweiligen Display-Controller benötigte Format um. Zum Beispiel sind es beim ILI9341-LCD-Controller 64K Farben im 565-Format.

Die Standardeinstellung für Befehle, die einen Farbparameter brauchen, kann mit dem Befehl COLOUR (kann auch COLOR geschrieben werden) festgelegt werden. Das ist praktisch, wenn dein Programm ein einheitliches Farbschema verwendet. Du kannst dann die Standardeinstellungen festlegen und die Kurzform der Zeichenbefehle in deinem ganzen Programm verwenden.

Der Befehl COLOUR hat das Format: `COLOUR Vordergrundfarbe, Hintergrundfarbe`

Schriftarten

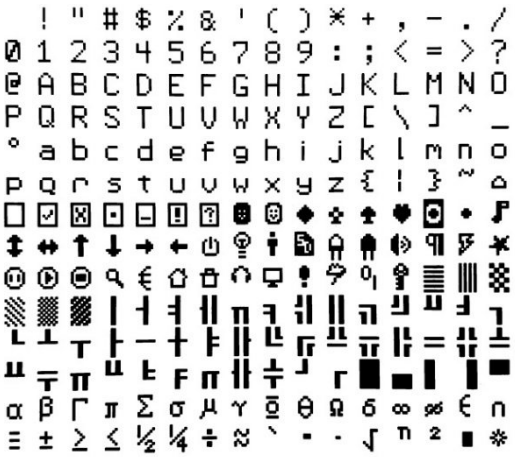
Es gibt acht integrierte Schriftarten. Diese sind:

Schrift Nummer	Größe (Breite x Höhe)	Zeichen Set	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	Große Schriftart für VGA-Versionen.
3	16 x 24	Alle 95 ASCII-Zeichen	Eine große Schriftart für HDMI-Versionen und LCD-Bildschirme.
4	10 x 16	Alle 95 ASCII-Zeichen plus 7F bis FF (hex)	Eine Schriftart mit erweiterten Grafikzeichen. Gut für hochauflösende Displays.
5	24 x 32	Alle 95 ASCII-Zeichen	Extra große Schriftart, sehr klar.
6	32 x 50	0 bis 9 plus ein paar Symbole	Zahlen plus Dezimalpunkt, Plus-, Minus-, Gleichheits-, Grad- und Doppelpunktzeichen. Echt 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-Videoausgabe eine Größe von 16 x 16 Pixel hat, bei HDMI- und LCD-Bildschirmen aber eine Größe von 16 x 24.

In allen Schriftarten (einschließlich Schriftart Nr. 6) wurde das Backquote-Zeichen (60 hexadezimal oder 96 dezimal) durch das Gradzeichen (º) ersetzt.

Schriftart Nr. 1 (die Standardschriftart) und Schriftart Nr. 4 haben einen erweiterten Zeichensatz, der alle Zeichen von CHR$(32) bis CHR$(255) oder 20 bis FF (hex) abdeckt, wie rechts dargestellt.



Eingebettete Schriftarten

Bei Bedarf können zusätzliche Schriftarten in ein BASIC-Programm eingebettet werden. Diese Schriftarten funktionieren genauso wie die integrierte Schriftart (d. h. sie werden mit dem Befehl FONT ausgewählt oder im Befehl TEXT angegeben).

Das Format einer eingebetteten Schriftart ist:

```
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 Schriftart.

Der Hauptteil der Schriftart besteht aus einer Folge von 8-stelligen Hexadezimalwörtern, 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 in einem Programm platziert werden, und MMBasic überspringt sie. Dieses Format entspricht dem von Micromite verwendeten Format.

Weitere Schriftarten und Infos 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.

Bildschirmkoordinaten

Alle Koordinaten und Messungen auf dem Bildschirm werden in Pixeln angegeben, wobei die X-Koordinate die horizontale Position und die Y-Koordinate die vertikale Position angibt. Die obere linke Ecke des Bildschirms hat die Koordinaten X = 0 und Y = 0, und die Werte steigen, wenn man sich nach unten und nach rechts auf dem Bildschirm bewegt.

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 Displays (die Y-Achse) in Pixeln zurück.
- MM.INFO(FONTHEIGHT)
  Gibt die Höhe der aktuellen Standardschriftart (in Pixeln) zurück. Alle Zeichen einer Schriftart haben die gleiche Höhe.
- MM.INFO(FONTWIDTH)
  Gibt die Breite eines Zeichens in der aktuellen Schriftart zurück (in Pixeln). Alle Zeichen haben die gleiche Breite.

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 Kommas hintereinander verwenden, um einen fehlenden Parameter anzuzeigen. Der fünfte Parameter des Befehls LINE ist zum Beispiel 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, die standardmäßig auf -1 gesetzt ist, was bedeutet, dass keine Füllung verwendet werden soll.

Die grundlegenden Zeichenbefehle sind:

- `CLS` *`C`*
  Löscht den Bildschirm in der Farbe C. Wenn C nicht angegeben ist, wird die aktuelle Standard-Hintergrundfarbe verwendet.
- `PIXEL X, Y,` *`C`*
  Leuchtet ein Pixel in der Farb t. Wenn C nicht angegeben ist, wird die aktuelle Standard-Vordergrundfarbe benutzt.
- `LINE X1, Y1, X2, Y2,` *`LW, C`*
  Zeichnet eine Linie, die bei X1 und Y1 anfängt und bei X2 und Y2 endet.
  LW ist die Breite der Linie und gilt nur für horizontale oder vertikale Linien. Wenn nichts angegeben ist oder die Linie diagonal verläuft, ist der Standardwert 1. 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 anfängt und W Pixel breit und 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 Rechteck mit abgerundeten Ecken, das bei X und Y beginnt und W Pixel breit und H Pixel hoch ist.
  R ist der Radius der Ecken des Kastens. 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 ist. Bei einem Seitenverhältnis von 0,5 wird beispielsweise ein Oval gezeichnet, dessen Breite halb so groß ist wie seine Höhe.

- `TEXT X, Y, STRING, ALIGNMENT, FONT, SCALE, C, BC`
  Zeigt eine Zeichenfolge an, die bei X und Y anfängt. ALIGNMENT ist 0, 1 oder 2 Zeichen (ein Zeichenfolgenausdruck oder eine Variable ist auch erlaubt), wobei der erste Buchstabe die horizontale Ausrichtung um X ist und L, C oder R für links-, zentriert- oder rechtsbündigen Text sein kann und der zweite Buchstabe die vertikale Ausrichtung um Y ist 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 standardmäßig auf die durch den Befehl FONT festgelegten Werte eingestellt. C ist die Zeichenfarbe und BC ist die Hintergrundfarbe. Sie sind optional und standardmäßig auf die durch den Befehl COLOUR festgelegten Werte eingestellt.

- `GUI BITMAP X, Y, BITS, WIDTH, HEIGHT, SCALE, C, BC`
  Zeigt die Bits in einer Bitmap an, beginnend bei X und Y. HEIGHT und WIDTH sind die Abmessungen der Bitmap, wie sie auf dem LCD-Bildschirm angezeigt werden, und sind standardmäßig auf 8x8 eingestellt. SCALE, C und BC sind die gleichen wie beim Befehl TEXT. Die Bitmap kann eine Ganzzahl oder eine Zeichenfolgenvariable oder -konstante sein und wird mit dem ersten Byte als ersten Bits der obersten Zeile (zuerst Bit 7, dann Bit 6 usw.) gezeichnet, 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.

- `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 Vordergrundfarbe verwendet.

- `ARC x, y, r1, [r2], a1, a2 [, c]`
  Zeichnet einen Kreisbogen mit einer bestimmten Farbe und Breite zwischen zwei Radien (in Grad definiert). Die Parameter für den Befehl ARC sind die x- und y-Koordinaten des Mittelpunkts des Bogens, der innere und äußere Radius, der Start- und Endwinkel des Bogens und die Farbe des Bogens. Der Nullpunkt liegt bei 12 Uhr.

Gedrehter Text

Wie oben beschrieben, kann die Ausrichtung des Textes im Befehl TEXT 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, wobei jedes Zeichen unter dem vorherigen von oben nach unten verläuft.

I Der Text wird umgekehrt (d. h. auf dem Kopf stehend) angezeigt.

U Der Text wird um 90º gegen den Uhrzeigersinn gedreht.

D Der Text wird um 90º im Uhrzeigersinn gedreht.

Als Beispiel wird der folgende Text „LCD-Anzeige" vertikal am linken Rand des Anzeigefelds und vertikal zentriert angezeigt:

```
TEXT 0, 250, „LCD-Anzeige", „LMV", 5
```

Die Positionierung bezieht sich auf die obere linke Ecke des Zeichens, wenn man es normal ansieht. Also, wenn du 100,100 umdrehst, ist der obere linke Pixel des ersten Zeichens bei 100,100 und der Text ist dann über y=101 und links von x=101. Genauso wird „R" in der Ausrichtungszeichenfolge aus der Perspektive des Zeichens gesehen, egal wie es ausgerichtet ist (nicht vom Bildschirm aus).

Transparenter Text

Der VGA- oder HDMI-Videoausgang oder LCD-Displays, die SSD1963, ST7796S, ILI9341, ST7789_320 oder ILI9488 mit angeschlossenem MISO verwenden, können transparenten Text anzeigen.

In diesem Fall ermöglicht der Befehl TEXT die Verwendung von -1 für die Hintergrundfarbe. Das bedeutet, dass der Text über den Hintergrund gezeichnet wird, wobei das Hintergrundbild durch die Lücken zwischen 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 werden Framebuffer oder Layer-Puffer entweder mit vorab zugewiesenem Speicher oder mit Speicher aus dem Benutzerspeicher erstellt.

Framebuffer können zum Erstellen von Bilddaten verwendet werden, die auf das physische Display kopiert werden können. Layer-Puffer werden in der Regel zum Erstellen von Teilbildern verwendet, die über ein Hintergrunddisplaybild gelegt werden können und eine effiziente Methode zum Verschieben von Anzeigeelementen über einen statischen Hintergrund bieten.

Alle Standard-Grafikzeichnungsbefehle können auf einem Framebuffer oder Layer-Puffer genauso verwendet werden wie beim Schreiben auf die physische Anzeige. Der Befehl FRAMEBUFFER WRITE wird verwendet, um das Ziel der Grafikausgabe mithilfe eines *Codes* zu steuern.

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 sind:

```
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 über das Hauptanzeigebild gelegt, wenn es auf den Bildschirm ausgegeben wird. s Fall von LCD-Displays wird der Befehl FRAMEBUFFER MERGE verwendet, um das endgültige Bild aus einem Framebuffer und einem Layer-Puffer 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-Versionen Modus 2 und Modus 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 Funktion. Ab Version 6.00.00 sind es separate Befehle. Der Unterschied besteht darin, dass BLIT eine einfache Speicheroperation ist, bei der Daten von einem Display oder Speicher auf ein Display oder einen Speicher kopiert werden. Sprites sind komplexer und ermöglichen es dem Programmierer, Elemente über einem Hintergrund anzuzeigen und sie dann über den Hintergrund zu bewegen, ohne das Hintergrundbild zu beschädigen. Darüber hinaus 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 Lesen aus seinem Framebuffer unterstützt, und auch die Blit-Funktionalität ist in diesem Fall eingeschränkt. Sprites sind für alle Versionen der Firmware aktiviert, wenn sie auf einem In-Memory-Framebuffer und VGA- und HDMI-Versionen der Firmware direkt mit dem Bildschirm verwendet werden.

Sprites werden immer als RGB121-Nibbles mit 2 Pixeln pro Byte gespeichert. Im Gegensatz dazu werden BLIT-Puffer als RGB888-Werte gespeichert und können daher mit Vollfarb-LCD-Displays verwendet werden. Dies geht natürlich zu Lasten eines deutlich höheren Speicherbedarfs.

Weitere Infos zur Verwendung von Sprites findest du unter dem Befehl SPRITE und der Funktion zusammen mit Anhang G.

Wenn das Display transparenten Text anzeigen kann, kannst du mit dem BLIT-Befehl einen Teil des gerade angezeigten Bildes in einen Speicherpuffer kopieren und später wieder auf das Display kopieren. Das ist praktisch, wenn du etwas über den Hintergrund zeichnen und später wieder entfernen willst, ohne das Bild im Hintergrund zu beschädigen. Beispiele hierfür sind Spiele, in denen sich eine Figur vor einer Landschaft bewegt, oder die sich bewegende Nadel eines fotorealistischen Messgeräts.

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 angezeigte Bild in den Puffer, WRITE kopiert den Puffer auf das Display und CLOSE gibt den Puffer frei und holt sich den verwendeten Speicher zurück. LOAD lädt eine Bilddatei in den Puffer.

BLIT LOAD und BLIT WRITE funktionieren auf jedem Display, während BLIT und BLIT READ nur auf Displays funktionieren, die transparenten Text unterstützen (d. h. mit SSD1963, ILI9341, ST7789_320 oder ILI9488 mit angeschlossenem MISO) sowie auf VGA- und HDMI-Displays und allen In-Memory-Framebuffern.

Mit diesen Befehlen kann ein Teil der Anzeige an einen anderen Ort kopiert werden (durch Kopieren in einen Puffer und anschließendes Schreiben an eine andere Stelle), aber einfacher ist es, eine alternative Version des BLIT-Befehls wie folgt zu verwenden:

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

Damit wird ein Teil des Bildes an x1/y1 an den Ort x2/y2 kopiert. w und h geben die Breite und Höhe des zu kopierenden Bildes an. Der Quell- und der Zielbereich können sich überlappen, und der BLIT-Befehl führt die Kopie korrekt aus.

Diese Form des BLIT-Befehls ist besonders nützlich, um Grafiken zu erstellen, die horizontal oder vertikal scrollen können, wenn neue Daten hinzugefügt werden.

Außerdem bietet die Firmware die Befehle BLIT MEMORY, BLIT COMPRESSED, BLIT FRAMEBUFFER und BLIT MERGE. Diese erweiterten Befehle können zum Programmieren von Spielen mit Hunderten von Anzeigeelementen verwendet werden, wie z. B. der Portierung von PETSCII-Robotern auf MMBasic.

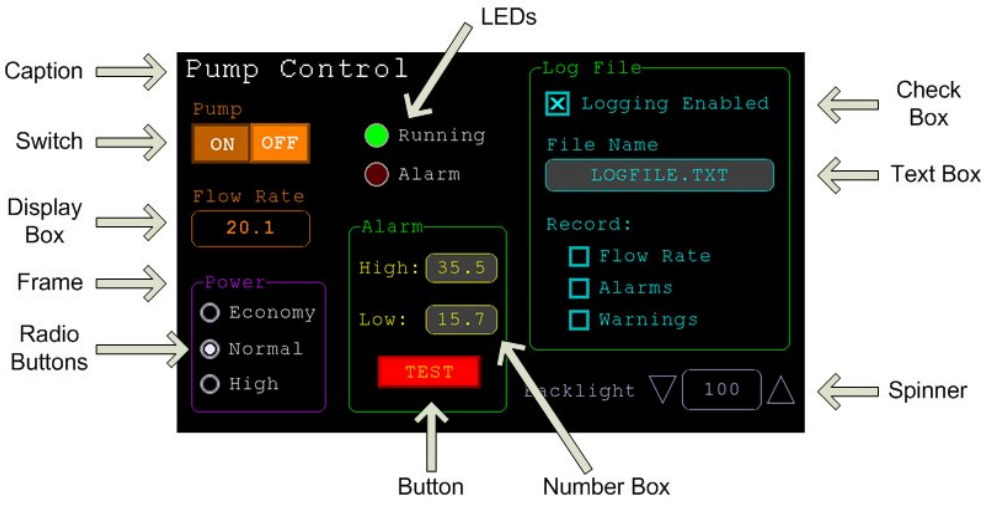
Bild laden

Mit den Befehlen LOAD IMAGE und LOAD JPG kann ein Bild aus dem Flash-Dateisystem oder von der SD-Karte geladen und auf dem LCD-Display angezeigt werden. Damit kann ein Logo gezeichnet oder den auf dem Display gezeichneten Grafiken ein verzierter Hintergrund hinzugefügt werden.

Erweiterte Grafik

NICHT VERFÜGBAR IN DEN VERSIONEN VGA/HDMI UND WEBMITE RP2040

Die PicoMite-Firmware enthält eine Reihe von erweiterten Grafikfunktionen, mit denen Programmierer ganz einfach berührungsempfindliche Steuerelemente auf einem LCD-Bildschirm erstellen können, wie in der Abbildung gezeigt. Dazu gehören Bildschirmschalter, Schaltflächen, Anzeigeleuchten, Tastaturen usw.



MMBasic zeichnet die Steuerung und animiert sie (d. h., ein Schalter scheint bei Berührung gedrückt zu werden). Der BASIC-Programmierer muss lediglich einen einzigen Befehl aufrufen, um die grundlegenden Details der Steuerung festzulegen.

Mit diesen Funktionen lässt sich ganz einfach ein Bedienfeld erstellen, um beliebige Steuerungsfunktionen wie Drehmaschinen, Motorsteuerungen, Heizungssysteme, kleine industrielle Prozesse usw. zu verwalten.

Die erweiterten Grafikfunktionen werden ausführlich im Dokument *„Advanced Graphics Functions.pdf"* beschrieben, das in der Firmware-Download-Datei enthalten ist.

3D-Engine

NICHT IN 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 ILI9341-basierten LCD-Display. Das Programm wird beendet und kehrt zur Eingabeaufforderung zurück, wenn der Bildschirm berührt wird.

Zuerst musst du die Anzeige- und Touch-Optionen konfigurieren, 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                                 ' 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"
  WENN TOUCH(X) <> -1 DANN ENDE
LOOP
```

Dieses Programm fängt damit an, 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 Feld mit roten Wänden und einem dunkelblauen Innenraum.

Danach geht das Programm in eine Endlosschleife, wo es drei Sachen macht:

1. Es zeigt die aktuelle Uhrzeit in dem zuvor gezeichneten Kasten an. Die Zeichenfolge wird sowohl horizontal als auch vertikal in der Mitte des Kastens zentriert gezeichnet. Beachte, dass der Befehl TEXT die Standardschriftart und -farben überschreibt, um seine eigenen Parameter festzulegen.
2. Zeichnet das Datum zentriert in der unteren Hälfte des Bildschirms. In diesem Fall verwendet der Befehl TEXT die zuvor festgelegten Standardschriftart und -farben.
3. Es prüft, ob der Bildschirm berührt wird. Das 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 anders):

WiFi- und Internetfunktionen

NUR WEBMITE-VERSION

Dieses Kapitel gibt einen Überblick über die WiFi- 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 ein paar Punkte beachten:

- Die Implementierung der Internetprotokolle beansprucht viele Ressourcen des Prozessors (insbesondere RAM), daher sollte die WebMite-Firmware nur verwendet werden, wenn WLAN- und Internetkonnektivität 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 zusammenarbeitet, und enthält ein paar einfache Beispiele, aber es geht nicht auf HTML, TCP und die vielen anderen Protokolle und Konventionen ein. Diese können für Neulinge verwirrend sein, die sich daher mit einigen der vielen im Internet verfügbaren Einführungen vertraut machen sollten.
- Bei der Verarbeitung komplexer Internetprotokolle kann die WebMite-Firmware verwirrt werden und hängen bleiben oder einen Fehler ausgeben und zur Eingabeaufforderung zurückkehren. Um dies zu ermöglichen, sollten Programme die WATCHDOG-Funktion verwenden, um solche Situationen zu beheben. Es wird auch empfohlen, das Programm von Zeit zu Zeit (z. B. einmal pro Woche) mit CPU RESTART einen Neustart zu erzwingen, um sicherzustellen, dass die Protokollstacks zurückgesetzt werden.

Verbindung zu einem WLAN-Netzwerk herstellen

Bevor du irgendetwas tust, musst du die WLAN-Funktion auf dem WebMite aktivieren. Dazu gibst du den folgenden Befehl in die Befehlszeile ein:

```
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 Zeichenfolgen, und wenn Zeichenfolgenkonstanten verwendet werden, sollten sie wie in diesem Beispiel in Anführungszeichen gesetzt werden:

```
OPTION WIFI „MyNetwork", „secret"
```

Nach der Eingabe startet das WebMite neu, was bedeutet, dass du die USB-Verbindung und den Zugriff auf die Konsole verlierst. Wenn du die Verbindung schnell wiederherstellst, wird die folgende Meldung angezeigt:

```
Verbindung zum WLAN wird hergestellt...
Verbunden 192.168.1.52
```

Die in der letzten Zeile angegebene Adresse ist die IP-Adresse, die dem WebMite vom Router zugewiesen wurde und je nach Netzwerk variiert. Wenn der WebMite keine Verbindung herstellen kann, wird folgende Meldung angezeigt:

```
Verbindung zum WLAN wird hergestellt...
Verbindung fehlgeschlagen.
```

Wenn die Verbindung nicht klappt, solltest du die Einstellungen deines WLAN-Routers checken:

- 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 auch 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 brauchen, bis die Verbindung hergestellt ist. Wenn du also OPTION AUTORUN ON benutzt, solltest du am Anfang deines Programms etwas wie das hier einfügen:

```
DO WHILE MM.INFO(IP-ADRESSE) = "0.0.0.0"
  IF TIMER > 5000 THEN CPU RESTART
LOOP
```

Dadurch wird 5 Sekunden lang auf eine Verbindung gewartet und bei fehlender Verbindung ein Neustart durchgeführt.

Fernzugriff auf die Konsole

Du kannst dich über Telnet per WLAN mit der Konsole des WebMite verbinden. Das ist praktisch, wenn das Gerät an einem schwer zugänglichen Ort steht. Sobald die Verbindung über Telnet hergestellt ist, kannst du alles machen, was du normalerweise über die USB-Konsole machen würdest, einschließlich des Ausführens des Editors.

Um diese Funktion zu aktivieren, gib Folgendes in die Befehlszeile ein: `OPTION TELNET CONSOLE ON`

Das musst du nur einmal eingeben und es wird bei jedem Neustart deines WebMite gespeichert. Dadurch wird der WebMite neu gestartet, sodass du dich wieder mit der USB-Konsole verbinden musst.

Tera Term (http://tera-term.en.lo4d.com) unterstützt Telnet und wird für diese Aufgabe empfohlen, da es eine gute VT100-Emulation bietet 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 und daher die empfohlene Methode für die Übertragung von Dateien zum und vom WebMite.

Ein TFTP-Server auf dem WebMite wird automatisch aktiviert, wenn du mit einem WLAN-Netzwerk verbunden bist, sodass hier nichts weiter erforderlich ist. Du benötigst einen TFTP-Client für deinen PC, und es gibt viele verschiedene Implementierungen für Windows, Mac OS und Linux. Beachte, dass in vielen Tutorials zu TFTP die Einrichtung eines TFTP-Servers behandelt wird – dies ist jedoch nicht erforderlich, du benötigst lediglich einen Client.

Bei Windows ist ein TFTP-Client im Betriebssystem dabei, aber du musst ihn über die Systemsteuerung aktivieren. Dazu gehst du wie folgt vor:

```
Systemsteuerung -> Programme und Funktionen -> Windows-Funktionen
aktivieren oder deaktivieren
```

Scroll dann in der Liste nach unten und setz ein Häkchen bei `TFTP-Client`.

Um eine Datei von deinem PC an den WebMite zu senden, gib in einem Befehls- oder PowerShell-Fenster auf deinem Windows-PC Folgendes ein („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-Adresse GET Dateiname
```

Um die Funktionen des TFTP-Clients aufzulisten, gib Folgendes ein:

```
TFTP -h
```

Ein alternativer und einfacher grafischer Windows-TFTP-Client ist: http://www.3iii.dk/linux/dd-wrt/tftp2.exe

Zeit abrufen

Ein üblicher erster Schritt in einem Programm ist, die Uhrzeit/das Datum abzurufen und die Uhr im WebMite einzustellen. Damit wird auch überprüft, ob der WebMite eine Verbindung zum Internet herstellen kann.

Die Zeit wird mit dem Befehl `WEB NTP` wie folgt abgerufen:

```
WEB NTP [timeoffset [, NTPserver$]]
```

Dabei ist „timeoffset” die Zeitzone, in der du dich befindest, und „NTPserver$” der Name oder die IP-Adresse des zu verwendenden Zeitservers. Dieser letzte Parameter ist optional. Wenn er weggelassen wird, verwendet die Firmware einen öffentlichen Zeitserver-Pool. Wenn auch der Parameter „timeoffset” weggelassen wird, 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
NTP-Antwort erhalten: 08/03/2023 05:34:57
```

Wenn der WebMite keinen Internetzugang hat, bekommst du eine Fehlermeldung. Diese 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 „Fehler beim Herstellen der Internetverbindung“: CPU
RESTART
```

Wir versuchen den Befehl WEB NTP zweimal, falls der erste Versuch wegen eines Zeitfehlers nicht klappt (das kann passieren) – beim zweiten Mal sollte es aber klappen.

Implementierung eines Webservers

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 in der Befehlszeile zu konfigurieren:

```
OPTION TCP SERVER PORT nn
```

Dabei ist „nn” die zu verwendende Portnummer (normalerweise 80 für eine Webseite). Der Befehl lautet in der Regel:

```
OPTION TCP SERVER PORT 80
```

Wie bei den anderen oben aufgeführten OPTION-Befehlen muss dieser Befehl nur einmal eingegeben werden und wird bei jedem Neustart des WebMite gespeichert. Dadurch wird auch der WebMite neu gestartet, und wenn du dich schnell wieder verbindest, siehst du Folgendes (mit einer anderen IP-Adresse):

```
Server wird unter 192.168.1.52 auf Port 80 gestartet
```

Mit dem obigen Schritt hast du 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 vom TCP-Server empfangen wird. Diese Subroutine kann den Befehl `WEB TCP READ` verwenden, um die eingehende Anfrage vom Remote-Client zu lesen, und der Befehl `WEB TRANSMIT PAGE` kann verwendet werden, um die angeforderte Webseite zu senden.

Hier ist zum Beispiel das komplette Programm zum Implementieren 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     '<hier irgendwas machen>'
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       Wenn (p% <> 0) Und (t% > p%) Dann
14             WEB TRANSMIT PAGE a%, „index.html”
15       ENDIF
16    NEXT a%
17 END SUB
```

Hier kommt die detaillierte Beschreibung des Programms:

Zeile 1 Zuerst wird ein 4K-Byte-Puffer für die eingehende Anfrage vom Client erstellt. In diesem Beispiel werden die Long-String-Befehle in MMBasic für die Verarbeitung der Daten verwendet, und dies ist der Puffer dafür (eine Beschreibung von Long Strings findest du im nächsten Kapitel mit dem Titel *„Long Strings“*). Die Größe dieses Puffers begrenzt die vom Client empfangene Datenmenge.

Zeile 2 Hier wird der Server gestartet und die Interrupt-Subroutine für die Verarbeitung eingehender Anfragen als „`WebInterrupt`” festgelegt.

Zeile 4 Dies ist Ihre Hauptprogrammschleife, die normalerweise Daten sammelt, Ausgänge umschaltet usw.

Zeile 7 Das ist die Unterroutine, die jedes Mal aufgerufen wird, wenn eine Anfrage vom Browser des Remote-Clients kommt.

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 und 12 Hol dir die Positionen der Schlüsselwörter in der Nachricht.

Zeile 13 Überprüfe, ob die Schlüsselwörter vorhanden sind und in der richtigen Reihenfolge stehen.

Zeile 14 Die Seite senden. Das ist 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>Dies ist eine Überschrift</h1>` enthalten kann. Weitere Informationen findest du unter *„Eine typische Webseite*“ weiter unten.

Einfügen von Daten in die Webseite

Normalerweise musst du Daten, die vom BASIC-Programm generiert wurden, in die übertragene Webseite einfügen. Das geht ganz einfach, indem du den Namen der BASIC-Variablen in geschweifte Klammern (d. h. { und }) in den Text der Webseite einfügst.

Wenn dein Programm zum Beispiel eine Variable namens `CurrentTemp` mit dem Wert 24 hat, die die aktuelle Temperatur angibt, würde die folgende Webseite: `Die Temperatur beträgt {CurrentTemp}` im Browser des Kunden wie folgt angezeigt werden: `Die Temperatur beträgt 24`.

Der Bezeichner zwischen den geschweiften Klammern kann eine Gleitkommazahl, eine Ganzzahl oder eine Zeichenfolge, 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 von 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 in deiner Webseite eine öffnende geschweifte Klammer brauchst, kannst du zwei als Paar verwenden (z. B. {{), die dann in eine einzelne öffnende Klammer umgewandelt werden. Eine schließende geschweifte Klammer ohne öffnendes Gegenstück bleibt unverändert.

Senden mehrerer Seiten

Wenn ein Remote-Client Daten anfordert, ohne eine Seite anzugeben, sendet er die Anfrage als `GET / HTTP`, wobei der Schrägstrich die Standardseite für den Server darstellt (normalerweise index.html). Im obigen Beispiel wurde dies nicht überprüft, sondern für alle Anfragen wurde einfach dieselbe Seite gesendet.

Wenn deine Seite aber 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 bewerkstelligen, indem die angeforderten Daten zwischen den Schlüsselwörtern GET und HTML überprüft werden. Ersetze beispielsweise 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-ÜBERTRAGUNGSSEITE a%,"page2.html"
ENDIF
```

Das kannst du auf so viele Seiten ausweiten, wie du brauchst.

Ein Bild senden

Du kannst ein Bild mit dem folgenden HTML-Code `<img src="pix.jpg"/>` in deine Webseite einfügen. Wenn der Remote-Browser das liest, sendet er die folgende Anfrage `GET /pix.jpg HTTP`. Du kannst dann das angeforderte Bild mit dem fett gedruckten 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"
SONST WENN s$ = "/pix.jpg" DANN
   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 besser.

Der Parameter „image/jpeg” ist als MIME-Typ bekannt, 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 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"
SONST WENN s$ = "/pix.jpg" DANN
   WEB TRANSMIT FILE a%,"pix.jpg","image/jpeg"
SONST
   WEB TRANSMIT CODE a%, 404
ENDIF
```

Live-Grafikdaten auf einer Webseite

Zahlen und Text auf einer Webseite sind nützlich, aber oft möchtest du auch grafische Elemente wie Kreisdiagramme, Liniendiagramme, historische Trends usw. einfügen, die aus den vom Programm gesammelten Daten abgeleitet wurden. Das kannst du auf Umwegen machen, indem du WebMite mit einem virtuellen Anzeigefeld konfigurierst, dann mit den Standard-Zeichenbefehlen (Pixel, Linie, Kreis usw.) auf diesem -Display zeichnest und das Ganze als BMP-Bild speicherst. Diese Datei kann dann als Bild in die Webseite eingebunden werden. Im Einzelnen läuft das so ab:

Zuerst muss ein virtuelles Display konfiguriert werden. Es gibt zwei, die du verwenden kannst: VIRTUAL_C ist ein Bild mit 320 x 240 Pixeln und 16 Farben, und VIRTUAL_M ist ein monochromes Bild mit 640 x 480 Pixeln. Zum Beispiel:

```
OPTION LCDPANEL VIRTUAL_C
```

Wie bei den anderen OPTION-Befehlen muss dieser Befehl an der Eingabeaufforderung eingegeben werden, führt zu einem Neustart und muss nur einmal eingegeben werden.

Dann 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(rot), RGB(blau)
LINE 10, 10, 200, 200, 1, RGB(yellow)
```

Wenn du fertig bist, speicher dieses Bild:

```
SAVE COMPRESSED IMAGE "graph.bmp"
```

In die 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 wie oben beschrieben vom Remote-Browser geladen wird (siehe *Senden eines Bildes*). 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, wenn neue Daten aufgezeichnet werden. Da dies aber nur nötig ist, wenn der Remote-Browser das Bild anfordert, sollte es nicht zu einer übermäßigen Abnutzung des Flash-Speichers kommen.

Ein vollständiger Allzweck-Server

Auf den vorherigen Seiten wurden die einzelnen Komponenten beschrieben, aus denen ein Webserver besteht. Nachfolgend findest du ein Beispiel für einen kompletten und integrierten Allzweck-Server, der die meisten Anfragen eines Browsers bearbeiten kann. Er überprüft die Erweiterung 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
   '<hier eine Verarbeitung durchführen>'
LOOP
```

```
' Subroutine zur Bearbeitung aller Webserver-Anfragen
SUB WebInterrupt
  LOCAL a%, p1%, p2%, file$, buff%(4096/8)
  FOR a% = 1 To MM.INFO(MAX CONNECTIONS)
    WEB TCP READ a%, buff%()
    P1% = LINSTR(buff%(),"GET")
    P2% = LINSTR(buff%(),"HTTP")
    Wenn (p1% <> 0) Und (p2% <> 0) Und (p2% > p1%) Dann
      file$ = LCASE$(LGetStr$(buff%(), p1% + 4, p2% - p1% - 5))
      WENN file$ = "/" DANN file$ = "/index.html"
      BEI FEHLER ÜBERSPRINGEN
      file$ FÜR EINGABE ALS #1 ÖFFNEN        ' Überprüfe, ob die Datei da ist
      WENN MM.ERRNO DANN WEB TRANSMIT CODE a%, 404 : FORTSETZEN FÜR
      #1 schließen
      WÄHL FALL RECHTS$(Datei$, 4)
        CASE "html", ".htm", ".txt"
          WEB TRANSMIT PAGE a%, file$
        CASE ".bmp", ".png", ".gif"
          WEB TRANSMIT FILE a%, file$, "image/" + RIGHT$(file$, 3)
        CASE ".jpg", "jpeg"
          WEBÜBERTRAGUNGSDATEI a%, Datei$, „image/jpeg“
        FALL ".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 der SD-Karte gespeichert werden müssen und ihre Namen nur Kleinbuchstaben enthalten dürfen (das Flash-Dateisystem unterscheidet zwischen Groß- und Kleinschreibung).

Eine typische Webseite

Die Webseite, die mit dem Befehl WEB TRANSMIT PAGE übertragen werden soll, muss nach 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>Temperaturüberwachung</h2>
Die Temperatur beträgt {CurrentTemp}
```

Oder du kannst eine komplexe Seite senden. Diese haben normalerweise einen Kopf- und einen Hauptteil, unterteilen den Text in Absätze und verwenden das Break-Tag für Abstände. Hier ist das Grundgerüst einer solchen Seite:

```
<html>
<head>
<title>WebMite</title>
</head>
<body>
<h1>Das ist eine Überschrift</h1>
<br>
<p>Die Temperatur ist {CurrentTemp}</p>
</body>
</html>
```

Im Internet gibt's viele Ressourcen mit HTML-Tutorials für Anfänger. Ein typisches Beispiel ist http://www.simplehtmlguide.com/. Außerdem gibt's viele WYSIWYG-HTML-Editoren. Zum Beispiel: https://onlinehtmleditor.dev/

Eingabefelder und Steuerung

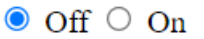
Mit HTML-Eingabefeldern kannst du Schaltflächen, Kontrollkästchen, Optionsfelder usw. auf deiner Webseite platzieren, über die der Benutzer Anfragen an den WebMite senden kann. Auf diese Weise kann der Benutzer aus der Ferne Dinge ein- und ausschalten, Steuerungsparameter einstellen und vieles mehr – alles über die vom WebMite bereitgestellte Webseite.

Dazu musst du ein *HTML-Formular* in die Webseite einfügen, das ein oder mehrere Eingabefelder enthält. Es gibt 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$} onClick='this.form.submit()'> Aus
    <input name='RB' type='radio' value='ON' {onb$} onClick='this.form.submit()'> Ein
</form>
```

Dadurch werden zwei Optionsfelder erstellt, die im Browser so aussehen:



Beachte, dass der obige HTML-Code zwei BASIC-Variablen (`offb$` und `onb$`) enthält, die durch den Befehl WEB TRANSMIT PAGE ersetzt werden. Diese Variablen steuern, wie die Schaltfläche angezeigt wird. Wenn sie auf eine leere Zeichenfolge gesetzt sind, wird die Schaltfläche als nicht aktiviert angezeigt. Wenn sie auf die Zeichenfolge „`checked='checked'`" gesetzt sind, wird die Schaltfläche als aktiviert angezeigt. Diese Variablen sollten beim Start des Programms initialisiert werden.

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 Schaltfläche 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"
    ' <Code zum Ausschalten des Geräts hier einfügen>
    offb$ = "checked='checked'" : onb$ = ""
    WEBÜBERTRAGUNGSSEITE a%, "index.html"
SONST WENN s$ = "/?RB=ON"
    ' <Code hier einfügen, um das Gerät einzuschalten>
    offb$ = "" : onb$ = "checked='checked'"
    WEBÜBERTRAGUNGSSEITE a%, "index.html"
```

Im Grunde macht dieser Code nur Folgendes: Er schaltet das Gerät wie gewünscht ein oder aus, setzt die Variablen `onb$` und `offb$`, um den neuen Status der Schaltflächen anzuzeigen, und schickt dann die ganze Webseite zurück an den Browser.

Dieses Beispiel lässt viele Details außer Acht, und du kannst es extrem kompliziert machen, wenn du mehrere Eingaben mit Schaltflächen, Texteingaben, Dropdown-Listen, Passwortfeldern, Anfragen für Datei-Uploads und vielem mehr hinzufügen möchtest. Dazu musst du dich jedoch tiefer mit HTML-Codierung befassen. Ein Beispiel hierfür findest du im Projekt *„Garden Watering Controller"* 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 verwaltet. Der erste lautet:

```
WEB OPEN TCP CLIENT Domain$, PortNumber
```

Damit wird eine TCP-Verbindung zu `Domain$` (z. B. „openweathermap.org") über die angegebene `PortNumber` (normalerweise 80 für eine Webseite) hergestellt.

Wenn die Verbindung offen ist, 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, bei dem es sich normalerweise um eine lange Zeichenfolgenvariable wie `buff%( 4096/8 )` handelt. 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$" einfach „`GET / HTTP`" sein, 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 komplettes 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. Dabei handelt es sich um eine 32-stellige Hexadezimalzahl, mit der du die Abfrage durchführen kannst. Diese Zahl sollte den Dummy-Schlüssel in der ersten Zeile ersetzen.

```
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 benutzt, um den gewünschten Wert aus der Antwort im JSON-Format (JavaScript Object Notation) zu holen.

Wenn du dieses Programm ausführst, solltest du was wie das hier sehen:

```
Verbunden
Aktuelle Temperatur in Paris: 13,54
```

Verwendung von UDP

Das User Datagram Protocol (UDP) ist ein schnelles und leichtgewichtiges Protokoll, das zum Senden von Nachrichten zwischen Geräten verwendet wird. Es kann unzuverlässig sein, da es nicht garantiert, dass Datenpakete zugestellt werden oder in der richtigen Reihenfolge ankommen. Es wird jedoch häufig für zeitkritische Anwendungen verwendet, bei denen Geschwindigkeit Vorrang vor Zuverlässigkeit hat.

Das Einrichten eines UDP-Servers ist ähnlich wie das Einrichten eines TCP-Servers. Zuerst wird UDP mit der folgenden Option in 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
```

Wobei „InterruptSub” der Name deiner Subroutine ist, die immer dann aufgerufen wird, wenn der UDP-Server eine Nachricht bekommt. 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 mit folgendem Befehl senden:

```
WEB UDP send ip_address$, port_nbr, message$
```

In der folgenden Demo werden zwei WebMites verwendet, die mit OPTION WIFI im selben Netzwerk konfiguriert sind.

Bevor du loslegst, musst du beide WebMites mit OPTION UDP SERVER PORT 77 einrichten.

Notier dir die IP-Adresse des ersten WebMites und gib das folgende Programm ein:

```
WEB UDP interrupt myint
Do
   '<hier einige Verarbeitungsschritte durchführen>'
Loop

' Unterprogramm unterbrechen
Sub myint
   Drucke „Empfangen “ + mm.message$ + „ von “ + mm.address
   Pause 100
   WEB UDP send mm.address$, 77, „WebMite #1 message”
Ende Sub
```

Starte dann das Programm, das zunächst in einer Schleife nichts macht.

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
   '<hier einige Verarbeitungsschritte durchführen>'
Loop

' Unterprogramm unterbrechen
Sub myint
   Drucke „Empfangen “ + mm.message$ + „ von “ + mm.address$
```

```
    Pause 100
    WEB UDP send mm.address$, 77, „WebMite #2 message”
Ende Sub
```

Das Ergebnis ist ein echt schneller Ping-Pong-Austausch von UDP-Nachrichten zwischen den beiden WebMites.

E-Mails senden

Wenn du ein Remote-Gerät wie WebMite hast, ist es praktisch, wenn es E-Mails verschicken kann, um Fehler zu melden, über seinen Status zu berichten und so weiter. WebMite kann das über das SMTP-Protokoll machen, um sich mit einem Server zu verbinden, der die E-Mail dann an den Empfänger weiterleitet.

Im folgenden Beispiel wird der kostenlose SMTP-Relay-Dienst von SMTP2GO verwendet, der 1000 E-Mails pro Monat erlaubt (mehr als genug für das WebMite). Beachte, dass sie keine Registrierungsanfragen von Leuten mit einer generischen kostenlosen E-Mail-Adresse (z. B. xxx@gmail.com) annehmen.

Um loszulegen, musst du dich kostenlos bei 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 die Zeichenfolge „ `nnnnnnnnn` ” in der ersten Zeile des folgenden Programms ersetzen, während das Base64-kodierte Passwort die Zeichenfolge „xxxxxxxxxxxxx” in der zweiten Zeile ersetzen sollte. Die anderen vier Zeilen am Anfang des Programms sollten ebenfalls durch deine Daten ersetzt werden:

```
CONST userBase64$ =  "nnnnnnnnnnnn"
CONST paswdBase64$ = "xxxxxxxxxxxx"
CONST mailfrom$ =    "from@server.com"
CONST mailto$ =      "to@server.com"
CONST subject$ =     "Test-E-Mail"
CONST message$ =     "Test von SMTP2GO "

CONST cr = Chr$(13)+Chr$(10)
DIM buff%(4096/8), body$

body$ = "Von: " + mailfrom$ + cr + "An: " + mailto$ + cr
body$ = body$ + "Betreff: " + 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-ANFRAGE „AUTH LOGIN” + cr, buff%()
WEB-TCP-CLIENT-ANFRAGE userBase64$ + cr, buff%()
WEB-TCP-CLIENT-ANFRAGE paswdBase64$ + cr, buff%()
WEB-TCP-CLIENT-ANFRAGE „MAIL FROM: ‟ + mailfrom$ + cr, buff%()
WEB-TCP-CLIENT-ANFRAGE „RCPT TO: ‟ + mailto$ + cr, buff%()
WEB-TCP-CLIENT-ANFRAGE „DATA” + cr, buff%()
PAUSE 300
WEB-TCP-CLIENT-ANFRAGE body$, buff%()
PAUSE 300
WEB TCP-CLIENT SCHLIESSEN
WENN LINSTR(buff%(), „250 OK”) = 0 DANN
  PRINT „E-Mail-Versand fehlgeschlagen”
Sonst
  Drucken „E-Mail erfolgreich gesendet‟
EndIf
```

Beachte, dass die im obigen Programm verwendete E-Mail-Adresse `mailfrom$ genau` die gleiche sein muss wie die, die du bei der Registrierung des verifizierten Absenders bei SMTP2GO angegeben hast. Wenn sie nicht übereinstimmen, lehnt SMTP2GO die E-Mail ab (dies ist eine Anti-Spam-Maßnahme).

Heutzutage ist die Nutzung eines SMTP-Relay-Dienstes kompliziert, weil die SMTP-Protokolle der verschiedenen Anbieter unterschiedlich sind und es viele Schutzmaßnahmen gegen Spam gibt. Dieses Beispiel bezieht sich speziell auf das SMTP-Protokoll von SMTP2GO, aber du kannst auch andere Dienste nutzen. Dazu musst du das Programm aber an deren SMTP-Protokoll anpassen.

Base 64-Kodierung

Base64 ist ein System zur Umwandlung von Binärdaten in eine Textzeichenfolge, die nur ASCII-Zeichen verwendet (d. h. es gibt keine Steuerzeichen). Es wurde entwickelt, um das Senden von Binärdaten über das Internet mit Protokollen zu vereinfachen, die keine Binärdaten akzeptieren, und viele Protokolle erfordern seine Verwendung.

Die MATH BASE64-Kodierungs-/Dekodierungsfunktion 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 verwenden, um den Benutzernamen und das Passwort in Base64 zu konvertieren, kannst du dies im Programm mit dieser Funktion tun.

Zum Beispiel:

```
DIM n, userBase64$, paswdBase64$
CONST user$ = "MyUserName"          ' Benutzername im Klartext
CONST paswd$ = "MyPassword"         ' Passwort im Klartext
...
' Benutzernamen und Passwort verschlüsseln
n = MATH(BASE64 ENCODE, user$, userBase64$)
n = MATH(BASE64 ENCODE, paswd$, paswdBase64$)
...
```

MQTT-Client

MQTT ist ein Protokoll, mit dem Clients wie WebMite Nachrichten auf einem Server (auch MQTT-Broker genannt) posten oder abrufen können. Das ist so ähnlich wie ein Schwarzes Brett oder ein Webforum, wo Leute Nachrichten posten, die andere später lesen können – der Hauptunterschied ist, dass MQTT für die Kommunikation zwischen Maschinen gedacht ist.

Ein typisches Beispiel wäre ein batteriebetriebenes WebMite, das den Wasserstand in einem Stausee überwacht. Zweimal täglich würde es sich einschalten, die Messung durchführen, 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, such einfach bei Google nach „kostenloser MQTT-Broker".

Der WebMite nutzt fünf Befehle, um Nachrichten zu senden oder abzurufen:

```
WEB MQTT CONNECT  Verbindung zu einem MQTT-Broker herstellen.
WEB MQTT PUBLISH  Veröffentliche Inhalte in einem MQTT-Thema (d. h. poste
eine Nachricht).
WEB MQTT SUBSCRIBE      Abonniere ein MQTT-Thema (also hol dir
Nachrichten).
WEB MQTT UNSUBSCRIBE    Ein MQTT-Thema abbestellen.
WEB MQTT CLOSE    Schließ die MQTT-Verbindung.
```

Ping

Das WebMite antwortet auf eine Ping-Nachricht, sodass du überprüfen kannst, ob es aktiv und erreichbar ist. Wenn es mit dem öffentlichen Internet verbunden ist, kannst du einen kostenlosen Dienst wie https://uptimerobot.com/ nutzen, der dich benachrichtigt, wenn es nicht mehr läuft.

Audio-Streaming

Mit den Befehlen WEB OPEN TCP STREAM und WEB TCP CLIENT STREAM kannst du zusammen mit dem Befehl PLAY STREAM ganz einfach eine grundlegende Internetradiofunktion einrichten. Das zeigt der Code unten, der das britische Programm ClassicFM empfängt. Hinweis: Für dieses Programm brauchst du einen VS1053-Audio-Codec.

```
Option escape
Option default none
' Anfrage für die Radioseite (ClassicFM) erstellen
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\nVerbindung: schließen\r\n\r\n"

'Erstelle einen Ringpuffer zum Lesen des Internet-Streams und
'Lese- und Schreibzeiger
Dim buff%(4095),w%,r%

' VS1053 einrichten und sagen, dass er aus dem Ringpuffer abspielen soll
Stream abspielen buff%(), r%, w%

' Internetradio-Website öffnen
WEB open tcp stream a$,80

' Anfrage zum Starten des Streams über den angegebenen Ringpuffer senden
WEB TCP CLIENT STREAM q$, buff%(), r%, w%

'Lehn dich zurück und hör zu
Do : Pause 500: Loop
```

Lange Zeichenfolgen

Lange Zeichenfolgen sind eine Reihe von Befehlen und Funktionen, mit denen MMBasic Zeichenfolgen unbegrenzter Länge bearbeiten kann. Sie sind besonders nützlich, wenn es um Daten geht, die über WLAN und das Internet gesendet werden. Standardzeichenfolgen 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 verfügbare RAM-Kapazität begrenzt sind.

Lange Zeichenfolgenvariablen

Variablen zum Speichern langer Zeichenfolgen müssen als Integer-Arrays definiert werden. Die Routinen für lange Zeichenfolgen speichern keine Zahlen in diesen Arrays, sondern verwenden sie nur als Speicherblöcke zum Speichern langer Zeichenfolgen.

Beim Erstellen dieser Arrays sollten sie als eindimensionale Integer-Arrays definiert werden, wobei die Anzahl der Elemente auf die Anzahl der Zeichen festgelegt wird, die für die maximale Zeichenfolgenlänge erforderlich sind, geteilt durch acht. 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 speichern 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 ist Null). Wenn diese Variablen an die Funktionen für lange Zeichenfolgen übergeben werden, sollten sie als Variablenname gefolgt von leeren Klammern eingegeben werden. 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 Zeichenfolge beträgt“ LLEN(longarg())
END SUB
```

Und so könnte man sie aufrufen:

```
MySub str1()
```

Befehle für lange Zeichenfolgen

Diese sind ausführlich in den Abschnitten „ *-Befehle* und *-Funktionen”* und „ ” dieses Handbuchs beschrieben. Die Befehle lauten:

LONGSTRING AES128 ENCRYPT/DECRYPT	Verschlüsselt oder entschlüsselt eine lange Zeichenfolge
LONGSTRING APPEND array%(), string$	Fügt eine normale Zeichenfolge an eine lange Zeichenfolge an
LONGSTRING BASE64 ENCODE/DECODE	Kodiert oder dekodiert eine lange Zeichenfolge mit Base 64
LONGSTRING CLEAR array%()	Löscht (d. h. setzt auf leer) eine lange Zeichenkette
LONGSTRING COPY dest%(), src%()	Kopiert eine lange Zeichenfolge
LONGSTRING CONCAT dest%(), src%()	Zwei lange Zeichenfolgen zusammenfügen
LONGSTRING LCASE array%()	Lange Zeichenfolge in Kleinbuchstaben umwandeln
LONGSTRING LEFT dest%(), src%(), nbr	Nimm die ersten nbr Zeichen aus einer langen Zeichenkette
LONGSTRING LOAD array%(), nbr, string$	Zeichen in eine lange Zeichenfolge kopieren
LONGSTRING MID dest%(), src%(), start, nbr	Hol dir Zeichen aus der Mitte einer langen Zeichenkette
LONGSTRING PRINT [#n,] src%() [;]	Druckt eine lange Zeichenkette
LONGSTRING REPLACE array%() , string$, start	Zeichen in einer langen Zeichenfolge ersetzen
LONGSTRING RESIZE addr%(), nbr	Länge einer langen Zeichenkette festlegen
LONGSTRING RIGHT dest%(), src%(), nbr	Die angegebene Anzahl von Zeichen aus einer langen Zeichenfolge holen

LONGSTRING SETBYTE addr%(), nbr, data	Ein Byte in einer langen Zeichenkette setzen
LONGSTRING TRIM array%(), nbr	Zeichen am Anfang einer langen Zeichenfolge entfernen
LONGSTRING UCASE array%()	Lange Zeichenfolge in Großbuchstaben umwandeln
LMID(array%(),start [,num])=string$	Text in eine lange Zeichenfolge einfügen/ersetzen

Funktionen für lange Zeichenfolgen

r = LGETBYTE(array%(), n)	Gibt den Wert eines Bytes in einer langen Zeichenfolge zurück
r$ = LGETSTR$(array%(), start, length)	Gibt einen Teil eines langen Strings als normalen String zurück.
r = LINSTR(array%(), search$ [,start] [,size])	Gibt die Position einer Zeichenfolge in einer langen Zeichenfolge zurück
r = LLEN(array%())	Gibt die Länge einer langen Zeichenfolge zurück.
r= LINPUT(array%(),fnbr,nbr)	Liest nbr Bytes aus einer Datei und gibt die Anzahl der gelesenen Bytes zurück
MATH(BASE64 ENCODE/DECODE)	Kodiert oder dekodiert Daten mit Base 64

MMBasic-Eigenschaften

Namenskonventionen

Bei Befehlsnamen, Funktionsnamen, Bezeichnungen, Variablennamen usw. wird nicht zwischen Groß- und Kleinschreibung unterschieden, sodass „Run" und „RUN" dasselbe bedeuten 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 ist beispielsweise „%", sodass eine automatisch erstellte Variable namens nbr% eine Ganzzahl ist. Es gibt drei Arten von Variablen:

1. Gleitkomma. Diese können eine Zahl 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 ist „!" und Gleitkomma ist die Standardeinstellung, wenn eine Variable ohne Suffix erstellt wird.
2. 64-Bit-Ganzzahl. Diese können Zahlen mit bis zu 19 Dezimalstellen ohne Genauigkeitsverlust speichern, aber keine Bruchteile (d. h. den Teil nach dem Dezimalpunkt). Das Suffix für eine Ganzzahl ist „%".
3. Zeichenfolgen. Diese speichern eine Folge von Zeichen (z. B. „Tom"). Das Suffix für eine Zeichenfolge ist das Symbol „$" (z. B. name$, s$ usw.). Zeichenfolgen können bis zu 255 Zeichen lang sein.

Variablennamen und Bezeichnungen können mit einem Buchstaben oder einem Unterstrich beginnen und beliebige Buchstaben oder Zahlen, den Punkt (.) und den Unterstrich (_) enthalten. Sie können bis zu 31 Zeichen lang sein. Ein Variablenname oder eine Bezeichnung darf nicht mit einem Befehl oder einer Funktion oder einem der folgenden Schlüsselwörter übereinstimmen: THEN, ELSE, TO, STEP, FOR, WHILE, UNTIL, MOD, NOT, AND, OR, XOR, AS. Beispielsweise ist step = 5 nicht zulässig.

Infos 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 Dezimalkonstante, mit &H für eine Hexadezimalzahl, mit &O für eine Oktalzahl oder mit &B für eine Binärzahl anfangen. Zum Beispiel ist &B1000 dasselbe wie die Dezimalkonstante 8. Konstanten, die mit &H, &O oder &B anfangen, werden immer als 64-Bit-Ganzzahlkonstanten behandelt.

Dezimalen können mit einem Minuszeichen (-) oder Pluszeichen (+) beginnen und mit „E" gefolgt von einer Exponentialzahl enden, um die Exponentialdarstellung anzuzeigen. Zum Beispiel ist 1,6E+4 dasselbe wie 16000. Wenn die Dezimalzahl einen Dezimalpunkt oder einen Exponenten enthält, wird sie als Gleitkommazahl behandelt, sonst als 64-Bit-Ganzzahl.

Zeichenfolgenkonstanten werden in doppelte Anführungszeichen (") gesetzt. Beispiel: „Hello World".

Implementierungsmerkmale

Maximales Programm – siehe Tabelle. Beachte, dass MMBasic das Programm beim Speichern im Flash-Speicher in Tokens zerlegt, sodass die endgültige Größe im Flash-Speicher von der Größe des Klartexts abweichen kann.

Die maximale Länge einer Befehlszeile beträgt 255 Zeichen.

Die maximale Länge eines Variablennamens oder einer Bezeichnung 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 ist 50.

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

Die maximale Anzahl von benutzerdefinierten Labels, Unterprogrammen und Funktionen (zusammen) – siehe Tabelle.

Die maximale Anzahl konfigurierbarer Interrupt-Pins beträgt 10.

Zahlen werden als Gleitkommazahlen mit doppelter Genauigkeit oder als 64-Bit-Ganzzahlen mit Vorzeichen 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, ist ± 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.

Die maximale Anzahl globaler Variablen und Konstanten findest du in der Tabelle.

Die maximale Anzahl lokaler Variablen – siehe Tabelle

Die maximale Anzahl von 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:

	Maximale Programmgröße	Maximale Frequenz	Maximale Anzahl von Unterprogrammen oder Funktionen	Maximale Array-Dimensionen	Standardanzahl globaler Variablen	Standardanzahl lokaler Variablen *
PicoMite RP2040	132 KB	420 MHz	256	6	256	240
PicoMiteUSB RP2040	132 KB	420 MHz	256	6	256	240
PicoMiteVGA RP2040	100 KB	378 MHz	256	6	240	240
PicoMiteVGAUSB RP2040	100 KB	378 MHz	256	6	240	240
WebMite RP2040	88 KB	252 MHz	256	6	240	240
PicoMite RP2350	324 KB	396 MHz	512	5	512	240
PicoMiteUSB RP2350	324 KB	396 MHz	512	5	512	240
PicoMiteVGA RP2350	184 KB	378 MHz	512	5	480	256
PicoMiteVGAUSB RP2350	184 KB	378 MHz	512	5	480	256
PicoMiteHDMI RP2350	176 KB	372 MHz	512	5	480	256
PicoMiteHDMIUSB RP2350	176 KB	372 MHz	512	5	480	256
WebMite RP2350	204 KB	252 MHz	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 hat viele Funktionen von Microsofts GW-BASIC. Es gibt zwar ein paar Unterschiede, die aus technischen und praktischen Gründen gemacht wurden, aber die meisten Standard-BASIC-Befehle und -Funktionen sind im Grunde gleich. Ein Online-Handbuch für GW-BASIC findest du unter http://www.antonis.de/qbebooks/gwbasman/index.html. Dort gibt's eine ausführlichere Beschreibung der Befehle und Funktionen.

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

Vordefinierte schreibgeschützte Variablen

Diese Variablen werden von MMBasic festgelegt und können vom laufenden Programm nicht geändert werden. Beachte, dass sie selbst keine Funktion haben, sondern ausgegeben oder einer Variablen zugewiesen werden müssen.

Beispiel:

```
> PRINT MM.VER
6.0002
>
```

oder in einem Programm:

```
Wenn MM.VER < 6.0002 Dann Fehlermeldung „Version 6.00.02 oder höher
erforderlich”
```

MM.VER	Gibt die Versionsnummer der Firmware als Fließkommazahl im Format aa.bbcc zurück, wobei aa die Hauptversionsnummer, bb die Nebenversionsnummer und cc die Revisionsnummer ist. Beispielsweise würde Version 6.03.00 den Wert 6.03 und Version 6.03.01 den Wert 6.0301 zurückgeben.
MM.ADDRESS$	NUR WEBMITE-VERSION 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 aktuellen Programm übergeben wurden, wenn ein MMBasic-Programm läuft. Details findest du unter den Befehlen RUN und *. Die Funktion gibt eine leere Zeichenfolge zurück, wenn Programme aus dem Editor oder mit der 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 physisches Display konfiguriert ist, sonst 0.
MM.ERRNO 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 Zeichenfolge, 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 Zeichenfolge zurückgesetzt.
MM.FLAGS	Gibt den Wert des Systemflags-Registers zurück
MM.FONTHEIGHT MM.FONTWIDTH	Gibt die Höhe oder Breite der aktuellen Schriftart in Pixeln zurück
MM.HRES MM.VRES	Ganzzahlen, die die aktuelle horizontale und vertikale Auflösung des VGA/HDMI-Videoausgangs oder des LCD-Bildschirms (falls konfiguriert) in Pixeln angeben.
MM.HEIGHT MM.WIDTH	Gibt die Anzahl der Zeichen über die physische Anzeige mit der aktuellen Schriftart oder die Anzahl der Zeichen unter der Anzeige mit der aktuellen Schriftart zurück.

MM.HPOS 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 Eingangsspannung wie Pico und Pico2 gibt MM.SUPPLY die Spannung zurück.
MM.INFO() MM.INFO$()	Diese beiden Versionen können austauschbar verwendet werden, aber für eine gute Programmierpraxis solltest du diejenige verwenden, die dem zurückgegebenen Datentyp entspricht.
MM.INFO$(AUTORUN)	Gibt die Einstellung des Befehls OPTION AUTORUN zurück
MM.INFO(ADC)	Gibt die Nummer des Puffers zurück, der gerade zum Lesen bereit ist, wenn ADC RUN (1 oder 2) verwendet wird. Gibt 0 zurück, wenn nichts bereit ist.
MM.INFO(ADC DMA)	Gibt „true" (1) zurück, wenn ADC DMA aktiv ist.
MM.INFO(BOOT)	Zeigt den Grund für den letzten Neustart des Pico an Gibt Folgendes zurück: Neustart – Das Gerät wurde mit CPU RESTART oder einem OPTION-Befehl neu gestartet. S/W Watchdog – Das Gerät wurde durch einen Software-Watchdog-Timeout neu gestartet. H/W Watchdog – Das Gerät wurde durch einen Hardware-Watchdog-Timeout neu gestartet. Firmware-Update – Das Gerät wurde nach einem Firmware-Update neu gestartet Einschalten – Das Gerät wurde eingeschaltet Reset-Schalter – Das Gerät wurde mit dem Reset-Schalter neu gestartet. Unbekannter Code &Hn – unbekannter Grund – bitte gib den Code und die Version an RP2040/2350
MM.INFO(BOOT COUNT)	Gibt an, wie oft der Pico seit der letzten Formatierung des Flash-Laufwerks neu gestartet wurde.
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 Zeichenfolge zurück.
MM.INFO$(LCDPANEL)	Gibt den Namen des konfigurierten LCD-Panels oder eine leere Zeichenfolge zurück.
MM.INFO(LCD320)	Gibt „true" zurück, wenn das Display mit dem Befehl OPTION LCD320 für eine Auflösung von 320 x 240 geeignet ist
MM.INFO$(SDCARD)	Gibt den Status der SD-Karte zurück. Gültige Rückgabewerte sind: DISABLED, NOT PRESENT, READY und UNUSED
MM.INFO$(CURRENT)	Gibt den Namen des aktuellen Programms zurück, wenn es aus einer Datei geladen wurde, oder NONE, wenn es nach einem NEW-, AUTOSAVE-, XMODEM- oder EDIT-Befehl aufgerufen wird.
MM.INFO$(PATH)	Gibt den Pfad des aktuellen Programms zurück oder NONE, wenn der Befehl nach einem NEW- oder EDIT-Befehl aufgerufen wird.
MM.INFO(DISK SIZE)	Gibt die Kapazität des Flash-Dateisystems oder der SD-Karte, je nachdem, welches Laufwerk gerade aktiv ist, in Byte zurück.
MM.INFO$(DRIVE)	Sagt dir, welches Laufwerk gerade aktiv ist: „A:" oder „B:".

MM.INFO(DATEI EXISTIERT fname$)	Gibt 1 zurück, wenn die angegebene Datei existiert, gibt -1 zurück, wenn fname$ ein Verzeichnis ist, ansonsten gibt es 0 zurück.
MM.INFO(EXISTS DIR Verzeichnisname$)	Gibt einen booleschen Wert zurück, der angibt, ob das angegebene Verzeichnis existiert.
MM.INFO(FREIER SPEICHERPLATZ)	Gibt den freien Speicherplatz auf dem Flash-Dateisystem oder der SD-Karte zurück, je nachdem, welches Laufwerk gerade aktiv ist.
MM.INFO(FILESIZE file$)	Gibt die Größe von file$ in Bytes zurück oder 0, wenn die Datei nicht gefunden wird.
MM.INFO$(MODIFIED file$)	Sagt dir, wann die Datei file$ geändert wurde, oder gibt eine leere Zeichenfolge zurück, wenn sie nicht gefunden wird.
MM.INFO$(SYSTEM I2C)	Gibt „I2C“, „I2C2“ oder „Nicht festgelegt“ zurück, je nach Status von OPTION SYSTEM I2C
MM.INFO(FCOLOUR)	Gibt die aktuelle Vordergrundfarbe zurück.
MM.INFO(BCOLOUR)	Gibt die aktuelle Hintergrundfarbe zurück.
MM.INFO(FONT)	Gibt die Nummer der gerade aktiven Schriftart zurück.
MM.INFO(FONT ADDRESS n)	Gibt die Adresse des Speicherplatzes mit der Adresse von FONT n zurück.
MM.INFO(FONT POINTER n)	Gibt einen ZEIGER auf den Anfang von FONT n im Speicher zurück.
MM.INFO(FONTHEIGHT) MM.INFO(FONTWIDTH)	Ganzzahlen, die die Höhe und Breite der aktuellen Schriftart (in Pixeln) angeben.
MM.INFO(FLASH)	Sagt dir, aus welchem Flash-Slot das Programm geladen wurde, falls das möglich ist.
MM.INFO(FLASH ADDRESS n)	Gibt die Adresse des Flash-Steckplatzes n zurück.
MM.INFO(HEAP)	Gibt die Menge an freiem MMbasic-Heap-Speicher zurück. Der MMBasic-Heap wird für Zeichenfolgen, Arrays und verschiedene andere temporäre und permanente Puffer (z. B. Audio) verwendet.
MM.INFO(HPOS) MM.INFO(VPOS)	Die aktuelle horizontale und vertikale Position (in Pixeln) nach dem letzten Grafik- oder Druckbefehl.
MM.INFO(ID)	Gibt die eindeutige ID des Pico zurück.
MM.INFO$(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$(MODBUFF-ADRESSE)	Gibt die Adresse im Speicher des Puffers zurück, der zum Speichern von MOD-Dateien verwendet wird.
MM.INFO$(OPTION Option)	

	Gibt den aktuellen Wert einer Reihe von Optionen zurück, die beeinflussen, wie ein Programm ausgeführt wird. „option" kann einer der folgenden Werte sein: AUTORUN, AUDIO, BASE, BREAK, CONSOLE, DEFAULT, EXPLICIT, KEYBOARD, ANGLE, HEIGHT, WIDTH, FLASH SIZE
MM.INFO$(PIN pinno)	Gibt den Status des E/A-Pins „pinno" zurück. Gültige Rückgabewerte sind: OFF, DIN, DOUT, AIN usw.
MM.INFO(PINNO GPnn)	Gibt die physikalische Pin-Nummer für eine bestimmte GP-Nummer zurück. GPnn kann eine Zeichenfolge ohne Anführungszeichen (GP01), eine Zeichenfolgenliteral („GP01") oder eine Zeichenfolgenvariable sein. D. h. A$ = „GP01": MM.INFO(PINNO A$).
MM.INFO(PIO RX DMA)	Zeigt an, ob der PIO RX DMA-Kanal beschäftigt ist
MM.INFO(PIO TX DMA)	Zeigt an, ob der PIO TX DMA-Kanal gerade beschäftigt ist
MM.INFO$(PLATTFORM)	Gibt die zuvor mit OPTION PLATFORM festgelegte Zeichenfolge zurück.
MM.INFO(PROGRAMM)	Gibt die Adresse im Speicher des aktuell laufenden Programms zurück.
MM.INFO(PS2)	Zeigt die letzte Rohmeldung an, die über die PS2-Schnittstelle empfangen wurde, wenn diese aktiviert ist.
MM.INFO(PWM-ZÄHLER)	Gibt die Anzahl der vom Chip unterstützten PWM-Kanäle zurück.
MM.INFO(PWM-EINSCHALTDauer C%, n%)	Gibt den aktuellen Arbeitszyklus in Taktzahlen des PWM-Kanals C%, N% zurück. Dabei ist N%=0 für A und 1 für B.
MM.INFO$(SOUND)	Gibt die aktuelle Aktivität am Audioausgang zurück (AUS, PAUSE, TON, WAV, FLAC, MP3, SOUND)
MM.INFO(SPI-GESCHWINDIGKEIT	Gibt die tatsächliche Geschwindigkeit des SYSTEM SPI zurück oder einen Fehler, wenn nicht eingestellt
MM.INFO(STACK)	Gibt den C-Stack-Zeiger zurück. Komplexer oder rekursiver Basic-Code kann zu dem Fehler „Stack overflow, expression too complex at depth %" führen. Das passiert, wenn der Stack unter &H 2003f800 liegt. Durch Überwachen des Stacks kann der Programmierer Vereinfachungen des Basic-Codes identifizieren, um den Fehler zu vermeiden.
MM.INFO$(SYSTEM I2C)	Gibt I2C, I2C2 oder NOT SET zurück, je nach Anwendbarkeit.
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-Taktrate läuft.
MM.INFO(FLIESENHÖHE) MM.INFO(SPUR)	<u>NUR VGA- UND HDMI-VERSIONEN</u> Gibt die aktuelle Einstellung der Kachelhöhe zurück.
MM.INFO$(TOUCH)	Gibt den Namen der FLAC-, MP3-, WAV- oder MIDI-Datei zurück, die gerade über den Audioausgang abgespielt wird.
MM.INFO(USB n)	Zeigt den Status des Touch-Controllers an. Mögliche Werte sind: „Deaktiviert", „Nicht kalibriert" und „Bereit".

MM.INFO(USB-VID n) MM.INFO(USB-PID n) MM.INFO(VARCNT) MM.INFO$(LINE) MM.INFO(UPTIME) MM.INFO(GÜLTIGE CPUSPEED-Geschwindigkeit %) MM.INFO(VERSION) MM.INFO(WRITEBUFF) MM.INFO(TCP PORT) MM.INFO(UDP-PORT)	Gibt den Gerätecode für jedes an Kanal „n“ angeschlossene Gerät zurück, der eine Zahl zwischen 1 und 4 ist. Der zurückgegebene Gerätecode kann sein: 0 = nicht in Gebrauch, 1 = Tastatur, 2 = Maus, 128 = PS4, 129 = PS3, 130 = SNES/Generisch 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 zwei oder mehr Tastaturen oder Mäuse angeschlossen sind oder drei oder mehr Gamepads, werden die zusätzlichen Geräte dem höchsten verfügbaren Kanal zugewiesen. Gibt die VID des USB-Geräts auf Kanal n zurück Gibt die PID des USB-Geräts auf Kanal n zurück Gibt die Anzahl der im MMBasic-Programm verwendeten Variablen zurück. Gibt die aktuelle Zeilennummer als Zeichenfolge zurück. LIBRARY wird zurückgegeben, wenn es sich in der Bibliothek befindet, und UNKNOWN, wenn es nicht in einem Programm ist. Hilft bei der Diagnose während der Komponententests. Gibt die Zeit seit dem Start in Sekunden als Fließkommazahl zurück. Gibt 1 zurück, wenn „speed%” für OPTION CPUSPEED „speed%” gültig ist Die Versionsnummer der Firmware (MM.VER konvertiert in diese). Gibt die Adresse im Speicher des aktuellen Puffers zurück, der für Zeichenbefehle verwendet wird. <u>NUR WEBMITE</u> Gibt den als Server eingestellten TCP-Port zurück oder 0, wenn keiner eingestellt ist Gibt den als Server eingestellten UDP-Port zurück oder 0, wenn keiner eingestellt ist.
MM.INFO(TCPIP-STATUS) MM.INFO(WIFI-STATUS)	<u>NUR WEBMITE</u> Gibt den TCPIP-Status der Verbindung zurück Gibt den WIFI-Status der Verbindung zurück. Gültige Rückgaben sind: 0 WLAN ist ausgefallen 1 Mit WLAN verbunden 2 Mit WLAN verbunden, aber keine IP-Adresse (nur TCPIP STATUS) 3 Mit WLAN verbunden und hat eine IP-Adresse (nur TCPIP-STATUS) -1 Verbindung ist fehlgeschlagen -2 Keine passende SSID gefunden (könnte außerhalb der Reichweite oder nicht verfügbar sein) -3 Authentifizierung fehlgeschlagen
MM.MESSAGE$	<u>NUR WEBMITE</u> Gibt den Inhalt des zuletzt empfangenen UDP-Datagramms oder des zuletzt empfangenen MQTT-Pakets zurück

MM.TOPIC$	Nur WEBMITE Gibt das Thema des zuletzt empfangenen MQTT-Pakets zurück
MM.ADDRESS$	Nur WEBMITE 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 ganzzahlige Variable gesetzt, um das Ergebnis der Operation anzuzeigen: 0 = Gerät nicht gefunden, 1 = Gerät gefunden, 2 = Zeitüberschreitung beim Gerät.
MM.I2C	Nach einem I2C-Schreib- oder Lesebefehl wird diese Ganzzahlvariable gesetzt, um das Ergebnis der Operation wie folgt anzuzeigen: 0 = Der Befehl wurde ohne Fehler ausgeführt. 1 = NACK-Antwort erhalten 2 = Befehl abgelaufen
MM.PERSISTENT	Gibt einen Wert zurück, der mit dem Befehl SAVE PERSISTENT gespeichert wurde.
MM.PS2	Gibt den letzten Code zurück, der über die PS2-Schnittstelle empfangen wurde, wenn diese aktiviert ist.
MM.WATCHDOG	Eine ganze Zahl, die wahr ist, wenn MMBasic aufgrund eines Watchdog-Timeouts neu gestartet wurde (siehe Befehl WATCHDOG), andernfalls falsch.

Optionen

Diese Tabelle listet die verschiedenen Optionsbefehle auf, mit denen MMBasic konfiguriert und seine Funktionsweise geändert werden kann. Optionen, die als dauerhaft gekennzeichnet sind, werden in einem 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, wodurch die USB-Konsolenschnittstelle zurückgesetzt wird. Das Programm im Speicher geht nicht verloren, da es in einem nichtflüchtigen Flash-Speicher gespeichert ist.

	Permanent?	
OPTION WINKEL RADIANE | GRAD		Dieser Befehl wechselt die trigonometrischen Funktionen zwischen Grad und Radiant um. Gilt für SIN, COS, TAN, ATN, ATAN2, MATH ATAN3, ACOS, ASIN
OPTION AUDIO PWMnApin, PWMnBpin oder OPTION AUDIO DISABLE	✓	Konfiguriert einen der PWM-Kanäle als Audioausgang. „PWMnApin" ist der linke Audiokanal, „PWMnBpin" der rechte. Beide Pins müssen zum selben Audiokanal gehören. Beispiel: OPTION AUDIO GP18, GP19 würde PWM1A und PWM1B an den Pins 24 und 25 verwenden. Diese Option verhindert die Verwendung dieser Pins im BASIC-Programm. Der Audioausgang wird mit PWM erzeugt, daher ist ein Tiefpassfilter am Ausgang nötig. Der Audioausgang des Raspberry Pi Pico ist ziemlich verrauscht. Mit OPTION POWER und/oder einer Stromversorgung über einen separaten 3,3-V-Linearregler kann man das reduzieren. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.
OPTION AUDIO SPI CSpin, CLKpin, MOSIpin oder OPTION AUDIO DISABLE	✓	Konfiguriert den Audioausgang so, dass er an einen MCP48n2-DAC geleitet wird, der an die angegebenen Pins angeschlossen ist. Der LDAC-Pin am DAC sollte mit GND verbunden werden.
OPTION AUDIO VS1053 CLKpin, MOSIpin, MISOpin, XCSpin, XDCSpin, DREQpin, XRSTpin oder OPTION AUDIO DISABLE	✓	Konfiguriert den Audioausgang so, dass er an einen VS1053-CODEC weitergeleitet wird. Damit kannst du neben den anderen unterstützten Formaten auch MP3- und MIDI-Dateien abspielen und es wird auch die Echtzeit-MIDI-Ausgabe unterstützt. Weitere Infos findest du unter dem Befehl PLAY.
OPTION AUDIO I2S BCLKpin, DINpin oder OPTION AUDIO DISABLE	✓	Konfiguriert den Audioausgang so, dass er an einen I2S-DAC weitergeleitet wird, der an die angegebenen Pins angeschlossen ist. Der LRCK-Pin am DAC sollte mit dem nächsten aufeinanderfolgenden GPIO-Pin zu BCLKpin verbunden werden.

OPTION AUTOREFRESH OFF | ON		Schwarz-Weiß-Displays können nur bildschirmweise aktualisiert werden. Mit OPTION AUTOREFRESH OFF/ON kannst du steuern, ob ein Schreibbefehl das Display sofort aktualisiert oder nicht. Wenn AUTOREFRESH auf OFF steht, kann der Befehl REFRESH zum Auslösen des Schreibvorgangs verwendet werden. Dies gilt für die folgenden Displays: N5110, SSD1306I2C, SSD1306I2C32, SSD1306SPI und ST7920
OPTION AUTORUN ON [,NORESET] oder OPTION AUTORUN n [,NORESET] oder OPTION AUTORUN OFF	✓	Sagt MMBasic, dass es beim Einschalten oder Neustart automatisch ein Programm starten soll. ON führt dazu, dass das aktuelle Programm im Programmspeicher ausgeführt wird. Wenn du „n“ angibst, wird der entsprechende Speicherplatz im Flash-Speicher ausgeführt. „n“ muss zwischen 1 und 3 liegen. Wenn du den optionalen Parameter „NORESET” angibst, bleibt AUTORUN auch dann aktiv, wenn das Programm einen Systemfehler verursacht (standardmäßig führt das dazu, dass die Firmware alle OPTION AUTORUN-Einstellungen löscht). OFF deaktiviert die Autorun-Option und ist die Standardeinstellung für ein neues Programm. Durch Drücken der Unterbrechungstaste (standardmäßig STRG-C) auf der Konsole wird das laufende Programm unterbrochen und die Eingabeaufforderung wieder angezeigt.
OPTION BASE 0 | 1		Legt den niedrigsten Wert für Array-Indizes auf entweder 0 oder 1 fest. Dies muss vor der Deklaration von Arrays verwendet werden und wird beim Einschalten auf den Standardwert 0 zurückgesetzt.
OPTION BAUDRATE nn		Setzt die Baudrate der seriellen Konsole (falls konfiguriert).
OPTION BREAK nn		Setz den Wert der Break-Taste auf den ASCII-Wert „nn”. Diese Taste wird benutzt, um ein laufendes Programm zu unterbrechen. Der Wert der Break-Taste wird beim Einschalten und beim Ausführen eines Programms auf die Taste STRG-C gesetzt, kann aber mit diesem Befehl in einem Programm auf eine beliebige Tastaturtaste geändert werden (z. B. setzt OPTION BREAK 4 die Break-Taste auf die Taste STRG-D). Wenn du diese Option auf Null setzt, wird die Break-Funktion komplett deaktiviert.
OPTION CASE LOWER | UPPER | TITLE	✓	Ändere die Groß-/Kleinschreibung für Befehls- und Funktionsnamen, wenn du den Befehl LIST benutzt. Standardmäßig ist TITLE eingestellt, aber mit OPTION CASE UPPER kannst du den alten Standard von MMBasic wiederherstellen.
OPTION COLOURCODE ON oder OPTION COLOURCODE OFF	✓	Aktiviert oder deaktiviert die Farbcodierung für die Ausgabe des Editors. Schlüsselwörter werden cyanfarben, Kommentare gelb usw. dargestellt. Die Standardeinstellung ist OFF. Das Schlüsselwort COLORCODE (US-Schreibweise) kann ebenfalls verwendet werden. Das funktioniert bei VGA/HDMI-Video und der seriellen Konsole mit einem Terminalemulator mit VT100-Emulation (z. B. Tera Term). Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.

OPTION CONSOLE-Ausgabe		Legt fest, wo Druckanweisungen ausgegeben 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 Fortsetzungszeilen im Editor und mit dem Befehl LIST. Zeilenfortsetzungen werden durch ein Leerzeichen gefolgt von einem Unterstrich am Ende einer Zeile angezeigt. Wenn diese Option aktiviert ist, teilt der Editor die Zeilen beim Lesen aus der Datei automatisch und fügt die erforderlichen Fortsetzungszeichen hinzu. Beim Beenden des Editors werden die Fortsetzungszeichen vor dem Speichern entfernt. Im Editor kann der Benutzer lange Zeilen erstellen, indem er eigene Fortsetzungszeichen hinzufügt. Dies erleichtert die Verwendung eines kleinen Bildschirms als Konsole erheblich.
OPTION CPUSPEED Geschwindigkeit	✓	<u>NICHT BEI HDMI- ODER VGA-VERSIONEN</u> Ändert die CPU-Taktrate. „speed" ist die CPU-Taktfrequenz in KHz im Bereich von 48000 bis 396000. Geschwindigkeiten über 200 MHz (150 MHz für den RP2350) gelten als Übertaktung, da dies die angegebene Höchstgeschwindigkeit des Standard-Raspberry Pi Pico ist. Die Standardgeschwindigkeit beträgt 200000 für den RP2040 und 150000 für den RP2350. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.
OPTION COUNT pin1, pin2, pin3, pin4	✓	Legt fest, welche Pins als Zähleingänge verwendet werden sollen. Standardmäßig sind dies GP6, GP7, GP8 und GP9. Der Befehl SETPIN definiert den Zählermodus. 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. Wenn OPTION DEFAULT NONE verwendet wird, muss der Typ aller Variablen explizit definiert sein, sonst kommt es zum Fehler „Variablentyp nicht angegeben". Wenn ein Programm ausgeführt wird, ist der Standardwert auf FLOAT gesetzt, um die Kompatibilität mit Microsoft BASIC und früheren Versionen von MMBasic zu gewährleisten.
OPTION DEFAULT COLOURS foreground [,background]	✓	Legt die Standardfarben für den Vordergrund und Hintergrund sowohl für den Monochrom- als auch für den Farbmodus fest. Die Farbe muss eine der folgenden sein: Weiß, Gelb, Lila, Braun, Fuchsia, Rostrot, Magenta, Rot, Cyan, Grün, Cerulean, Mittelgrün, Kobalt, Myrte, Blau und Schwarz. Ein numerischer Wert kann nicht verwendet werden. Der Standardwert ist Weiß, Schwarz. Wenn der Hintergrund weggelassen wird, ist er standardmäßig schwarz.
OPTION DEFAULT MODE n	✓	Damit wird der Standardanzeigemodus beim Booten festgelegt. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.
OPTION DISK SAVE fname$ OPTION DISK LOAD fname	✓	Mit diesen Befehlen kann der Benutzer den kompletten Satz der definierten Optionen in einer Datei speichern und wiederherstellen. Die Datei kann dann mit 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]	✓	Legt die Eigenschaften des für die Konsole verwendeten Anzeigeterminals fest. Sowohl der Befehl LIST als auch der Befehl EDIT benötigen diese Informationen, um den Text für die Anzeige korrekt zu formatieren. „lines“ ist die Anzahl der Zeilen auf dem Display und „chars“ ist die Breite des Displays in Zeichen. Die Standardeinstellung ist 24 Zeilen x 80 Zeichen. Wenn du diese Option änderst, bleibt sie auch nach dem Ausschalten gespeichert. Die Maximalwerte sind 100 Zeilen und 240 Zeichen. 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 (wenn die Option in den Terminaleinstellungen aktiviert ist). **Diese Option ist nicht verfügbar, wenn ein LCD-Display als Konsole verwendet wird.**
OPTION ESCAPE		Aktiviert die Möglichkeit, Escape-Sequenzen in String-Konstanten einzufügen. Siehe Abschnitt *„Sonderzeichen in Strings*“.
OPTION EXPLICIT		Wenn du diesen Befehl am Anfang eines Programms einfügst, musst du jede Variable explizit mit den Befehlen DIM, LOCAL oder STATIC deklarieren, bevor du sie im Programm verwenden kannst. Diese Option ist standardmäßig deaktiviert, wenn ein Programm ausgeführt wird. Wenn sie verwendet wird, muss sie angegeben werden, bevor Variablen verwendet werden.
OPTION FAST AUDIO ON| OFF		Bei Verwendung des Befehls PLAY SOUND können Änderungen an Sounds, Lautstärken 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 herunterfährt und dann wieder hochfährt. Dies verbessert die Audioausgabe erheblich, führt jedoch zu einer kurzen Verzögerung des Befehls PLAY SOUND (im schlimmsten Fall 3 ms). Diese Verzögerung kann durch die Verwendung von OPTION FAST AUDIO ON in einem Programm vermieden werden. Die hörbaren Klicks können dann wieder auftreten, aber das liegt im Ermessen des Programmierers. Dies ist eine temporäre Option, die bei jeder Ausführung eines Programms auf OFF zurückgesetzt wird.
OPTION FNKey string	✓	Definiert die Zeichenfolge, die generiert wird, wenn eine Funktionstaste an der Eingabeaufforderung gedrückt wird. „FNKey” kann F1 und F5 bis F9 sein. Durch Setzen von string$ auf "" wird die gespeicherte Funktion gelöscht und die ursprüngliche Funktion der Taste wiederhergestellt. Beispiel: OPTION F8 „RUN “+chr$(34)+”myprog” +chr$(34)+chr$(13)+chr$(10). Dieser Befehl muss in der Eingabeaufforderung ausgeführt werden (nicht in einem Programm).

OPTION GPS	✓	Konfiguriert den PicoMite für die Verbindung mit einem GPS-Modul. Diese Integration ermöglicht es dem PicoMite, NMEA-Datenströme automatisch im Hintergrund zu analysieren und Standort-, Zeit- und Geschwindigkeitsdaten über die Funktion `GPS()` für das Benutzerprogramm verfügbar zu machen. Das wird ausführlich in der Datei *Option_GPS_User_Manual.pdf* beschrieben, die im ZIP-Archiv mit der Firmware enthalten ist.
OPTION HEARTBEAT ON/OFF [HEARTBEATpin]	✓	Aktiviert oder deaktiviert die Ausgabe der Heartbeat-LED. Beim Pico-W wird der Heartbeat über einen Pin gesteuert, der vom CWY43-Chip kontrolliert wird. NICHT WEBMITE-VERSION Standardmäßig ist der Heartbeat bei RP2350A-Chips auf GP25 aktiviert. Wenn er deaktiviert ist, kann das Programm die LED über GP25 steuern. Bei RP2350B-Chips ist der Heartbeat nicht aktiviert. Wenn der Heartbeat deaktiviert ist, kann der Befehl sowohl zum Aktivieren als auch zum optionalen Festlegen des zu verwendenden Pins (standardmäßig GP25) verwendet werden.
OPTION HDMI-PINS clockpositivepin, d0positivepin, d1positivepin, d2positivepin	✓	NUR HDMI-VERSION Legen Sie die für den HDMI-Videoausgang verwendeten I/O-Pins fest. Dies ist nur bei nicht standardmäßigen PCB-Layouts erforderlich. Die positiven HDMI-Signalpins werden nach „nbr" unten eingestellt. Gültige Werte sind 0-7, und die Pins dürfen sich für jeden Kanal nicht überschneiden. Wenn „nbr" eine gerade Zahl ist, ist der negative Ausgang auf dem physischen Pin+1, wenn „nbr" eine ungerade Zahl ist, ist er auf dem physischen Pin-1. nbr — HSTX Nbr — Physikalischer Pin 0 — HSTX0 — GP12 1 — HSTX1 — GP13 2 — HSTX2 — GP14 3 — HSTX3 — GP15 4 — HSTX4 — GP16 5 — HSTX5 — GP17 6 — HSTX6 — GP18 7 — HSTX7 — GP19 Die Standardeinstellung ist: OPTION HDMI-PINS 2, 0, 6, 4 Das heißt: CK+ und CK- sind GP14 und GP15 zugewiesen D0+ und D0- sind GP12 und GP13 zugeordnet D1+ und D1- sind GP18 und GP19 zugewiesen D2+ und D2- sind GP16 und GP17 zugewiesen
OPTIONSTASTATUR nn [,capslock] [,numlock] [,repeatstart] [,repeatrate] oder OPTION KEYBOARD DISABLE	✓	Konfiguriere eine Tastatur. Diese kann für die Konsoleneingabe verwendet werden, und alle eingegebenen Zeichen sind über alle Befehle verfügbar, die von der Konsole gelesen werden (seriell über USB). „nn" ist ein zweistelliger Code, der das Tastaturlayout angibt. 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 die Befehlszeile eingegeben werden und führt zu

		einem Neustart. Diese Einstellung kann mit folgendem Befehl zurückgesetzt werden: OPTION KEYBOARD DISABLE Die optionalen Parameter „capslock" und „numlock" sind True/False-Ganzzahlen, die den Anfangszustand der Tastatur festlegen (Standard ist 0 und 1). Die optionalen Parameter „repeatstart" und „repeatrate" legen die Zeit für die erste Wiederholung einer gedrückten Taste und nachfolgende Wiederholungen fest. Für eine USB-Tastatur sind sie 100 bis 2000 ms und 25 bis 2000 ms. Bei einer PS2-Tastatur liegen sie zwischen 0 und 3, was 250 ms, 500 ms, 750 ms und 1000 ms bedeutet (Standardwert ist 1), und zwischen 0 und 31, was 33 ms bis 500 ms gemäß der PS2-Tastatur-Spezifikation bedeutet (Standardwert ist 12 oder 100 ms).
OPTIONSTASTATUR I2C	✓	Konfiguriert die Unterstützung für die Solderparty bbq20 mini I2C-Tastatur. Hinweis: OPTION SYSTEM I2C muss vor der Ausführung dieses Befehls eingestellt werden.
OPTION KEYBOARD PINS clockpin, datapin	✓	Ermöglicht dem Benutzer die Auswahl der Pins, die für den Anschluss einer PS2-Tastatur verwendet werden sollen. Der Standardwert ist Pin 11 (GP8) und Pin 12 (GP9). Die PS2-Tastatur muss deaktiviert sein (OPTION KEYBOARD DISABLE).
OPTION KEYBOARD REPEAT repeatstart , repeatrate	✓	<u>NUR USB-TASTATUR</u> Legt die Zeit für die erste Wiederholung einer gedrückten Taste (100–2000) und nachfolgende Wiederholungen (25–2000) in Millisekunden fest.
OPTION LCD-PINS clkpin, mosipin, misopin	✓	Die Firmware unterstützt die Trennung der SPI-Pins, die zum Ansteuern eines LCD-Displays verwendet werden, von denen, die für Touch und/oder die SD-Karte verwendet werden. Durch diese Trennung kann die Firmware den zweiten Prozessor für die Arbeit mit dem Display nutzen, ohne dass andere Funktionen beeinträchtigt werden. Das Einstellen der LCD-SPI-Pins ist eine Voraussetzung für die Verwendung von gepufferten Treibern 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 sind, müssen diese auf einem anderen SPI-Kanal liegen.
OPTION LCDPANEL OPTION LCDPANEL VIRTUAL_C oder OPTION LCDPANEL VIRTUAL_M OPTION LCDPANEL-Optionen	✓	<u>NICHT VGA- ODER HDMI-VERSIONEN</u> Konfiguriert ein LCD-Panel bei Versionen, die ein angeschlossenes LCD unterstützen. Konfiguriert ein virtuelles LCD-Panel ohne physisch angeschlossenes Panel. VIRTUAL _C = Farbe, 4 Bit, 320 x 240 VIRTUAL _M = Monochrom, 640 x 480 Mit dieser Funktion kann ein Programm grafische Bilder auf diesem virtuellen Panel zeichnen und sie dann als BMP-Datei speichern. Nützlich zum Erstellen eines Grafikbildes für den Export ohne angeschlossenes Display. Konfiguriert die PicoMite-Firmware für die Verwendung mit einem angeschlossenen LCD-Panel.

oder OPTION LCDPANEL DISABLE		Schau dir das Kapitel *„LCD-Anzeigen”* an, um mehr zu erfahren. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.
OPTION LCDPANEL CONSOLE [Schriftart [, fc [, bc [, blight]]] [,NOSCROLL] oder OPTION LCDPANEL NOCONSOLE		Konfiguriert das LCD-Display für die Verwendung als Konsolenausgabe. Das LCD muss transparenten Text unterstützen (z. B. die Controller SSD1963_x, ILI9341 oder ST7789_320). „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. Der optionale Befehl NOSCROLL ändert die Firmware so, dass bei der Ausgabe auf die letzte Zeile des Displays das Display nicht scrollt, sondern gelöscht wird und die Ausgabe oben auf dem Display fortgesetzt wird. Dadurch können Displays, die kein Lesen unterstützen, als Konsolengerät verwendet werden, und: Beachte, dass bei anderen Displays als dem SSD1963 das Scrollen für jede Konsolenausgabe sehr langsam ist, daher wird empfohlen, für diese Displays die Option NOSCROLL zu verwenden. Diese Einstellung wird im Flash gespeichert und beim Start automatisch angewendet. Um sie zu deaktivieren, benutze den Befehl OPTION LCDPANEL NOCONSOLE. Dieser Befehl muss an der Eingabeaufforderung ausgeführt werden.
OPTION LCDPANEL USER hres, vres		Konfiguriert einen vom Benutzer geschriebenen Anzeigetreiber in MMBasic. Eine Beschreibung zum Schreiben des Treibers findest du in der Datei „User Display Driver.txt” in der PicoMite-Firmware-Distribution.
OPTION LCDPANEL CONSOLE [Schriftart [, fc [,bc]] oder OPTION LCDPANEL NOCONSOLE	✓	<u>NUR VGA- UND HDMI-VERSIONEN</u> Ändert die Standardschriftart, die auf dem VGA- oder HDMI-Display verwendet wird. „fc” ist die Vordergrundfarbe und „bc” ist die Hintergrundfarbe. 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, verwenden Sie den Befehl OPTION CONSOLE. Bei SSD1963-basierten Displays im Querformat und SPI-Displays im Hochformat nutzt die Firmware H/W-Scrolling, um die Leistung der Displaykonsole zu verbessern.
OPTION LCD320 EIN/AUS		<u>NICHT VGA- ODER HDMI-VERSIONEN</u> Damit werden 16-Bit-LCD-Displays im 320x240-Modus aktiviert oder deaktiviert, sodass beispielsweise Spiele auf diesen größeren LCD-Displays möglich sind. Bei 800x480-Displays wird das 320x240-Bild um den Faktor 2 skaliert und nimmt den Bildschirmbereich 80,0 bis 719,479 ein. Bei 480x272-Displays wird das 320x240-Bild in einem Fenster angezeigt und nimmt den Bildschirmbereich 80,16 bis 399,255 ein.
OPTION LEGACY ON oder OPTION LEGACY OFF		Damit schaltest du den Kompatibilitätsmodus mit den Grafikbefehlen ein oder aus, die im ursprünglichen Colour Maximite verwendet wurden. Die Befehle COLOUR, LINE, CIRCLE und PIXEL verwenden die alte Syntax, und alle Zeichenbefehle akzeptieren Farben im Bereich von 0 bis

		7. Hinweise: • Schlüsselwörter wie RED, BLUE usw. sind nicht implementiert und sollten daher bei Bedarf als Konstanten definiert werden. • Die Syntax der Legacy-Befehle findest du im Colour Maximite MMBasic Language Manual. Du kannst es unter https://geoffg.net/OriginalColourMaximite.html runterladen.
OPTIONSLISTE		Hier werden die Einstellungen aller Optionen aufgelistet, die von ihrer Standardeinstellung geändert wurden und dauerhaft sind. OPTION LIST zeigt auch 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 gesamten Variablen (MM.INFO(MAX VARS)) fest, die als lokale Variablen zugewiesen werden sollen. Im Allgemeinen sollte das nicht nötig sein, aber wenn du zum Beispiel viele globale Konstanten zuweisen möchtest und nicht viele lokale Variablen benötigst, kannst du damit das Gleichgewicht ändern. Dies ist eine temporäre Option, die bis zum Zurücksetzen des Prozessors bestehen bleibt. Sie kann nicht aufgerufen werden, wenn bereits Variablen definiert wurden.
OPTION MILLISECONDS ON|OFF		Damit wird die Ausgabe von Millisekunden in der Funktion TIME$ aktiviert oder deaktiviert. D. h. HH:MM:SS.mmm Der Millisekundenzähler wird auf Null gesetzt, wenn 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 diese Option aktiviert ist, wird ein Mod-Puffer mit einer Größe von 128 KB erstellt. Dies kann mit „sizeinK" überschrieben werden. Beachte, dass diese Option einen Teil des Flash-Dateisystems reserviert (d. h. das Flash-Dateisystem wird verkleinert). Standardmäßig deaktiviert. Hinweis: Diese Option ist bei einem RP2350 mit aktiviertem PSRAM nicht erforderlich. In diesem Fall wird die MOD-Datei in den Speicherplatz im PSRAM geladen.
OPTION MOUSE CLKpin, DATApin OPTION MAUS DEAKTIVIEREN	✓ ✓	NUR FÜR NICHT-USB-FIRMWARE Stell die Pins so ein, dass sie für den Anschluss einer PS2-Maus genutzt werden können. Mit diesem Befehl wird die Maus beim Booten automatisch konfiguriert und du kannst Interrupts einrichten und Werte lesen, ohne dass du dafür extra Befehle brauchst. Das ist anders als bei MOUSE OPEN, wo die Maus nur verbunden wird, wenn das Programm läuft. Die PS2-Maus MUSS deaktiviert sein. Deaktiviert die automatische Verbindung zu einer PS2-Maus und gibt die Pins für die normale Nutzung frei.
OPTION MOUSE SENSITIVITY f!	✓	NUR USB-FIRMWARE Standardmäßig betreibt die Firmware eine USB-Maus im Boot-Modus. Das heißt, sie gibt 8-Bit-X- und Y-Positionen und den Status der drei Standardtasten zurück. Durch Einstellen von OPTION MOUSE

		SENSITIVITY weist die Firmware die Maus an, mit ihrer vollen Leistungsfähigkeit zu arbeiten. Das heißt, sie versucht, den gesamten von der Maus empfangenen USB-Bericht zu interpretieren, einschließlich der Radposition, aller Tasten und der 8-, 12- oder 16-Bit-x/y-Positionsinformationen. Durch Einstellen von „f$“ wird der vollständige Mausbericht aktiviert und die x/y-Positionen werden um „f!“ skaliert. Die Firmware kann nicht alle Maustypen unterstützen. Wenn dies bei einer bestimmten Maus zu Problemen führt, setze sie mit OPTION MOUSE SENSITIVITY 0 auf den Startmodus zurück. Die Aktivierung dieser Option mit dem Wert 1,0 bei Verwendung einer standardmäßigen Microsoft Basic Optical-Maus 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 Standardprüfung auf Unterbrechungen und Strg-C am Ende jedes Befehls deaktiviert. Wenn du ihn auf ON setzt, können zeitkritische Prozesse ohne Unterbrechungsrisiko ausgeführt werden. Der Befehl sollte aber vorsichtig verwendet werden, da das Programm sonst nur mit einem H/W-Reset gestoppt werden kann.
OPTION PICO EIN/AUS	✓	ALLE VERSIONEN AUSSER WEBMITE Wenn dieser Befehl auf AUS gesetzt ist, sind die Pins GP23, GP24 und GP29 nicht für die normale Pico-Verwendung eingerichtet und sofort verfügbar. Standardmäßig ist er für RP2350A und RP2040 auf EIN und für RP2350B auf AUS gesetzt.
OPTION PIN-Nr.		Stell „nbr” als PIN (Persönliche Identifikationsnummer) für den Zugriff auf die Konsolenaufforderung ein. „nbr” kann eine beliebige Zahl größer als Null mit bis zu acht Stellen sein. Wenn ein laufendes Programm aus irgendeinem Grund versucht, zur Befehlszeile zurückzukehren, fragt MMBasic diese Nummer ab, bevor die Eingabeaufforderung angezeigt wird. Das ist eine Sicherheitsfunktion, denn ohne Zugriff auf die Befehlszeile kann ein Eindringling weder das Programm im Speicher auflisten oder ändern noch die Funktionsweise von MMBasic in irgendeiner Weise modifizieren. Um diese Funktion zu deaktivieren, gibst du die PIN-Nummer Null ein (d. h. OPTION PIN 0). Eine dauerhafte Sperre kann durch Eingabe von 99999999 als PIN-Nummer aktiviert werden. Wenn eine dauerhafte Sperre aktiviert ist oder die PIN-Nummer verloren geht, kann das Problem nur durch erneutes Laden der PicoMite-Firmware behoben werden. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.
OPTION PLATFORM name$	✓	Ermöglicht es einem Benutzer, eine bestimmte Hardwarekonfiguration zu identifizieren, die dann in Programmen zur Steuerung des Programmablaufs verwendet werden kann. „name$” kann bis zu 31 Zeichen lang sein. Dies ist eine dauerhafte Option. MM.INFO$(PLATFORM) gibt diese Zeichenfolge zurück. Das kann zum Beispiel für eine bestimmte Hardware-Konfiguration verwendet werden: `OPTION PLATFORM „GameMite”` Dann können Programme, die auf dieser oder anderen Plattformen laufen, Folgendes verwenden:

		`IF MM.INFO$(PLATFORM) = "GameMite" THEN …`
OPTION POWER PFM | PWM	✓	Ändert den Betrieb des 3,3-V-Schaltnetzteils. Standardmäßig läuft das im PFM-Modus. PWM sorgt für weniger Geräusche, ist aber nicht so energieeffizient. Denk dran, dass das System bei hoher Belastung unabhängig von dieser Einstellung im PWM-Modus läuft.
OPTION PSRAM PIN n oder OPTION PSRAM PIN DISABLE	✓	Aktiviert/deaktiviert die PSRAM-Unterstützung. „n" ist der PSRAM-Chip-Select-Pin (CS) und kann GP0, GP8, GP19 oder GP47 sein. Normalerweise wird GP47 für Pimoroni-Boards verwendet. Standardmäßig ist die Option deaktiviert. Nach dem Einschalten ist der Inhalt des PSRAM unbestimmt. Lösche ihn vor der Verwendung mit RAM ERASE.
OPTION RESET	✓	Setzt alle gespeicherten Optionen auf ihre Standardwerte zurück. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.
OPTION RESET cfg oder OPTION RESET LIST	✓	Setzt alle Optionen für die Konfiguration „cfg" auf ihre Standardwerte zurück. OPTION RESET LIST zeigt alle verfügbaren Konfigurationen an. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.
OPTION RESOLUTION nn [,cpuspeedinKhz]		<u>NUR FÜR HDMI- UND VGA-VERSIONEN</u> Bei Firmware mit HDMI-Video stellst du die Videoauflösung auf „nn" ein. Dabei ist „nn": 640x480 oder 640 720x400 oder 720 800x600 oder 800 (nur RP2350) 848x480 oder 848 (nur RP2350) 1280x720 oder 1280 (nur HDMI) 1024 x 768 oder 1024 (nur HDMI) Für 640x480 kann die Bildwiederholfrequenz auf 60 Hz (252 MHz oder 378 MHz) oder 75 Hz (315 MHz) eingestellt werden, indem man „cpuspeedinKHz" an den Befehl hängt (also 252000, 378000 oder 315000). Jede VGA- und HDMI-Auflösung kann in verschiedenen Modi betrieben werden, die mit dem Befehl MODE eingestellt werden. 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 Zeit und Datum aus dem RTC beim Booten und jede Stunde. Wenn diese Option aktiviert ist und der RTC nicht reagiert, wird jedes laufende Programm mit einer Fehlermeldung abgebrochen. An der Befehlszeile wird eine Informationsmeldung ausgegeben.

		Dieser Befehl muss an der Befehlszeile (nicht in einem Programm) ausgeführt werden.
OPTION SDCARD CSpin [,CLKpin, MOSIpin, MISOpin] oder OPTION SDCARD DISABLE	✓	Legen Sie die für die SD-Kartenschnittstelle zu verwendenden E/A-Pins fest oder deaktivieren Sie sie. Wenn die optionalen Pins nicht angegeben werden, verwendet die SD-Karte die durch OPTION SYSTEM SPI angegebenen Pins. 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. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.
OPTION SDCARD COMBINED CS	✓	Hiermit wird festgelegt, dass der Touch-Chip-Auswahl-Pin auch für die SD-Karte verwendet wird. In diesem Fall ist eine externe Schaltung erforderlich, um die SD-Chip-Auswahl zu implementieren. Siehe „SD-Karten" im Kapitel „Programm- und Datenspeicherung".
OPTION SERIAL CONSOLE uartapin, uartbpin [,B] OPTION SERIAL CONSOLE DISABLE	✓	Legt fest, dass der Zugriff auf die Konsole über einen seriellen Hardware-Port (anstelle eines virtuellen seriellen Ports über USB) erfolgt. „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, ist nicht wichtig. 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-Port gesendet. Zurück zur normalen USB-Konsole. Diese Befehle müssen an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.
OPTION SYSTEM I2C sdapin, sclpin [,SLOW/FAST]	✓	Gib den I^2C-Port und die Pins an, die von Systemgeräten (LCD-Panel und RTC) genutzt werden sollen. Die PicoMite-Firmware nutzt einen bestimmten I^2C-Port für Systemgeräte, während der andere für den Programmierer bleibt. Dieser Befehl legt fest, welche Pins genutzt werden sollen und somit welcher der I^2C-Ports genutzt wird. Die dem SYSTEM I2C zugewiesenen Pins stehen für andere MMBasic SETPIN-Einstellungen nicht zur Verfügung, können aber für zusätzliche I^2C-Geräte unter Verwendung des Standard-I2C-Befehls verwendet werden. Hinweis: I2C(2) OPEN und I2C(2) CLOSE sind in diesem Fall nicht verfügbar. Standardmäßig wird der I^2C-Port mit einer Geschwindigkeit von 400 kHz und einer Zeitüberschreitung von 100 ms geöffnet. Die I^2C-Frequenz kann mit dem optionalen dritten Parameter eingestellt werden, der die Werte FAST = 400 kHz oder SLOW = 100 kHz annehmen kann. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.

OPTION SYSTEM SPI CLKpin, MOSIpin, MISOpin oder OPTION SYSTEM SPI DISABLE	✓	Legen Sie den SPI-Port und die Pins für die Verwendung durch Systemgeräte (SD-Karte, LCD-Panel usw.) fest oder deaktivieren Sie sie. Die PicoMite-Firmware nutzt einen bestimmten Hardware-SPI-Port für Systemgeräte, während der andere für den Benutzer verfügbar 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. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.
OPTION TAB 2 | 3 | 4 | 8	✓	Legt den Abstand für die Tabulatortaste fest. Der Standardwert ist 2.
OPTION TCP SERVER PORT n	✓	NUR WEBMITE Startet bei jedem Neustart von WebMite einen TCP-Server auf Port „n“. Normalerweise nutzen HTTP-Server Port 80. Verwende „OPTION TCP SERVER PORT 0”, um die Funktion zu deaktivieren. Wenn der Server läuft, kann er auf bis zu MM.INFO(MAX CONNECTIONS) reagieren.
OPTION TELNET-KONSOLE AUS|NUR|EIN	✓	NUR WEBMITE Konfiguriert die Handhabung der Konsole über Telnet. ON = Konsolenausgabe wird an USB und Telnet gesendet ONLY = Konsolenausgabe nur an Telnet AUS = Konsolenausgabe nur an USB
OPTION TFTP OFF|ON	✓	NUR WEBMITE Schaltet den TFTP-Server ein oder aus. Standardmäßig ist er eingeschaltet.
OPTION TOUCH T_CS-Pin, T_IRQ-Pin [, Beep] oder OPTION TOUCH DISABLE	✓	NICHT VGA- ODER HDMI-VERSIONEN Konfiguriert MMBasic für die berührungsempfindliche Funktion eines angeschlossenen LCD-Bildschirms. „T_CS-Pin” und „T_IRQ-Pin” sind die I/O-Pins, die für die Chipauswahl bzw. den Touch-Interrupt verwendet werden (es können beliebige freie Pins verwendet werden). Die restlichen Pins werden mit denen verbunden, die mit dem Befehl OPTION SYSTEM SPI angegeben wurden. „Beep” ist ein optionaler Pin, der mit einem kleinen Summer/Beeper verbunden werden kann, um ein „Klicken” oder einen Piepton zu erzeugen, wenn eine erweiterte Grafiksteuerung berührt wird (z. B. Optionsfeld, Schalter usw.). Dies wird in *Advanced Graphics Functions.pdf* beschrieben. Dieser Befehl muss an der Eingabeaufforderung (nicht in einem Programm) ausgeführt werden.
OPTION TOUCH FT6336 IRQpin, RESETpin [,BEEPpin] [,sensitivity]	✓	NICHT VGA- ODER HDMI-VERSIONEN Aktiviert die Touch-Unterstützung für den kapazitiven Touch-Chip FT6336. Die Empfindlichkeit ist eine Zahl zwischen 0 und 255 – der Standardwert ist 50, je niedriger, desto empfindlicher. SDA und SCK sollten an gültige I2C-Pins angeschlossen und mit OPTION SYSTEM I2C eingerichtet werden. Sieh dir auch die TOUCH-

		Funktion an.
OPTION VCC-Spannung		Legt die Spannung (Vcc) fest, die an den Raspberry Pi Pico angelegt wird. Bei Verwendung der ADC-Pins zur Spannungsmessung verwendet die PicoMite-Firmware die Spannung am Pin mit der Bezeichnung VREF als Referenz. Diese Spannung kann mit einem DMM genau gemessen und mit diesem Befehl für eine genauere Messung eingestellt werden. Der Parameter wird nicht gespeichert und sollte im Programm initialisiert werden. Der Standardwert, wenn nicht festgelegt, ist 3,3.
OPTION UDP-SERVER-PORT n	✓	NUR WEBMITE-VERSION 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 zusätzlichen Zeichen verworfen. Verwende „OPTION UDP SERVER PORT 0", um die Funktion zu deaktivieren.
OPTION VGA-PINS HSYNC-Pin, BLUE-Pin	✓	Nur VGA-Version Ändert die Pins, die für die VGA-Bildschirmausgabe verwendet werden, und macht das PCB-Design oder die Verdrahtung flexibler. „HSYNCpin" zeigt den Anfang eines Paares aufeinanderfolgender GP-nummerierter Pins an, die mit HSYNC und VSYNC verbunden sind. „BLUEpin" zeigt an, wo vier aufeinanderfolgende GP-nummerierte Pins anfangen, die mit BLUE, GREEN_LSB, GREEN_MSB und RED verbunden sind.
OPTION WEB-MELDUNGEN EIN/AUS		NUR WEBMITE-VERSION Deaktiviert informative Web-Meldungen, wenn auf „OFF" gesetzt. Standardmäßig ist „ON" eingestellt.
OPTION WIFI ssid$, passwd$, [name$] [,ipaddress$, mask$, gateway$]	✓	NUR WEBMITE-VERSION Konfiguriert die Firmware so, dass sie sich beim Neustart automatisch mit einem WLAN-Netzwerk verbindet. „ssid$" ist der Name des Netzwerks und „password$" ist das Zugangspasswort. Beide sind Zeichenfolgen und wenn Zeichenfolgenkonstanten verwendet werden, sollten sie in Anführungszeichen gesetzt werden. Optional kann ein Name für das Gerät angegeben werden „name$", andernfalls wird ein Name aus der eindeutigen Geräte-ID erstellt. Optional können eine statische IP-Adresse, eine IP-Maske und eine Gateway-Adresse als „ipaddress$", „mask$" und „gateway$" angegeben werden. 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)	Startet einen Kommentar, und jeder darauf folgende Text wird ignoriert. Kommentare können an beliebiger Stelle in einer Zeile platziert werden.
*Datei	Der Sternchenbefehl ist eine Abkürzung für RUN, die nur an der MMBasic-Eingabeaufforderung verwendet werden kann. Beispiel: * RUN *foo RUN „foo” *"foo bar" RUN „foo bar” *foo –wombat RUN „foo”, „--wombat” *foo „wom” RUN „foo”, CHR$(34) + „wom” + CHR$(34) *foo --wom="bat" RUN „foo”, „--wom=” + CHR$(34) + „bat” + CHR$(34) String-Ausdrücke werden von diesem Befehl nicht unterstützt/ausgewertet; alle angegebenen Argumente werden als Literalstring an den Befehl RUN übergeben.
? (Fragezeichen)	Abkürzung für den Befehl PRINT.
/* */	Start und Ende von mehrzeiligen Kommentaren. /* und */ müssen die ersten Zeichen ohne Leerzeichen am Anfang einer Zeile sein und von einem Leerzeichen oder einem Zeilenende gefolgt werden (d. h. es handelt sich um 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:“ in der Befehlszeile
ADC ADC OPEN freq, n_channels [,interrupt]	Die ADC-Befehle bieten eine alternative Methode zur Aufzeichnung analoger Eingänge und sind für die schnelle Aufzeichnung vieler Messwerte in einem Array gedacht. Dabei 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. Die Pin-Reihe umfasst GP26, GP27, GP28 und GP29 für RP2040 und RP2350A oder GP40, GP41, GP42, GP43 auf RP2350B. Wenn die Anzahl der Kanäle eins ist, wird immer GP26 (ADC0) verwendet, wenn zwei, werden GP26 und GP27 (ADC0 und ADC1) verwendet usw. Die Abtastung mehrerer Kanäle erfolgt sequenziell (es gibt nur einen ADC). Die Pins sind auf die Funktion festgelegt, wenn ADC OPEN aktiv ist. Die maximale Gesamtfrequenz ist CPU-Geschwindigkeit/96 (z. B. 520 kHz, wenn alle vier Kanäle mit einer CPU-Einstellung von 200 MHz abgetastet werden sollen). Denk dran, dass eine Gesamtabtastfrequenz von über 500 kHz eine Übertaktung des ADC bedeutet. Der optionale Interrupt-Parameter gibt an, welcher Interrupt nach Abschluss der Konvertierung aufgerufen werden soll. Wenn nichts angegeben ist, wird die Konvertierung blockiert.
ADC-FREQUENZ freq	Damit änderst du die Abtastfrequenz der ADC-Umwandlung, ohne schließen und wieder öffnen zu müssen.
ADC CLOSE	Gibt die Pins für die normale Verwendung frei.

ADC START array1!() [,array2!()] [,array3!()] [,array4!()] [,C1min] [,C1max] [,C2min] [,C2max] [,C3min] [,C3max] [,C4min] [,C4max]	Startet die Wandlung in die angegebenen Arrays. Die Arrays müssen Fließkommazahlen sein und die gleiche Größe haben. Die Größe der Arrays bestimmt die Anzahl der Wandlungen. Start kann wiederholt aufgerufen werden, sobald der ADC OPEN ist. „Cxmin" und „Cxmax" skalieren die Messwerte. Wenn zum Beispiel C1min=200 und C1max=100 sind, werden Werte zwischen 200 und 100 für äquivalente Spannungen von 0 bis 3,3 erzeugt. Wenn die Skalierung nicht benutzt wird, werden die Ergebnisse als Spannung zwischen 0 und OPTION VCC (Standardwert 3,3 V) zurückgegeben.
ADC RUN array1% (),array2%)	Führt den ADC kontinuierlich im doppelt gepufferten Modus aus. Der ADC füllt zuerst array1% und dann array2% und dann 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 aktuell zum Lesen verfügbaren Puffers zurück (1 oder 2).
ARRAY ADD in(), Wert ,out()	Fügt den Wert „value" zu jedem Element der Matrix in() hinzu (oder hängt ihn für Zeichenfolgen an) und speichert das Ergebnis in out(). Funktioniert für Arrays beliebiger Dimensionen von Zeichenfolgen sowie für Ganzzahlen und Gleitkommazahlen (Konvertierung zwischen Ganzzahlen und Gleitkommazahlen möglich). Die Einstellung von num auf 0 oder „" ist optimiert und eine schnelle Methode zum Kopieren eines gesamten Arrays. in() und out() können dasselbe Array sein.

ARRAY INSERT targetarray(), [d1] [,d2] [,d3] [,d4] [,d5] , sourcearray()	Das ist das Gegenteil von ARRAY SLICE, hat eine sehr ähnliche Syntax und ermöglicht es dir zum Beispiel, einen einzelnen Vektor in einem Array von Vektoren mit einem einzigen Befehl oder ein eindimensionales Array von Zeichenfolgen in ein zweidimensionales Array von Zeichenfolgen zu ersetzen. Die Arrays können numerisch oder Zeichenfolgen sein, und „sourcearray" und „destinationarray" müssen gleich sein (Hinweis: Bei numerischen Arrays kann zwischen Ganzzahlen und Gleitkommazahlen konvertiert werden).
	
	Beispiel:
	`OPTION BASE 1`
	`DIM targetarray(3,4,5)`
	`DIM sourcearray(4)=(1,2,3,4)`
	`ARRAY INSERT targetarray(), 2, , 3, sourcearray()`
	Setzt die Elemente 2,1,3 = 1 und 2,2,3 = 2 und 2,3,3 = 3 und 2,4,3 = 4

ARRAY SET value, array()	Setzt alle Elemente in array() auf den Wert „value". Der Wert kann eine Zahl oder eine Zeichenfolge sein, und „array" muss gleich bleiben (Hinweis: Konvertierung zwischen Ganzzahlen und Gleitkommazahlen möglich). Beachte, dass dies die schnellste Methode ist, um ein Array zu löschen, indem es auf Null oder eine leere Zeichenfolge gesetzt wird.

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 mit einer FOR-Schleife. Der Ausschnitt wird festgelegt, indem du für alle Indizes des Quellarrays außer einem einen Wert angibst. Der Befehl sollte so viele Indizes enthalten, einschließlich des leeren, wie das Quellarray Dimensionen hat. Die Arrays können numerisch oder Zeichenfolgen sein, und „sourcearray" und „destinationarray" müssen gleich sein (Hinweis: Bei numerischen Arrays kann zwischen Ganzzahlen und Gleitkommazahlen konvertiert werden).
	Beispiel:
	`OPTION BASE 1`
	`DIM a(3,4,5)`
	`DIM b(4)`
	`ARRAY SLICE a(), 2, , 3, b()`
	Kopiert die Elemente 2,1,3 und 2,2,3 und 2,3,3 und 2,4,3 in das Array b().

ARC x, y, r1, [r2], a1, a2 [, c]	Zeichnet einen Kreisbogen mit einer bestimmten Farbe und Breite zwischen zwei Radien (in Grad angegeben). Die Parameter für den Befehl ARC sind: x: X-Koordinate des Mittelpunkts des Bogens y: Y-Koordinate des Mittelpunkts des Bogens r1: Innenradius des Bogens r2: äußerer Radius des Bogens – kann weggelassen werden, wenn er 1 Pixel breit ist a1: Startwinkel des Bogens in Grad a2: Endwinkel des Bogens in Grad c: Farbe des Bogens (wenn nichts angegeben ist, wird die Vordergrundfarbe genommen) Null Grad ist die 12-Uhr-Position.
ASTRO	Berechnet die Position eines Himmelsobjekts anhand der manuell mit dem Befehl LOCATION eingestellten Zeit und Position. Dies ist einer von mehreren Befehlen, die hochpräzise astronomische Berechnungen durchführen, die für die Ausrichtung von Teleskopen oder die Navigation geeignet sind. Eine detaillierte Beschreibung findest du in der Datei *„GPS_Astro_Reference.pdf"*, die im ZIP-Archiv zum Herunterladen der Firmware enthalten ist.

AUTOSAVE oder AUTOSAVE CRUNCH oder AUTOSAVE APPEND oder AUTOSAVE N	Hiermit startest du den automatischen Programmeingabemodus. Dieser Befehl nimmt Textzeilen von der seriellen Konsole und speichert sie im Programmspeicher. Dies ist eine Möglichkeit, ein BASIC-Programm auf den Raspberry Pi Pico zu übertragen. Das zu übertragende Programm kann in einen Terminalemulator eingefügt werden, und dieser Befehl erfasst den Textstrom und speichert ihn im Programmspeicher. Er kann auch zum direkten Eingeben eines kleinen Programms über die Konsoleneingabe verwendet werden. 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. Mit F2 kannst du den Modus verlassen und das Programm sofort ausführen. Die Option CRUNCH weist MMBasic an, vor dem Speichern alle Kommentare, Leerzeilen und unnötigen Leerzeichen aus dem Programm zu entfernen. Dies kann bei großen Programmen verwendet werden, damit sie in den begrenzten Speicher passen. CRUNCH kann mit dem einzelnen Buchstaben C abgekürzt werden. Die Option APPEND lässt das vorhandene Programm unverändert und hängt die neuen Daten aus der seriellen Eingabe an dessen Ende an. Die Option N führt die automatische Speicherung wie gewohnt aus, gibt die Daten aber nicht an die Konsole zurück. Sie wartet auf das Senden des ersten Zeichens und verwendet dann einen Timer, der überprüft, ob zwischen den Zeichen mehr als 100 ms liegen. Wenn diese Verzögerung festgestellt wird, wechselt das Programm zurück in den normalen Eingabemodus und gibt die Meldung „Enter ctrl-Z, F1 oder F2 zum Beenden“ Du kannst dann weitere Zeichen eingeben, die gespeichert werden sollen, oder einen der normalen Befehle zum Beenden verwenden. Dieser Befehl kann jederzeit mit Strg-C abgebrochen werden, wodurch der Programmspeicher unverändert bleibt.
BACKLIGHT n [,DEFAULT] HINTERGRUNDBELEUCHTUNG n [,FreqInHz]	<u>NICHT-VGA- ODER HDMI-VERSIONEN</u> Stellt die Hintergrundbeleuchtung des Displays ein, gültige Werte sind 0 bis 100. Wenn DEFAULT angegeben ist, stellt die Firmware die Hintergrundbeleuchtung beim Einschalten automatisch auf diesen Wert ein. Das ist besonders nützlich im Batteriebetrieb, wo eine Reduzierung der Hintergrundbeleuchtung die Batterielebensdauer erheblich verlängern kann. Einige Schaltungen sind zu langsam, um die Standard-PWM-Frequenz der Hintergrundbeleuchtung zu verwenden, die gewählt wurde, um Störungen des Audiosignals zu vermeiden. In diesem Fall kann eine Benutzerfrequenz angegeben werden. Dies ist eine temporäre Option, die bei jedem Neustart neu eingestellt werden muss.
BEZIER x%(),y%() [,n] [,Farbe]	Zeichnet eine Bezier-Kurve mit einer unbegrenzten Anzahl von Kontrollpunkten. „x%()” und „y%()” sind eindimensionale Integer-Arrays, die die Koordinaten der Kontrollpunkte enthalten. Die Bezier-Kurve beginnt immer am ersten Kontrollpunkt. Die Arrays müssen die gleiche Dimension haben. Der optionale Parameter „n” legt fest, wie viele Kontrollpunkte für die Darstellung verwendet werden sollen. Wenn er weggelassen wird, bestimmen die Größe von „x%()” und „y%()” die Anzahl. N muss >=2 und <= der Array-Größe sein. Der optionale Parameter „colour“ sagt, in welcher Farbe die Kurve gezeichnet wird (standardmäßig weiß).

BIT(var%, bitno) = Wert	Setzt ein bestimmtes Bit (0-63) in einer ganzzahligen Variablen. „Wert“ kann 0 oder 1 sein. 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.
BLIT	BLIT ist eine einfache Speicheroperation, bei der Daten von einem Display oder Speicher auf ein Display oder einen Speicher kopiert werden. Hinweise: • Es stehen 32 Puffer von #1 bis #32 zur Verfügung. • Bei der Angabe der Puffernummer ist das Symbol # optional. • Alle anderen Argumente sind in Pixeln angegeben.
BLIT READ [#]b, x, y, w, h	BLIT READ kopiert einen Teil der Anzeige in den Speicherpuffer '#b'. Die Quellkoordinaten sind 'x' und 'y', die Breite des zu kopierenden Anzeigebereichs ist 'w' und die Höhe ist 'h'. Bei Verwendung dieses Befehls wird der Speicherpuffer automatisch erstellt und ausreichend Speicher zugewiesen. Dieser Puffer kann mit dem Befehl BLIT CLOSE freigegeben und der Speicher zurückgewonnen werden.
BLIT WRITE [#]b, x, y [,mode]	BLIT WRITE kopiert den Speicherpuffer '#b' auf die Anzeige. Die Zielkoordinaten sind 'x' und 'y'. Der optionale Parameter „mode” ist standardmäßig auf 0 gesetzt und legt fest, wie die gespeicherten Bilddaten beim Schreiben geändert werden. Es ist das bitweise AND der folgenden Werte: &B001 = von links nach rechts gespiegelt &B010 = von oben nach unten gespiegelt &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 Anzeigetafeln (nicht nur auf Tafeln mit dem ILI9341-Controller). Beispiel: BLIT LOAD #1,"image1", 50,50,100,100 lädt einen Bereich von 100 Pixeln im Quadrat mit der oberen linken Ecke bei 50,50 aus dem Bild image1.bmp
BLIT CLOSE [#]b	BLIT CLOSE schließt den Speicherpuffer „#b”, damit er für einen anderen BLIT READ-Vorgang genutzt werden kann und der verwendete Speicher wiederhergestellt wird.
BLIT MERGE Farbe, x, y, w, h	<u>NICHT VGA- ODER HDMI-VERSIONEN</u> Kopiert einen Bereich des Framebuffers, der durch die Pixelkoordinaten „x” und „y” der oberen linken Ecke definiert ist und eine Breite von „w” und eine Höhe von „h” hat, auf das LCD-Display. Als Teil des Kopiervorgangs überlagert es das LCD-Display mit Pixeln aus dem Layer-Puffer, die nicht auf die angegebene „Farbe” eingestellt sind. Die Farbe wird als Zahl zwischen 0 und 15 angegeben, die Folgendes darstellt: Schwarz, Blau, Myrte, Kobalt, Mittelgrün, Cerulean, Grün, Cyan, Rot, Magenta, Rost, Fuchsia, Braun, Flieder, Gelb und Weiß Für den Betrieb müssen sowohl ein Framebuffer als auch ein Layer-Puffer erstellt worden sein. 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” legen die obere linke Ecke des Bereichs „width” und „height” auf dem Quell-Framebuffer fest, der kopiert werden soll. „xout” und „yout” zeigen die linke obere Ecke des Bereichs auf dem Ziel-Framebuffer an, der die Kopie bekommen soll. Der optionale Parameter „colour” gibt eine Pixelfarbe auf der Quelle an, die nicht kopiert wird. Wenn er weggelassen wird, werden alle Pixel kopiert. Die Farbe wird als Zahl zwischen 0 und 15 angegeben, die Folgendes bedeutet: Schwarz, Blau, Myrte, Kobalt, Mittelgrün, Cerulean, Grün, Cyan, Rot, Magenta, Rost, Fuchsia, Braun, Flieder, Gelb und Weiß Für die Ausführung müssen sowohl ein Framebuffer als auch ein Layer-Buffer erstellt worden sein. Bei ILI9341-, ST7789_320- und ILI9488-Displays wird automatisch auf die Bildausblendung gewartet, bevor mit dem Kopieren begonnen wird.
BLIT MEMORY Adresse, x, y [,col]	Kopiert einen Speicherbereich, der als gepacktes Array von Farbnibbles behandelt wird, in die aktuelle Grafikausgabe, wie durch FRAMEBUFFER WRITE angegeben. Die Farbe wird als Zahl zwischen 0 und 15 angegeben, die Folgendes darstellt: Schwarz, Blau, Myrte, Kobalt, Mittelgrün, Cerulean, Grün, Cyan, Rot, Magenta, Rost, Fuchsia, Braun, Flieder, Gelb und Weiß Das erste Wort des Speicherbereichs, der bei „Adresse%” anfängt, muss die Breite und Höhe des zu kopierenden Bereichs als 16-Bit-Ganzzahlen enthalten, wobei die Breite die unteren 16 Bits sind. Die Adresse muss an einer Wortgrenze ausgerichtet sein (durch 4 teilbar). Wenn der optionale Parameter „col” angegeben wird, wird diese bestimmte Farbe nicht kopiert. Wenn das oberste Bit der Breite oder Höhe auf 1 gesetzt ist, werden die Farbdaten als komprimiert behandelt (die restlichen 15 Bits werden als Breite und/oder Höhe verwendet). Der Komprimierungsalgorithmus ist einfach: Jedes Byte enthält eine Zählung im unteren Nibble (1–15) und eine Farbe im oberen Nibble (0–15). Wenn mehr als 15 Pixel dieselbe Farbe haben, werden zusätzliche Bytes für diese Farbe verwendet.
BLIT COMPRESSED Adresse %, x, y [,col]	Funktioniert genauso wie BLIT MEMORY, geht aber davon aus, dass die Daten komprimiert sind, und ignoriert das oberste Bit in der 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” legen die obere linke Ecke des Bereichs mit der „width” und „height” auf dem Flash-Slot fest, der kopiert werden soll. „xout” und „yout” sagen, wo oben links der Bereich auf dem Ziel-Framebuffer ist, der die Kopie bekommen soll. Der optionale Parameter „Farbe” sagt, welche Pixelfarbe im Flash-Slot nicht kopiert werden soll. Wenn das weggelassen wird, werden alle Pixel kopiert. Die Farbe wird als Zahl zwischen 0 und 15 angegeben, die Folgendes bedeutet: Schwarz, Blau, Myrte, Kobalt, Mittelgrün, Cerulean, Grün, Cyan, Rot, Magenta, Rostrot, Fuchsia, Braun, Flieder, Gelb und Weiß

BLIT x1, y1, x2, y2, w, h	Kopiere einen Teil des Bildschirms in einen anderen Teil des Bildschirms. 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". Alle Argumente sind in Pixeln angegeben. Wenn die Ausgabe auf einem LCD-Bildschirm angezeigt werden soll, muss es sich um einen der folgenden Controller handeln: SSD19863, ILI9341_8, ILI9341, ILI9488 (wenn MISO angeschlossen) oder ST7789_320.
BOX x, y, w, h [, lw] [,c] [,fill]	Zeichnet ein Feld auf dem Display mit der oberen linken Ecke bei „x" und „y" mit einer Breite von „w" Pixeln und einer Höhe von „h" Pixeln. „lw" ist die Breite der Seiten des Kastens und kann Null sein. Der Standardwert ist 1. „c" gibt die Farbe an und ist standardmäßig die Standard-Vordergrundfarbe, wenn nichts anderes angegeben ist. „fill" ist die Füllfarbe. Sie kann weggelassen oder auf -1 gesetzt werden, dann wird das Feld nicht gefüllt. 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, sonst wird ein Fehler ausgegeben. „w", „h", „lw", „c" und „fill" können entweder Arrays oder einzelne Variablen/Konstanten sein. Eine Definition der Farben und Grafikkoordinaten findest du im Kapitel *„Grafikbefehle und -funktionen".*
BYTE(var$, byteno)=value	Setzt ein bestimmtes Byte in einer Zeichenfolge auf einen ganzzahligen Wert. „Wert" kann im Bereich von 0 bis 255 liegen. Die Byte-Nummer kann zwischen 1 und der aktuellen Länge der Zeichenfolgenvariablen liegen. Dies entspricht MID$(var$,byteno,1)=CHR$(value), wird jedoch viel schneller ausgeführt. Siehe auch die Funktion BYTE.
CALL usersubname$ [,usersubparameters,..]	Das ist eine coole Möglichkeit, benutzerdefinierte Unterprogramme programmgesteuert aufzurufen (siehe auch die Funktion CALL()). Oft kannst du damit komplizierte SELECT- und IF THEN ELSEIF ENDIF-Klauseln vermeiden und es wird viel effizienter verarbeitet. „usersubname$" kann eine beliebige Zeichenfolge, Variable oder Funktion sein, die zum Namen einer normalen Benutzer-Subroutine (kein integrierter Befehl) aufgelöst wird. Die „usersubparameters" sind die gleichen Parameter, die auch beim direkten Aufruf der Unterroutine verwendet würden. Eine typische Anwendung wäre das Schreiben einer beliebigen Art von Emulator, bei dem eine von vielen Unterroutinen in Abhängigkeit von einer Variablen aufgerufen werden soll. Außerdem bietet sie die Möglichkeit, einen Unterroutinenamen als Variable an eine andere Unterroutine oder Funktion zu übergeben.
KAMERA	NICHT VGA- ODER HDMI-VERSIONEN Befehl zur Unterstützung des OV7670-Kameramoduls.

CAMERA OPEN XLKpin, PLKpin, HSpin, VSCpin, RETpin, D0pin	Damit wird die Kamera initialisiert. Es wird ein 12-MHz-Takt auf XLK (PWM) ausgegeben und überprüft, ob die Signale auf PLK, VS und HS richtig empfangen werden. Die Kamera wird auf eine Auflösung von 160 x 120 (QQVGA) eingestellt, was das Maximum ist, das mit dem verfügbaren Speicher möglich ist. Aktivieren Sie OPTION SYSTEM I2C in der PicoMite-Firmware und verbinden Sie SCL und SDA mit den entsprechenden Pins (auf dem Kameramodul möglicherweise mit SIOC und SIOD gekennzeichnet). Diese Verbindungen müssen einen Pullup auf 3,3 V haben – empfohlen wird 2K7. Andere Pins werden gemäß dem Befehl OPEN verdrahtet. (Hinweis: VS kann auf Ihrem Modul als VSYNC, HS als HREF, PLK als PCLK, RET als RESET und XLK als XCLK gekennzeichnet sein. D0pin legt den Anfang eines Bereichs von 8 aufeinanderfolgenden Pins fest (z. B. GP0 – GP7).
KAMERA-AUFNAHME [Skala, [x , y]]	Damit wird ein Bild von der Kamera (RGB565) aufgenommen und auf einem LCD-Bildschirm angezeigt. Damit der Befehl funktioniert, muss ein SPI-LCD angeschlossen und aktiviert sein. (ILI9341 und ST7789_320 werden empfohlen). Die Skalierung ist standardmäßig auf 1 und x, y jeweils auf 0 eingestellt. Standardmäßig wird ein Bild mit einer Größe von 160 x 120 auf dem LCD-Bildschirm ausgegeben, wobei die obere linke Ecke bei 0,0 auf dem LCD-Bildschirm liegt. Wenn du den Maßstab auf 2 setzt, wird das Bild auf einem 320 x 240 Display angezeigt. Durch Einstellen der Parameter x und y wird die obere linke Ecke des Bildes auf dem LCD-Bildschirm versetzt. Die Aktualisierungsrate in einer Endlosschleife beträgt 7 FPS auf dem Display im Maßstab 1:1 und 5 FPS skaliert auf 320 x 240. Wenn das Display über MISO verkabelt ist, kann das Bild mit dem Befehl SAVE IMAGE auf der Festplatte gespeichert werden.
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 Intensitäts- und Farbinformationen getrennt sind und nur ein Byte für ein Graustufenbild mit 256 Stufen benötigt wird, was ideal für die Bewegungserkennung ist. image% ist ein Array der Größe 160x120 Bytes (DIM image%(160,120/8-1) Beim Aufruf des Befehls enthält es ein gepacktes 8-Bit-Graustufenbild. Die Variable change! gibt den Prozentsatz zurück, um den sich das Bild seit dem letzten Aufruf des Befehls verändert hat. Wenn „scale” eingestellt ist, wird optional das Bilddelta auf dem Bildschirm angezeigt, also der Unterschied 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.
KAMERATEST tnum	Aktiviert oder deaktiviert ein Testsignal von der Kamera. tnum=2 erzeugt Farbbalken und tnum=0 stellt die visuelle Eingabe wieder her.

KAMERA-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 Einstellung der verschiedenen Kameraregister noch weiter verbessert werden.
CAT S$, N$	Verbindet die Zeichenfolgen, indem N$ an S$ angehängt wird. Das ist im Grunde dasselbe wie `S$ = S$ + N$`, läuft aber etwas schneller.
CHAIN fname$ [cmdline$]	Ermöglicht es einem Programm, ein anderes Programm auszuführen, wobei der Variablenbereich erhalten bleibt – es wird empfohlen, den Befehl in einem Programm der obersten Ebene und nicht innerhalb einer Subroutine zu verwenden. Wenn der optionale Parameter „cmdline$" angegeben ist, wird dieser in MM.CMDLINE$ an das verkettete Programm übergeben.
CHDIR dir$	Ändert das aktuelle Arbeitsverzeichnis auf dem Standardlaufwerk zu „dir$". 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 'x' und 'y' und dem Radius 'r' auf dem Bildschirm. 'lw' ist optional und steht für die Linienbreite (Standardwert ist 1). „c" ist die optionale Farbe und ist standardmäßig die aktuelle Vordergrundfarbe, wenn nichts anderes angegeben wird. Das optionale „a" ist eine Fließkommazahl, 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, wodurch das Feld nicht gefüllt wird. 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', „y" und „r" müssen entweder Arrays oder einzelne Variablen/Konstanten sein, sonst kommt es zu einem Fehler. „lw", „a", „c" und „fill" können entweder Arrays oder einzelne Variablen/Konstanten sein. 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. Siehe ERASE zum Löschen bestimmter Array-Variablen.
CLOSE [#]fnbr [,[#]fnbr] …	Schließ die Datei(en), die du vorher mit der Dateinummer „#fnbr" geöffnet hast. Das # ist optional. Sieh dir auch den Befehl OPEN an.
CLS [Farbe]	Löscht den Bildschirm des LCD-Panels. Optional kann „Farbe" angegeben werden, die beim Löschen des Bildschirms als Hintergrundfarbe verwendet wird.
CMM2 LOAD oder CMM2 RUN	Lädt und/oder führt ein Programm von der Festplatte mit dem CMM2-Programmlademodus aus. Das beinhaltet eine aggressive Komprimierung des Programms und unterstützt #INCLUDE-Dateien und #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 alle Bearbeitungen von Programmen offline oder direkt von und auf die Festplatte erfolgen müssen, da die Quelldateien aus der mit diesen Befehlen geladenen Version nicht

	rekonstruiert werden können.
COLOUR fore [, back] oder COLOR fore [, back]	Legt die Standardfarbe für Befehle (PRINT usw.) fest, die auf dem angeschlossenen LCD-Bildschirm angezeigt werden. „fore" ist die Vordergrundfarbe, „back" ist die Hintergrundfarbe. Der Hintergrund ist optional und wird, wenn nicht angegeben, standardmäßig auf eine zuvor festgelegte Hintergrundfarbe oder, falls zuvor nicht geändert, auf Schwarz gesetzt.
COLOUR MAP inarray%(), outarray%() [,colourmap%()]	Dieser Befehl erzeugt 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 diesen Indexwert in colourmap% zugeordnet.
CONFIGURE cfg LISTE KONFIGURIEREN	Konfiguriert eine Karte gemäß dem in „cfg" angegebenen Äquivalent von OPTION RESET. Listet alle verschiedenen Konfigurationen auf, die für die Firmware-Version verfügbar sind.
CONST id = Ausdruck [, id = Ausdruck] … usw.	Erstellt eine Konstante, die nach dem Erstellen nicht mehr geändert werden kann. „id" ist der Bezeichner, der denselben Regeln wie Variablen folgt. Der Bezeichner kann ein Typ-Suffix (!, % oder $) haben, das aber nicht erforderlich ist. Wenn es angegeben wird, muss es mit dem Typ von „Ausdruck" übereinstimmen. „Ausdruck" ist der Wert des Bezeichners und kann ein normaler Ausdruck (einschließlich benutzerdefinierter Funktionen) sein, der bei der Erstellung der Konstante ausgewertet wird. Eine außerhalb einer Subroutine oder Funktion definierte Konstante ist global und im gesamten Programm sichtbar. Eine innerhalb einer Subroutine oder Funktion definierte Konstante ist lokal für diese Routine und verdeckt eine globale Konstante mit demselben Namen.
CONTINUE	Setzt die Ausführung eines Programms fort, das durch eine END-Anweisung, einen Fehler oder STRG-C angehalten wurde. Das Programm wird mit der nächsten Anweisung nach dem vorherigen Haltepunkt neu gestartet. Beachte, dass es nicht immer möglich ist, das Programm korrekt fortzusetzen – das gilt vor allem für komplexe Programme mit Grafiken, verschachtelten Schleifen und/oder verschachtelten Unterprogrammen und Funktionen.
CONTINUE DO oder CONTINUE FOR	Springe zum Ende einer DO/LOOP- oder einer FOR/NEXT-Schleife. Die Schleifenbedingung wird dann geprüft und wenn sie noch gültig ist, wird die Schleife mit der nächsten Iteration fortgesetzt.
COPY fname1$ TO fname2$ COPY fname$ TO dirname$	Kopiere eine Datei von „fname1$" nach „fname2$". Beide sind Zeichenfolgen. Sowohl in „fname$" als auch in „fname$" kann ein Verzeichnispfad verwendet werden. Wenn die Pfade unterschiedlich sind, wird die in „fname$" angegebene Datei mit dem angegebenen Dateinamen in den in „fname2$" angegebenen Pfad kopiert. Die Dateinamen können die Laufwerksangabe enthalten, wenn du auf ein nicht aktives Laufwerk kopierst oder von einem nicht aktiven Laufwerk kopierst (siehe Befehl DRIVE). Kopieren mit Platzhaltern. Die Massenkopie wird ausgelöst, wenn fname$ ein „*"- oder ein „?"-Zeichen enthält. dirname$ muss ein gültiger Verzeichnisname sein und darf NICHT mit einem Schrägstrich enden.

CPU RESTART	Löst einen Neustart der Prozessoren aus. Dadurch werden alle Variablen gelöscht und alles zurückgesetzt (z. B. Timer, COM-Ports, I2C usw.), ähnlich wie beim Einschalten, jedoch ohne den Startbildschirm. Wenn die OPTION AUTORUN gesetzt wurde, wird das Programm an der angegebenen Flash-Position oder im Programmspeicher neu gestartet.
CPU-RUHEZUSTAND 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] …] hex [[hex[…] hex [[hex[…] END CSUB	Definiert den Binärcode für ein eingebettetes Maschinencode-Programmmodul, das in C oder ARM-Assembler geschrieben ist. Das Modul erscheint in MMBasic als Befehl „name” und kann wie ein integrierter Befehl verwendet werden. In einem Programm können mehrere eingebettete Routinen verwendet werden, wobei jede ein anderes Modul mit einem anderen „name” 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 beim Speichern des Programms automatisch in MMBasic programmiert. Jedes „Hex“ muss genau acht Hexadezimalziffern haben, die die Bits in einem 32-Bit-Wort darstellen, und durch ein oder mehrere Leerzeichen oder Zeilenumbrüche getrennt sein. Der Befehl muss mit einem passenden END CSUB beendet 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. Der Typ jedes Parameters kann in der Definition angegeben werden. Zum Beispiel: `CSUB MySub integer, integer, string` Dies gibt an, dass es drei Parameter gibt, wobei die ersten beiden Ganzzahlen und der dritte eine Zeichenfolge sind. Hinweis: • Es können bis zu zehn Argumente angegeben werden („arg1”, „arg2” usw.). • Wenn eine Variable oder ein Array als Argument angegeben wird, bekommt die C-Routine einen Zeiger auf den Speicher, der der Variable oder dem Array zugewiesen ist, und die C-Routine kann diesen Speicher ändern, um einen Wert an den Aufrufer zurückzugeben. Bei Arrays sollten sie mit leeren Klammern übergeben werden, z. B. arg(). In der CSUB wird das Argument als Zeiger auf das erste Element des Arrays bereitgestellt. • Konstanten und Ausdrücke werden als Zeiger auf einen temporären Speicherplatz, der den Wert enthält, an die eingebettete C-Routine übergeben.
DATA Konstante[,Konstante]..	Speichert numerische Konstanten und Zeichenfolgenkonstanten, auf die mit READ zugegriffen werden kann. Im Allgemeinen sollten Zeichenfolgenkonstanten in doppelte Anführungszeichen ("") gesetzt werden. Eine Ausnahme besteht, wenn die Zeichenfolge nur aus alphanumerischen Zeichen besteht, die keine MMBasic-Schlüsselwörter darstellen (wie THEN, WHILE usw.). In diesem Fall sind Anführungszeichen nicht erforderlich. Numerische Konstanten können auch Ausdrücke wie 5 * 60 sein.

DATE$ = "DD-MM-YY[YY]" oder DATE$ = "DD/MM/YY[YY]" oder DATE$ =”YYYY-MM-DD” oder DATE$ = „JJJJ/MM/TT”	Stell das Datum der internen Uhr/des internen Kalenders ein. DD, MM und YY sind Zahlen, zum Beispiel: DATE$ = "28-7-2014" Mit OPTION RTC AUTO ENABLE startet die PicoMite-Firmware mit dem in RTC programmierten DATE$. Ohne OPTION RTC AUTO ENABLE startet die PicoMite-Firmware beim Einschalten mit dem Datum „01-01-2024”.
DEFINEFONT #Nbr hex [[hex[…] hex [[hex[…] END DEFINEFONT	Damit wird eine eingebettete Schriftart definiert, die zusammen mit den integrierten Schriftarten auf einem angeschlossenen LCD-Bildschirm verwendet werden oder diese ersetzen kann. Diese funktionieren genauso wie die integrierten Schriftarten (d. h. sie werden mit dem Befehl FONT ausgewählt oder im Befehl TEXT angegeben). Eine Auswahl eingebetteter Schriftarten und eine vollständige Beschreibung ihrer Erstellung findest du im Ordner *„Embedded Fonts*” in der ZIP-Datei mit der PicoMite-Firmware. „#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. Jedes „Hex” muss genau acht Hexadezimalziffern umfassen und durch Leerzeichen oder Zeilenumbrüche vom nächsten getrennt sein. • Es können mehrere Zeilen mit „hex”-Wörtern verwendet werden, wobei der Befehl mit einem passenden END DEFINEFONT beendet wird. • In einem Programm können mehrere eingebettete Schriftarten verwendet werden, wobei jede eine andere Schriftart mit einer anderen Schriftartnummer definiert. • Während der Ausführung überspringt MMBasic alle DEFINEFONT-Befehle, sodass sie an beliebiger Stelle im Programm platziert werden können. • Fehler im Datenformat werden beim Speichern des Programms gemeldet.
DEVICE BITSTREAM pinno, n_transitions, array%()	Dieser Befehl wird verwendet, um eine extrem genaue Bitsequenz auf dem angegebenen Pin zu erzeugen. Der Pin muss zuvor als Ausgang eingerichtet und auf den erforderlichen Startpegel eingestellt worden sein. Hinweise: • Das Array enthält die Länge jedes Pegels im Bitstrom in Mikrosekunden. Die maximal zulässige Periode beträgt 65,5 ms. • Der erste Übergang passiert sofort, wenn der Befehl ausgeführt wird. • Die letzte Periode im Array wird ignoriert, außer zur Definition der Zeit, bevor die Steuerung zum Programm oder zur Befehlszeile zurückkehrt. • Der Pin bleibt im Startzustand, wenn die Anzahl der Übergänge gerade ist, und im entgegengesetzten Zustand, wenn die Anzahl der Übergänge ungerade ist.
DEVICE CAMERA	Siehe Befehl CAMERA
GERÄT GAMEPAD	Siehe Befehl GAMEPAD
GERÄT HUMID	Schau mal unter dem Befehl HUMID nach
GERÄT KEYPAD	Schau mal den Befehl KEYPAD an
DEVICE MOUSE	Schau mal unter dem Befehl MOUSE nach
GERÄT LCD	Siehe Befehl LCD

DEVICE SERIALTX pinno, Baudrate, ostring$	Gibt „ostring$“ als seriellen Datenstrom auf „pinno“ aus. „baudrate“ kann zwischen 110 und 230400 liegen (für 230400 muss die CPU möglicherweise übertaktet werden). Beachte, dass das Programm während der Übertragung angehalten wird und Unterbrechungen ignoriert werden.
DEVICE SERIALRX pinno, baudrate, istring$, timeout_in_ms, status% [,nbr] [,terminators$]	Gibt serielle Daten auf „pinno” ein. „baudrate” kann zwischen 110 und 230400 liegen (für 230400 muss die CPU möglicherweise übertaktet werden). „status%” gibt Folgendes zurück: -1 = Zeitüberschreitung (Hinweis: Verwende LEN(istring$), um die Anzahl der empfangenen Zeichen anzuzeigen) 2 = Anzahl der angeforderten Zeichen erreicht 3 = Endzeichen erfüllt „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. Das Programm wird angehalten und Unterbrechungen werden ignoriert, während dieser Befehl ausgeführt wird.
DEVICE WII	Siehe Befehl WII
GERÄT WS2812	Siehe WS2812-Befehl

DIM [Typ] decl [,decl]..
wobei „decl” Folgendes ist:
var [Länge] [Typ] [init]
„var” ist ein Variablenname mit optionalen Dimensionen
„length” wird benutzt, um die maximale Größe der Zeichenfolge auf „n” zu setzen, wie in LENGTH n
„type” ist entweder FLOAT oder INTEGER oder STRING (dem Typ kann das Schlüsselwort AS vorangestellt werden – wie in AS FLOAT)
'init' ist der Wert, mit dem die Variable initialisiert wird, und besteht aus:
= <Ausdruck>
Für eine einfache Variable wird ein Ausdruck verwendet, für ein Array eine Liste von durch Kommas getrennten Ausdrücken, die in Klammern stehen.
 	
Beispiele:
DIM nbr(50)
DIM INTEGER nbr(50)
DIM name AS STRING
	Deklariert eine oder mehrere Variablen (d. h. macht den Variablennamen und seine Eigenschaften dem Interpreter bekannt).
	Wenn OPTION EXPLICIT benutzt wird (was wir empfehlen), sind die Befehle DIM, LOCAL oder STATIC die einzigen Möglichkeiten, eine Variable zu erstellen. Wenn diese Option „ ” nicht benutzt wird, ist die Verwendung des Befehls DIM optional, und wenn er nicht benutzt wird, wird die Variable automatisch erstellt, wenn sie zum ersten Mal referenziert wird.
	Der Typ der Variablen (also Zeichenfolge, Gleitkomma oder Ganzzahl) kann auf drei Arten angegeben werden:
	Durch Verwendung eines Typ-Suffixes (d. h. !, % oder $ für Float, Integer oder String). Beispiel:
	`DIM nbr%, amount!, name$`
	Durch Verwendung eines der Schlüsselwörter FLOAT, INTEGER oder STRING direkt nach dem Befehl DIM und vor der Auflistung der Variablen. Der angegebene Typ gilt dann für alle aufgelisteten Variablen (d. h. er muss nicht wiederholt werden). Beispiel:
	`DIM STRING first_name, last_name, city`
	Verwende die Microsoft-Konvention, bei der das Schlüsselwort „AS” und das Typ-Schlüsselwort (also FLOAT, INTEGER oder STRING) nach jeder Variablen steht. Wenn du diese Methode verwendest, musst du den Typ für jede Variable angeben und kannst ihn von Variable zu Variable ändern.
	Zum Beispiel:
	`DIM Betrag AS FLOAT, Name AS STRING`
	Gleitkomma- oder Ganzzahlvariablen werden bei ihrer 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. Beispiel:
	`DIM STRING city = "Perth", house = "Brick"`
	Der Initialisierungswert kann ein Ausdruck sein (auch mit anderen Variablen) und wird ausgewertet, wenn der DIM-Befehl ausgeführt wird. Weitere Beispiele für die Syntax findest du im Kapitel *„Variablen definieren und*

DIM a, b$, nbr(100), strn$(20)

DIM a(5,5,5), b(1000)

DIM strn$(200) LÄNGE 20

DIM STRING strn(200) LÄNGE 20

DIM a = 1234, b = 345

DIM STRING strn = "Text"

DIM x%(3) = (11, 22, 33, 44)

verwenden".

Neben einfachen Variablen deklariert der Befehl DIM auch Array-Variablen (also indizierte Variablen mit mehreren Dimensionen). Nach dem Namen der Variablen werden die Dimensionen durch eine Liste von Zahlen angegeben, die durch Kommas getrennt und in Klammern gesetzt sind. Zum Beispiel:

```
DIM array(10, 20)
```

Jede Zahl gibt den Indexbereich in jeder Dimension an. Normalerweise beginnt die Indizierung jeder Dimension bei 0, aber mit dem Befehl OPTION BASE kann dies auf 1 geändert werden.

Das obige Beispiel gibt ein zweidimensionales Array mit 11 Elementen (0 bis 10) in der ersten Dimension und 21 Elementen (0 bis 20) in der zweiten Dimension an. Die Gesamtzahl der Elemente beträgt 231, und da jede Gleitkommazahl 8 Byte benötigt, werden insgesamt 1848 Byte Speicher zugewiesen.

Zeichenfolgen belegen standardmäßig 255 Byte (d. h. Zeichen) Speicherplatz pro Element, was bei der Definition von Zeichenfolgen-Arrays schnell zu einer hohen Speicherauslastung führen kann. In diesem Fall kann mit dem Schlüsselwort LENGTH die jedem Element zuzuweisende Speichermenge und damit die maximale Länge der zu speichernden Zeichenfolge angegeben werden. Diese Zuweisung („n“) kann zwischen 1 und 255 Zeichen liegen.

Beispiel: `DIM STRING s(5, 10)` deklariert ein Zeichenfolgenarray mit 66 Elementen, das 16.896 Byte Speicherplatz verbraucht, während

```
DIM STRING s(5, 10) LENGTH 20
```

nur 1.386 Byte Speicherplatz verbraucht. Beachte, dass die jedem Element zugewiesene Speichermenge n + 1 beträgt, da das zusätzliche Byte verwendet wird, um die tatsächliche Länge der in jedem Element gespeicherten Zeichenfolge zu verfolgen.

Wenn einem Element des Arrays eine Zeichenfolge zugewiesen wird, die länger als „n“ ist, wird ein Fehler ausgegeben. Ansonsten verhalten sich mit dem Schlüsselwort LENGTH erstellte Zeichenfolgen-Arrays genauso wie andere Zeichenfolgen-Arrays. Dieses Schlüsselwort kann auch mit Nicht-Array-Zeichenfolgenvariablen verwendet werden, spart jedoch keinen Speicherplatz.

Im obigen Beispiel kannst du auch die Microsoft-Syntax verwenden, bei der der Typ nach dem Längenqualifizierer angegeben wird. Zum Beispiel:

```
DIM s(5, 10) LENGTH 20 AS STRING
```

Der Längenparameter kann für einfache (nicht als Array definierte) Zeichenfolgen mit den folgenden Einschränkungen verwendet werden: Beim RP2040 wird eine Länge von mehr als 9 ignoriert und standardmäßig werden 255 Byte zugewiesen. Beim RP2350 wird eine Länge von mehr als 15 ignoriert und standardmäßig 255 Byte zugewiesen. Wenn die Länge kleiner oder gleich dem Grenzwert ist, wird die Zeichenfolge im Variablenheader gespeichert und 256 Byte werden eingespart. Dies ist wahrscheinlich besonders nützlich für kurze konstante Zeichenfolgen.

Arrays können auch bei ihrer Deklaration initialisiert werden, indem am Ende der Deklaration ein Gleichheitszeichen (=) gefolgt von einer in Klammern gesetzten Liste von Werten hinzugefügt wird. Beispiel:

```
DIM INTEGER nbr(4) = (22, 44, 55, 66, 88)
```

oder

```
DIM s$(3) = ("foo", "boo", "doo", "zoo")
```

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, dann die zweite usw.

Beachte auch, dass die Initialisierungswerte nach dem LENGTH-Qualifizierer (falls verwendet) und nach der Typdeklaration (falls verwendet) stehen

	müssen.
DO <Anweisungen> LOOP	Diese Struktur führt eine Endlosschleife aus. Mit dem Befehl EXIT DO kannst du die Schleife beenden oder die Steuerung muss explizit mit Befehlen wie GOTO oder EXIT SUB (in einer Subroutine) außerhalb der Schleife übertragen werden.
DO WHILE Ausdruck <Anweisungen> LOOP	Wiederholt die Schleife, solange „Ausdruck" wahr ist (dies entspricht der älteren WHILE-WEND-Schleife). Wenn der Ausdruck zu Beginn falsch ist, werden die Anweisungen in der Schleife nicht ausgeführt, auch nicht einmal.
DO <Anweisungen> LOOP UNTIL Ausdruck	Wiederholt die Schleife, bis der Ausdruck nach UNTIL wahr ist. Weil die Prüfung am Ende der Schleife gemacht wird, werden die Anweisungen in der Schleife mindestens einmal ausgeführt, auch wenn der Ausdruck wahr ist.
DO <Anweisungen> LOOP WHILE Ausdruck	Wiederholt die Schleife, bis der Ausdruck nach WHILE falsch ist. Da die Prüfung am Ende der Schleife gemacht wird, werden die Anweisungen in der Schleife mindestens einmal ausgeführt, auch wenn der Ausdruck falsch ist.
DRAW3D	NICHT IN DER WEBMITE-VERSION VERFÜGBAR Die 3D-Engine hat Befehle zum Bearbeiten von 3D-Bildern, wie zum Beispiel das Einstellen der Kamera, Erstellen, Ausblenden, Drehen usw. Eine ausführliche Beschreibung findest du im Dokument *„3D_Graphics_User_Manual.*pdf" im PicoMite-Firmware-Download.
DRIVE drive	Legt das aktive Laufwerk als „drive$" fest. „drive$" kann „A:" oder „B:" sein, wobei A das Flash-Laufwerk und B die SD-Karte ist, falls konfiguriert.
EDIT oder EDIT fname oder DATEI fname$ BEARBEITEN	Ruf den Vollbild-Editor auf. Wenn du einen Dateinamen eingibst, lädt der Editor die Datei von der aktuellen Festplatte (A: oder B:), damit du sie bearbeiten kannst, und speichert sie beim Beenden mit F1 oder F2 auf der Festplatte. Wenn die Datei nicht da ist, wird sie beim Beenden erstellt. Das aktuelle Programm, das im Flash-Speicher gespeichert ist, bleibt davon unberührt. Wenn du eine vorhandene Datei bearbeitest, wird beim Beenden auch eine Sicherungskopie mit der Endung .bak erstellt. Wenn fname$ eine andere Erweiterung als .bas enthält, wird die Farbcodierung während der Bearbeitung vorübergehend deaktiviert. Wenn keine Erweiterung angegeben ist, geht die Firmware von .bas aus. Durch das Bearbeiten einer Datei von der Festplatte können Nicht-Basic-Dateien wie HTML- oder Sprite-Dateien bearbeitet werden, ohne dass sie beim Tokenisierungsprozess, der beim Speichern im Flash-Speicher stattfindet, beschädigt werden. EDIT und EDIT fname$ können nur über die Befehlszeile aufgerufen werden. Wenn du eine Datei in einem Programm bearbeiten möchtest, kannst du den Befehl EDIT FILE fname$ verwenden. Der Befehl muss im obersten Programmlevel verwendet werden und nicht innerhalb einer Subroutine. 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. Details zur Verwendung des Editors findest du im Kapitel *Vollbild-Editor*.
ELSE	Fügt eine optionale Standardbedingung in eine mehrzeilige IF-Anweisung ein. Mehr Infos findest du unter „Mehrzeilige IF-Anweisung".

ELSEIF Ausdruck THEN oder ELSE IF Ausdruck THEN	Fügt eine optionale sekundäre Bedingung in eine mehrzeilige IF-Anweisung ein. Mehr Infos findest du unter der mehrzeiligen IF-Anweisung.
END [noend] oder 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 implizierten END-Befehl endet. Sie wird nicht ausgeführt, wenn das Programm mit dem Abbruchzeichen (d. h. Strg-C) beendet wird. Der optionale Parameter „noend" kann verwendet werden, um die Ausführung der Unterroutine MM.END zu blockieren, z. B. „END noend". Wenn „cmd$" angegeben ist, wird es nach Beendigung des Programms wie an der Eingabeaufforderung ausgeführt. Hinweis: Wenn „END cmd$" verwendet wird, aber eine Unterroutine MM.END vorhanden ist, wird diese ausgeführt und cmd$ ignoriert.
END CSUB	Markiert das Ende einer C-Subroutine. Siehe den Befehl CSUB. Jedes CSUB muss genau eine passende END CSUB-Anweisung haben.
END FUNCTION	Markiert das Ende einer benutzerdefinierten Funktion. Siehe den Befehl FUNCTION. Jede Funktion muss genau eine passende END FUNCTION-Anweisung haben. Verwende EXIT FUNCTION, wenn du aus einer Funktion innerhalb ihres Körpers zurückkehren musst.

ENDIF
oder
END IF
	Beendet eine mehrzeilige IF-Anweisung.
	Weitere Infos findest du unter der mehrzeiligen IF-Anweisung.

END SUB	Markiert das Ende einer benutzerdefinierten Subroutine. Sieh dir den Befehl SUB an. Jede Subroutine muss genau eine passende END SUB-Anweisung haben. Verwende EXIT SUB, wenn du aus einer Subroutine innerhalb ihres Hauptteils zurückkehren musst.
END TYPE	Markiert das Ende einer benutzerdefinierten Struktur. Sieh dir den Befehl TYPE an.
ERASE variable [,variable]..	Löscht globale Variablen und gibt den ihnen zugewiesenen Speicher frei. Das funktioniert mit Array-Variablen und normalen (Nicht-Array-)Variablen. Arrays können mit leeren Klammern (z. B. `dat()`) oder einfach durch Angabe des Variablennamens (z. B. `dat`) angegeben werden. Verwende CLEAR, um alle Variablen gleichzeitig zu löschen (einschließlich Arrays).
ERROR [Fehlermeldung$]	Löst einen Fehler aus und beendet das Programm. Das wird normalerweise beim Debuggen oder zum Abfangen von Ereignissen verwendet, die nicht auftreten sollten. 'error_msg$' ist optional und ist die Meldung, die auf der Konsole angezeigt wird.
EXECUTE Befehl$	Damit wird der Basic-Befehl „command$" ausgeführt. Die Verwendung sollte auf Basic-Befehle beschränkt sein, die nacheinander ausgeführt werden, z. B. funktioniert die GOTO-Anweisung nicht richtig. Zu den getesteten und funktionierenden Elementen gehören GOSUB, Unterprogrammaufrufe und andere einfache Anweisungen (wie PRINT und

	einfache Zuweisungen). Mehrere Anweisungen, die durch : getrennt sind, sind nicht erlaubt und führen zu einem Fehler. Der Befehl setzt einen internen Watchdog, bevor er den angeforderten Befehl ausführt. Wenn die Kontrolle nicht zum Befehl zurückkehrt, wie bei einer GOTO-Anweisung, läuft der Timer ab. In diesem Fall bekommst du die Meldung „Befehlszeitüberschreitung". Du kannst EXECUTE nicht aus Code heraus aufrufen, der mit EXECUTE ausgeführt wurde. RUN ist ein Sonderfall und bricht den Timer ab, sodass du den Befehl bei Bedarf zum Verketten von Programmen verwenden kannst.
EXIT DO EXIT FOR EXIT FUNCTION EXIT SUB	EXIT DO sorgt dafür, dass du eine DO..LOOP-Schleife vorzeitig beenden kannst. EXIT FOR ermöglicht einen vorzeitigen Ausstieg aus einer FOR..NEXT-Schleife. EXIT FUNCTION ermöglicht einen vorzeitigen Ausstieg aus einer definierten Funktion. EXIT SUB ermöglicht einen vorzeitigen Ausstieg aus einer definierten Unterroutine. Der alte Standard von EXIT allein (Beenden einer do-Schleife) wird auch unterstützt.
FILES [fspec$] [,sort]	Listet Dateien in beliebigen Verzeichnissen auf dem Standard-Flash-Dateisystem oder der SD-Karte auf. „fspec$" (falls angegeben) kann einen Pfad und Suchplatzhalter im Dateinamen enthalten. Fragezeichen (?) stehen für ein beliebiges Zeichen und ein Sternchen (*) steht für eine beliebige Anzahl von Zeichen. Wenn nichts angegeben wird, werden alle Dateien aufgelistet. Zum Beispiel: * Alle Einträge suchen *.TXT Alle Einträge mit der Erweiterung TXT suchen E*.* Alle Einträge suchen, die mit E beginnen X?X.* Alle Dateinamen mit drei Buchstaben finden, die mit X beginnen und enden mydir/* Alle Einträge im Verzeichnis mydir suchen Achtung: Wenn du Platzhalter im Pfadnamen benutzt, kommt es zu einem Fehler „sort" legt die Sortierreihenfolge wie folgt fest: Größe in aufsteigender Reihenfolge Zeit in absteigender Reihenfolge Name nach Dateiname (Standard, wenn nicht anders angegeben) Typ nach Dateierweiterung
FILL x, y, Füllfarbe [,Rahmenfarbe]	Füllt einen Bereich der Anzeige mit einer Farbe. Wenn der Befehl ohne die optionale Angabe „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. Wenn die optionale „bordercolour" angegeben wird, ersetzt „fillcolour" alle bereits vorhandenen Farben, bis die angegebene „bordercolour" erreicht wird. Beachte, dass dies auf TFT-Displays mit ungepufferten Treibern langsam sein kann.

FLAG(n%)=Wert	Setzt ein Bit in einem Systemflag-Register. N% kann zwischen 0 und 63 liegen (d. h. es sind 64 Flag-Bits verfügbar). Der Wert kann 0 oder 1 sein. Siehe auch den Befehl FLAGS und die Funktion FLAG sowie MM.FLAGS
FLAGS=Wert	Setzt alle Bits im Systemflag-Register auf den angegebenen Wert. Schau dir auch den Befehl FLAG, die Funktion FLAGS und MM.FLAGS an.
FLASH FLASH LIST FLASH LIST n [,all] FLASH ERASE n FLASH ERASE ALL FLASH SPEICHERN n FLASH-LADEN n FLASH RUN n FLASH-KETTE n FLASH ÜBERSCHREIBEN n FLASH-DISK-LADEN n, Dateiname$ [,O[ÜBERSCHREIBEN]] FLASH LOAD IMAGE n, Dateiname$ [,O/ÜBERSCHREIBEN]	Verwalte 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. Einer dieser Flash-Speicherplätze kann mit dem Befehl OPTION AUTORUN n automatisch geladen und ausgeführt werden, wenn das Gerät eingeschaltet wird. Im Folgenden steht „n“ für eine Zahl zwischen 1 und 3. Zeigt eine Liste aller Flash-Speicherplätze einschließlich der ersten Zeile des Programms an. Listet das auf Speicherplatz n gespeicherte Programm auf. Mit ALL wird die Liste ohne Seitenumbrüche angezeigt. Löscht einen Flash-Programmspeicherplatz. Löscht alle Flash-Programmspeicherplätze. Speichert das aktuelle Programm an dem angegebenen Flash-Speicherplatz. Lade ein Programm vom angegebenen Flash-Speicherplatz in den Programmspeicher. Führt das Programm im Flash-Speicherplatz n aus, löscht alle Variablen. Ändert den Programmspeicher nicht. Führt das Programm an Speicherplatz n aus und lässt alle Variablen so, wie sie sind (damit kann das Programm viel größer sein als der Programmspeicher). Ändert den Programmspeicher nicht. Hinweis: Wenn das verkettete Programm den Befehl READ benutzt, muss es vor dem ersten Lesen RESTORE aufrufen. Löscht einen Flash-Programmspeicherplatz und speichert dann das aktuelle Programm an dem angegebenen Flash-Speicherplatz. 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, mit Befehlen wie PEEK und MEMORY COPY unter Verwendung der Adresse des Flash-Speicherplatzes geladen und abgerufen werden. Wenn der optionale Parameter OVERWRITE (oder O) angegeben wird, wird der Inhalt des Flash-Slots überschrieben, ohne dass ein Fehler ausgegeben wird. Dieser Befehl lädt eine angegebene BMP-Datei im RGB121-Format in den Flash-Speicherplatz. Der Flash-Speicherplatz sollte vorher gelöscht worden sein, oder die Angabe des optionalen Parameters O oder OVERWRITE erzwingt eine Löschung. Beachte, dass die Firmware die ersten beiden Wörter im Flash-Speicherplatz auf die Breite und Höhe des gespeicherten Bildes setzt. 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. Mit diesem Befehl wird sichergestellt, dass bei einem Stromausfall nach einem Schreibbefehl keine Daten verloren gehen.
FONT [#]font-number, scaling	Damit wird die Standardschriftart für die Anzeige von Text auf einem LCD-Bildschirm oder der Videoausgabe festgelegt. Schriftarten werden als Zahl angegeben, z. B. #2 (das # ist optional). Details zu den verfügbaren Schriftarten findest du im Kapitel *„Grafikbefehle und -funktionen*”. „scaling” kann zwischen 1 und 15 liegen und vergrößert die Pixel, sodass die angezeigten Zeichen entsprechend breiter und höher werden. Bei einer Skalierung von 2 werden Höhe und Breite verdoppelt.
FOR counter = start TO finish [STEP increment]	Startet eine FOR-NEXT-Schleife, wobei der „Zähler” zunächst auf „start” gesetzt wird und in Schritten von „inkrement” (Standardwert ist 1) erhöht wird, bis der „Zähler” größer als „end” ist. Der „increment” kann eine ganze Zahl oder eine Gleitkommazahl sein. Beachte, dass die Verwendung einer Gleitkommazahl für „increment” Rundungsfehler im „counter” verursachen kann, was dazu führen kann, dass die Schleife zu früh oder zu spät beendet wird. „increment” kann negativ sein. In diesem Fall sollte „finish” kleiner als „start” sein, und die Schleife zählt rückwärts. Siehe auch den Befehl NEXT.
FRAMEBUFFER FRAMEBUFFER CREATE FRAMEBUFFER LAYER FRAMEBUFFER WRITE where/where$ FRAMEBUFFER CLOSE [welcher] FRAMEBUFFER COPY von, nach [,b]	<u>NICHT HDMI- UND VGA-VERSIONEN</u> 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 Grafikobjekte über die Hintergrundanzeige abzuspielen. Erstellt einen Framebuffer „F” mit einem RGB121-Farbraum und einer Auflösung, die zum konfigurierten SPI-Farbdisplay passt. Erstellt einen Framebuffer „L“ mit einem RGB121-Farbraum und einer Auflösung, die zum konfigurierten SPI-Farbdisplay passt. Legt das Ziel für nachfolgende Grafikbefehle fest. „where“ kann N, F oder L sein, wobei N die tatsächliche Anzeige ist. Es kann eine Zeichenfolgenvariable oder ein Literal verwendet werden. Schließt einen Framebuffer und gibt den Speicher frei. Der optionale Parameter „which“ kann F oder L sein. Wenn er weggelassen wird, werden beide geschlossen. Macht eine superoptimierte Vollbildkopie von einem Framebuffer auf einen anderen. „from” und „to” können N, F oder L sein, wobei N die physische Anzeige ist. Du kannst nur von N auf Displays kopieren, die BLIT und transparenten Text unterstützen. Die Firmware komprimiert oder erweitert die RGB-Auflösung automatisch, wenn du zwischen nicht übereinstimmenden Framebuffern kopierst. Natürlich gehen beim Kopieren von RGB565 nach RGB121 Informationen verloren, aber für viele Anwendungen (z. B. Spiele) sind 16 Farbstufen mehr als ausreichend.

FRAMEBUFFER WAIT	Beim Kopieren auf ein LCD-Display kann der optionale Parameter „b“ verwendet werden (FRAMEBUFFER COPY F/L, N, B). Dadurch wird die Firmware angewiesen, den Kopiervorgang mit der zweiten CPU im Raspberry Pi Pico durchzuführen, und die Steuerung kehrt sofort zum Basic-Programm zurück.
FRAMEBUFFER-ZUSAMMENFÜHREN [Farbe] [,Modus] [,Aktualisierungsrate]	Hält die Verarbeitung an, bis das LCD-Display in die Bildausblendung geht. Implementiert für ILI9341-, ST7789_320- und ILI9488-Displays. Wird verwendet, um Artefakte beim Schreiben auf den Bildschirm zu reduzieren. Kopiert den Inhalt des Layer-Puffers und des Framebuffers auf das LCD-Display und lässt dabei alle Pixel einer bestimmten Farbe weg. Voraussetzungen für den Befehl sind, dass sowohl FRAMEBUFFER als auch LAYERBUFFER erstellt wurden. FRAMEBUFFER MERGE – schreibt den Inhalt des Framebuffers auf den physischen Bildschirm und überschreibt dabei alle Pixel im Framebuffer, die im Layerbuffer gesetzt sind (nicht Null). FRAMEBUFFER MERGE col – schreibt den Inhalt des Framebuffers auf den physischen Bildschirm und überschreibt dabei alle Pixel im Framebuffer, die im Layerbuffer sind und nicht auf die transparente Farbe „col” gesetzt sind. Die Farbe wird als Zahl zwischen 0 und 15 angegeben, die Folgendes bedeutet: 0:SCHWARZ,1:BLAU,2:MYRTLE,3:KOBALT,4:MITTLERES GRÜN,5:CERULEAN,6:GRÜN,7:CYAN,8:ROT,9:MAGENTA,10:ROST,11:FUCHSIA,12:BRAUN,13:LILA,14:GELB,15:WEISS FRAMEBUFFER MERGE col,B – wie oben, nur dass die Übertragung auf den physischen Bildschirm auf der zweiten CPU passiert und die Steuerung sofort zu Basic zurückkehrt FRAMEBUFFER MERGE col,R [,updaterate] – Setzt die zweite CPU so, dass sie die physische Anzeige mit der Zusammenführung der beiden Puffer ständig aktualisiert. Setzt automatisch FRAMEBUFFER WRITE F, wenn nicht schon F oder L eingestellt ist. Standardmäßig wird der Bildschirm so schnell wie möglich aktualisiert (bei 200 MHz aktualisiert ein ILI9341 im SPI-Modus etwa 13 Mal pro Sekunde, im 8-Bit-Parallelmodus erreicht der ILI9341 27 FPS). Wenn „updaterate” eingestellt ist, wird der Bildschirm mit der in Millisekunden angegebenen Rate aktualisiert (es sei denn, diese ist geringer als die schnellstmögliche auf dem Display). Hinweis: FRAMEBUFFER WRITE kann nicht auf N gesetzt werden, während die kontinuierliche zusammengeführte Aktualisierung aktiv ist.
FRAMEBUFFER SYNC	FRAMEBUFFER MERGE col,A – bricht die kontinuierlichen Aktualisierungen ab. Außerdem wird die automatische Aktualisierung durch Löschen des Layerbuf oder Framebuffers, durch Strg-C oder durch END abgebrochen. Wartet, bis die letzte Hintergrundzusammenführung oder Hintergrundkopie auf der zweiten CPU abgeschlossen ist, um ein Zeichnen ohne Bildfehler zu ermöglichen.
FRAMEBUFFER	<u>NUR HDMI- UND VGA-VERSIONEN</u> Mit dem Framebuffer-Befehl kannst du einen Teil des variablen Speichers für Framebuffer, Layer-Puffer oder beides reservieren und diese dann auf coole Weise nutzen, um Tearing-Artefakte zu vermeiden und/oder Grafikobjekte über die Hintergrundanzeige abzuspielen.
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 zum aktuellen Anzeigemodus passen
FRAMEBUFFER LAYER [Farbe]	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 Zahl 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 Ebenenanwendung nicht unterstützt wird, fungiert ein Ebenenpuffer als weiterer Framebuffer.
FRAMEBUFFER-EBENE OBEN [Farbe]	Nur RP2350: Erstellt einen zweiten Ebenenpuffer „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 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 der zweiten Ebene nicht unterstützt wird, fungiert er als weiterer Bildspeicher.
FRAMEBUFFER WRITE wo/wo$	Gibt das Ziel für nachfolgende Grafikbefehle an. „where“ kann N, F, 2, T oder L sein, wobei N die tatsächliche Anzeige ist. Eine Zeichenfolgenvariable kann verwendet werden.
FRAMEBUFFER CLOSE [welcher]	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 von, nach [,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. Wenn der optionale Parameter „b” angegeben wird, wird die Verarbeitung angehalten, bis der Monitor in die Bildausblendung geht.
FRAMEBUFFER WARTEN	Hält die Verarbeitung an, bis die nächste Bildausblendung beginnt.

FUNCTION xxx (arg1 [,arg2, …]) [AS <type>}
  <Anweisungen>
  <Anweisungen>
  xxx = <Rückgabewert>
END FUNCTION
	Definiert eine aufrufbare Funktion. Das ist so, als würdest du MMBasic während der Ausführung deines Programms eine neue Funktion hinzufügen.
	„xxx" ist der Name der Funktion und muss den Regeln für die Benennung von Variablen entsprechen. Der Typ der Funktion kann durch einen Typ-Suffix (z. B. xxx$) oder durch Angabe des Typs mit AS <Typ> am Ende der Funktionsdefinition festgelegt werden. Beispiel:
	`FUNCTION xxx (arg1, arg2) AS STRING`
	'arg1', 'arg2' usw. sind die Argumente oder Parameter der Funktion (die Klammern sind immer erforderlich, auch wenn keine Argumente vorhanden sind). Ein Array wird mit leeren Klammern angegeben, z. B. arg3(). Der Typ des Arguments kann durch einen Typ-Suffix (z. B. arg1$) oder durch Angabe des Typs mit `AS <Typ> (z. B. arg1 AS STRING)` festgelegt werden.
	Das Argument kann auch eine andere definierte Funktion oder dieselbe Funktion sein, wenn Rekursion verwendet werden soll (der Rekursionsstapel ist begrenzt).
	Um den Rückgabewert der Funktion festzulegen, setzt man den Wert auf den Namen der Funktion. Zum Beispiel:
	`FUNCTION SQUARE(a)`
	`  SQUARE = a * a`
	`END FUNCTION`
	Jede Definition muss eine END FUNCTION-Anweisung haben. 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.
	Du benutzt die Funktion, indem du ihren Namen und ihre Argumente in einem Programm verwendest, genau wie bei einer normalen MMBasic-Funktion. Zum Beispiel:
	`PRINT QUADRAT(56,8)`
	Wenn die Funktion aufgerufen wird, wird jedes Argument im Aufrufer mit dem Argument in der Funktionsdefinition abgeglichen. Diese Argumente sind nur innerhalb der Funktion verfügbar.
	Funktionen können mit einer variablen Anzahl von Argumenten aufgerufen werden. Alle in der Funktionsliste ausgelassenen Argumente werden auf Null oder eine Null-Zeichenkette gesetzt.
	Argumente in der Liste des Aufrufers, die eine Variable sind und den richtigen Typ haben, werden per Referenz an die Funktion übergeben. Das heißt, dass alle Änderungen am entsprechenden Argument in der Funktion auch in die Variable des Aufrufers kopiert werden und daher nach Beendigung der Funktion noch zugänglich sind. 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 generiert, wenn dies nicht möglich ist.
	Arrays werden durch Angabe des Array-Namens mit leeren Klammern (z. B. arg()) übergeben, immer per Referenz und müssen vom richtigen Typ sein.
	Du darfst nicht mit Befehlen wie GOTO in eine Funktion springen oder aus ihr herausspringen. Das kann unvorhersehbare Nebenwirkungen haben, die dir den Tag verderben könnten.

GAMEPAD COLOUR Kanal, Farbe	Ändert die Farbe des Displayfelds auf einem PS4-Controller auf dem USB-Kanal „channel". „colour" wird als Standard-RGB888-Wert festgelegt, z. B. RGB(RED).

GAMEPAD HAPTIC Kanal links, rechts	Lässt die linken und rechten Vibrationsmotoren eines PS4-Controllers auf dem USB-Kanal „Kanal" laufen. „links" und „rechts" müssen eine Zahl zwischen 0 (aus) und 255 (maximal) sein.
GAMEPAD INTERRUPT ENABLE Kanal, int [,mask]	Aktiviert Interrupts beim Drücken der Tasten eines USB-Gamepads. Der optionale Parameter „mask" legt fest, welche Schalter den Interrupt auslösen (standardmäßig alle). „mask" ist eine Bitmap, die der Ausgabe der Funktion DEVICE(GAMEPAD channel,B) entspricht.
GAMEPAD INTERRUPT DISABLE Kanal	Deaktiviert Interrupts vom Gamepad auf dem angegebenen Kanal.
GAMEPAD MONITOR	Verwende GAMEPAD MONITOR, bevor du ein Gamepad anschließt. Nach dem Anschließen wird bei jeder Änderung der Tasten ein Vorher-Nachher-Bericht angezeigt.
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 diese Funktion, 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 Bitnummer an diesem Index für die Daten, die dem entsprechenden Bit entsprechen. Weitere Informationen zur Bitverwendung (0-15) findest du unter DEVICE(GAMEPAD n,B).
GOTO Ziel	Leitet die Programmausführung zum Ziel weiter, das eine Zeilennummer oder eine Bezeichnung sein kann.
GUI BITMAP x, y, Bits [, Breite] [, Höhe] [, Skalierung] [, c] [, bc]	Zeigt die Bits in einer Bitmap auf einem VGA/HDMI-Monitor oder LCD-Bildschirm an, beginnend bei „x" und „y" auf einem angeschlossenen Gerät. „height" und „width" sind die Abmessungen der Bitmap, wie sie auf dem Gerät angezeigt werden, und sind standardmäßig auf 8x8 eingestellt. „scale" ist optional und ist standardmäßig auf den Wert gesetzt, der mit dem Befehl FONT festgelegt wurde. „c" ist die Zeichenfarbe und „bc" ist die Hintergrundfarbe. Sie sind optional und sind standardmäßig auf die aktuellen Vordergrund- und Hintergrundfarben eingestellt. Die Bitmap kann eine Ganzzahl oder eine Zeichenfolgenvariable oder -konstante sein und wird mit dem ersten Byte als ersten Bits der obersten Zeile (zuerst Bit 7, dann Bit 6 usw.) gezeichnet, 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 oder GUI CALIBRATE a,b,c,d,d	NICHT VGA- UND HDMI-VERSIONEN Dieser Befehl wird verwendet, um die Touch-Funktion eines LCD-Bildschirms zu kalibrieren. Er zeigt eine Reihe von Zielen auf dem Bildschirm an und wartet darauf, dass jedes Ziel präzise berührt wird. Der Befehl kann auch mit fünf Argumenten verwendet werden, die die Kalibrierungswerte angeben. In diesem Fall wird die Kalibrierung durchgeführt, ohne dass Ziele angezeigt werden oder eine Eingabe vom Benutzer erforderlich ist. Um die Werte zu ermitteln, verwenden Sie die OPTION LIST, **nachdem** Sie das Display normal kalibriert haben. Beachten Sie, dass diese Werte für dieses Display spezifisch sind und erheblich variieren können.

GUI-TEST LCDPANEL	Testet ein Anzeigegerät (LCD, VGA usw.). Es zeichnet kontinuierlich eine animierte Anzeige mit Farbcirkeln auf dem Display.
GUI RESET LCDPANEL	NICHT VGA- UND HDMI-VERSIONEN Reinitialisiert das konfigurierte LCD-Panel. 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 dann kann dieser Befehl verwendet werden, um das Display neu zu initialisieren.
GUI-TEST TOUCH	NICHT VGA- UND HDMI-VERSIONEN Testet die Touch-Funktion des LCD-Bildschirms. Der Bildschirm wird gelöscht und MMBasic wartet auf eine Berührung, die einen weißen Punkt auf dem Display anzeigt, der die genaue Berührungsposition auf dem Bildschirm markiert. Jedes in die Konsole eingegebene Zeichen beendet den Test.
HELP suchtext	Der Befehl „help" sucht nach einer Datei namens „help.txt" auf Laufwerk A:. Diese Datei kann vom Benutzer oder von der Community erstellt worden sein und muss ein bestimmtes Format haben. Bei jedem Hilfseintrag muss die erste Zeile eine Suchzeichenfolge sein, der ein ~-Zeichen vorangestellt ist. Diese wird von der Hilfefunktion zum Auffinden eines Eintrags verwendet und nicht angezeigt. Der „Suchtext" kann ? für die Ersetzung eines einzelnen Zeichens oder * für die Ersetzung mehrerer Zeichen (oder keiner) enthalten. Nach dem Suchstring gibt die nächste Zeile normalerweise die Syntax eines bestimmten Befehls oder einer bestimmten Funktion an. Alle folgenden Zeilen sind weitere Erläuterungen. z. B. *~COLOR* *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, wenn nicht angegeben, standardmäßig schwarz angezeigt.* Der Befehl gibt alle Einträge zurück, die mit dem „searchtext" übereinstimmen, und diese werden so paginiert, dass sie zum Konsolengerät passen. Verschiedene Versionen von help.txt sind verfügbar unter https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=17865
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). „pin" ist der mit dem Sensor verbundene I/O-Pin. Jeder I/O-Pin kann verwendet werden. „tvar" ist die Variable, die die gemessene Temperatur speichert, und „hvar" ist das Gleiche für die Luftfeuchtigkeit. Beide müssen vorhanden sein und beide müssen Fließkommavariablen sein. Beispiel: `HUMID 3, TEMP!, HUMIDITY!` Die Temperatur wird in ºC gemessen und die Luftfeuchtigkeit in Prozent relativer Luftfeuchtigkeit. Beide werden mit einer Auflösung von 0,1 gemessen. Wenn ein Fehler auftritt (Sensor nicht angeschlossen oder fehlerhaftes Signal), werden beide Werte auf 1000,0 gesetzt. Normalerweise sollte der DHT22 mit 3,3 V betrieben werden, damit sein Ausgang für den Raspberry Pi Pico unter 3,6 V bleibt (beim Pico 2 gibt's dieses

	Problem nicht), und der Signalpin sollte mit einem 1K- bis 10K-Widerstand (4,7K empfohlen) auf 3,3 V hochgezogen werden. Der optionale Parameter DHT11 ändert die Timings, damit er 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 Geschwindigkeit, Timeout	Aktiviert das erste I²C-Modul im Master-Modus. „speed" ist die zu verwendende Taktrate (in KHz) und muss entweder 100, 400 oder 1000 sein. „timeout" ist ein Wert in Millisekunden, nach dessen Ablauf die Sende- und Empfangskommandos des Masters unterbrochen werden, wenn sie nicht abgeschlossen sind. Der Mindestwert ist 100. Der Wert Null deaktiviert das Timeout (dies wird jedoch nicht empfohlen).
I2C WRITE addr, option, sendlen, senddata [,sendata ..]	Sendet Daten an das I²C-Slave-Gerät. „addr" ist die I²C-Adresse des Slaves. „option" kann 0 für den normalen Betrieb oder 1 sein, um die Kontrolle über den Bus nach dem Befehl zu behalten (eine Stoppbedingung wird nach Abschluss des Befehls nicht gesendet). „sendlen" ist die Anzahl der zu sendenden Bytes. „senddata" sind die zu sendenden Daten – diese können auf verschiedene Weise angegeben werden (alle gesendeten Werte liegen zwischen 0 und 255). Hinweise: • 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 angegeben werden, das mit leeren Klammern (d. h. ohne Dimensionen) angegeben wird. Die „sendlen"-Bytes des Arrays werden beginnend mit dem ersten Element gesendet. Beispiel: `I2C WRITE &H6F, 0, 3, ARRAY()` Die Daten können eine String-Variable sein (keine Konstante). Beispiel: `I2C WRITE &H6F, 0, 3, STRING$`
I2C READ addr, option, rcvlen, rcvbuf	Ruft Daten vom I2C-Slave-Gerät ab. „addr" ist die I²C-Adresse des Slaves. „option" kann 0 für den normalen Betrieb oder 1 sein, um die Kontrolle über den Bus nach dem Befehl zu behalten (eine Stoppbedingung wird nach Abschluss des Befehls nicht gesendet). „rcvlen" ist die Anzahl der zu empfangenden Bytes. „rcvbuf" ist die Variable oder das Array, in dem die empfangenen Daten gespeichert werden – das kann sein: • Eine Zeichenfolgenvariable. Die Bytes werden als aufeinanderfolgende Zeichen gespeichert. • Ein eindimensionales Array von Zahlen, das mit leeren Klammern angegeben wird. 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 CHECK addr	Setzt die schreibgeschützte Variable MM.I2C auf 0, wenn ein Gerät auf die Adresse „addr" antwortet. MM.I2C wird auf 1 gesetzt, wenn keine Antwort kommt.

I2C CLOSE I2C SLAVE	Deaktiviert das Master-I²C-Modul. Dieser Befehl sendet auch ein Stopp-Signal, wenn der Bus noch gehalten wird. Siehe Anhang B
I2C2	Die gleichen Befehle wie für I2C (oben), aber für den zweiten I²C-Kanal.
IF expr THEN stmt [: stmt] oder IF expr THEN stmt ELSE stmt	Wertet den Ausdruck „expr" aus und führt die Anweisung nach dem Schlüsselwort THEN aus, wenn er wahr ist, oder springt zur nächsten Zeile, wenn er falsch ist. Wenn weitere Anweisungen in der Zeile vorhanden sind (getrennt durch Doppelpunkte (:)), werden diese ebenfalls ausgeführt, wenn sie wahr sind, oder übersprungen, wenn sie falsch sind. Das Schlüsselwort ELSE ist optional. Wenn es vorhanden ist, werden die darauf folgenden Anweisungen ausgeführt, wenn „expr" als falsch ausgewertet wird. Die Konstruktion „THEN-Anweisung" kann auch ersetzt werden durch: GOTO Zeilennummer | Label'. Diese Art von IF-Anweisung steht komplett in einer Zeile.

IF Ausdruck THEN
<Anweisungen>
[ELSEIF Ausdruck THEN
<Anweisungen>]
[ELSE
<Anweisungen>]
ENDIF
	Mehrzeilige IF-Anweisung mit optionalen ELSE- und ELSEIF-Fällen, die mit ENDIF endet. Jede Komponente steht in einer eigenen Zeile.
	Wertet „Ausdruck" aus und führt die Anweisung(en) nach THEN aus, wenn der Ausdruck wahr ist, oder optional die Anweisung(en) nach der ELSE-Anweisung, wenn er falsch ist. Die ELSEIF-Anweisung (falls vorhanden) wird ausgeführt, wenn die vorherige Bedingung falsch ist, und 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 [,inkrement]	Erhöht die Variable „var" entweder um 1 oder, falls angegeben, um den Wert in „increment". „increment" kann negativ sein, was zu einer Verringerung führt. Funktional entspricht dies `var = var + increment,` wird aber viel schneller verarbeitet.
INPUT ["prompt$";] var1 [,var2 [, var3 [, etc]]]	Nimmt eine Liste von Werten, die durch Kommas (,) getrennt sind und über die Konsole eingegeben werden, und ordnet sie einer sequenziellen Liste von Variablen zu. Wenn der Befehl zum Beispiel lautet: INPUT a, b, c und Folgendes über die Tastatur eingegeben wird: 23, 87, 66 Dann ist a = 23 und b = 87 und c = 66 Die Liste der Variablen kann eine Mischung aus Float-, Integer- oder String-Variablen sein. Die in die Konsole eingegebenen Werte müssen dem Typ der Variablen entsprechen. Wenn ein einzelner Wert eingegeben wird, ist kein Komma erforderlich (dieser Wert darf jedoch kein Komma enthalten). „prompt$" ist eine Zeichenfolgenkonstante (keine Variable oder kein Ausdruck) und wird, wenn angegeben, zuerst ausgegeben. Normalerweise wird die Eingabeaufforderung mit einem Semikolon (;) beendet, und in diesem Fall wird nach der Eingabeaufforderung ein Fragezeichen ausgegeben. Wenn die Eingabeaufforderung mit einem Komma (,) statt mit einem Semikolon (;) beendet wird, wird das Fragezeichen unterdrückt.
INPUT #nbr, Liste von Variablen	Wie oben, nur dass die Eingabe von einem seriellen Port oder einer Datei gelesen wird, die vorher für INPUT als „nbr" geöffnet wurde. Sieh dir den Befehl OPEN an.

INTERRUPT [myint]	Dieser Befehl löst eine Software-Unterbrechung aus. Die Unterbrechung wird mit INTERRUPT „myint" eingerichtet, wobei „myint" der Name einer Subroutine ist, die ausgeführt wird, wenn die Unterbrechung ausgelöst wird. Verwende INTERRUPT 0, um den Interrupt zu deaktivieren. Verwende INTERRUPT ohne Parameter, um den Interrupt auszulösen. Hinweis: Der Interrupt kann auch aus einem CSUB
IR dev, key , int oder IR CLOSE	Decodiert NEC- oder Sony-Infrarot-Fernbedienungssignale. Ein IR-Empfängermodul ist nötig, um das IR-Licht zu erfassen und das Signal zu demodulieren. Es kann an jeden Pin angeschlossen werden, aber dieser Pin muss vorher mit dem Befehl `SETPIN n, IR` Die Dekodierung des IR-Signals läuft 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). „int" ist eine benutzerdefinierte Subroutine, die aufgerufen wird, wenn eine neue Taste gedrückt 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. Der Befehl „IR CLOSE" beendet den IR-Decoder. 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 die normale Verwendung, aber wenn Sie nach einem bestimmten numerischen Code suchen, der von einem Hersteller bereitgestellt wird, sollten Sie die Bits umkehren. Wie das geht, wird hier beschrieben: http://www.thebackshed.com/forum/forum_posts.asp?TID=8367 Weitere Infos findest du im Kapitel *„Spezielle Hardwaregeräte*".
IR-SEND-Pin, dev, key	Erzeugt ein 12-Bit-Infrarotsignal nach dem Protokoll der Sony-Fernbedienung. „pin" ist der zu verwendende I/O-Pin. Das kann jeder I/O-Pin sein, der automatisch als Ausgang konfiguriert wird und an eine Infrarot-LED angeschlossen werden sollte. Im Ruhezustand ist der Pegel niedrig, hohe Pegel zeigen an, wann die LED eingeschaltet werden soll. „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. 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] oder KEYPAD CLOSE	Überwacht und entschlüsselt Tastendrücke auf einer 4x3- oder 4x4-Tastatur. Die Überwachung der Tastatur läuft im Hintergrund und das Programm läuft nach diesem Befehl ohne Unterbrechung weiter. „var" sollte eine numerische Variable sein und ihr Wert wird jedes Mal aktualisiert, wenn ein Tastendruck erkannt wird. „int" ist eine benutzerdefinierte Subroutine, die aufgerufen wird, wenn ein neuer Tastendruck empfangen wird. In der Interrupt-Subroutine kann das Programm die Variable „var" überprüfen und entsprechende Maßnahmen ergreifen. r1, r2, r3 und r4 sind Pin-Nummern, die für die vier Reihenverbindungen zum Tastenfeld verwendet werden, und c1, c2, c3 und c4 sind die Spaltenverbindungen. c4 ist optional und wird nur bei 4x4-Tastenfeldern verwendet. Dieser Befehl konfiguriert diese Pins automatisch wie erforderlich. 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 der Wert 20 für A, 21 für B, 22 für C und 23 für D zurückgegeben. Der Befehl KEYPAD CLOSE beendet die Tastaturfunktion und setzt den I/O-Pin wieder auf einen nicht konfigurierten Zustand zurück. Weitere Infos findest du im Abschnitt *„Spezielle Hardwaregeräte*".

KEYPAD keymapmap!(), var!,int, startcolpin, nocols, startrowpin, norows	NUR RP2350-VERSIONEN
	Konfigurier und verwende eine Tastatur mit einer beliebigen Anzahl von Zeilen und Spalten und weise jeder Taste benutzerdefinierte Rückgabecodes zu.
	„keymapmap!()" ist ein zweidimensionales Array von Floats, das die vom Benutzer angegebene Zuordnung für jede Taste enthält. „ " Die Array-Dimensionen müssen mit der angegebenen Anzahl von Spalten und Zeilen übereinstimmen.
	„var!" ist die Float-Variable, die den zurückgegebenen Wert enthält. „int" ist die Interrupt-Routine, die beim Drücken einer Taste aufgerufen wird.
	'startcolpin' ist der Pin, der den Bereich der zusammenhängenden GP-Pins beginnt, die für den Spaltenscan verwendet werden, während 'nocols' die Anzahl der Pins in diesem Bereich ist. 'startrowpin' ist der Pin, der 'norows' zusammenhängender GP-Pins beginnt, die für den Zeilenscan verwendet werden.
	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.
	Das Programm könnte so aussehen:
	`OPTION BASIS 1`
	`DIM keymap(3,4)=(1,4,7,10, 4,5,6,0, 3,6,9,11)`
	`KEYPAD-Tastenbelegung(), keyret, myint, GP1, 3, GP4, 4`
	`DO`
	`  ' Hauptprogramm-Verarbeitungsschleife`
	`LOOP`
	`SUB myint`
	`  PRINT keyret`
	`END SUB`

KILL file$ [,all]	Löscht die durch „file$" angegebene Datei. Die Dateiendung muss angegeben werden. Wenn fname$ ein „*" oder ein „?" enthält, wird eine Massenlöschung gestartet. Wenn du den optionalen Parameter „all" benutzt, musst du nur einmal bestätigen. Wenn „all" nicht angegeben ist, musst du für jede Datei einzeln bestätigen.
LCD INIT d4, d5, d6, d7, rs, en oder LCD line, pos, text$ oder LCD CLEAR oder LCD CLOSE	Zeigt Text auf einem LCD-Zeichenanzeigemodul an. Dieser Befehl funktioniert mit den meisten 1-zeiligen, 2-zeiligen oder 4-zeiligen LCD-Modulen, die den Controller KS0066, HD44780 oder SPLC780 verwenden (dies kann jedoch nicht garantiert werden). Der Befehl LCD INIT wird verwendet, um das LCD-Modul für die Verwendung zu initialisieren. „d4" bis „d7" sind die E/A-Pins, die mit den Eingängen D4 bis D7 des LCD-Moduls verbunden sind (die Eingänge D0 bis D3 sollten mit Masse verbunden sein). „rs" ist der Pin, der mit dem Registerauswahleingang des Moduls verbunden ist (manchmal auch als CMD bezeichnet). „en" ist der Pin, der mit dem Freigabe- oder Chipauswahleingang des Moduls verbunden ist. Der R/W-Eingang des Moduls sollte immer geerdet sein. Die oben genannten I/O-Pins werden durch diesen Befehl automatisch als Ausgänge festgelegt. 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 Zeichenfolge, die den Text enthält, der auf das LCD-Display geschrieben werden soll. „pos" kann auch C8, C16, C20 oder C40 sein. In diesem Fall wird die Zeile gelöscht und der Text auf einem Display mit 8, 16, 20 oder 40 Zeilen zentriert. Beispiel: `LCD 1, C16, „Hallo"` LCD CLEAR löscht alle auf dem LCD angezeigten Daten und LCD CLOSE beendet die LCD-Funktion und setzt alle E/A-Pins wieder auf den nicht konfigurierten Zustand zurück. Weitere Infos findest du im Kapitel *„Spezielle Hardwaregeräte*".
LCD CMD d1 [, d2 [, etc]] oder LCD DATA d1 [, d2 [, etc]]	Diese Befehle senden ein oder mehrere Bytes als Befehl (LCD CMD) oder als Daten (LCD DATA) an ein LCD-Display. Jedes Byte ist eine Zahl zwischen 0 und 255 und muss durch Kommas getrennt werden. Das LCD muss zuvor mit dem Befehl LCD INIT initialisiert worden sein (siehe oben). Diese Befehle können verwendet werden, um ein nicht standardmäßiges LCD im „Rohmodus" anzusteuern, oder um spezielle Funktionen wie Scrollen, Cursor und benutzerdefinierte Zeichensätze zu aktivieren. Die erforderlichen Befehls- und Datenwerte findest du im Datenblatt deines LCD.
LET Variable = Ausdruck	Weist der Variablen den Wert von „Ausdruck" zu. LET wird automatisch angenommen, wenn eine Anweisung nicht mit einem Befehl beginnt. Beispiel: `Var = 56`
LIBRARY SAVE oder LIBRARY DELETE oder BIBLIOTHEK AUFZEICHNEN	Die Bibliothek ist ein spezieller Teil des Programmspeichers, der Programmcode wie Unterprogramme, Funktionen und CFunktionen enthalten kann. Diese Routinen sind für den Programmierer nicht sichtbar, stehen aber dem laufenden Programm zur Verfügung und funktionieren genauso wie die in MMBasic eingebauten Befehle und Funktionen. Jeder Code in der Bibliothek, der nicht in einer Unterroutine oder Funktion enthalten ist, wird unmittelbar vor der Ausführung eines Programms ausgeführt. Dies kann zum Initialisieren von Konstanten, zum Festlegen von Optionen usw. verwendet werden. Eine ausführliche Erläuterung findest du

oder LIBRARY LIST ALL oder BIBLIOTHEKS-DISK SPEICHERN fname$ oder LIBRARY DISK LOAD fname$	unter der Überschrift *„Die Bibliothek”* in diesem Handbuch. 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 sind weiterhin verfügbar). LIBRARY SAVE nimmt alles, was sich im normalen Programmspeicher befindet, komprimiert es (entfernt redundante Daten wie Kommentare) und hängt es an den Bibliotheksbereich an (der Hauptprogrammspeicher ist dann leer). Der Code in der Bibliothek wird nicht in LIST oder EDIT angezeigt und wird nicht gelöscht, wenn ein neues Programm geladen oder NEW verwendet wird. LIBRARY DELETE entfernt die Bibliothek und gibt Flash-Steckplatz 3 für die normale Verwendung zurück (OPTION RESET macht dasselbe). LIBRARY LIST listet den Inhalt der Bibliothek auf. Verwende ALL, um ohne Seitenbestätigungen aufzulisten. LIBRARY DISK SAVE fname$ speichert die aktuelle Bibliothek als Binärdatei, sodass sie später mit LIBRARY DISK LOAD fname$ wiederhergestellt werden kann. So lassen sich Bibliotheken für einzelne Programme einfach speichern, wiederherstellen und verteilen. Außer den versionsspezifischen Funktionen in der Bibliothek (WEB, VGA, GUI) können Bibliotheken auch versionsübergreifend genutzt werden.
LINE x1, y1, x2, y2 [,[-] LW [, C]]	Zeichnet eine Linie auf dem Display, die bei den Koordinaten „x1” und „y1” anfängt und bei „x2” und „y2” endet. „LW” ist die Breite der Linie. Wenn nichts angegeben ist, ist der Standardwert 1. Bei Linien mit einer definierten Breite bestimmen die Koordinaten x1 und y1 den Pixel oben links der dicken Linie. Das heißt, die Linie befindet sich rechts von der angegebenen Position oder darunter auf dem Bildschirm. Wenn die Breite als negativer Wert angegeben wird, gilt die Breite für Linien in alle Richtungen, und sie werden auf die angegebenen Start- und Zielkoordinaten zentriert. „C“ ist eine ganze Zahl, die die Farbe angibt, und ist standardmäßig auf die aktuelle Vordergrundfarbe eingestellt. Alle Parameter können als Arrays ausgedrückt werden, und die Software zeichnet die Anzahl der Linien, die durch die Abmessungen des kleinsten Arrays bestimmt wird. „x1”, „y1”, „x2” und „y2” müssen alle Arrays oder alle einzelne Variablen/Konstanten sein, sonst wird ein Fehler erzeugt. „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 sind die gleichen wie beim Befehl LINE oben. Diese Version nutzt aber variable Intensitätswerte der angegebenen Farbe, um die „versetzte” Qualität von 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(), Farbe	Dieser Befehl erzeugt ein Liniendiagramm der in „x()” und „y()” angegebenen Koordinatenpaare. Das Diagramm hat n-1 Segmente, wobei die Arrays x und y n Elemente enthalten.
LINE INPUT [prompt$,] string-variable$	Liest eine ganze Zeile aus der Konsoleneingabe in „string-variable$”. „prompt$“ ist eine Zeichenfolgenkonstante (keine Variable oder kein Ausdruck) und wird, wenn angegeben, zuerst ausgegeben. Ein Fragezeichen wird nur ausgegeben, wenn es Teil von „prompt$“ ist. Im Gegensatz zu INPUT liest dieser Befehl eine ganze Zeile und hält nicht bei durch Kommas getrennten Datenelementen an.

LINE INPUT #nbr, string-variable$	Wie oben, nur dass die Eingabe von einem seriellen Kommunikationsport oder einer Datei gelesen wird, die vorher für INPUT als „nbr" geöffnet wurde. Sieh dir den Befehl OPEN an.
LINE PLOT ydata() [,nbr] [,xstart] [,xinc] [,ystart] [,yinc] [,colour]	Zeichnet ein Liniendiagramm aus einem Array von Datenpunkten der y-Achse. „ydata()" ist ein Array aus Floats oder Integers, die gezeichnet werden sollen. „nbr" ist die Anzahl der zu zeichnenden Liniensegmente – standardmäßig ist das der kleinere Wert aus der Array-Größe und MM.HRES-2, wenn nichts angegeben ist. „xstart" ist die x-Koordinate, bei der mit dem Zeichnen begonnen werden soll – Standardwert ist 0 „xinc" ist die Schrittweite entlang der x-Achse, um jede Koordinate zu zeichnen – Standardwert ist 1 „ystart" ist die Position in ydata, an der die Darstellung beginnen soll – Standardwert ist der Array-Anfang. Hinweis: berücksichtigt OPTION BASE „yinc" ist die Schrittweite für den Index in ydata, die für jeden zu zeichnenden Punkt hinzugefügt wird „colour" ist die Farbe, in der die Linie gezeichnet wird
LIST [fname$] oder LIST ALL [fname$]	Zeigt ein Programm auf der Konsole an. LIST allein listet das Programm mit einer Pause nach jedem Bildschirm voll auf. LIST ALL zeigt das Programm ohne Pausen an. Das ist praktisch, wenn du das Programm auf einen Terminalemulator auf einem PC übertragen willst, der den Eingabestrom in eine Datei speichern kann. Wenn du das optionale „fname$" angibst, wird die Datei im Flash-Dateisystem oder auf der SD-Karte aufgelistet.
LIST PINS	Listet den aktuellen Status aller Pins auf dem Prozessor auf.
LIST SYSTEM I2C	Listet eine Übersicht aller I2C-Geräte auf, die mit dem System-I2C-Bus verbunden sind.
LIST COMMANDS oder LIST FUNCTIONS	Zeigt alle gültigen Befehle oder Funktionen an
LIST VARIABLES [s%()]	Zeigt alle globalen Variablen und Konstanten an. Wenn der Befehl in einer Subroutine aufgerufen wird, werden auch die Variablen angezeigt, die von dieser Subroutine und allen sie aufrufenden Subroutinen verwendet werden. Wenn der optionale Parameter s$() verwendet wird, werden die Variablen so aufgelistet, dass s%() als Longstring behandelt wird (siehe Befehl LONGSTRING).
LMID(array%(),start [,num])=string$	Fügt „string$" an der Stelle „start" in den langen String „array%()" ein und ersetzt dabei „num" vorhandene Zeichen. Wenn „num" nicht angegeben ist, wird es anhand der Länge von „string$" berechnet. „num" kann 0 sein. In diesem Fall wird „string$" an der angegebenen Stelle eingefügt.

LOAD file$ [,R]	Lädt ein Programm namens „file$" aus dem Flash-Dateisystem oder von der SD-Karte in den Programmspeicher. Wenn das optionale Suffix ‚R hinzugefügt wird, wird das Programm sofort ohne Aufforderung ausgeführt (in diesem Fall muss „file$" eine String-Konstante sein). Der Befehl RUN macht dasselbe und erlaubt die Verwendung einer String-Variablen. Wenn keine Erweiterung angegeben ist, wird „.BAS" an den Dateinamen angehängt.
LOAD CONTEXT [KEEP]	Stellt den Variablenbereich auf den zuvor gespeicherten Zustand zurück und behält optional die gespeicherten Variablen bei, um bei Bedarf ein zweites LOAD zu ermöglichen. Siehe auch SAVE CONTEXT
LOAD IMAGE/BMP fname$ [,x] [,y] [,mode] [,ximage] [,yimage]	Lädt eine BMP-Datei aus „fname$" und schreibt sie auf das aktuelle Ausgabegerät (Anzeige oder Framebuffer), beginnend bei „x", „y" (Standardwert ist 0,0). Der optionale Parameter „mode" gibt an, ob die Ausgabe gedithert werden soll. Die Bits 0 und 1 legen das Ausgabeformat fest, und Bit 2 bestimmt die Art des zu verwendenden Ditherings. Standardmäßig ist der Modus auf -1 gesetzt, was bedeutet, dass kein Dithering angewendet werden soll. DITHER_FLOYD_STEINBERG_RGB121 0 DITHER_FLOYD_STEINBERG_RGB222 1 DITHER_FLOYD_STEINBERG_RGB332 2 DITHER_ATKINSON_RGB121 4 DITHER_ATKINSON_RGB222 5 DITHER_ATKINSON_RGB332 6 „ximage" und „yimage" sagen, wo in der Bilddatei mit dem Schreiben auf den Bildschirm angefangen werden soll (Standard ist 0,0). Wenn keine Erweiterung angegeben wird, wird „.BMP" an den Dateinamen angehängt. Alle Arten des BMP-Formats werden unterstützt, einschließlich Schwarzweiß- und Echtfarben-24-Bit-Bildern.
LOAD JPG file$ [, x] [, y] [,mode] [,ximage] [,yimage]	Lädt eine JPG-Datei von „fname$" und schreibt sie auf das aktuelle Ausgabegerät (Display oder Framebuffer), beginnend bei „x", „y" (Standardwert ist 0,0). Der optionale Parameter „mode" sagt, ob die Ausgabe gedithert werden soll. Die Bits 0 und 1 legen das Ausgabeformat fest, und Bit 2 bestimmt die Art des zu verwendenden Ditherings. Standardmäßig ist der Modus auf -1 gesetzt, was bedeutet, dass kein Dithering angewendet werden soll. DITHER_FLOYD_STEINBERG_RGB121 0 DITHER_FLOYD_STEINBERG_RGB222 1 DITHER_FLOYD_STEINBERG_RGB332 2 DITHER_ATKINSON_RGB121 4 DITHER_ATKINSON_RGB222 5 DITHER_ATKINSON_RGB332 6 „ximag" und „yimage" sagen, wo in der Bilddatei mit dem Schreiben auf den Bildschirm angefangen werden soll (Standard ist 0,0). Wenn keine Erweiterung angegeben wird, wird „.JPG" an den Dateinamen angehängt. Progressive JPG-Bilder werden nicht unterstützt.

LOAD PNG fname$ [, x] [, y] [,transparent] [,alphacut]	NUR RP2350-VERSIONEN Lädt und zeigt eine PNG-Datei „fname“ an. Wenn keine Dateiendung angegeben wird, wird automatisch „.png“ an den Dateinamen angehängt. Die Datei muss im RGBA8888-Format vorliegen, was die normale Standardeinstellung ist. Wenn angegeben, geben „x” und „y” an, wo auf dem Display oder im Framebuffer das Bild angezeigt wird. Der optionale Parameter „transparent” (Standardwert 0) gibt einen der Farbcodes (0–15) an, die den Pixeln in der PNG-Datei mit einem Alpha-Wert unter „alphacut” (Standardwert 20) zugewiesen werden. Wenn „transparent” auf -1 gesetzt ist, wird das PNG-Bild so geschrieben, dass Pixel mit einem Alpha-Wert unter „alphacut” komplett weggelassen werden.
LOCAL variable [, variables] Die vollständige Syntax findest du unter DIM.	Definiert eine Liste von Variablennamen als lokal für die Subroutine oder Funktion. Dieser Befehl verwendet genau die gleiche Syntax wie DIM und erstellt Variablen, die nur innerhalb der Subroutine oder Funktion sichtbar sind. Sie werden automatisch gelöscht, wenn die Subroutine oder Funktion beendet wird.
LOCATION	Legt einen manuellen Standort (Breitengrad/Längengrad) und eine Zeit für die Verwendung mit dem Befehl ASTRO fest. Dies ist einer von mehreren Befehlen, die hochpräzise astronomische Berechnungen durchführen, die für die Ausrichtung oder Navigation von Teleskopen geeignet sind. Eine detaillierte Beschreibung findest du in der Datei *GPS_Astro_Reference.pdf*, die im ZIP-Archiv mit der Firmware enthalten ist.
LONGSTRING	Die LONGSTRING-Befehle ermöglichen die Bearbeitung von Zeichenfolgen, die länger als die normale MMBasic-Grenze von 255 Zeichen sind. Variablen zum Speichern langer Zeichenfolgen müssen als eindimensionale Integer-Arrays definiert werden, wobei die Anzahl der Elemente auf die Anzahl der Zeichen festgelegt wird, die für die maximale Zeichenfolgenlänge erforderlich sind, geteilt durch acht. Der Grund für die Division durch acht ist, dass jede Ganzzahl in einem MMBasic-Array acht Bytes belegt. Beachte, dass die Routinen für lange Zeichenfolgen nicht auf Überlauf in 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 die Longstring in in%() und speichert das Ergebnis in out%(). Für die Modi CBC und CTR erzeugt die Verschlüsselung einen zufälligen Initialisierungsvektor und hängt out%() mit dem IV vor. Wenn ein expliziter IV angegeben wird, wird dieser anstelle des zufälligen Vektors verwendet und an out%() angehängt. Bei der CBC- und CTR-Entschlüsselung geht die Firmware davon aus, dass die ersten 16 Bytes von in%() der Initialisierungsvektor sind. Wenn du eine Nachricht ohne IV übertragen willst, 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 verwenden kann. Dies kann mit LONGSTRING CONCAT erfolgen, um die eingehende Nachricht zu einem Longstring hinzuzufügen, 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. Durch die Codierung wird die Länge um 4/3 erhöht, durch die Decodierung um 3/4 verringert.
LONGSTRING CLEAR array %()	Löscht die Longstring-Variable „array%()". Das heißt, sie wird auf eine leere Zeichenfolge gesetzt.
LONGSTRING COPY dest%(), src%()	Kopiere eine lange Zeichenfolge in eine andere. „dest%()" ist die Zielvariable und „src%()" ist die Quellvariable. Der Inhalt von „dest%()" wird überschrieben.
LONGSTRING CONCAT dest%(), src%()	Füge eine lange Zeichenfolge an eine andere an. „dest%()" ist die Zielvariable und „src%()" ist 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 lange Zeichenfolgenvariable sein.
LONGSTRING LEFT dest%(), src%(), nbr	Kopiert die ersten 'nbr' Zeichen von 'src%()' nach 'dest%()' und überschreibt alles, was sich in 'dest%()' befand. , d. h. Kopieren vom Anfang von 'src%()'. 'src%()' und 'dest%()' müssen Long-String-Variablen sein. 'nbr' muss eine ganzzahlige Konstante oder ein Ausdruck sein.
LONGSTRING LOAD array %(), nbr, string$	Kopiert 'nbr' Zeichen von „string$" in die lange Zeichenfolgenvariable „array%()" und überschreibt dabei alles, was sich in „array%()" befand.
LONGSTRING MID dest%(), src%(), start, nbr	Kopiert 'nbr' Zeichen von 'src%()' nach 'dest%()', beginnend bei der Zeichenposition 'start', und überschreibt dabei den bisherigen Inhalt von 'dest%()'. Kopiert also aus der Mitte von 'src%()'. „nbr" ist optional. Wenn es weggelassen wird, werden die Zeichen von „start" bis zum Ende der Zeichenkette kopiert. „src%()" und „dest%()" müssen lange Zeichenkettenvariablen sein. „start" und „nbr" müssen ganzzahlige Konstanten oder Ausdrücke sein.
LONGSTRING PRINT [#n,] src%() [;]	Druckt die in „src%()" gespeicherte Longstring in die Datei oder den COM-Port, die bzw. der als „#n" geöffnet ist. Wenn „#n" nicht angegeben ist, wird die Ausgabe an die Konsole gesendet. Füge ein Semikolon hinzu, um CR/LF zu unterdrücken.
LONGSTRING REPLACE array%() , string$, start	Ersetzt Zeichen in der normalen MMBasic-Zeichenkette „string$" in einer vorhandenen langen Zeichenkette „array%()", beginnend an der Position „start" in der langen Zeichenkette.
LONGSTRING RESIZE addr%(), nbr	Setzt die Größe des Longstrings auf „nbr". Das überschreibt die Größe, die von anderen Longstring-Befehlen festgelegt wurde, also sei vorsichtig damit. Typischerweise wird das benutzt, wenn man einen Longstring als Byte-Array verwenden will.
LONGSTRING RIGHT dest%(), src%(), nbr	Kopiert die rechten „nbr" Zeichen von „src%()" nach „dest%()" und überschreibt dabei den Inhalt von „dest%()". Das heißt, es wird vom Ende von „src%()" kopiert. „src%()" und „dest%()" müssen Longstring-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	Schneidet „nbr“ Zeichen vom Anfang einer langen Zeichenkette ab. „array%()“ muss eine lange Zeichenfolgenvariable sein. „nbr“ muss eine ganzzahlige Konstante oder ein Ausdruck sein.
LONGSTRING UCASE array%()	Wandelt alle Kleinbuchstaben in „array%()” in Großbuchstaben um. „array%()” muss eine lange Zeichenfolgenvariable sein.
LOOP [UNTIL Ausdruck]	Beendet eine Programmschleife: siehe DO.
MAP	NUR HDMI-VERSION 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. RGB555-Format) eingestellt werden. Weitere Infos findest du unter der MAP-Funktion.
MAP(n) = rgb%	Damit wird allen Pixeln mit dem 4- oder 8-Bit-Farbwert „n” die 24-Bit-Farbe „rgb%” zugewiesen. Die Änderung wird nach dem Befehl MAP SET aktiviert.
MAP MAXIMITE	Damit wird die Farbtabelle auf die Farben eingestellt, die im ursprünglichen Colour Maximite verwendet wurden.
MAP GREYSCALE	Damit wird die Farbtabelle auf 16 oder 32 Graustufen gesetzt (je nach MODE). MAP GRAYSCALE ist auch okay.
MAP SET	Damit wird MMBasic dazu gebracht, die Farbtabelle (festgelegt mit MAP(n)=rgb%) während des nächsten Austastintervalls zu aktualisieren.
MAP RESET	Damit wird die Farbtabelle auf die Standardfarben zurückgesetzt.
MAP	PICOMITE RP2350 NUR GEPUFFERTE TREIBER Mit den MAP-Befehlen kann der Programmierer die Farben festlegen, die im 8-Bit-RGB332-Farbmodus verwendet werden. Jeder Wert in der 8-Bit-Farbpalette kann auf eine unabhängige 24-Bit-Farbe (d. h. RGB888-Format) eingestellt werden. Weitere Infos findest du unter der MAP-Funktion.
MAP(n) = rgb%	Damit 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 Farbtabelle (festgelegt mit MAP(n)=rgb%) während des nächsten Austastintervalls.
MAP RESET	Damit wird die Farbtabelle auf die Standardfarben zurückgesetzt.
MAP	NUR VGA-VERSION 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 eingestellt werden. Weitere Infos 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 verfügbaren 16 VGA-RGB121-Farben umgewandelt, die durch das Widerstandsnetzwerk festgelegt sind. Die Änderung wird nach dem Befehl „MAP SET” aktiviert.

MAP MAXIMITE

Damit wird die Farbtabelle auf die Farben eingestellt, die im ursprünglichen Colour Maximite verwendet wurden.

MAP SET

Dadurch aktualisiert MMBasic die Farbtabelle (festgelegt mit MAP(n)=rgb%) während des nächsten Austastintervalls.

MAP RESET

Damit wird die Farbtabelle auf die Standardfarben zurückgesetzt, die im 4-Bit-Modus wie folgt lauten:

„n"	Farbe	Wert
15	WEISS	RGB(255, 255, 255)
14	GELB	RGB(255, 255, 0)
13	LILA	RGB(255, 128, 255)
12	BRAUN	RGB(255, 128, 0)
11	FUCHSIA	RGB(255, 64, 255)
10	ROST	RGB(255, 64, 0)
9	MAGENTA	RGB(255, 0, 255)
8	ROT	RGB(255, 0, 0)
7	CYAN	RGB(0, 255, 255)
6	GRÜN	RGB(0, 255, 0)
5	CERULEAN	RGB(0, 128, 255)
4	MITTLERES GRÜN	RGB(0, 128, 0)
3	KOBALT	RGB(0, 64, 255)
2	MYRTLE	RGB(0, 64, 0)
1	BLAU	RGB(0, 0, 255)
0	SCHWARZ	RGB(0, 0, 0)

MATH

Der Befehl „math" macht viele einfache mathematische Berechnungen, die man in BASIC programmieren kann, aber es ist schneller, Schleifenstrukturen in der Firmware zu programmieren, und es ist auch besser, weil sie nach dem Debuggen für alle da sind, ohne dass man das Rad neu erfinden muss. Hinweis: Zweidimensionale mathematische Matrizen werden immer mit DIM matrix(n_columns, n_rows) angegeben, und natürlich entsprechen die Dimensionen OPTION BASE. Quaternionen werden als Array mit 5 Elementen w, x, y, z, magnitude gespeichert.

MATH RANDOMIZE [n]

Sät den Mersenne-Twister-Algorithmus.
Wenn n nicht angegeben ist, ist der Startwert die Zeit in Mikrosekunden seit dem Start.
Der Mersenne-Twister-Algorithmus liefert viel bessere Zufallszahlen als die integrierte Funktion der C-Bibliothek. Hinweis: Der RP2350 hat einen H/W-Zufallszahlengenerator.

Einfache Array-Arithmetik

MATH SET nbr, array()

MATH SCALE in(), scale ,out()

Siehe ARRAY SET

Skaliert die Matrix in() um den Skalarwert scale und speichert das Ergebnis in out(). Funktioniert für Arrays beliebiger Dimensionen, sowohl für Ganzzahlen als auch für Gleitkommazahlen, und kann zwischen diesen konvertieren. Die Einstellung b auf 1 ist optimiert und die schnellste Methode, um ein gesamtes Array zu kopieren.

MATH ADD in(), num ,out()

MATH INTERPOLATE in1(), in2(), Verhältnis, out()

Siehe ARRAY ADD

Dieser Befehl wendet die folgende Gleichung auf jedes Array-Element an:
out = (in2 - in1) * ratio + in1
Arrays können beliebig viele Dimensionen haben, müssen aber unterschiedlich

MATH WINDOW in(), minout, maxout, out() [,minin!, maxin!]	sein und die gleiche Gesamtzahl an Elementen haben. Der Befehl funktioniert mit ganzzahligen und Gleitkomma-Arrays in beliebiger Kombination. Dieser Befehl nimmt das Array „in" 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. Dieser Befehl kann das Skalieren von Daten für die Darstellung usw. erheblich vereinfachen.
MATH SLICE Quellarray(), [d1] [,d2] [,d3] [,d4] [,d5] , Zielarray()	Beispiel: DIM IN(2)=(1,2,3) DIM OUT(2) MATH WINDOW IN(),7,3,OUT(),LOW,HIGH Gibt OUT(0)=7, OUT(1)=5,OUT(2)=3,LOW=1,HIGH=3 zurück
MATH INSERT Zielarray(), [d1] [,d2] [,d3] [,d4] [,d5] , Quellarray()	Siehe ARRAY SLICE
MATH POWER inarray(), potenz, outarray()	Siehe ARRAY INSERT
MATH SHIFT inarray%(), nbr, outarray%() [,U]	Hebt jedes Element in „inarray()" auf die angegebene „Potenz" und speichert das Ergebnis in „outarray()".
Matrix-Arithmetik	Dieser Befehl verschiebt alle Elemente von inarray%() um ein Bit und speichert das Ergebnis in outarray%() (kann mit inarray%()) übereinstimmen. nbr kann zwischen -63 und 63 liegen. Positive Zahlen werden nach links verschoben (mit der Potenz von 2 multipliziert). Negative Zahlen werden nach rechts verschoben. Der optionale Parameter ,U erzwingt eine vorzeichenlose Verschiebung.
MATH M_INVERSE array!(), inversearray!()	Dies gibt die Umkehrung von array!() in inversearray!() zurück. Das Array muss quadratisch sein, und du bekommst eine Fehlermeldung, wenn das Array nicht umgekehrt werden kann (Determinante = 0). array!() und inversearray!() können nicht identisch sein.
MATH M_PRINT array()	Schneller Mechanismus zum Drucken einer 2D-Matrix mit einer Zeile pro Zeile.
MATH M_TRANSPOSE in(), out()	Transponiere die Matrix in() und speichere das Ergebnis in matrix out(). Beide Arrays müssen 2D 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 2D sein, müssen aber nicht quadratisch sein. Wenn sie nicht quadratisch sind, müssen die Arrays die Dimensionen in1(m,n) in2(p,m) ,out(p,n) haben.
Vektor-Arithmetik	
MATH V_PRINT array() [,hex]	Schneller Mechanismus zum Drucken eines kleinen Arrays in einer einzigen Zeile. „hex" druckt in Hexadezimal.
MATH V_NORMALISE inV(), outV()	
MATH V_MULT matrix(), inV(), outV()	
MATH V_CROSS inV1(),	

inV2(), outV()	
MATH V_ROTATE x, y, a, xin(), yin(), xout(), yout()	Konvertiert einen Vektor inV() in eine Einheitsskala und speichert das Ergebnis in outV(). (sqr(x*x + y*y +....)=1 Die Anzahl der Elemente im Vektor ist nicht begrenzt.
Quaternion-Arithmetik MATH Q_INVERT inQ(), outQ()	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 Q_VECTOR x, y, z, outVQ()	Berechnet das Kreuzprodukt zweier Vektoren mit drei Elementen inV1() und inV2() und speichert das Ergebnis in outV().
MATH Q_CREATE theta, x, y, z, outRQ() MATH Q_EULER Gierwinkel, Neigungswinkel, Rollwinkel, outRQ()	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, kann aber mit dem Befehl OPTION ANGLE geändert werden.
	Invertiert das Quaternion in inQ() und speichert das Ergebnis in outQ().
	Konvertiert einen durch x, y und z angegebenen Vektor in einen normalisierten Quaternion-Vektor outVQ() mit der ursprünglichen Größe.
MATH Q_MULT inQ1(), inQ2(), outQ()	Erzeugt einen normalisierten Rotationsquaternion outRQ(), um Quaternion-Vektoren um die Achsen x, y, z um einen Winkel von theta zu drehen. Theta wird in Radianten angegeben.
MATH Q_ROTATE , RQ(), inVQ(), outVQ()	Erzeugt ein normalisiertes Rotationsquaternion outRQ(), um Quaternion-Vektoren entsprechend den definierten Gier-, Neigungs- und Rollwinkeln zu drehen Wenn der Vektor vor dem „Betrachter" liegt, schaut Yaw von der Spitze des Vektors und dreht sich im Uhrzeigersinn, Pitch dreht die Spitze von der Kamera weg und Roll dreht sich um die z-Achse im Uhrzeigersinn. Die Gier-, Neigungs- und Rollwinkel sind standardmäßig in Radianten angegeben, berücksichtigen aber die Einstellung von OPTION ANGLE.
	Multipliziert zwei Quaternionen inQ1() und inQ2() und speichert das Ergebnis in outQ().
	Dreht den Quaternion-Vektor inVQ() um den Drehquaternion RQ() und speichert das Ergebnis in outVQ()

MATH C_ADD array1(), array2(), array3() MATH C_SUB array1(), array2(), array3() MATH C_MUL array1(), array2(), array3() MATH C_DIV array1(), array2(), array3() MATH C_AND array1(), array2(), array3() MATH C_OR array1(), array2(), array3() MATH C_XOR array1(), array2(), array3()	Diese Befehle machen Zelle-für-Zelle-Operationen an Arrays. array1 und array2 sind die Parameter und array3 ist die Ausgabe. Alle Arrays müssen die gleiche Größe und den gleichen Typ haben (Float oder Integer). Es gibt keine Einschränkungen hinsichtlich der Anzahl der Dimensionen und keine Einschränkungen hinsichtlich der Verwendung desselben Arrays zweimal oder sogar dreimal in den Parametern. Beispiel: MATH C_MUL a%(),a%(),a%() quadriert alle Werte im Array a%()

MATH FFT signalarray!(), FFTarray!()	Führt eine schnelle Fourier-Transformation der Daten in „signalarray!“ durch. „signalarray“ muss ein Fließkommawert sein und die Größe muss eine Potenz von 2 sein (z. B. s(1023), wenn OPTION BASE Null ist). „FFTarray” muss ein Fließkommawert sein und die Dimension 2*N haben, wobei N gleich dem Signalarray ist (z. B. f(1,1023), wenn OPTION BASE Null ist). Der Befehl gibt die FFT als komplexe Zahlen zurück, wobei der Realteil in f(0,n) und der Imaginärteil in f(1,n) liegt.
MATH FFT INVERSE FFTarray!(), signalarray!()	Führt eine inverse schnelle Fourier-Transformation der Daten in „FFTarray!” durch. „FFTarray” muss ein Fließkommawert sein und die Dimension 2*N haben, wobei N eine Potenz von 2 sein muss (z. B. f(1,1023), wenn OPTION BASE gleich Null ist), mit dem Realteil in f(0,n) und dem Imaginärteil in f(1,n). „signalarray” muss ein Gleitkommawert sein und die einzelne Dimension muss mit dem FFT-Array übereinstimmen. Der Befehl gibt den Realteil der inversen Transformation in „signalarray“ zurück.
MATH FFT MAGNITUDE signalarray!(),magnitudearray!()	Erzeugt Amplituden für Frequenzen für die Daten in „signalarray!”. „signalarray“ muss ein Fließkommawert sein und die Größe muss eine Potenz von 2 sein (z. B. s(1023), wenn OPTION BASE gleich Null ist). „magnitudearray“ muss ein Fließkommawert sein und die Größe muss mit der des Signal-Arrays übereinstimmen. Der Befehl gibt die Amplitude des Signals bei verschiedenen Frequenzen nach der folgenden Formel zurück: Frequenz an der Array-Position N = N * Abtastfrequenz / Anzahl der Abtastwerte
MATH FFT PHASE signalarray(), phasearray()	Erzeugt Phasen für Frequenzen für die Daten in „signalarray!“. „signalarray” muss ein Fließkommawert sein und die Größe muss eine Potenz von 2 sein (z. B. s(1023), wenn OPTION BASE Null ist). „phasearray” muss ein Fließkommawert sein und die Größe muss die gleiche sein wie die des Signal-Arrays. 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. Ax, ay und az sind die Beschleunigungen in den drei Richtungen und sollten in Einheiten der Standardbeschleunigung angegeben werden. Gx, gy und gz sind die Momentanwerte der Drehgeschwindigkeit, die in Radianten pro Sekunde angegeben werden sollten. 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 zwischen den drei Eingaben konsistent sind. Bei Verwendung des MPU-9250 sind beispielsweise ax, ay, az, gx, gy, gz, my, mx, -mz basierend auf den Messwerten des Sensors die richtigen Eingaben. Pitch, Roll und Yaw sollten Fließkomma-Variablen sein und die Ausgaben der Sensorfusion enthalten. Die SENSORFUSION-Routine misst automatisch die Zeit zwischen aufeinanderfolgenden Aufrufen und nutzt diese für ihre internen Berechnungen. Der Madwick-Algorithmus nimmt einen optionalen Parameter p1. Dieser wird als Beta in der Berechnung verwendet. Wenn er nicht angegeben wird, ist der

	Standardwert 0,5. Der Mahony-Algorithmus nimmt zwei optionale Parameter p1 und p2. Diese werden als Kp und Ki in der Berechnung verwendet. Wenn sie nicht angegeben werden, sind die Standardwerte 10,0 bzw. 0,0. Ein vollständiges 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() MATH SINC x_in(), y_in(), n, window, freq, x_out(), y_out()	Wende einen Fenster-Sinc-Filter an, um Koordinatendaten zu glätten oder zu interpolieren. Ein Sinc-Filter ist ein idealer Tiefpassfilter, der hochfrequente Störgeräusche entfernt und gleichzeitig die Signalform beibehält. Er eignet sich besonders gut für die Neuberechnung (Interpolation). Parameter: x_in(), y_in() — Eingabe-Koordinaten-Arrays (Float) n — Anzahl der Eingabepunkte m — Anzahl der Ausgabepunkte (optional). Wenn weggelassen oder m=n: Glättungsmodus (Ausgabegröße n). Wenn m!=n: Interpolations-/Resampling-Modus (Ausgabegröße m) Fenster — Filterkernelgröße (muss ungerade sein; gerade Werte werden erhöht). Typische Werte: 15 bis 101. Größere Werte sorgen für eine schärfere Trennlinie, sind aber langsamer. freq — Normalisierte Grenzfrequenz (0,0 < Freq <= 0,5). 0,5 ist die Nyquist-Frequenz (keine Filterung). Typische Werte: 0,1 (starke Glättung) bis 0,4 (leichte Glättung). x_out(), y_out() — Ausgabe-Koordinaten-Arrays (müssen groß genug sein, um die Ausgabe aufzunehmen)
MATH PID INIT-Kanal, pid_params!(), Callback	Dieser Befehl richtet einen PID-Regler ein, der automatisch im Hintergrund laufen kann. Bis zu 8 PID-Regler können gleichzeitig laufen (Kanäle 1 bis 8). „callback" ist eine MMbasic-Subroutine, die mit der durch die Abtastzeit definierten Rate aufgerufen wird. Details dazu, was in der Subroutine enthalten sein sollte, findest du in der Funktion MATH(PID …). Das Array pid_params!() muss für alle aufgelisteten Elemente dimensioniert werden, einschließlich der Reglerspeicherparameter (DIM pid_params!(13)), und mit den erforderlichen Einstellungen initialisiert werden. PID-Konfiguration Element 0 = Kp Element 1 = Ki Element 2 = Kd Element 3 = tau ' Zeitkonstante des derivativen Tiefpassfilters Element 4 = limMin 'Ausgangsbegrenzungen Element 5 = limMax Element 6 = limMinInt 'Integrator-Grenzen Element 7 = limMaxInt Element 8 = T 'Abtastzeit (in Sekunden) Controller „Speicher" Element 9 = Integrator Element 10 = prevError Element 11 = Differenzierer

	Element 12 = vorherigeMessung Element 13 = Ausgang
MATH PID START-Kanal	Startet einen vorher initialisierten PID-Regler auf dem angegebenen Kanal
MATH PID STOP-Kanal	Stoppt einen vorher initialisierten PID-Regler auf dem angegebenen Kanal und löscht die internen Datenstrukturen Siehe https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=17263 Ein Beispiel für die Einrichtung und den Betrieb eines PID-Reglers findest du unter
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 unter Verwendung der angegebenen AES128-Verschlüsselungsmethode in „out". Die Parameter können jeweils eine Zeichenfolge, ein Integer-Array oder ein Float-Array sein, wobei jede Kombination möglich ist. Der Schlüssel muss 16 Elemente lang sein (16*8=128 Bit), „in" und „out" müssen ein Vielfaches von 16 Elementen lang sein. Wenn „out" als Zeichenfolge angegeben wird (z. B. out$), muss die Zeichenfolgenvariable vorhanden sein und sollte auf leer gesetzt werden (DIM out$=""). Die maximale Anzahl von Elementen 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. Für die 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. In beiden Fällen wird der IV vor die Ausgabe gesetzt. Bei CBC und CTR müssen die ersten 16 Elemente der Eingabe der Initialisierungsvektor sein, damit die Entschlüsselung klappt. Wenn du eine Nachricht ohne IV übertragen willst, solltest du den IV vor dem Senden der Nachricht mit Standard-MMBasic-Manipulationen entfernen. In diesem Fall muss der Empfänger sowohl den IV als auch die Schlüssel en kennen und eine vollständige Nachricht erstellen, bevor er DECRYPT verwendet, indem er den IV an die eingehende Nachricht anhängt.

MEMORY	Zeigt die aktuell belegte Speichermenge an. Beispiel:
	`Programm:`
	`  0K ( 0%) Programm (0 Zeilen)`
	`180K (100 %) Frei`
	
	`Gespeicherte Variablen:`
	`16K (100 %) Frei`
	
	`RAM:`
	`  0K (0 %) 0 Variablen`
	`  0K (0 %) Allgemein`
	`228 KB (100 %) frei`
	
	Hinweise:
	• Die Speicherauslastung wird auf die nächsten 1K Byte gerundet.
	• Der allgemeine Speicher (RAM) wird von Arrays, Strings, seriellen E/A-Puffern usw. genutzt.

MEMORY SET Adresse, Byte, Anzahl der Bytes MEMORY SET BYTE Adresse, Byte, Anzahl der Bytes MEMORY SET SHORT Adresse, Short, Anzahl der Shorts MEMORY SET WORD Adresse, Wort, Anzahl der Wörter MEMORY SET INTEGER Adresse, Integerwert, Anzahl der Integer [,Inkrement] MEMORY SET FLOAT Adresse, Fließkommawert, Anzahl der Fließkommawerte [,Inkrement]	Dieser Befehl setzt einen Speicherbereich auf einen bestimmten Wert. BYTE = Ein Byte pro Speicheradresse. SHORT = Zwei Bytes pro Speicheradresse. WORD = Vier Bytes pro Speicheradresse. FLOAT = Acht Bytes pro Speicheradresse. „increment" ist optional und regelt, wie der Zeiger „address" bei der Ausführung der Operation erhöht wird. Wenn zum Beispiel increment=3 ist, wird nur jedes dritte Element des Ziels gesetzt. Der Standardwert ist 1.
MEMORY COPY Quelladresse, Zieladresse, Anzahl der Bytes [,Quellinkrement] [,Zielinkrement] MEMORY COPY INTEGER Quelladresse, Zieladresse, Anzahl der Ganzzahlen [,Quellinkrement] [,Zielinkrement] MEMORY COPY FLOAT Quelladresse, Zieladresse, Anzahl der Fließkommazahlen [,Quellinkrement] [,Zielinkrement]	Dieser Befehl kopiert einen Speicherbereich in einen anderen. COPY INTEGER und FLOAT kopieren acht Bytes pro Vorgang. „sourceincrement" ist optional und regelt, wie der Zeiger „sourceaddress" während der Ausführung der Operation erhöht wird. Wenn zum Beispiel sourceincrement=3 ist, wird nur jedes dritte Element der Quelle kopiert. Der Standardwert ist 1. „destinationincrement" funktioniert ähnlich und wirkt sich auf den Zeiger „destinationaddress" aus.
MEMORY PRINT #]fnbr , nbr, address%/array() MEMORY INPUT [#]fnbr , nbr, address%/array()	Diese Befehle speichern oder lesen „nbr" Datenbytes aus dem Speicher oder in einen offenen Datei auf der Festplatte. Der zu speichernde Speicher kann als ganzzahliges Array angegeben werden. In diesem Fall wird die Anzahl der zu speichernden oder zu lesenden Bytes mit der Array-Größe verglichen. Alternativ kann eine Speicheradresse verwendet werden. In diesem Fall findet keine Überprüfung statt, und Benutzerfehler können zu einem Absturz der Firmware führen.

MEMORY PACK source%()/sourceaddress%, dest%()/destaddress%, number, size 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. Die beiden Arrays sind immer normale Integer-Arrays, aber das gepackte Array kann 2, 4, 8, 16 oder 64 Werte „enthalten". So kann ein einzelnes Integer-Array-Element 2 32-Bit-Wörter, 4 16-Bit-Werte, 8 Bytes, 16 Nibbles oder 64 Boolesche Werte (Bits) speichern. „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. Alternativ können Speicheradressen verwendet werden. In diesem Fall kann keine Überprüfung stattfinden, und Benutzerfehler könnten zu einem Absturz der Firmware führen.
MKDIR dir$	Erstellt das Verzeichnis „dir$" im Standard-Flash-Dateisystem oder auf der SD-Karte.
MID$(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 ist, wird standardmäßig die Länge von „str2$" verwendet.
MODUS 1 oder MODUS 2 oder MODUS 3 (nur RP2350)	Nur VGA-Versionen VGA-Video unterstützt mehrere Auflösungen (siehe OPTION AUFLÖSUNG). Dieser Befehl wählt den Modus „n" je nach Auflösung aus: OPTION AUFLÖSUNG 640 x 480 MODUS 1 640 x 480 x 2 Farben (monochrom). Standard bei Start. Die Breite der Kacheln ist auf 8 Pixel festgelegt. Die Höhe der Kacheln ist standardmäßig auf 12 Pixel eingestellt, kann aber zwischen 8 und MM.HRES liegen. Die Farben der Kacheln werden mit der Standardnotation RGB888 angegeben. Diese wird in RGB121 umgewandelt. Ein Framebuffer (F) und ein Layer-Puffer (L) können erstellt werden. Diese haben keinen Einfluss auf die Anzeige und verbrauchen keinen Benutzerspeicher, können aber beide zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. MODUS 2 320 x 240 x 16 Farben. 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 verbraucht 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 verbraucht keinen Benutzerspeicher. Alle in den Ebenenpuffer geschriebenen Pixel werden automatisch auf dem Display angezeigt und überlagern alles, was sich im Hauptdisplaypuffer befindet. Es kann eine Farbe festgelegt werden (0-15: Standardwert ist 0), die nicht angezeigt wird, sodass der Hauptdisplaypuffer durchscheinen kann. Mit der Map-Funktion können die Standard - Farben der 16 verfügbaren Farben überschrieben werden. Die Hardware ist auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind. MODUS 3 640 x 480 x 16 Farben. 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 verbraucht 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 verbraucht keinen Benutzerspeicher. Alle in den Ebenenpuffer geschriebenen Pixel werden automatisch auf dem Display angezeigt und überlagern alles, 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. Mit der Map-Funktion können die Standardfarben der 16 verfügbaren Farben überschrieben werden. Die Hardware ist auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind.

OPTION AUFLÖSUNG 720 x 400

MODUS 1 720 x 400 x 2 Farben (monochrom). Standard bei Start.
Die Breite der Kacheln ist auf 8 Pixel festgelegt. Die Höhe der Kacheln ist standardmäßig auf 12 Pixel eingestellt, kann aber zwischen 8 und MM.HRES liegen. Die Farben der Kacheln werden mit der Standardnotation RGB888 angegeben. Diese wird in RGB121 umgewandelt. Ein Framebuffer (F) und ein Layer-Puffer (L) können erstellt werden. Diese haben keinen Einfluss auf die Anzeige und verbrauchen keinen Benutzerspeicher, können aber beide zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden.

MODUS 2 360 x 200 x 16 Farben.
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 verbraucht 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 verbraucht keinen Benutzerspeicher. Alle Pixel, die in den Ebenenpuffer geschrieben werden, erscheinen automatisch auf dem Display 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 durchscheinen kann. Es gibt eine Zuordnungsfunktion, mit der die Standardfarben der 16 verfügbaren Farben überschrieben werden können. Bei VGA ist die Hardware auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind.

MODUS 3 720 x 400 x 16 Farben.
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 Pixel, die in den Ebenenpuffer geschrieben werden, werden automatisch auf dem Display angezeigt 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 durchscheinen kann. Mit der Map-Funktion können die Standardfarben der 16 verfügbaren Farben überschrieben werden. Bei VGA ist die Hardware auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind.

OPTION AUFLÖSUNG 800 x 600 (nur RP2350)

MODUS 1 800 x 600 x 2 Farben (monochrom). Standard beim Start.
Die Breite der Kacheln ist auf 8 Pixel festgelegt. Die Höhe der

Kacheln ist standardmäßig auf 12 Pixel eingestellt, kann aber zwischen 8 und MM.HRES liegen. Die Farben der Kacheln werden mit der Standardnotation RGB888 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 verbrauchen keinen Benutzerspeicher, aber beide können zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden.

MODUS 2 400 x 300 x 16 Farben.
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 verbraucht 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 verbraucht keinen Benutzerspeicher. Alle in den Ebenenpuffer geschriebenen Pixel werden automatisch auf dem Display angezeigt und überlagern alles, was sich im Hauptdisplaypuffer befindet. Es kann eine Farbe festgelegt werden (0-15: Standardwert ist 0), die nicht angezeigt wird, sodass der Hauptdisplaypuffer durchscheinen kann. Mit der Map-Funktion können die Standardfarben der 16 verfügbaren Farben überschrieben werden. Die Hardware ist auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind.

MODUS 3 800 x 600 x 16 Farben.
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 verbraucht keinen Benutzerspeicher. Alle in den Ebenenpuffer geschriebenen Pixel werden automatisch auf dem Display angezeigt und überlagern alles, 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. Mit der Map-Funktion können die Standardfarben der 16 verfügbaren Farben überschrieben werden. Die Hardware ist auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind.

<u>OPTION AUFLÖSUNG 848 x 480 (nur RP2350)</u>

MODUS 1 848 x 480 x 2 Farben (monochrom). Standard beim Start.
Die Breite der Kacheln ist auf 8 Pixel festgelegt. Die Höhe der Kacheln ist standardmäßig auf 12 Pixel eingestellt, kann aber zwischen 8 und MM.HRES liegen. Die Farben der Kacheln werden mit der Standardnotation RGB888 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 verbrauchen keinen Benutzerspeicher, können aber beide zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden.

MODUS 2 424 x 240 x 16 Farben.
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 verbraucht 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 verbraucht

	keinen Benutzerspeicher. Alle in den Ebenenpuffer geschriebenen Pixel werden automatisch auf dem Display angezeigt und überlagern alles, 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. Mit der Map-Funktion können die Standardfarben der 16 verfügbaren Farben überschrieben werden. Die Hardware ist auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind. MODUS 3 848 x 48 x 16 Farben. 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 verbraucht 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 verbraucht keinen Benutzerspeicher. Alle in den Ebenenpuffer geschriebenen Pixel werden automatisch auf dem Display angezeigt und überlagern den Inhalt des Hauptanzeigepuffers ( ). Es kann eine Farbe festgelegt werden (0-15: Standardwert ist 0), die nicht angezeigt wird, sodass der Hauptanzeigepuffer durchscheinen kann. Mit der Map-Funktion können die Standardfarben der 16 verfügbaren Farben überschrieben werden. Die Hardware ist auf die 16 Farben beschränkt, die durch das Widerstandsnetzwerk definiert sind.
MODUS n	<u>NUR HDMI-VERSIONEN</u> HDMI-Video unterstützt eine Reihe von Auflösungen (siehe OPTION AUFLÖSUNG). Dieser Befehl wählt den Modus „n" je nach Auflösung aus: <u>OPTION AUFLÖSUNG 640 x 480</u> MODUS 1 640 x 480 x 2 Farben (monochrom). Standard bei Start. Benutz den Befehl TILE wie gewohnt. Die Breite der Kacheln ist auf 8 Pixel festgelegt. Die Höhe der Kacheln ist standardmäßig auf 12 Pixel eingestellt, kann aber zwischen 8 und MM.HRES liegen. Die Farben der Kacheln werden mit der Standardnotation RGB888 angegeben. Diese wird in RGB555 umgewandelt. Es können ein Framebuffer (F) und ein Layer-Puffer (L) erstellt werden. Diese können zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. MODUS 2 320 x 240 x 16 Farben. Ein Framebuffer (F) kann erstellt werden. Dieser kann zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Puffer erstellt werden. Alle Pixel, die in den Layer-Puffer geschrieben werden, erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptbildschirm-Puffer befindet. Es kann eine Farbe angegeben werden (0-15: Standardwert ist 0), die nicht angezeigt wird, sodass der Hauptbildschirm-Puffer durchscheinen kann. Es gibt eine Zuordnungsfunktion, mit der die Standardfarben überschrieben werden können. MODUS 3 640 x 480 x 16 Farben. Farbzuordnung zur RGB555-Palette. Ein Bildspeicher (F) kann erstellt werden. Er kann zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Ebenenpuffer erstellt werden. Alle Pixel,

die in den Ebenenpuffer geschrieben werden, werden automatisch auf dem Display angezeigt 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 durchscheinen kann.

MODUS 4 320 x 240 x 32768 Farben.
Dies ist volles RGB555, wodurch Farbbilder in guter Qualität angezeigt werden können. Ein Bildspeicher (F) und ein Ebenenpuffer (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.

MODUS 5 320 x 240 x 256 Farben.
Ein Framebuffer (F) kann 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-Puffer erstellt werden. Dieser nutzt keinen Benutzerspeicher. Alle Pixel, die in den Ebenenpuffer geschrieben werden, werden automatisch auf dem Display angezeigt 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. Mit der Zuordnungsfunktion können die Standardfarben der 256 verfügbaren Farben überschrieben werden. Jede der 256 Farben kann einer beliebigen RGB555-Farbe zugeordnet werden.

<u>OPTION AUFLÖSUNG 720 x 400</u>

MODUS 1 720 x 400 x 2 Farben (monochrom). Standard beim Start. Benutze den Befehl TILE wie gewohnt. Die Breite der Kacheln ist auf 8 Pixel festgelegt. Die Höhe der Kacheln ist standardmäßig auf 12 Pixel eingestellt, 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-Puffer (L) erstellt werden. Diese können zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden.

MODUS 2 360 x 200 x 16 Farben.
Ein Framebuffer (F) kann erstellt werden. Dieser kann zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Puffer erstellt werden. Alle Pixel, die in den Layer-Puffer geschrieben werden, erscheinen automatisch auf dem Bildschirm und liegen über allem, was sich im Hauptbildschirm-Puffer befindet. Es kann eine Farbe angegeben werden (0-15: Standardwert ist 0), die nicht angezeigt wird, sodass der Hauptbildschirm-Puffer durchscheinen kann. Es gibt eine Zuordnungsfunktion, mit der die Standardfarben überschrieben werden können.

MODUS 3 720 x 400 x 16 Farben.
Farbzuordnung zur RGB555-Palette. Ein Bildspeicher (F) kann erstellt werden. Er kann zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Ebenenpuffer erstellt werden. Alle Pixel, die in den Ebenenpuffer geschrieben werden, werden automatisch auf dem Display angezeigt 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 durchscheinen kann. MODUS 4 360 x 200 x 32768 Farben. Dies ist volles RGB555, wodurch Farbbilder in guter Qualität angezeigt werden können. Ein Bildspeicher (F) und ein Ebenenpuffer (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. MODUS 5 360 x 200 x 256 Farben. Ein Framebuffer (F) kann erstellt werden. Dies hat keine Auswirkungen auf die Anzeige. Er kann zum Erstellen von Bildern und zum Kopieren auf den Bildschirm (N) verwendet werden. Zusätzlich kann ein Layer-Puffer erstellt werden. Dieser belegt keinen Benutzerspeicher. Alle Pixel, die in den Ebenenpuffer geschrieben werden, erscheinen automatisch auf dem Display und liegen über allem, was sich im Hauptanzeigepuffer befindet. Es kann eine Farbe angegeben werden (0-255: Standardwert ist 0), die nicht angezeigt wird, sodass der Hauptanzeigepuffer durchscheinen kann. Mit der Zuordnungsfunktion können die Standardfarben der 256 verfügbaren Farben überschrieben werden. Jede der 256 Farben kann einer beliebigen RGB555-Farbe zugeordnet werden. <u>OPTION AUFLÖSUNG 800 x 600 (Hinweis: reduziert den verfügbaren Heap-Speicher)</u> MODUS 1 800 x 600 x 2 Farben mit RGB332-Kacheln (benutze den Befehl TILE wie gewohnt) MODUS 2 400 x 300 x 16 Farben mit optionaler Ebene und Farbauftragung auf RGB332-Palette MODUS 3 800 x 400 x 16 Farben mit optionaler Ebene und Farbabbildung auf RGB332-Palette MODUS 5 400 x 300 x 256 Farben mit optionaler Ebene (kein Speicherverbrauch) <u>OPTIONALE AUFLÖSUNG 848 x 480 (Anmerkung: reduziert den verfügbaren Heap-Speicher)</u> MODUS 1 848 x 480 x 2 Farben mit RGB332-Kacheln (benutze den Befehl TILE wie gewohnt) MODUS 2 424 x 240 x 16 Farben mit optionaler Ebene und Farbabbildung auf RGB332-Palette MODUS 3 848 x 480 x 16 Farben mit optionaler Ebene und Farbabbildung auf RGB332-Palette MODUS 5 424 x 240 x 256 Farben mit optionaler Ebene (kein Speicherverbrauch) <u>OPTIONALE AUFLÖSUNG 1280 x 720</u> MODUS 1 1280 x 720 x 2 Farben mit RGB332-Kacheln (benutze den Befehl TILE wie gewohnt) MODUS 2 320 x 180 x 16 Farben mit 2 optionalen Ebenen und Farbabbildung auf RGB332-Palette MODUS 3 640 x 360 x 16 Farben mit optionaler Ebene und Farbabbildung auf RGB332-Palette MODUS 5 320 x 180 x 256 Farben mit optionaler Ebene (kein Speicherverbrauch)

	OPTIONALE AUFLÖSUNG 1024 x 768 MODUS 1 1024 x 768 x 2 Farben mit RGB332-Kacheln (benutze den Befehl TILE wie gewohnt) MODUS 2 256 x 192 x 16 Farben mit 2 optionalen Ebenen und Farbabbildung auf die RGB332-Palette MODUS 3 512 x 384 x 16 Farben MODUS 5 256 x 192 x 256 Farben mit optionaler Ebene und Farbabbildung auf RGB332-Palette OPTIONALE AUFLÖSUNG 1024 x 600 MODUS 1 1024 x 600 x 2 Farben mit RGB332-Kacheln (benutze den Befehl TILE wie gewohnt) MODUS 2 256 x 150 x 16 Farben mit 2 optionalen Ebenen und Farbabbildung auf RGB332-Palette MODUS 3 512 x 300 x 16 Farben mit optionaler Ebene und Farbabbildung auf RGB332-Palette MODUS 5 256 x 150 x 256 Farben mit optionaler Ebene und Farbzuordnung zur RGB332-Palette OPTIONALE AUFLÖSUNG 800 x 480 (Anmerkung: reduziert den verfügbaren Heap-Speicher) MODUS 1 800 x 480 x 2 Farben mit RGB332-Kacheln (benutze den Befehl TILE wie gewohnt) MODUS 2 400 x 240 x 16 Farben mit 2 optionalen Ebenen und Farbabbildung auf RGB332-Palette MODUS 3 800 x 480 x 16 Farben mit optionaler Ebene und Farbabbildung auf RGB332-Palette MODUS 5 400 x 240 x 256 Farben mit optionaler Ebene und Farbzuordnung zur RGB332-Palette
MODUS 400 oder MODUS 800	NUR RP2350 PICOMITE-VERSIONEN Wechselt zwischen den Modi 400 x 240 und 800 x 480 für SSD1963-Puffertreiber
MOUSE MOUSE INTERRUPT ENABLE Kanal, int MOUSE INTERRUPT DISABLE-Kanal MOUSE SET Kanal, x-Koordinate, y-Koordinate [, Radanzahl]	Für alle Varianten des Befehls. Bei USB-Firmware ist „channel" der USB-Anschluss, an den die Maus angeschlossen ist (1-4). Weitere Infos findest du unter MM.INFO(USB n). Bei PS2-Firmware ist „channel" fest auf den Wert 2 eingestellt. „int" ist eine benutzerdefinierte Subroutine, die aufgerufen wird, wenn die linke Maustaste gedrückt wird. Deaktiviert eine Unterbrechung der linken Maustaste Legt die aktuelle Position fest, die von der Maus zurückgegeben wird: x, y und optional die Positionen des Rads
MOUSE OPEN-Kanal, CLKpin, DATApin	NICHT-USB-VERSIONEN – NUR FÜR EINE PS2-MAUS 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 Ausführung des Programms zu

MOUSE CLOSE-Kanal	konfigurieren, im Gegensatz zu OPTION MOUSE, das die Maus dauerhaft konfiguriert. Der Kanal ist für die Kompatibilität mit der USB-Mausfunktionalität enthalten und muss auf 2 gesetzt werden. Wenn keine Maus angeschlossen ist, wird eine Fehlermeldung angezeigt, und der Befehl kann erneut aufgerufen werden, sobald die Maus angeschlossen ist. Schließt den Zugriff auf die Maus und stellt die Pins wieder auf normale Verwendung zurück. Der Befehl gibt eine Fehlermeldung aus, wenn OPTION MOUSE eingestellt wurde.
NEU	Löscht den Programmspeicher und alle Variablen, einschließlich gespeicherter Variablen.
NEXT [Zählervariable] [, Zählervariable] usw.	NEXT kommt am Ende einer FOR-NEXT-Schleife; siehe FOR. Die „Zählvariable" gibt genau an, welche Schleife gerade bearbeitet wird. Wenn keine „Zählvariable" angegeben ist, geht NEXT automatisch zur innersten Schleife. Man kann auch mehrere Zählvariablen angeben, wie hier: `NEXT x, y, z`
ON ERROR ABORT oder ON ERROR IGNORE oder BEI FEHLER ÜBERSPRINGEN [nn] oder BEI FEHLER LÖSCHEN Oder BEI FEHLER NEU STARTEN	Hiermit wird festgelegt, was passiert, wenn während der Ausführung eines Programms ein Fehler auftritt. Das gilt für alle von MMBasic gefundenen Fehler, wie z. B. Syntaxfehler, falsche Daten, fehlende Hardware usw. 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. ON ERROR RESTART führt dazu, dass MMBasic einen Hardware-Reset durchführt, und wenn du OPTION AUTORUN eingestellt hast, wird das Programm natürlich sauber neu gestartet. ON ERROR IGNORE bewirkt, dass alle Fehler ignoriert werden. ON ERROR SKIP ignoriert einen Fehler in einer Reihe von Befehlen (angegeben durch die Zahl „nn"), die nach diesem Befehl ausgeführt werden. „nn" ist optional, die Standardeinstellung ist eins, wenn nichts angegeben ist. Nachdem die Anzahl der Befehle abgeschlossen ist (mit oder ohne Fehler), kehrt MMBasic zum Verhalten von ON ERROR ABORT zurück. Wenn ein Fehler auftritt und ignoriert/übersprungen wird, wird die schreibgeschützte Variable MM.ERRNO auf einen Wert ungleich Null gesetzt und MM.ERRMSG$ wird auf die Fehlermeldung gesetzt, die normalerweise generiert würde. Diese werden durch ON ERROR CLEAR auf Null und eine leere Zeichenfolge zurückgesetzt. Sie werden auch gelöscht, wenn das Programm ausgeführt wird und wenn ON ERROR IGNORE und ON ERROR SKIP verwendet werden. ON ERROR IGNORE kann das Debuggen eines Programms sehr erschweren, daher wird dringend empfohlen, nur ON ERROR SKIP zu verwenden.
ON KEY-Ziel oder ON KEY ASCIIcode, target	Die erste Version des Befehls setzt einen Interrupt, der die benutzerdefinierte Subroutine „Ziel" aufruft, sobald ein oder mehrere Zeichen im Eingabepuffer der seriellen Konsole warten. Beachte, dass alle Zeichen, die im Eingabepuffer warten, in der Interrupt-Subroutine gelesen werden sollten, da sonst automatisch ein weiterer Interrupt erzeugt wird, sobald das Programm aus dem Interrupt zurückkehrt. Mit der zweiten Version kannst du eine Interrupt-Routine mit einem bestimmten Tastendruck verknüpfen. Das läuft auf einer niedrigen Ebene für die serielle Konsole und wenn es aktiviert ist, wird die Taste nicht in den Eingabepuffer gelegt, sondern löst nur den Interrupt aus. Es nutzt einen separaten Interrupt vom

	einfachen ON KEY-Befehl, sodass es bei Bedarf gleichzeitig genutzt werden kann. In beiden Varianten kannst du den Interrupt deaktivieren, indem du die Zahl Null als Ziel verwendest, z. B.: ON KEY 0. oder ON KEY ASCIIcode, 0
ON PS2 Ziel	Dies löst einen Interrupt aus, sobald die PicoMite-Firmware eine Nachricht von der PS2-Schnittstelle sieht. Verwende MM.info(PS2), um die empfangene Rohmeldung zu melden. So kann der Programmierer sowohl das Drücken als auch das Loslassen der Taste erfassen. Die Scancodes (Set 2) findest du unter https://wiki.osdev.org/PS/2_Keyboard.
ONEWIRE RESET-Pin oder ONEWIRE WRITE-Pin, Flag, Länge, Daten [, Daten…] oder ONEWIRE READ Pin, Flag, Länge, Daten [, Daten…]	Befehle für die Kommunikation mit 1-Wire-Geräten. ONEWIRE RESET setzt den 1-Wire-Bus zurück ONEWIRE WRITE sendet eine bestimmte Anzahl von Bytes ONEWIRE READ liest eine bestimmte Anzahl von Bytes „Pin" ist der zu verwendende I/O-Pin (befindet sich im hinteren Anschluss). Es kann jeder Pin sein, der für digitale I/O geeignet ist. „flag" ist eine Kombination aus den folgenden Optionen: 1 – Reset vor dem Befehl senden 2 – Reset nach Befehl senden 4 – Nur ein Bit statt eines Bytes an Daten senden/empfangen 8 – Starke Pullup-Funktion nach dem Befehl ausführen (der Pin wird auf High gesetzt und Open Drain deaktiviert) „length" ist die Länge der zu sendenden oder zu empfangenden Daten „data" sind die zu sendenden Daten oder die zu empfangende Variable. Die Anzahl der Datenelemente muss mit dem Parameter „length" übereinstimmen. Siehe auch *Anhang C*.
OPEN fname$ FOR mode AS [#]fnbr	Öffnet eine Datei zum Lesen oder Schreiben. „fname" ist der Dateiname mit einer optionalen Erweiterung, die durch einen Punkt (.) getrennt ist. Lange Dateinamen mit Groß- und Kleinbuchstaben werden unterstützt. Das Dateisystem auf der SD-Karte unterscheidet NICHT zwischen Groß- und Kleinschreibung, das Flash-Dateisystem hingegen schon. Ein Verzeichnispfad kann mit einem Backslash als Verzeichnistrennzeichen angegeben werden. Das übergeordnete Verzeichnis des aktuellen Verzeichnisses kann mit dem Verzeichnisnamen „.." (zwei Punkte) und das aktuelle Verzeichnis mit „." (ein Punkt) angegeben werden. Beispiel: OPEN ".\dir1\dir2\filename.txt" FOR INPUT AS #1 Der Modus kann INPUT, OUTPUT, APPEND oder RANDOM sein. INPUT öffnet die Datei zum Lesen und gibt eine Fehlermeldung aus, wenn die Datei nicht vorhanden ist. OUTPUT öffnet die Datei zum Schreiben und überschreibt automatisch alle vorhandenen Dateien mit demselben Namen. APPEND öffnet die Datei auch zum Schreiben, überschreibt aber keine vorhandene Datei; stattdessen werden alle Schreibvorgänge an das Ende der Datei angehängt. Wenn keine Datei vorhanden ist, verhält sich der Modus APPEND wie der Modus OUTPUT (d. h. die Datei wird erstellt und dann zum Schreiben geöffnet). RANDOM öffnet die Datei zum Lesen und Schreiben und ermöglicht den zufälligen Zugriff mit dem Befehl SEEK. Beim Öffnen wird der Lese-/Schreibzeiger am Ende der Datei positioniert. Wenn die Datei nicht da ist, wird sie erstellt. „fnbr" ist die Dateinummer (1 bis 10). Das # ist optional. Bis zu 10 Dateien

	können gleichzeitig geöffnet sein und sich entweder auf dem Laufwerk A: oder C: oder auf beiden befinden. Die Befehle INPUT, LINE INPUT, PRINT, WRITE und CLOSE sowie die Funktionen EOF() und INPUT$() von , verwenden alle „fnbr", um die Datei zu identifizieren, die gerade bearbeitet wird. Siehe auch ON ERROR und MM.ERRNO für die Fehlerbehandlung.
OPEN comspec$ AS [#]fnbr	Öffnet einen seriellen Kommunikationsport zum Lesen und Schreiben. Es stehen zwei Ports zur Verfügung (COM1: und COM2:), die beide gleichzeitig geöffnet werden können. Eine vollständige Beschreibung mit Beispielen findest du in *Anhang A*. Mit „fnbr" kann der Port mit jedem Befehl oder jeder Funktion, die eine Dateinummer verwendet, zum Schreiben und Lesen genutzt werden.
OPEN comspec$ AS GPS [,timezone_offset] [,monitor]	Öffnet einen seriellen Kommunikationsport zum Lesen von einem GPS-Empfänger. Die Daten können dann mit der Funktion GPS() abgerufen werden . Das wird ausführlich in der Datei *Option_GPS_User_Manual.pdf* beschrieben, die im ZIP-Archiv zum Herunterladen der Firmware enthalten ist.
OPTION	Schau dir den Abschnitt *„Optionen"* weiter oben in diesem Handbuch an.
PAUSE delay	Hält die Ausführung des laufenden Programms für „Verzögerung" ms an. Das kann ein Bruchteil sein. Zum Beispiel entspricht 0,2 200 µs. Die maximale Verzögerung beträgt 2147483647 ms (etwa 24 Tage). Beachte, dass Interrupts während einer Pause erkannt und verarbeitet werden.
PIN(Pin) = Wert	Bei einem als digitaler Ausgang konfigurierten „Pin" wird der Ausgang auf niedrig („Wert" ist Null) oder hoch („Wert" ist ungleich Null) gesetzt. Du kannst einen Ausgang auf hoch oder niedrig setzen, bevor er als Ausgang konfiguriert wird, und diese Einstellung ist dann die Standardausgabe, wenn der Befehl SETPIN wirksam wird. Siehe die Funktion PIN() zum Lesen von einem Pin und den Befehl SETPIN zum Konfigurieren. Eine allgemeine Beschreibung der Ein-/Ausgabefunktionen der PicoMite-Firmware findest du im Kapitel *„Verwendung der E/A-Pins*".
PIO	Der Prozessorchip im Raspberry Pi Pico mit den RP2040-Prozessoren hat ein programmierbares I/O-System mit zwei gleichen PIO-Geräten (pio%=0 oder pio%=1), die wie spezielle CPU-Kerne funktionieren. Der Raspberry Pi Pico 2 mit den RP2350-Prozessoren verfügt über drei PIO-Geräte. Eine detailliertere Beschreibung der Programmierung von PIOs findest du im Anhang F. Die PIO-Befehle werden wie MMBasic-Befehle behandelt. Gültige Befehle sind: jmp (<Bedingung>) <Ziel> wait <Polarität> gpio <gpio_num> wait <Polarität> pin <Pin-Nummer> wait <Polarität> irq (vorherige | nächste) <irq_num> (rel) wait <Polarität> jmppin (+ <Pin-Offset>) in <Quelle>, <Bitanzahl> out <Ziel>, <Bitanzahl> Push (iffull) push (iffull) block

	push (iffull) noblock
	ziehen (wenn leer)
	Pull (wenn leer) Block
	ziehen (wenn leer) ohne Block
	mov <Ziel>, (op) <Quelle>
	irq (vorherige | nächste) <irq_num> (rel)
	irq (vorherige | nächste) set <irq_num> (rel)
	irq (vorherige | nächste) nowait <irq_num> (rel)
	irq (vorherige | nächste) warte <irq_num> (rel)
	irq (vorherige | nächste) löschen <irq_num> (rel)
	set <Ziel>, <Wert>
	
	Nur RP2350
	mov rxfifo[y], isr
	mov rxfifo[<index>], isr
	mov osr, rxfifo[y] NB
	mov osr, rxfifo[<Index>] Hinweis: Nur RP2350
	
	Gültige Anweisungen:
	.program name
	.line n/next
	.label name
	.wrap Ziel
	.wrap
	.Seite festlegen n
	.end program [Liste]
	
	PIO-Befehle werden zwischen einem Befehl „PIO ASSEMBLE n" und einer Anweisung „.end program" benutzt. Andere Anweisungen können auch dazwischen stehen.

PIO assemble pio,linedata$	Dieser Befehl setzt textbasierten PIO-Assembler-Code zusammen und lädt ihn, einschließlich Sprungmarken. Verwendung: PIO assemble pio,".program anything", um den Assembler zu starten. Verwendung: PIO assemble pio,".side_set n [opt] [pindirs]", wenn Side Set verwendet wird. Dies ist erforderlich, um die Op-Codes korrekt zu erstellen, wenn ein oder mehrere Side-Set-Pins verwendet werden. Das Pin-Control-Register wird nicht geladen, da es spezifisch für die Zustandsmaschine ist. Beachte auch, dass der Parameter „opt" den Op-Code bei Anweisungen mit einem Side-Parameter ändert. Verwendung: PIO assemble pio, „.line n", um ab einer anderen Zeile als 1 zu assemblieren – das ist optional. Verwendung: PIO assemble pio,".end program [list]", um die Assemblierung zu beenden und das PIO zu programmieren. Der optionale Parameter LIST bewirkt eine Hex-Ausgabe der Op-Codes auf dem Terminal. Verwendung: PIO assemble pio,"label:" zum Definieren einer Bezeichnung. Dies muss als separater Befehl erscheinen. Verwendung: PIO assemble „'.wrap target", um anzugeben, wo das Programm

	umbrechen soll. Siehe PIO(.wrap target) für die Verwendung. Verwendung: PIO assemble „.wrap“, um anzugeben, wo das Programm von „.wrap target“ zurückkehren soll. Informationen zur Verwendung findest du unter PIO(.wrap).
PIO DMA RX pio, sm, nbr, data%() [,completioninterrupt] [,transfersize] [,loopbackcount]	Verwendung: PIO assemble pio "Befehl [Parameter]", um die eigentlichen PIO-Befehle zu definieren, die in Maschinencode umgewandelt werden sollen. Richtet DMA-Übertragungen von PIO zum MMBasic-Speicher ein.
PIO DMA TX pio, sm, nbr, data%() [,completioninterrupt] [,transfersize] [,loopbackcount]	pio gibt an, welche der beiden pio-Instanzen verwendet werden soll (0 oder 1). sm gibt an, welche der Zustandsmaschinen verwendet werden soll (0-3). nbr gibt an, wie viele 32-Bit-Wörter übertragen werden sollen. Sieh unten für den Sonderfall, wenn nbr auf Null gesetzt wird. data%() ist das Array, das die PIO-Daten entweder bereitstellt oder empfängt. Der optionale Parameter completioninterrupt ist der Name einer MMBasic-Subroutine, die aufgerufen wird, wenn die DMA abgeschlossen ist und im Fall von DMA_OUT der FIFO geleert wurde. Wenn der optionale Interrupt nicht verwendet wird, kann der Status des DMA mit den folgenden Funktionen überprüft werden: MM.INFO(PIO RX DMA) MM.INFO(PIO TX DMA) Der optionale Parameter transfersize ermöglicht es dem Benutzer, die normalen 32-Bit-Übertragungen zu überschreiben und 8, 16 oder 32 auszuwählen. Der optionale Parameter loopbackcount gibt an, wie viele Datenelemente gelesen oder geschrieben werden sollen, bevor der DMA am Anfang des Puffers wieder von vorne beginnt. Der Parameter muss eine Potenz von 2 zwischen 2 und 32768 sein. Wegen einer Einschränkung im RP2040/RP2350 muss das MMBasic-Array im Speicher an die Anzahl der Bytes in der Schleife angepasst werden, wenn loopbackcount benutzt wird. Wenn das Array also 64 Ganzzahlen lang ist, was 512 Bytes entspricht, muss das Array im Speicher an einer 512-Byte-Grenze ausgerichtet werden. Alle MMBasic-Arrays sind auf eine Grenze von 256 Byte ausgerichtet, aber um ein Array zu erstellen, das garantiert auf eine Grenze von 512 Byte oder mehr ausgerichtet ist, muss der Befehl PIO MAKE RING BUFFER verwendet werden. Wenn „loopbackcount“ gesetzt ist, kann „nbr“ auf 0 gesetzt werden. In diesem Fall läuft die Übertragung kontinuierlich und füllt den Puffer wiederholt, bis sie explizit gestoppt wird. Wenn „nbr” und „loopbackcount” beide angegeben und identisch sind, startet der DMA nach Abschluss automatisch neu (nur TX).
PIO DMA RX OFF PIO DMA TX OFF	Bricht einen laufenden DMA ab.
PIO INTERRUPT pio, sm [,RXinterrupt] [,TXinterrupt]	Legt grundlegende Interrupts für PIO-Aktivitäten fest. Verwende den Wert 0 für RXinterrupt oder TXinterrupt, um einen Interrupt zu deaktivieren. Nicht benötigte Werte weglassen. Der RX-Interrupt wird ausgelöst, wenn ein Wort vom PIO-Code in den angegebenen FIFO „geschoben“ wurde. Die Daten MÜSSEN im Interrupt gelesen werden, um ihn zu löschen.
PIO INIT MACHINE pio%, Zustandsmaschine%, Taktfrequenz [,pinctrl] [,execctrl] [,shiftctrl] [,startinstruction] [,sideout [,setout] [,outout]	

	Der TX-Interrupt wird ausgelöst, wenn der angegebene FIFO VOLL ist und der PIO-Code ihn nun „gezogen” hat. Initialisiert PIO „pio%” mit der Zustandsmaschine „statemachine%”. „clockspeed” ist die Taktrate der Zustandsmaschine in kHz. Die ersten vier optionalen Argumente sind Variablen, die die Initialisierungswerte der Zustandsmaschinenregister und die Adresse der ersten auszuführenden Anweisung enthalten (Standardwert ist Null). Diese bestimmen, wie die PIO funktioniert. 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 die Anweisung sofort auf dem angegebenen pio und der angegebenen Zustandsmaschine aus.
PIO WRITE pio, state_machine, count, data0 [,data1..]	Schreibt die Datenelemente in die angegebene PIO und Zustandsmaschine. Der Schreibvorgang ist blockierend, daher muss die Zustandsmaschine in der Lage sein, die bereitgestellten Daten zu übernehmen. Hinweis: Dieser Befehl wird in zukünftigen Versionen wahrscheinlich zusätzliche Funktionen brauchen.
PIO WRITEFIFO a,b,c,d	Schreibt in eines der 4 einzelnen FIFO-Register. „a” ist der PIO (0 oder 1), „b” ist die Zustandsmaschine (0...3), „c” ist das FIFO-Register *0…3) und „d” sind die Daten% (32-Bit-Ganzzahlwert).
PIO READ pio, state_machine, count, data% [()]	Liest die Datenelemente aus dem angegebenen pio und der angegebenen Zustandsmaschine. Das Lesen ist nicht blockierend, daher muss die Zustandsmaschine in der Lage sein, die angeforderten Daten bereitzustellen. Wenn count eins ist, kann eine Ganzzahl zum Empfangen der Daten verwendet werden, andernfalls sollte ein Ganzzahl-Array angegeben werden. Hinweis: Dieser Befehl wird in zukünftigen Versionen wahrscheinlich zusätzliche Funktionen brauchen.
PIO START pio, statemachine	Startet eine bestimmte Zustandsmaschine auf pio.
PIO STOP pio, statemachine	Stoppt eine bestimmte Zustandsmaschine auf pio.
PIO CLEAR pio	Das stoppt die auf allen Zustandsmaschinen angegebene pio und setzt die Steuerregister und Interrupt-Flags für die Zustandsmaschinen PINCTRL, EXECTRL und SHIFTCTRL auf die Standardwerte zurück.
PIO PROGRAM pio	Sagt, dass die nächsten Zeilen PIO-Assembler-Befehle für das PIO „pio” sind, bis eine „.end program”-Anweisung kommt.
PIO PROGRAM pio,array%()	Programmier den ganzen pio-Programmspeicher mit den Daten in array%(). Sieh Anhang F.
PIO PROGRAM LINE pio, line, instruction	Programmier nur die angegebene Zeile in einem PIO-Programm.

PIO SYNC pio, Zustandsmaschinen, [,vorherige Zustandsmaschinen] [,nächste Zustandsmaschinen]	Synchronisiert die Uhren der PIO-Zustandsmaschinen „pio" gibt den Referenz-PIO für den Befehl an. „statemachines", „prevstatemachines" und „nextstatemachines" sind Bitmaps (0 bis 15), die angeben, welche Zustandsmaschinenuhren synchronisiert werden sollen. Um also festzulegen, dass alle Zustandsmaschinen von PIO0 und PIO1 synchronisiert werden sollen, könntest du Folgendes verwenden: PIO SYNC 0,15,,15 oder PIO SYNC 1,15,15
PIO SET BASE 0/16	PIO-Befehle funktionieren nur mit 32 GPIO-Ports. Beim RP2350B sagt dieser Befehl dem System, dass es GP0-GP31 (0) oder GP16-GP47 (16) verwenden soll.
PIO CONFIGURE pio, sm, clock [,startaddress] [,sidesetbase] [,sidesetno] [,sidesetout] [,setbase] [,setno] [,setout] [,outbase] [,outno] [,outout] [,inbase] [,jmppin] [,wraptarget] [,wrap] [,sideenable] [,sidepindir] [,pushthreshold] [,pullthreshold] [,autopush] [,autopull] [,inshiftdir] [,outshiftdir] [,joinrxfifo] [,jointxfifo] [,joinrxfifoget] [,joinrxfifoput]	Die Parameter in diesem Befehl sind im Grunde die gleichen, die du im Befehl PIO INIT verwenden würdest, plus die Hilfsfunktionen PINCTRL, SHIFTCTRL und EXECCTRL, aber alles in einem einzigen Befehl zusammengefasst. Das ist nötig, weil das Pico SDK im Hintergrund ein paar clevere Sachen macht, um die RP2350B-Schnittstelle zu verwalten. „sidesetbase", „sidebase outbase", „inbase" und „jmppin" sind Pin-Definitionen. Du kannst diese entweder als GPno oder als Pin-Nummer (z. B. GP3 oder 5) angeben. 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 gültig. Sie sind alle standardmäßig auf den Basiswert gesetzt, mit Ausnahme von „jmppin" (Standardwert -1), das explizit gesetzt werden muss, wenn du ein „jmppin" verwenden möchtest, da dies die Einstellung des erforderlichen Statusbits auslöst. „clock" ist die gewünschte PIO-Taktrate in Hz. „startaddress" ist die PIO-Anweisung, die die Ausführung startet – Standardwert ist 0. „sidesetno", „setno" und „outno" geben die Anzahl der Pins an, die für diese Funktionen verwendet werden können – Standardwert ist 0. „sidesetout", „setout" und „outout" sagen, ob diese Pins als Ausgänge konfiguriert werden sollen (1=ja, 0=nein) – Standardwert ist 0 „wraptarget" und „wrap" liegen im Bereich von 0 bis 31 und sind standardmäßig auf 0 und 31 eingestellt. „inshiftdir" und „outshiftdir" sind standardmäßig auf 1 gesetzt – verschiebt das Ausgangs-Schieberegister nach rechts und das Eingangs-Schieberegister nach rechts (Daten kommen von links rein). Alle anderen Parameter sind Boolesche Werte, die eine bestimmte Funktion aktivieren können – 1 zum Aktivieren, 0 zum Deaktivieren – alle sind standardmäßig auf 0 gesetzt. Einfaches Beispiel: *'PIO Konfigurieren Sie pio, sm, clock, startaddress,* *'sidesetbase, sidesetno, sidesetout,* *'setbase, setno, setout, outbase, outno, outout, inbase,* *'jmppin, wraptarget, wrap, sideenable, sidepindir,* *'pushthreshold, pullthreshold, autopush, autopull, inshiftdir, outshiftdir,* *'joinrxfifo, jointxfifo, joinrxfifoget, joinrxfifoput* *PIO-Assembler 1* *.Programmtest* *.Zeile 0* *.wrap Ziel* *Pins setzen,1* *Pins setzen, 0* *.wrap*

	.end program *SetPin gp45,pio1* *PIO-Basis 1,16 setzen* *PIO konfigurieren 1,0,1000000,,,,,gp45,1,1,,,,,,Pio(.wrap Ziel),Pio(.wrap)* *PIO starten 1,0* *Ausführen* *Schleife* Obwohl der Befehl PIO CONFIGURE viele Parameter hat, ist er echt einfach zu benutzen, 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, ohne die Kommas zu verändern. Wenn du alle Ersetzungen gemacht hast, lösch alle nachstehenden Kommas. Wenn die Zeile dann zu lang für den Editor ist, lösch die CRs nacheinander, beginnend am Ende der vorletzten Zeile und nach oben arbeitend. Auf diese Weise erhältst du einen gültigen Befehl, der einfach einzugeben und zu bearbeiten ist. Hinweis: Du kannst auch Fortsetzungszeilen verwenden, um die Bearbeitung zu vereinfachen (siehe OPTION CONTINUATION LINES).
PIXEL x, y [,c]	Lege einen Pixel auf einem Videoausgang oder einem angeschlossenen LCD-Bildschirm auf eine Farbe fest. „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. Wenn es weggelassen wird, wird die aktuelle Vordergrundfarbe verwendet. Alle Parameter können als Arrays ausgedrückt werden, und die Software zeichnet die Anzahl der Pixel entsprechend den Abmessungen des kleinsten Arrays. „x" und „y" müssen entweder beide Arrays oder beide einzelne Variablen/Konstanten sein, sonst wird ein Fehler ausgegeben. „c" kann entweder ein Array oder eine einzelne Variable oder Konstante sein. Eine Definition der Farben und Grafikkoordinaten findest du im Kapitel *Grafikbefehle und -funktionen*.
PLAY PLAY TONE links, rechts [,dauer] [,unterbrechen]	Dieser Befehl erzeugt verschiedene Audioausgaben. Siehe den Befehl OPTION AUDIO zum Einstellen der für die Ausgabe zu verwendenden I/O-Pins. Das Audiosignal ist ein pulsweitenmoduliertes Signal (PWM), sodass ein Tiefpassfilter erforderlich ist, um die Trägerfrequenz zu entfernen. Erzeugt zwei separate Frequenzen auf den linken und rechten Kanälen der Audioausgabe. „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, wird der Ton so lange abgespielt, bis er explizit gestoppt wird oder das Programm beendet wird. „interrupt" ist eine optionale Subroutine, die aufgerufen wird, wenn die Wiedergabe beendet wird. 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 Befehl PLAY TONE geändert werden.

PLAY FLAC file$ [, interrupt]	Spielt eine FLAC-Datei über den Soundausgang ab. „file$" ist die abzuspielende FLAC-Datei (die Erweiterung .flac wird angehängt, falls sie fehlt). Die Abtastrate kann bis zu 48 kHz in Stereo betragen (96 kHz, wenn der Pico übertaktet ist). Die FLAC-Datei wird im Hintergrund abgespielt. 'interrupt' ist optional und ist der Name einer Subroutine, die aufgerufen wird, wenn die Datei fertig abgespielt ist. Wenn file$ 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. „file$" ist die abzuspielende WAV-Datei (die Erweiterung .wav wird angehängt, falls sie fehlt). Die WAV-Datei muss PCM-codiert in Mono oder Stereo mit 8 oder 16 Bit Abtastrate sein. Die Abtastrate kann bis zu 48 kHz in Stereo betragen (96 kHz, wenn der Pico übertaktet ist). Die WAV-Datei wird im Hintergrund abgespielt. „interrupt" ist optional und ist der Name einer Subroutine, die aufgerufen wird, wenn die Datei fertig abgespielt ist.
PLAY ARRAY l%(), r%(), freq [,start] [,end] [,terminationinterrupt]	Gibt Daten über den Audioausgang aus, indem Daten in einem oder zwei Arrays verwendet werden: Dafür braucht man gepackte Arrays „l%" und „r%" (können für links und rechts dasselbe Array sein) mit 16-Bit-Werten im Bereich von -32768 bis 32767 und gibt die Werte im Array mit der angegebenen Abtastfrequenz aus. Wenn der optionale Parameter „start" angegeben wird, werden die Arrays ab der gepackten Position „start" ausgegeben. Wenn der optionale Parameter „end" angegeben wird, werden die Arrays bis zum Erreichen der gepackten Position „end" ausgegeben. Wenn der optionale Parameter „terminationinterrupt" angegeben wird, wird die Subroutine ausgeführt, sobald das letzte Array-Element ausgegeben wurde. Es ist wichtig, den Parameter „freq" zu verstehen. Das ist die Rate, mit der jedes Array-Element ausgegeben wird. Wenn das Array zum Beispiel 10 Zyklen einer Sinuswelle mit insgesamt 18000 Samples enthält, würde die Einstellung der Frequenz auf 1800 eine 1-Hz-Sinuswelle für 10 Sekunden ausgeben. Die Einstellung der Frequenz auf 18000 würde eine 10-Hz-Sinuswelle für 1 Sekunde ausgeben.
PLAY MP3 file$ [, interrupt]	Spielt eine MP3-Datei über den Soundausgang ab (NUR RP2350). „file$" ist die abzuspielende MP3-Datei (die Erweiterung .mp3 wird angehängt, falls sie fehlt). Die Abtastrate kann bis zu 48 kHz betragen. Die MP3-Datei wird im Hintergrund abgespielt. „interrupt" ist optional und ist der Name einer Subroutine, die aufgerufen wird, wenn die Datei fertig abgespielt ist. Wenn file$ ein Verzeichnis auf Laufwerk B: ist, spielt der Pico alle Dateien in diesem Verzeichnis nacheinander ab.
PLAY MODFILE Datei$ [, Unterbrechung]	Spielt eine MOD-Datei über den Soundausgang ab. „file$" ist die abzuspielende MOD-Datei (die Erweiterung .mod wird angehängt, falls sie fehlt). Die MOD-Datei läuft im Hintergrund und wird in einer Endlosschleife („ ") abgespielt. Wenn die optionale Option „interrupt" angegeben ist, wird diese aufgerufen, sobald die Datei einmal durchgespielt wurde, und die Wiedergabe wird dann beendet. Dieser Befehl nutzt vorzugsweise Speicherplatz im PSRAM, wenn er für den Dateipuffer aktiviert ist (nur RP2350). In diesem Fall

	muss ein Modpuffer nicht mit dem Befehl OPTION aktiviert werden.
PLAY MODSAMPLE Samplenum, channel [,volume]	Spielt ein bestimmtes Sample in 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 jede 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 Sounds gleichzeitig auf dem Audioausgang ab. „soundno" ist die Soundnummer und kann zwischen 1 und 4 liegen, sodass vier Sounds gleichzeitig auf jedem Kanal abgespielt werden können. „channelno" gibt den Ausgangskanal an. Er kann L (linker Lautsprecher), R (rechter Lautsprecher) oder B (beide Lautsprecher) sein. „type" gibt die zu verwendende Wellenform an. Sie kann S (Sinuswelle), Q (Rechteckwelle), T (Dreieckwelle), W (Sägezahnwelle), O (Null-Ausgabe), P (periodisches Rauschen), N (zufälliges Rauschen) oder U (benutzerdefiniert mit PLAY LOAD SOUND) sein. „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 werden alle folgenden Beispiele die Wellenform in ihrer ursprünglichen Tonhöhe wiedergeben: Abtastrate von 4000, Frequenz = 1 Abtastrate von 8000, Frequenz = 2 Abtastrate von 16000, Frequenz = 4 „Lautstärke" ist optional und muss zwischen 1 und 25 liegen. Der Standardwert ist 25. Wenn PLAY SOUND aufgerufen wird, werden alle anderen Audiofunktionen blockiert und bleiben blockiert, bis PLAY STOP aufgerufen wird. Die Ausgabe kann mit PLAY PAUSE und PLAY RESUME vorübergehend angehalten werden. Wenn du SOUND bei einem schon laufenden „soundno" aufrufst, wird die vorherige Ausgabe sofort ersetzt. Einzelne Sounds kannst du mit dem Typ „O" ausschalten. Wenn 4 Sounds gleichzeitig auf beiden Kanälen der Audioausgabe laufen, verbraucht das etwa 23 % der CPU.
PLAY PAUSE PLAY RESUME PLAY STOP	PLAY PAUSE hält die aktuell abgespielte Datei oder den Ton vorübergehend an. PLAY RESUME setzt die Wiedergabe eines angehaltenen Sounds fort. PLAY STOP beendet die Wiedergabe der Datei oder des Tons. Wenn das Programm aus irgendeinem Grund beendet wird, wird auch die Tonausgabe automatisch gestoppt.
PLAY VOLUME links, rechts	Passt die Lautstärke der Audioausgabe an. „links" und „rechts" sind die Pegel für den linken und rechten Kanal und können zwischen 0 und 100 liegen, wobei 100 die maximale Lautstärke ist. Es gibt eine lineare Beziehung zwischen dem eingestellten Pegel und der Ausgabe. Die Lautstärke ist standardmäßig auf Maximum eingestellt, wenn ein Programm gestartet wird.
PLAY NEXT	

VORHERIGE WIEDERGABE	Stoppt die Wiedergabe der aktuellen Audiodatei und startet die nächste im Verzeichnis Stoppt die Wiedergabe der aktuellen Audiodatei und startet die vorherige im Verzeichnis.
	SPEZIELLE VS1053-WIEDERGABE-BEFEHLE
MP3-Datei abspielen$ [, Unterbrechung]	Spielt eine MP3-Datei über den Soundausgang ab. „file$" ist die abzuspielende MP3-Datei (die Erweiterung .mp3 wird angehängt, falls sie fehlt). Die Abtastrate sollte 44100 Hz Stereo sein. Die MP3-Datei wird im Hintergrund abgespielt. „interrupt" ist optional und ist der Name einer Subroutine, die aufgerufen wird, wenn die Datei fertig abgespielt ist. Wenn file$ ein Verzeichnis auf dem Laufwerk B: ist, spielt der Pico alle Dateien in diesem Verzeichnis nacheinander ab.
PLAY HALT	Dieser Befehl funktioniert, wenn 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 für die Unterstützung von MP3-Hörbüchern entwickelt.

PLAY CONTINUE track$	Setzt die Wiedergabe des angegebenen MP3-Titels fort. „track$" ist der Name der Datei, die bei der Unterbrechung abgespielt wurde, wobei alle Dateiattribute entfernt wurden.
	z. B.
	PLAY MP3 „B:/mp3/mymp3.mp3"
	  etwas später
	PLAY HALT
	  später nochmal
	PLAY CONTINUE „mymp3"

MIDIFILE-Datei abspielen$ [, Unterbrechung]	Spielt eine MIDI-Datei über den Soundausgang ab. „file$" ist die MIDI-Datei, die abgespielt werden soll (die Erweiterung .mid wird angehängt, falls sie fehlt). Die MIDI-Datei wird im Hintergrund abgespielt. „interrupt" ist optional und ist der Name einer Subroutine, die aufgerufen wird, wenn die Datei fertig abgespielt ist. Wenn file$ ein Verzeichnis auf Laufwerk B: ist, spielt der Pico alle Dateien in diesem Verzeichnis nacheinander ab.
MIDI abspielen	Startet den Echtzeit-MIDI-Modus. In diesem Modus kannst du MIDI-Befehle an den VS1053 senden, um auszuwählen, welche Instrumente auf welchen Kanälen gespielt werden sollen, Noten einzuschalten und sie in Echtzeit auszuschalten.
MIDI-Befehl cmd%, Daten1%, Daten2% abspielen	Sendet einen MIDI-Befehl im Echtzeit-MIDI-Modus. Ein Beispiel wäre die Zuweisung eines Instruments zu einem Kanal. Z. B. PLAY MIDI CMD &B11000001,4 „Kanal 1 auf Instrument 4 einstellen".
MIDI TEST abspielen n	Spielt eine MIDI-Testsequenz ab, n=0 bis 3, 0 = normale Echtzeit, die anderen spielen Noten- und Instrumentensamples ab.
SPIELE NOTE EIN Kanal%, Note%, Anschlagstärke%	Schaltet die Note auf dem angegebenen Kanal ein, wenn du im Echtzeit-MIDI-Modus bist.
SPIELE NOTE AUS Kanal%, Note% [, Anschlagstärke%]	Schaltet die Note auf dem angegebenen Kanal im Echtzeit-MIDI-Modus aus.
STREAM ABSPIELEN buffer %(), readpointer%, writepointer%	

	Sendet Daten aus dem Ringpuffer „buffer%" an den VS1053-CODEC. Dieser Befehl startet einen Hintergrund-Ausgabestrom, bei dem alles, was sich im Puffer zwischen dem Lesepfeil und dem Schreibpfeil befindet, an den VS1053 gesendet wird, wobei der Lesepfeil fortlaufend aktualisiert wird. Kann für die Ausgabe beliebiger Wellenformen verwendet werden.
POKE BYTE addr%, byte oder POKE SHORT addr%, short% oder POKE WORD addr%, word% oder POKE INTEGER addr%, int% oder POKE FLOAT addr%, float! oder POKE VAR var, offset, byte oder POKE VARTBL, offset, byte	Setzt ein Byte oder ein Wort 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, sonst wird ein Fehler gemeldet. POKE BYTE setzt das Byte (d. h. 8 Bit) an der Speicherstelle „addr%" auf „byte". „addr%" sollte eine ganze Zahl sein. POKE SHORT setzt die kurze Ganzzahl (d. h. 16 Bit) an der Speicherstelle „addr%" auf „word%". „addr%" und „short%" sollten ganze Zahlen sein. POKE WORD setzt das Wort (also 32 Bit) an der Speicherstelle 'addr%' auf 'word%'. 'addr%' und 'word%' sollten ganze Zahlen sein. POKE INTEGER setzt die MMBasic-Ganzzahl (also 64 Bit) an der Speicherstelle 'addr%' auf 'int%'. 'addr%' und 'int%' sollten Ganzzahlen sein. POKE FLOAT setzt das Wort (also 64 Bit) an der Speicherstelle 'addr%' auf 'float!'. 'addr%' sollte eine ganze Zahl und 'float!' eine Gleitkommazahl sein. 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 Befehl [,Daten1] [,Daten2] [,Daten3]	Dieser Befehl sendet Befehle und zugehörige Daten an den Display-Controller für ein angeschlossenes Display. Damit kann der Programmierer Parameter für die Konfiguration des Displays ändern. Beispielsweise schaltet POKE DISPLAY &H28 ein SSD1963-Display aus und POKE DISPLAY &H29 schaltet es wieder ein. Funktioniert für alle Displays außer dem ST7790.
POKE DISPLAY HRES n 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] POLYGON n(), xarray%(), yarray%() [, bordercolour()] [, fillcolour()] 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 Vordergrundfarbe verwendet. 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. „n" kann ein Array sein, und die Farben können optional auch Arrays sein, wie folgt: POLYGON n(), xarray%(), yarray%() [, bordercolour()] [, fillcolour()] POLYGON n(), xarray%(), yarray%() [, bordercolour] [, fillcolour] Die Elemente von „array n()" sagen, wie viele xy-Koordinatenpaare in jedem der Polygone sind. Zum Beispiel würde DIM n(1)=(3,3) bedeuten, dass zwei Polygone mit jeweils drei Eckpunkten gezeichnet werden sollen. Die Größe des n-Arrays entscheidet, wie viele Polygone gezeichnet werden, es sei denn, es gibt ein Element mit dem Wert Null. In diesem Fall verarbeitet die Firmware nur die Polygone bis zu diesem Punkt. Die x-, y-Koordinatenpaare für alle Polygone werden in „xarray%()" und „yarray%()" gespeichert. Die

	Parameter „xarray%()“ und „yarray%()“ müssen mindestens so viele Elemente haben wie die Summe der Werte im Array n.
	Jedes Polygon kann geschlossen werden, wenn das erste und das letzte Element gleich sind. Ist das letzte Element nicht gleich dem ersten, erstellt die Firmware automatisch ein zusätzliches x,y-Koordinatenpaar, um das Polygon zu vervollständigen. Wird die Füllfarbe weggelassen, werden nur die Polygonkonturen gezeichnet.
	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 sowohl für den Rand als auch für die Füllung haben.
	Beispielsweise werden hier drei Dreiecke in Gelb, Grün und Rot gezeichnet:
	`DIM c%(2)=(3,3,3)`
	`DIM x%(8)=(100,50,150,100,50,150,100,50,150)`
	`DIM y%(8)=(50,100,100,150,200,200,250,300,300)`
	`DIM fc%(2)=(rgb(gelb),rgb(grün),rgb(rot))`
	`POLYGON c%(),x%(),y%(),fc%(),fc%()`

PORT(start, nbr [,start, nbr]…) = Wert	Legt mehrere I/O-Pins gleichzeitig fest (also mit einem Befehl). „start” ist eine I/O-Pin-Nummer, und das niedrigste Bit in „value” (Bit 0) wird zum Einstellen dieses Pins verwendet. Bit 1 wird zum Einstellen des Pins „start” plus 1 verwendet, Bit 2 zum Einstellen des Pins „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, führt zu einem Fehler. Das Start/nbr-Paar kann wiederholt werden, wenn eine zusätzliche Gruppe von Ausgangspins hinzugefügt werden muss. Zum Beispiel: `PORT(15, 4, 23, 4) = &B10000011` Setzt acht I/O-Pins. 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 gesetzt wird. 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) von 1 bis zur Anzahl der I/O-Pins auf dem Chip verwendet werden. Hinweis: Wenn die Pins mit der GPn-Syntax definiert werden, ignoriert die Firmware ungültige Pins, also `PORT(GP0, 8) = &B10000011` Setzt acht I/O-Pins. Die Pins GP0, GP1 und GP7 werden auf High gesetzt, während GP2, GP3, GP4, GP5 und GP6 auf Low gesetzt werden. Siehe die Funktion PORT, um gleichzeitig von mehreren Pins zu lesen.
PRINT Ausdruck [[,;]Ausdruck] … usw.	Gibt Text an die serielle Konsole aus, gefolgt von einem Zeilenumbruch/Zeilenvorschub-Paar. Es können mehrere Ausdrücke verwendet werden, die durch eines der folgenden Zeichen getrennt werden müssen: • Komma (,) getrennt werden, wodurch das Tabulatorzeichen ausgegeben wird • Semikolon (;), das nichts ausgibt (es wird nur zum Trennen von Ausdrücken verwendet). • Nichts oder ein Leerzeichen, das wie ein Semikolon wirkt. Ein Semikolon (;) oder ein Komma (,) am Ende der Ausdrucksliste unterdrückt die Ausgabe des Zeilenumbruch-/Zeilenvorschub-Paares am Ende einer Druckanweisung. Beim Drucken wird einer positiven Zahl ein Leerzeichen vorangestellt, einer negativen Zahl ein Minuszeichen (-), aber es folgt kein Leerzeichen. Ganzzahlen werden ohne Dezimalpunkt gedruckt, während Brüche mit

	Dezimalpunkt und den signifikanten Dezimalstellen gedruckt werden. Große oder kleine Gleitkommazahlen werden automatisch im wissenschaftlichen Zahlenformat gedruckt. Mit der Funktion TAB() kann ein Abstand zu einer bestimmten Spalte eingefügt werden, und mit der Funktion STR$() können Zeichenfolgen ausgerichtet oder anderweitig formatiert werden.
PRINT #nbr, Ausdruck [[,;]Ausdruck] … usw.	Wie oben, nur dass die Ausgabe an einen seriellen Kommunikationsport oder eine Datei geleitet wird, die für OUTPUT oder APPEND mit der Dateinummer „nbr" geöffnet ist. Sieh dir den Befehl OPEN an.
PRINT #GPS, Ausdruck [[,;]Ausdruck] … usw.	Schreibt eine NMEA-Zeichenkette an ein geöffnetes GPS-Gerät. 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]) Ausdruck oder PRINT @(x, [y], m) Ausdruck	Funktioniert auf der Terminal-Konsole eines angeschlossenen Computers oder auf einem VGA/HDMI-Bildschirm oder dem Display, wenn OPTION LCDPANEL CONSOLE aktiviert ist. Funktioniert wie der Standardbefehl PRINT, außer dass der Cursor an den Koordinaten x, y positioniert wird, die in Pixeln angegeben sind. Wenn y weggelassen wird, wird der Cursor an „x" in der aktuellen Zeile positioniert. Beispiel: PRINT @(150, 45) „Hallo Welt" Die @-Funktion kann überall in einem Druckbefehl verwendet werden. Beispiel: PRINT @(150, 45) „Hallo" @(150, 55) „Welt" Mit der Funktion @(x,y) kannst du den Cursor an einer beliebigen Stelle auf oder außerhalb des Bildschirms positionieren. Wenn du zum Beispiel PRINT @(-10, 0) "Hallo" eingibst, wird nur „allo" angezeigt, weil die ersten beiden Zeichen außerhalb des Bildschirms liegen und nicht angezeigt werden können. Die Funktion @(x,y) verhindert automatisch den Zeilenumbruch, der normalerweise passiert, wenn der Cursor über den rechten Bildschirmrand hinausgeht. Wenn „m" angegeben ist, läuft der Videomodus so ab: m = 0 Normaler Text (weiße Buchstaben, schwarzer Hintergrund) m = 1 Der Hintergrund wird nicht gezeichnet (d. h. transparent) m = 2 Das Video wird umgekehrt (schwarze Buchstaben, weißer Hintergrund) m = 5 Die aktuellen Pixel werden umgekehrt (transparenter Hintergrund).
PULSE-Pin, Breite	Erzeugt einen Impuls am „Pin" mit einer Dauer von „Breite" ms. „Breite" kann ein Bruchteil sein. Zum Beispiel entspricht 0,01 10 µs, was die Erzeugung sehr schmaler Impulse ermöglicht. Der erzeugte Impuls hat die entgegengesetzte Polarität zum Zustand des I/O-Pins, wenn der Befehl ausgeführt wird. Wenn der Ausgang beispielsweise auf „high" gesetzt ist, erzeugt der Befehl PULSE einen negativen Impuls. Hinweise: • „Pin" muss als Ausgang konfiguriert sein. • Bei einem Impuls von weniger als 3 ms beträgt die Genauigkeit ± 1 µs. • Bei einem Impuls von 3 ms oder mehr beträgt die Genauigkeit ± 0,5 ms. • Ein Impuls von 3 ms oder mehr läuft im Hintergrund. Bis zu fünf verschiedene und gleichzeitige Impulse können im Hintergrund laufen, und jeder kann durch einen neuen PULSE-Befehl zeitlich geändert oder durch einen PULSE-Befehl mit Null für „width" beendet werden.
PWM-Kanal, Frequenz,	Es stehen 8 separate PWM-Frequenzen (Kanäle 0 bis 7) und bis zu 16

[dutyA] [,dutyB][,phase] [,defer]	Ausgänge mit individuell gesteuerter Einschaltdauer zur Verfügung. Du kannst für jeden Kanal entweder PWMnA oder PWMnB oder beides ausgeben – ohne Einschränkung. Die Einschaltdauer wird als Prozentsatz angegeben, und du kannst einen negativen Wert verwenden, um die Ausgabe umzukehren (-100,0 <= duty <=100,0). Um nur Kanal B zu verwenden, benutze die Syntax: „PWM-Kanal, Frequenz, , dutyB“ – beachte das doppelte Komma vor dem gewünschten Tastverhältnis. Minimale Frequenz = (cpuspeed + 1) / (2^24) Hz. Die maximale Geschwindigkeit ist OPTION CPUSPEED/4. Bei sehr hohen Geschwindigkeiten werden die Arbeitszyklen zunehmend eingeschränkt. Phase ist ein Parameter, der bewirkt, dass die Wellenformen so zentriert werden, dass eine Wellenform mit einem kürzeren Arbeitszyklus zur gleichen Zeit beginnt und endet wie eine längere. Verwende 1, um diesen Modus zu aktivieren, und 0 (oder lass ihn weg), um normal zu laufen. Wenn der Parameter „deferredstart” auf 1 gesetzt ist, werden die PWM-Kanäle konfiguriert, aber die Ausgabe wird nicht gestartet. Sie können dann mit dem Befehl PWM SYNC gestartet werden. Dies kann verwendet werden, um unerwünschte Startartefakte zu vermeiden. Der PWM-Befehl kann auch Servos wie folgt ansteuern: `PWM 1,50,(Position_als_Prozentwert * 0,05 + 5)`
PWM SYNC s0 [,s1][,s2][,s3] [,s4][,s5][,s6][,s7]	Dies initiiert die 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 Arbeitszyklus – kann ein Float sein). Du kannst einen Versatz von -1 verwenden, um einen Kanal aus der Synchronisation auszuschließen.
PWM-Kanal, AUS	Stopp der Ausgabe auf „Kanal”.

RAM	RP2350 nur mit aktiviertem PSRAM Der RAM-Befehl ermöglicht den Zugriff auf bis zu 5 RAM-Programmslots (ähnlich wie Flash-Slots). RAM-Slots bleiben nach einem Hardware- und Software-Reset erhalten, aber nicht nach einem Neustart.
RAM-LISTE	Zeigt eine Liste aller RAM-Speicherplätze einschließlich der ersten Zeile des Programms an.
RAM-LISTE n [,alle]	Listet das auf Speicherplatz n gespeicherte Programm auf. Mit ALL wird ohne Seitenumbrüche aufgelistet.
RAM-LÖSCHEN n	Löscht einen RAM-Programmspeicherplatz.
RAM-LÖSCHUNG ALL	Löscht alle RAM-Programmspeicherplätze.
RAM SPEICHERN n	Speicher das aktuelle Programm an dem angegebenen RAM-Speicherplatz.
RAM-LADEN n	Lade ein Programm vom angegebenen RAM-Speicherplatz in den Programmspeicher.
RAM-LAUF n	Startet das Programm an Speicherplatz n, löscht alle Variablen. Ändert nichts am Programmspeicher.
RAM-Kette n	Führt das Programm an Speicherplatz n aus und lässt alle Variablen so, wie sie sind (damit kann das Programm, das viel größer ist als der Programmspeicher). Ändert den Programmspeicher. Hinweis: Wenn das verkettete Programm den Befehl READ verwendet, muss es vor dem ersten Lesevorgang den Befehl RESTORE aufrufen.
RAM ÜBERSCHREIBEN n	Löscht einen RAM-Programmspeicherplatz und speichert dann das aktuelle Programm an der angegebenen RAM-Position gespeichert.
RAM-DATEI LADEN n, fname$ [,O[ÜBERSCHREIBEN]]	Lädt die MMBasic-Datei fname$ in den angegebenen RAM-Speicherplatz. Wenn der optionale Parameter OVERWRITE (oder O) angegeben ist, wird der Inhalt des Flash-Speicherplatz ohne Fehlermeldung überschrieben.
RANDOMIZE nbr	Setzt den Zufallszahlengenerator mit „nbr" zurück. Auf dem RP2040 wird der Zufallszahlengenerator beim Einschalten mit Null initialisiert und erzeugt jedes Mal die gleiche Folge von Zufallszahlen. Um jedes Mal eine andere Zufallsfolge zu erzeugen, musst du einen anderen Wert für „nbr" verwenden (die TIMER-Funktion ist dafür praktisch). Dieser Befehl hat keine Wirkung auf dem RP2350, der über einen Hardware-Zufallsgenerator verfügt, der keine Initialisierung erfordert.
RBOX x, y, w, h [, r] [,c] [,fill]	Zeichnet ein Rechteck mit abgerundeten Ecken auf dem Videoausgang oder dem angeschlossenen LCD-Bildschirm, beginnend bei „x" und „y", das „w" Pixel breit und „h" Pixel hoch ist. „r" ist der Radius der Ecken des Kastens. Der Standardwert ist 10. „c" gibt die Farbe an und ist standardmäßig die Standard-Vordergrundfarbe, wenn nichts anderes angegeben ist. „fill" ist die Füllfarbe. Sie kann weggelassen oder auf -1 gesetzt werden, dann wird das Feld nicht gefüllt. 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", „y", „w" und „h" müssen alle Arrays oder alle

	einzelne Variablen/Konstanten sein, sonst kommt es zu einem Fehler „ ”. „r”, „c” und „fill” können entweder Arrays oder einzelne Variablen/Konstanten sein. Eine Definition der Farben und Grafikkoordinaten findest du im Kapitel *„Grafikbefehle und -funktionen*”.
REDIM [PRESERVE] array1(dimensions) [, array2(dimensions]… [,arrayn(dimensions]	Damit änderst du die Größe der angegebenen Arrays mit den angegebenen Dimensionen. Wenn der optionale Unterbefehl PRESERVE angegeben wird, werden die vorhandenen Daten in das neue Array kopiert. Das neue Array kann größer oder kleiner als das Original sein. Bei Zeichenfolgen-Arrays bleibt die ursprünglich angegebene LÄNGE erhalten. Beachte, dass bei mehrdimensionalen Arrays nur die letzte Dimension geändert werden kann, wenn PRESERVE verwendet wird. Bei Verwendung von PRESERVE muss genügend Speicherplatz vorhanden sein, damit sowohl das ursprüngliche Array als auch seine 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, wenn sie gelesen werden. Arrays können als Variablen verwendet werden (angegeben mit leeren Klammern, z. B. a()) und in diesem Fall wird die Größe des Arrays verwendet, um zu bestimmen, wie viele Elemente gelesen werden sollen. Wenn das Array mehrdimensional ist, wird die Dimension ganz links am schnellsten verschoben. Siehe auch DATA und RESTORE.
READ SAVE oder READ RESTORE	READ SAVE speichert den virtuellen Zeiger, den der Befehl READ benutzt, um auf die nächsten zu lesenden DATA zu zeigen. READ RESTORE stellt den vorher gespeicherten Zeiger wieder her. Dadurch können Unterprogramme Daten lesen und anschließend den Lesepointer wiederherstellen, um andere Teile des Programms, die möglicherweise dieselben Datenanweisungen lesen, nicht zu stören. Diese Befehle können verschachtelt werden.
REFRESH	Startet eine Aktualisierung des Bildschirms für E-Ink-Schwarzweiß-Displays. Diese können nur bildschirmweise aktualisiert werden. Wenn OPTION AUTOREFRESH auf OFF steht, kann dieser Befehl zum Auslösen des Schreibvorgangs verwendet werden. Dieser Befehl funktioniert mit den folgenden Displays: N5110, SSD1306I2C, SSD1306I2C32, SSD1306SPI, ST7920.
REM-Zeichenfolge	REM ermöglicht das Einfügen von Anmerkungen in ein Programm. Beachte, dass die Verwendung von einfachen Anführungszeichen (‘) im Microsoft-Stil zur Kennzeichnung von Anmerkungen auch unterstützt wird und bevorzugt ist.
RENAME old$ AS new$	Benenne eine Datei oder ein Verzeichnis von „old$” in „new$” um. Beide sind Zeichenfolgen. Sowohl in „old$” als auch in „new$” kann ein Verzeichnispfad verwendet werden. Wenn die Pfade unterschiedlich sind, wird die in „old$” angegebene Datei mit dem angegebenen Dateinamen in den in „new$” angegebenen Pfad verschoben.

RESTORE [Zeile]	Setzt die Zeilen- und Positionszähler für die READ-Anweisung zurück. Wenn „Zeile" angegeben ist, werden die Zähler auf den Anfang der angegebenen Zeile zurückgesetzt. „Zeile" kann eine Zeilennummer, eine Bezeichnung oder eine Variable mit diesen Werten sein. Wenn „Zeile" nicht angegeben ist, werden die Zähler auf den Anfang des Programms zurückgesetzt.
RMDIR dir	Entfernt oder löscht das Verzeichnis „dir$" auf dem Standard-Flash-Dateisystem oder der SD-Karte.
RTC GETTIME RTC SETTIME Jahr, Monat, Tag, Stunde, Minute, Sekunde RTC SETREG reg, Wert RTC GETREG reg, var	RTC GETTIME ruft das aktuelle Datum/die aktuelle Uhrzeit von einer Echtzeituhr vom Typ PCF8563, DS1307, DS3231 oder RV3028 ab und stellt die interne MMBasic-Uhr ein. Das Datum/die Uhrzeit kann dann mit den Funktionen DATE$ und TIME$ abgerufen werden. RTC SETTIME stellt die Zeit im Uhrenchip ein. „hour" muss im 24-Stunden-Format angegeben werden. „Jahr" kann zwei- oder vierstellig sein. Der Befehl RTC SETTIME akzeptiert auch ein einzelnes String-Argument im Format *TT/MM/JJ HH:MM*. Das heißt, das Datum/die Uhrzeit kann vom Benutzer über eine GUI-FORMATBOX mit dem Format DATETIME2 eingegeben werden (siehe *Advanced Graphics Functions.pdf*). Die Befehle RTC SETREG und GETREG können zum Einstellen oder Auslesen des Inhalts von Registern im Echtzeituhr-Chip verwendet werden. „reg" ist die Nummer des Registers, „value" ist die im Register zu speichernde Zahl und „var" ist eine Variable, die die aus dem Register gelesene Zahl empfängt. 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. Diese Chips sind I^2C-Geräte und müssen mit den beiden I^2C-Pins verbunden werden, wie in OPTION SYSTEM I2C angegeben, mit entsprechenden Pullup-Widerständen. Siehe auch den Befehl OPTION RTC AUTO ENABLE.
RUN oder RUN [Datei$] [, Befehlszeile$]	Ein Programm ausführen. Wenn „file$" nicht angegeben ist, wird das Programm ausgeführt, das gerade im Programmspeicher ist. Wenn „file$" angegeben ist, wird die genannte Datei aus dem Flash- oder SD-Karten-Dateisystem ausgeführt. Wenn „file$" keine Erweiterung „.BAS" enthält, wird diese automatisch hinzugefügt. Wenn „cmdline$" angegeben ist, wird sein Wert beim Ausführen des Programms an die Konstante MM.CMDLINE$ übergeben. Wenn „cmdline$" nicht angegeben ist, wird ein leerer Stringwert an MM.CMDLINE$ übergeben. Hinweise: • Sowohl „file$" als auch „cmdline$" können als Zeichenfolgenausdrücke angegeben werden. • Mit FLASH RUN n kannst du ein Programm ausführen, das in einem Flash-Slot gespeichert ist.
SAVE file$	Speichert das Programm im aktuellen Arbeitsverzeichnis des Flash-Dateisystems oder der SD-Karte als „file$". Beispiel: SAVE „TEST.BAS" Wenn keine Erweiterung angegeben wird, wird „.BAS" an den Dateinamen angehängt. 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 Top-Level-Programm 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. Siehe auch LOAD CONTEXT
SAVE IMAGE file$ [,x, y, w, h] oder SPEICHERE KOMPRIMIERTES BILD file$ [,x, y, w, h]	Speichert das aktuelle Bild auf dem Videoausgang oder dem LCD-Bildschirm als BMP-Datei. Der LCD-Bildschirm muss lesbar sein, zum Beispiel ein Bildschirm auf ILI9341-Basis oder ein VIRTUAL_M- oder VIRTUAL_C-Bildschirm. „file$" ist der Name der Datei. Wenn keine Erweiterung angegeben wird, wird „.BMP" an den Dateinamen angehängt. Das Bild wird als Echtfarben-24-Bit-Bild gespeichert. „x", „y", „w" und „h" sind optional und stehen für die Koordinaten („x" und „y" sind die Koordinaten oben links) und Abmessungen (Breite und Höhe) des zu speichernden Bereichs. Wenn nichts angegeben wird, wird der ganze Bildschirm gespeichert. Beachte, dass „width", wenn es benutzt wird, ein Vielfaches von 2 sein muss. SAVE COMPRESSED IMAGE funktioniert genauso, nur dass die Dateigröße durch RLE-Komprimierung reduziert wird.
SAVE PERSISTENT n%	Speichert den Wert n% an einem speziellen Speicherort, der einen Watchdog-Reset oder einen physischen Reset übersteht, aber nicht einen Neustart. Siehe auch MM.PERSISTENT oder MM.INFO(PERSISTENT)
SEEK [#]fnbr, pos	Positioniert den Lese-/Schreibzeiger in einer Datei, die im Flash-Dateisystem oder auf der SD-Karte geöffnet wurde, für den zufälligen Zugriff auf das Byte „pos". Das erste Byte in einer Datei hat die Nummer eins, also setzt SEEK #5,1 den Lese-/Schreibzeiger an den Anfang der als #5 geöffneten Datei.
SELECT CASE Wert CASE testexp [[, testexp] …] <Anweisungen> <Anweisungen> CASE test-n [[, test-n] …] <Anweisungen> <Anweisungen> CASE ELSE <Anweisungen> <Anweisungen> END SELECT	Führt eine von mehreren Anweisungsgruppen aus, je nachdem, was der Ausdruck ergibt. „value" ist der Ausdruck, der geprüft wird. Das kann eine Zahl, eine Zeichenfolgenvariable oder ein komplexer Ausdruck sein. „testexp" (oder test-n) ist der Wert, mit dem verglichen werden soll. Das kann sein: • Ein einzelner Ausdruck (z. B. 34, „Zeichenfolge" oder PIN(4)*5), dem er entsprechen kann • Ein Wertebereich in Form von zwei einzelnen Ausdrücken, die durch das Schlüsselwort „TO" getrennt sind (z. B. 5 TO 9 oder „aa" TO „cc") • Ein Vergleich, der mit dem Schlüsselwort „IS" beginnt (optional). Zum Beispiel: IS > 5, IS <= 10. 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" übereinstimmt, wird es automatisch mit CASE ELSE abgeglichen. Wenn CASE ELSE nicht vorhanden ist, führt das Programm keine <statements> aus und fährt mit dem Code nach END SELECT fort. Wenn eine Übereinstimmung gefunden wird, werden die <statements> nach der CASE-Anweisung bis END SELECT oder bis zu einem weiteren CASE ausgeführt, woraufhin das Programm mit dem Code nach END SELECT fortfährt. Es kann eine unbegrenzte Anzahl von CASE-Anweisungen verwendet werden, aber es darf nur ein CASE ELSE vorhanden sein, und dieses sollte das letzte

	vor dem END SELECT sein.
	Beispiel:
	`SELECT CASE nbr%`
	`  CASE 4, 9, 22, 33 TO 88`
	`  statements`
	`  CASE IS < 4, IS > 88, 5 TO 8`
	`  Anweisungen`
	`  CASE SONST`
	`  Anweisungen`
	`END SELECT`
	Jedes SELECT CASE muss genau eine passende END SELECT-Anweisung haben. Es können beliebig viele SELECT…CASE-Anweisungen in den CASE-Anweisungen anderer SELECT…CASE-Anweisungen verschachtelt werden.

SERVO-Kanal [PositionA] [,PositionB]	Steuert einen Standard-Servo. „positionA" und „positionB" können zwischen -20 und 120 liegen und erzeugen ein 50-Hz-Signal zwischen 800 µs und 2,2 ms. Wie beim PWM-Befehl müssen die Pins mit SETPIN n,PWM eingerichtet werden. Um nur Kanal B zu verwenden, benutze die Syntax: SERVO Kanal,,Position. Beachte die beiden Kommas, die anzeigen, dass Kanal A nicht eingestellt wird. Schau dir die Pinbelegung an, um den Kanal und den Unterkanal (A oder B) für jeden Pin anzugeben. Beispiele: 90°-Servo: 0 = 0° und 90 = 90° 180°-Servo: 0 = 0° und 90 = 180 Hinweis: Bei Werten < 0 oder > 90 kann der Strom ansteigen, wenn das Servo seine Endposition erreicht. 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. Schau dir das Kapitel *„Verwendung der I/O-Pins"* an, um eine allgemeine Beschreibung der Ein-/Ausgabefunktionen des Pico zu bekommen. „pin" ist der zu konfigurierende I/O-Pin, „cfg" ist der Modus, in den der Pin versetzt werden soll, und „option" ist ein optionaler Parameter. „cfg" ist ein Schlüsselwort und kann einer der folgenden Werte sein: OFF — Nicht konfiguriert oder inaktiv AIN — Analogeingang (d. h. Messung der Spannung am Eingang). ARAW — Schneller Analogeingang, der einen Wert zwischen 0 und 4095 zurückgibt. DIN — Digitaler Eingang Wenn „option" weggelassen wird, ist der Eingang hochohmig. Wenn „Option" das Schlüsselwort „PULLUP" oder „PULLDOWN" ist, wird ein konstanter Strom von etwa 50 µA verwendet, um den Eingangs-Pin auf 3,3 V hoch- oder herunterzuziehen. Wegen eines Fehlers in den RP2350-Chips wird empfohlen, einen Pulldown mit einem Widerstand von 8,2 K oder weniger zu machen. FIN — Frequenzeingang Mit „option" kann die Gate-Zeit (die Zeitdauer, die zum Zählen der Eingangszyklen verwendet wird) festgelegt werden. Es kann sich um eine beliebige Zahl zwischen 10 ms und 100000 ms handeln. Die Funktion PIN() gibt immer die korrekt skalierte Frequenz in Hz zurück, unabhängig von der verwendeten Gate-

	Zeit. Wenn „option" weggelassen wird, beträgt die Gate-Zeit 1 Sekunde. Die Pins können GP6, GP7, GP8 oder GP9 sein (kann mit OPTION COUNT geändert werden). PIN Periodeneingabe Mit „option" kann die Anzahl der Eingangszyklen angegeben werden, 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 in ms zurück, egal wie viele Zyklen für den Durchschnitt verwendet wurden. Wenn „option" weggelassen wird, wird die Periode von nur einem Zyklus verwendet. Die Pins können GP6, GP7, GP8 oder GP9 sein (kann mit OPTION COUNT geändert werden). CIN Zähleingang Mit „option" kann festgelegt werden, welche Flanke den Zählvorgang auslöst und ob Pullup oder Pulldown aktiviert ist. 2 gibt eine fallende Flanke mit Pullup an, 3 sagt, dass sowohl eine fallende als auch eine steigende Flanke einen Zählung ohne Pullup auslöst, 5 legt beide Flanken fest, aber mit Pullup. Wenn „Option" weggelassen wird, löst eine steigende Flanke den Zählvorgang aus. Wegen einem Fehler in den RP2350-Chips wird Pull-Down nicht empfohlen. Die Pins können GP6, GP7, GP8 oder GP9 sein (kann mit OPTION COUNT geändert werden). DOUT Digitaler Ausgang „option" wird in diesem Modus nicht benutzt. Die Funktionen PIN() und PORT() können auch verwendet werden, um den Wert auf einem oder mehreren Ausgangspins zurückzugeben. Siehe die Funktion PIN() zum Lesen von Eingängen und die Anweisung PIN()= zum Einstellen eines Ausgangs. Siehe den folgenden Befehl, wenn ein Interrupt konfiguriert ist.
SETPIN pin, cfg, target [, option]	Konfiguriert „pin" so, dass ein Interrupt gemäß „cfg" generiert wird. Jeder I/O-Pin, der für digitale Eingaben geeignet ist, kann so konfiguriert werden, dass er einen Interrupt generiert, wobei maximal zehn Interrupts gleichzeitig konfiguriert werden können. „cfg" ist ein Schlüsselwort und kann eines der folgenden sein: OFF Nicht konfiguriert oder inaktiv INTH Interrupt bei Eingang von niedrig auf hoch INTL Interrupt bei Eingang von hoch nach niedrig INTB Unterbrechung bei beiden (d. h. bei jeder Änderung des Eingangs) „target" ist eine vom Benutzer definierte Subroutine, die aufgerufen wird, wenn das Ereignis eintritt. Die Rückkehr aus dem Interrupt erfolgt über die Befehle END SUB oder EXIT SUB. „option" ist dieselbe wie in SETPIN pin DIN (oben) verwendet. In diesem Modus wird der Pin auch als digitaler Eingang konfiguriert, sodass der Wert des Pins immer mit der Funktion PIN() abgerufen werden kann. Eine allgemeine Beschreibung findest du im Kapitel *„ Verwendung der E/A-Pins*".

SETPIN GP25, DOUT | HEARTBEAT	NICHT IN DER WEBMITE-VERSION Diese Version von SETPIN steuert die integrierte LED. Wenn sie als DOUT konfiguriert ist, kann sie per Programm gesteuert ein- und ausgeschaltet werden. Wenn es als HEARTBEAT eingestellt ist, blinkt es bei eingeschaltetem Gerät immer mal wieder 1 Sekunde lang an und 1 Sekunde lang aus. Das ist der Standardzustand und wird wieder so, wenn das Benutzerprogramm nicht mehr läuft.
SETPIN p1[, p2 [, p3]], Gerät	Diese Befehle werden für die Pin-Zuweisung für spezielle Geräte verwendet. Die Pins müssen aus dem Pin-Bezeichnungsdiagramm 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 die Pins anhand ihrer Pin-Nummer (z. B. 1, 2) oder GP-Nummer (z. B. GP0, GP1) referenziert werden können. Beachte, dass bei der WebMite-Version: • SPI1 und SPI2 bei GP20 bis GP28 nicht verfügbar sind • COM1 und COM2 sind auf P20 bis GP28 nicht verfügbar • I2C ist auf Pin 34 (GP28) nicht verfügbar • Folgende Pins sind nicht verfügbar: GP29, GP25, GP24 und GP23
SETPIN rx, tx, COM1	Weise die Pins zu, die für die serielle Schnittstelle COM1 verwendet werden sollen. Gültige Pins sind RX: GP1, GP13 oder GP17 TX: GP0, GP12, GP16 oder GP28
SETPIN rx, tx, COM2	Weis die Pins zu, die für die serielle Schnittstelle COM2 verwendet werden sollen. Gültige Pins sind RX: GP5, GP9 oder GP21 TX: GP4, GP8 oder GP20
SETPIN rx, tx, clk, SPI	Weis die Pins zu, die für den SPI-Port SPI benutzt werden sollen. Gültige Pins sind RX: GP0, GP4, GP16 oder GP20 TX: GP3, GP7 oder GP19 CLK: GP2, GP6 oder GP18
SETPIN rx, tx, clk, SPI2	Weis die Pins zu, die für den SPI-Port SPI2 benutzt werden sollen. Gültige Pins sind RX: GP8, GP12 oder GP28 TX: GP11, GP15 oder GP27 CLK: GP10, GP14 oder GP26
SETPIN sda, scl, I2C	Verteil die Pins, die für den I^2C-Port I2C benutzt werden sollen. Gültige Pins sind SDA: GP0, GP4, GP8, GP12, GP16, GP20 oder GP28 SCL: GP1, GP5, GP9, GP13, GP17 oder GP21
SETPIN sda, scl, I2C2	Verteil die Pins, die für den I^2C-Port I2C2 benutzt werden sollen. Gültige Pins sind SDA: GP2, GP6, GP10, GP14, GP18, GP22 oder GP26 SCL: GP3, GP7, GP11, GP15, GP19 oder GP27

SETPIN-Pin, PWM[nx]	Pin zu PWMnx zuweisen „n" ist die PWM-Nummer (0 bis 7) und „x" ist der Kanal (A oder B). n und x sind optional. Der Setpin kann geändert werden, bis der PWM-Befehl gegeben wird. Dann wird der Pin für PWM gesperrt, bis PWMn,OFF gegeben wird.
SETPIN Pin, IR	Verteil die Pins für die Infrarot-Kommunikation (IR) (kann jeder Pin sein).
SETPIN-Pin, PIOn	Pin für die Verwendung durch PIO0, PIO1 oder PIO2 reservieren (nur RP2350) (Details zu PIO findest du in *Anhang F*).
SETPIN GP1, FFIN [,gate]	<u>NUR RP2350</u> Stellt GP1 als schnellen Frequenzeingang ein. Es können Eingänge bis zur CPU-Geschwindigkeit /2 aufgezeichnet werden. Mit „gate" kann die Gate-Zeit (die Zeitdauer zum Zählen der Eingangszyklen) festgelegt werden. Diese kann zwischen 10 ms und 100000 ms liegen. Die Funktion PIN() gibt immer die korrekt skalierte Frequenz in Hz zurück, unabhängig von der verwendeten Gate-Zeit. Wenn „option" weggelassen wird, beträgt die Gate-Zeit 1 Sekunde. Die Funktion nutzt den PWM-Kanal 0 zum Zählen, sodass sie mit keiner anderen Nutzung dieses PWM-Kanals kompatibel ist.
SETTICK period, target [, nbr]	Damit wird ein periodischer Interrupt (oder „Tick") eingerichtet. Es stehen vier Tick-Timer zur Verfügung („nbr" ist 1, 2, 3 oder 4). „nbr" ist optional, und wenn es nicht angegeben wird, wird Timer Nummer 1 verwendet. Die Zeit zwischen den Interrupts beträgt „period" Millisekunden und „target" ist die Interrupt-Subroutine, die aufgerufen wird, wenn das zeitgesteuerte Ereignis eintritt. Die Periode kann zwischen 1 und 2147483647 ms (etwa 24 Tage) liegen. Diese Interrupts können deaktiviert werden, indem „period" auf Null gesetzt wird (d. h. SETTICK 0, 0, 3 deaktiviert den Tick-Timer Nummer 3).
SETTICK PAUSE, Ziel [, Anzahl] oder SETTICK RESUME, Ziel [, nbr]	Den angegebenen Timer pausieren oder fortsetzen. Wenn er pausiert ist, wird der Interrupt verzögert, aber der aktuelle Zählstand bleibt erhalten.
SORT array() [,indexarray()] [,flags] [,startposition] [,elementstosort]	Dieser Befehl nimmt ein Array beliebigen Typs (Ganzzahl, Gleitkomma oder Zeichenfolge) und sortiert es an Ort und Stelle in aufsteigender Reihenfolge. Er hat einen optionalen Parameter „indexarray%()". Wenn dieser verwendet wird, muss es sich um ein Integer-Array handeln, das genauso groß ist wie das zu sortierende Array. Nach dem Sortieren enthält dieses Array die ursprüngliche Indexposition jedes Elements im zu sortierenden Array vor dem Sortieren. Alle Daten im Array werden überschrieben. So können verbundene Arrays sortiert werden. Der Parameter „flag" ist optional und gültige Flag-Werte sind: bit0: 0 (Standard, wenn weggelassen) normale Sortierung – 1 umgekehrte Sortierung bit1: 0 (Standard) groß-/kleinschreibungsabhängig – 1 Sortierung ist groß-/kleinschreibungsunabhängig (nur Zeichenfolgen). bit2: 0 (Standard) Normale Sortierung – 1 Leere Zeichenfolgen werden an das Ende des Arrays verschoben Die optionale „startposition" legt fest, bei welchem Element im Array die

Sortierung beginnen soll. Der Standardwert ist 0 (OPTION BASE 0) oder 1 (OPTION BASE 1).

Der optionale Parameter „elementstosort" sagt, wie viele Elemente im Array sortiert werden sollen. Standardmäßig sind das alle Elemente nach der „startposition".

Jeder der optionalen Parameter kann weggelassen werden. Um beispielsweise nur die ersten 50 Elemente eines Arrays zu sortieren, könntest du Folgendes verwenden:

```
SORT array(), , , ,50
```

Beispiel:

Das Array city$() könnte die Namen von Städten weltweit enthalten und lässt sich mit dem folgenden Befehl ganz einfach in aufsteigender alphabetischer Reihenfolge sortieren: SORT city$()

Der Befehl SORT funktioniert mit Zeichenfolgen, Gleitkommazahlen und Ganzzahlen, allerdings muss das zu sortierende Array eindimensional sein.

Oft sind 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.

Das Sortieren und Zugreifen auf diese Daten ist etwas komplexer, kann aber relativ einfach mit einem optionalen Parameter für den Befehl sort wie folgt durchgeführt werden:

SORT array(), indexarray%()

indexarray%() muss ein eindimensionales Integer-Array sein, das genauso groß ist 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%() war, wird überschrieben).

Um auf die sortierten Daten zuzugreifen, kopierst du zuerst das Array, das den Hauptschlüssel enthält, in ein temporäres Array und sortierst dieses unter Angabe von indexarray%(). Nach dem Sortieren kann indexarray%() zum Indizieren der ursprünglichen Arrays verwendet werden.

Zum Beispiel:

```
DIM city$(100),pop%(100),area!(100),sindex%(100),t$(100)
FOR i = 0 to 100
  t$(i) = city$(i)        ' temporäre Kopie der Schlüssel
NEXT i
SORT t$(), sindex%()      ' temporäres Array sortieren,
FÜR i = 0 bis 100
  k = sindex%(i)          ' Index zum ursprünglichen
Array
  DRUCKE city$(k),pop%(k),area!(k)' in sortierter
Reihenfolge aus
NEXT i
```

SPI OPEN Geschwindigkeit, Modus, Bits oder SPI READ nbr, array() oder SPI WRITE nbr, data1, data2, data3, … usw. oder SPI WRITE nbr, string$ oder SPI WRITE nbr, array() oder SPI CLOSE	Kommunikation über einen SPI-Kanal. Schau dir *Anhang D* an, um mehr Details zu erfahren. „nbr" ist die Anzahl der zu sendenden oder empfangenden Datenelemente. „data1", „data2" usw. können Float- oder Integer-Werte sein und im Fall von WRITE eine Konstante oder ein Ausdruck. Wenn „string$" benutzt wird, werden „nbr" Zeichen gesendet. „array" muss ein eindimensionales Array vom Typ Float oder Integer sein, und „nbr" Elemente werden gesendet oder empfangen.
SPI2	Dieselben Befehle wie für SPI (oben), aber für den zweiten SPI-Kanal.
SPRITE	<u>NUR VGA- UND HDMI-VERSIONEN</u> Die SPRITE-Befehle werden zum Bearbeiten kleiner Grafiken auf dem VGA- oder HDMI-Bildschirm verwendet und sind beim Schreiben von Spielen nützlich. Sprites funktionieren nur in Framebuffern in den Modi 2 und 3. Sprites werden aus Effizienzgründen immer als RGB121-„Nibbles" gespeichert. Die maximale Größe eines Sprites ist MM.HRES-1 und MM.VRES-1. Sieh dir auch den BLIT-Befehl und die SPRITE()-Funktionen an.
SPRITE CLOSE [#]n	Schließt das Sprite „n" und gibt seine Speicherressourcen frei, sodass die Sprite-Nummer wiederverwendet werden kann. Der Befehl gibt eine Fehlermeldung aus, wenn andere Sprites aus diesem kopiert werden, es sei denn, sie wurden zuvor geschlossen.
SPRITE CLOSE ALL	Schließt alle Sprites und gibt den gesamten Sprite-Speicher frei. Der Bildschirm bleibt unverändert.
SPRITE COPY [#]n, [#]m, nbr	Erstellt eine Kopie von Sprite „n" in „nbr" neuer Sprites, beginnend mit der Nummer „m". Kopierte Sprites verwenden dasselbe geladene Bild wie das Original, um Speicherplatz zu sparen.
SPRITE HIDE [#]n	Entfernt Sprite n von der Anzeige und ersetzt den gespeicherten Hintergrund. Um einen Bildschirm in einen früheren Zustand zurückzusetzen, sollten Sprites in umgekehrter Reihenfolge zu ihrer Schreibweise „LIFO" ausgeblendet werden.
SPRITE HIDE ALL	Blendet alle Sprites aus, sodass der Hintergrund bearbeitet werden kann. Die folgenden Befehle können nicht verwendet werden, wenn alle Sprites ausgeblendet sind: SPRITE SHOW (SAFE) SPRITE HIDE (SAFE, ALL) SPRITE SWAP SPRITE MOVE SPRITE SCROLLR SPRITE SCROLL
SPRITE WIEDERHERSTELLEN	Bringt die Sprites zurück, die vorher mit SPRITE HIDE ALL versteckt wurden.

SPRITE HIDE SAFE [#]n	Entfernt Sprite n von der Anzeige und ersetzt den gespeicherten Hintergrund. Blendet automatisch alle neueren Sprites sowie das angeforderte Sprite aus und ersetzt sie anschließend. Dadurch wird sichergestellt, dass Sprites, die von anderen Sprites überdeckt werden, entfernt werden können, ohne dass der Benutzer die Schreibreihenfolge verfolgen muss. Natürlich ist diese Version weniger leistungsfähig als die einfache Version und sollte nur verwendet werden, wenn die Gefahr besteht, dass das Sprite teilweise überdeckt wird.
SPRITE INTERRUPT sub	Gibt den Namen der Subroutine an, die aufgerufen wird, wenn eine Sprite-Kollision auftritt. In Anhang G wird beschrieben, wie du mit der Funktion SPRITE Details zu den kollidierten Objekten abfragen kannst.
SPRITE READ [#]b, x, y, w, h	Damit wird ein Teil der Anzeige in den Speicherpuffer '#b' kopiert. Die Quellkoordinaten sind 'x' und 'y', die Breite des zu kopierenden Anzeigebereichs ist 'w' und die Höhe ist 'h'. Wenn dieser Befehl verwendet wird, wird der Speicherpuffer automatisch erstellt und ausreichend Speicher zugewiesen. Dieser Puffer kann mit dem Befehl SPRITE CLOSE freigegeben und der Speicher zurückgewonnen werden.
SPRITE WRITE [#]b, x, y [,mode]	Kopiert das Sprite „#b“ auf die Anzeige. Die Zielkoordinaten sind „x“ und „y“. Der optionale Parameter „mode“ ist standardmäßig auf 4 gesetzt und legt fest, wie die gespeicherten Bilddaten beim Schreiben geändert werden. Es handelt sich um die bitweise UND-Verknüpfung der folgenden Werte: &B001 = von links nach rechts gespiegelt &B010 = von oben nach unten gespiegelt &B100 = transparente Pixel nicht kopieren
SPRITE LOAD fname$ [,start_sprite_number] [,mode]	Lädt die Datei „fname$”, die als originale Colour Maximite-Sprite-Datei formatiert sein muss. Das Dateiformat findest du im *Original-Handbuch* zur *Sprache* Colour Maximite *MMBasic*. Du kannst mehrere Sprite-Dateien laden, indem du für jede Datei eine andere „start_sprite_number” angibst. Du musst dafür sorgen, dass sich die Sprites nicht überlappen. Der Modus ist standardmäßig auf Null eingestellt, in diesem Fall werden die CMM1/CMM2-Farbcodes verwendet (Schwarz, Blau, Grün, Cyan, Rot, Magenta, Gelb, Weiß, Myrte, Kobalt, Mittelgrün, Cerulean, Rost, Fuchsia, Braun, Flieder). Wenn der Modus auf 1 gesetzt ist, werden die RGB121-Farbcodes verwendet: (Schwarz, Blau, Myrte, Kobalt, Mittelgrün, Cerulean, Grün, Cyan, Rot, Magenta, Rost, Fuchsia, Braun, Flieder, Gelb, Weiß).
SPRITE LOADARRAY [#]n, w, h, array%()	Erstellt das Sprite „n” mit der Breite „w” und der Höhe „h”, indem w*h RGB888-Werte aus „array%()” gelesen werden. Die RGB888-Werte müssen in der Reihenfolge der Spalten und dann der Zeilen beginnend oben links gespeichert werden. Dadurch kann der Programmierer einfache Sprites in einem Programm erstellen, ohne sie von der Festplatte laden oder vom Display lesen zu müssen. Die Firmware gibt eine Fehlermeldung aus, wenn „array%()” nicht groß genug ist, um die erforderliche Anzahl von Werten aufzunehmen.
SPRITE LOADBMP [#]b, fname$ [,x] [,y] [,w] [,h]	Lädt einen Blit-Puffer aus einer 24-Bit-BMP-Bilddatei. „x” und „y” sagen dir, wo im Bild das Laden anfangen soll, und „w” und „h” geben die Breite und Höhe des Bereichs an, der geladen werden soll. Beispiel: `SPRITE LOAD #1,"image1", 50,50,100,100` lädt einen Bereich von 100 Pixeln im Quadrat mit der oberen linken Ecke bei 50, 50 aus dem Bild image1.bmp.

SPRITE LOADPNG [#]b, fname$ [,transparent] [,alphacut]	Lädt SPRITE Nummer „b” aus der PNG-Datei „fname$”. Wenn keine Erweiterung angegeben ist, wird automatisch .png an den Dateinamen angehängt. Die Datei muss im RGBA8888-Format vorliegen, was die normale Standardeinstellung ist. 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. Die variable Transparenz kann dann mit dem Befehl SPRITE SET TRANSPARENT n oder FRAMEBUFFER LAYER n verwendet werden, um das Sprite mit dem transparenten Bereich ausgeblendet anzuzeigen.
SPRITE MOVE	Führt eine einzelne atomare Transaktion aus, die alle Sprites neu positioniert, für die zuvor mit dem Befehl SPRITE NEXT eine Positionsänderung festgelegt wurde. Kollisionen werden erkannt, sobald alle Sprites verschoben sind, und wie bei einem Scroll gemeldet.
SPRITE NEXT [#]n, x, y	Setzt die X- und Y-Koordinaten des Sprites, die beim nächsten Bildlauf oder bei Ausführung des Befehls SPRITE MOVE verwendet werden sollen. Durch die Verwendung von SPRITE NEXT anstelle von SPRITE SHOW können mehrere Sprites als Teil derselben atomaren Transaktion verschoben werden.
SPRITE SCROLL x, y [,col]	Verschiebt den Hintergrund und alle Sprites auf dem aktiven Framebuffer (L oder N) um „x” Pixel nach rechts und „y” Pixel nach oben. „x” kann eine beliebige Zahl zwischen -MM.HRES-1 und MM.HRES-1 sein, „y” kann eine beliebige Zahl zwischen -MM.VRES-1 und MM.VRES-1 sein. Sprites auf einer anderen Ebene als Null bleiben an ihrer Position auf dem Bildschirm. Standardmäßig wird das Bild beim Scrollen umgebrochen. Wenn 'col' angegeben ist, ersetzt die Farbe den Bereich hinter dem gescrollten Bild. Wenn 'col' auf -1 gesetzt ist, bleibt der gescrollte Bereich unverändert.
SPRITE SET TRANSPARENT n	Legt den Farbcode (0-15) fest, der als transparent verwendet wird, wenn Sprites über einem Hintergrund angezeigt werden (Standardwert ist 0).
SPRITE SHOW [#]n, x,y, layer [,options]	Zeigt das Sprite „n” auf dem Bildschirm an, wobei die obere linke Ecke bei den Koordinaten „x”, „y” liegt. Sprites kollidieren nur mit anderen Sprites auf derselben Ebene, Ebene Null oder mit dem Bildschirmrand. Wenn ein Sprite schon auf dem Bildschirm zu sehen ist, sorgt der SPRITE SHOW-Befehl das Sprite an die neue Position. Der Anzeigehintergrund wird als Teil des Befehls gespeichert und ersetzt, wenn das Sprite ausgeblendet oder weiter verschoben wird. Der Parameter „options“ ist optional und kann wie folgt eingestellt werden: Bit 0 gesetzt – von links nach rechts gespiegelt Bit 1 gesetzt – von oben nach unten gespiegelt Bit 2 gesetzt – schwarze Pixel werden nicht als transparent behandelt, Standardwert ist 0
SPRITE SHOW SAFE [#]n, x,y, layer [,orientation] [,ontop]	Zeigt ein Sprite an und gleicht automatisch alle anderen Sprites aus, die es überlappen. Wenn das Sprite noch nicht angezeigt wird, funktioniert der Befehl genauso wie SPRITE SHOW. Wenn das Sprite schon angezeigt wird, wird es verschoben und bleibt in seiner Position relativ zu anderen Sprites, basierend auf der ursprünglichen Reihenfolge, in der sie geschrieben wurden. Wenn also Sprite 1 vor Sprite 2 geschrieben wurde und es so verschoben wird, dass es Sprite 2 überlappt, wird es unter Sprite 2 angezeigt. Wenn der optionale Parameter „ontop” auf 1 gesetzt ist, wird das verschobene Sprite zum neuesten Sprite und liegt über allen anderen Sprites, die es überlappt. Details zum Orientierungsparameter findest du unter SPRITE SHOW.

SPRITE SWAP [#]n1, [#]n2 [,orientation]	Ersetzt das Sprite „n1" durch das Sprite „n2". Die Sprites müssen die gleiche Breite und Höhe haben, und „n1" muss angezeigt werden, sonst kommt es zu einem Fehler. Mehr Infos zum Orientierungsparameter findest du unter SPRITE SHOW. Das Ersatz-Sprite übernimmt den Hintergrund vom Original sowie dessen Position in der Reihenfolge der Zeichnung.
STAR	Berechnet die Position eines Himmelsobjekts anhand der aktuellen Uhrzeit und des Standorts von einem angeschlossenen GPS-Modul. Dies ist einer von mehreren Befehlen, die hochpräzise astronomische Berechnungen durchführen, die für die Ausrichtung von Teleskopen oder die Navigation geeignet sind. Eine detaillierte Beschreibung findest du in der Datei *GPS_Astro_Reference.pdf*, die im ZIP-Archiv mit der Firmware enthalten ist.
STATIC variable [, variables] Die vollständige Syntax findest du unter DIM.	Definiert eine Liste von Variablennamen, die lokal für die Subroutine oder Funktion sind. Diese Variablen behalten ihren Wert zwischen den Aufrufen der Subroutine oder Funktion (im Gegensatz zu Variablen, die mit dem Befehl LOCAL erstellt wurden). Dieser Befehl hat genau die gleiche Syntax wie DIM. Der einzige Unterschied besteht darin, dass die Länge des mit STATIC erstellten Variablennamens und die Länge des Unterprogramm- oder Funktionsnamens zusammen nicht mehr als 31 Zeichen betragen dürfen. Statische Variablen können mit einem Wert initialisiert werden. Diese Initialisierung wirkt sich nur beim ersten Aufruf der Unterroutine aus (nicht bei späteren Aufrufen).
STEPPER	NUR RP2350-VERSIONEN Dieser Befehl bietet ein komplettes System zur Steuerung von bis zu 3 Schrittmotorachsen (X, Y, Z) mit Unterstützung für die Ausführung von G-Code, Beschleunigungsplanung und Hardware-Endschaltern. Das wird in der Datei *„Stepper_Reference.pdf*" im Download-Zip-Paket der Firmware genau erklärt.
STRUCT	Mit diesem Befehl kannst du Strukturen (auch als benutzerdefinierte Typen bezeichnet) bearbeiten, mit denen verwandte Variablen unterschiedlicher Typen unter einem einzigen Namen zusammengefasst werden können. Das wird ausführlich in der Datei *MMBasic_Structures_Manual.pdf* beschrieben, die im ZIP-Archiv zum Firmware-Download enthalten ist.
SUB xxx (arg1 [,arg2, …]) <Anweisungen> <Anweisungen> END SUB	Definiert eine aufrufbare Subroutine. Das ist dasselbe, als würde man MMBasic während der Ausführung des Programms einen neuen Befehl hinzufügen. „xxx" ist der Name der Unterroutine und muss den Regeln für die Benennung von Variablen entsprechen. 'arg1', 'arg2' usw. sind die Argumente oder Parameter für die Unterroutine. Ein Array wird durch leere Klammern angegeben, z. B. arg3(). Der Typ des Arguments kann durch einen Typ-Suffix (z. B. arg1$) oder durch Angabe des Typs mit `AS <Typ> (z. B. arg1 AS STRING)` angegeben werden. Argumente in der Liste des Aufrufers, die eine Variable sind und den richtigen Typ haben, werden per Referenz an die Subroutine übergeben. Das heißt, dass alle Änderungen am entsprechenden Argument in der Subroutine auch in die Variable des Aufrufers kopiert werden und daher nach Beendigung der Subroutine darauf zugegriffen werden kann. 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 generiert, wenn dies nicht möglich ist. Arrays werden durch Angabe des Array-Namens mit leeren Klammern (z. B.

	arg()) übergeben, immer per Referenz und müssen vom richtigen Typ sein. Jede Definition muss eine END SUB-Anweisung haben. Wenn diese erreicht ist, kehrt das Programm zur nächsten Anweisung nach dem Aufruf der Unterroutine zurück. Der Befehl EXIT SUB kann für einen vorzeitigen Abbruch verwendet werden. Du benutzt die Subroutine, indem du ihren Namen und ihre Argumente in einem Programm wie einen normalen Befehl verwendest. Zum Beispiel: `MySub a1, a2` Wenn die Unterprogramm aufgerufen wird, wird jedes Argument im Aufrufer mit dem Argument in der Unterprogrammdefinition abgeglichen. Diese Argumente sind nur innerhalb der Unterprogramm verfügbar. Unterprogramme können mit einer variablen Anzahl von Argumenten aufgerufen werden. Alle in der Liste der Unterprogramm ausgelassenen Argumente werden auf Null oder eine Null-Zeichenkette gesetzt. Klammern um die Argumentliste sowohl im Aufrufer als auch in der Definition sind optional.

SYNC time% [,period]
oder
SYNC
	Mit dem Befehl SYNC kann der Benutzer sehr präzise zeitgesteuerte Wiederholungsaktionen (Genauigkeit 1–2 Mikrosekunden) implementieren.
	Dazu wird der Befehl erst mal mit dem Parameter time% aufgerufen. Damit wird eine sich wiederholende Uhr für time% Mikrosekunden eingerichtet. Der optionale Parameter „period" ändert die Zeit und kann „U" für Mikrosekunden, „M" für Millisekunden oder „S" für Sekunden sein.
	Sobald die Uhr eingerichtet ist, wird das Programm mit dem Befehl SYNC ohne Parameter daran synchronisiert. Dieser wartet, bis die Uhrperiode abgelaufen ist. Bei Perioden unter 2 ms ist dies nicht unterbrechbar. Bei Perioden über 2 ms reagiert das Programm auf Strg-C, aber nicht auf MMBasic-Interrupts.
	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. Das folgende Beispiel würde beispielsweise einen Servo mit der erforderlichen präzisen 50-Hz-Taktung ansteuern:
	`SYNC 20, M`
	`DO`
	`  SYNC`
	`  PULSE GP0,n`
	`LOOP`

TEMPR START pin [, Genauigkeit] [, Zeitlimit]	Mit diesem Befehl kannst du eine Umwandlung starten, die auf einem DS18B20-Temperatursensor läuft, der an „Pin" angeschlossen ist. Normalerweise reicht die Funktion TEMPR() allein aus, um eine Temperaturmessung durchzuführen, sodass die Verwendung dieses Befehls optional ist. Weitere Infos findest du im Abschnitt *„Temperatur messen*". Dieser Befehl startet die Messung am Temperatursensor. Das Programm kann dann während der Messung andere Aufgaben ausführen und später die Funktion TEMPR() verwenden, um den Messwert abzurufen. Wenn die Funktion TEMPR() vor Ablauf der Umwandlungszeit verwendet wird, wartet die Funktion die verbleibende Umwandlungszeit ab, bevor sie den Wert zurückgibt. Es können beliebig viele dieser Umwandlungen (an verschiedenen Pins) gestartet werden und gleichzeitig laufen. „precision" ist die Auflösung der Messung und ist optional. Es ist eine Zahl zwischen 0 und 3 mit folgender Bedeutung: 0 = 0,5 °C Auflösung, 100 ms Umwandlungszeit. 1 = 0,25 °C Auflösung, 200 ms Umwandlungszeit (dies ist die

	Standardeinstellung). 2 = 0,125 °C Auflösung, 400 ms Umwandlungszeit. 3 = 0,0625 °C Auflösung, 800 ms Umwandlungszeit. Der optionale Zeitüberschreitungsparameter setzt die oben genannten Umwandlungszeiten außer Kraft, um langsame Geräte zu berücksichtigen.
TEXT x, y, Zeichenfolge$ [,Ausrichtung$] [, Schriftart] [, Skalierung] [, c] [, bc]	Zeigt eine Zeichenfolge auf dem Videoausgang oder dem angeschlossenen LCD-Bildschirm an, beginnend bei „x" und „y". „string$" ist die anzuzeigende Zeichenfolge. Numerische Daten sollten in eine Zeichenfolge konvertiert und mit der Funktion Str$() formatiert werden. „alignment$" 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 (links), CENTER (zentriert) oder RIGHT (rechts) sein kann und der zweite Buchstabe die vertikale Ausrichtung um „y" angibt und T, M oder B für TOP (oben), MIDDLE (Mitte) oder BOTTOM (unten) sein kann. Die Standardausrichtung ist links/oben. Beispiel: „CM" zentriert den Text vertikal und horizontal. Die Zeichenfolge „alignment$" kann eine Konstante (z. B. „CM") oder eine Zeichenfolgenvariable sein. Aus Gründen der Abwärtskompatibilität mit früheren Versionen von MMBasic kann die Zeichenfolge auch ohne Anführungszeichen angegeben werden (z. B. CM). Ein dritter Buchstabe kann in der Ausrichtungszeichenfolge verwendet werden, um die Drehung des Textes anzugeben. Dies kann „N" für normale Ausrichtung, „V" für vertikalen Text, bei dem jedes Zeichen unter dem vorherigen von oben nach unten läuft, „I" für invertierten Text (d. h. auf dem Kopf stehend), „U" für eine Drehung des Textes um 90º gegen den Uhrzeigersinn und „D" für eine Drehung des Textes um 90º im Uhrzeigersinn sein. „font" und „scale" sind optional und werden standardmäßig durch die mit dem Befehl FONT festgelegten Werte ersetzt. „c" ist die Zeichenfarbe und „bc" ist die Hintergrundfarbe. Sie sind optional und standardmäßig auf die aktuellen Vordergrund- und Hintergrundfarben eingestellt. Eine Definition der Farben und Grafikkoordinaten findest du im Kapitel *Grafikbefehle und -funktionen.*
TILE x, y [,foreground] [,background] [,nbr_tiles_wide] [,nbr_tiles_high] TILE HEIGHT n	<u>NUR VGA- ODER HDMI-VERSIONEN MODUS 1</u> Legt die Farbe für eine oder mehrere Kacheln auf dem Bildschirm fest. Im Standard-Monochrom-Modus wird der Bildschirm in 80x40 Kacheln mit jeweils 8x12 Pixeln aufgeteilt. Dies entspricht der Schriftart 1 und ermöglicht eine vollständige Farbcodierung im Editor im Monochrom-Modus. Jeder Kachel kann eine andere Vordergrund- und Hintergrundfarbe zugewiesen bekommen, die aus den folgenden Farben ausgewählt werden kann: Weiß, Gelb, Lila, Braun, Fuchsia, Rostrot, Magenta, Rot, Cyan, Grün, Cerulean, Mittelgrün, Kobalt, Myrte, Blau und Schwarz. „x" und „y" sind die Koordinaten des Startblocks (0-79, 0-39). „foreground" und „background" sind die neuen ausgewählten Farben. „nbr_tiles_wide" und „nbr_tiles_high" sind die Anzahl der zu ändernden Kacheln. Die Änderung passiert sofort und hat keinen Einfluss auf den Text oder die Grafiken, die gerade in den Kacheln angezeigt werden (nur auf die Farben). Legt die Höhe der Kacheln fest. „n" kann zwischen 12 und 480 (RP2040) oder zwischen 8 und 480 (RP2350) liegen.

TIME$ = „HH:MM:SS” oder TIME$ = „HH:MM” oder 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 setzt TIME$ = "14:30" die Uhr auf 14:30 Uhr mit null Sekunden. Mit OPTION RTC AUTO ENABLE startet die PicoMite-Firmware mit der in RTC programmierten TIME$. Ohne OPTION RTC AUTO ENABLE startet die Firmware mit TIME$="00:00:00".
TIMER = msec	Setzt den Timer auf eine bestimmte Anzahl von Millisekunden zurück. Normalerweise wird das nur benutzt, um den Timer auf Null zurückzusetzen, aber du kannst ihn auf jede beliebige positive Zahl einstellen. Weitere Infos findest du unter der TIMER-Funktion.
TRACE ON oder TRACE OFF oder TRACE LIST nn	TRACE ON/OFF schaltet die Trace-Funktion ein bzw. aus. Diese Funktion zeigt während der Programmausführung die Nummer jeder Zeile (gezählt vom Anfang des Programms) in eckigen Klammern an. Das ist beim Debuggen von Programmen echt praktisch. TRACE LIST zeigt die letzten „nn” ausgeführten Zeilen im oben beschriebenen Format an. MMBasic protokolliert immer die ausgeführten Zeilen, sodass diese Funktion immer verfügbar ist (d. h. sie muss nicht eingeschaltet werden).
TRIANGLE X1, Y1, X2, Y2, X3, Y3 [, C [, FILL]]	Zeichnet ein Dreieck auf dem angeschlossenen Videoausgang oder LCD-Display mit den Ecken bei X1, Y1 und X2, Y2 und X3, Y3. „C” ist die Farbe des Dreiecks und standardmäßig die aktuelle Vordergrundfarbe. „FILL” ist die Füllfarbe und standardmäßig keine Füllung (sie kann auch auf -1 für keine Füllung gesetzt werden). Alle Parameter können als Arrays angegeben werden, und die Software zeichnet die Anzahl der Dreiecke, die durch die Abmessungen 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, sonst wird der Fehler „c” ausgegeben. „c” und „fill” können entweder Arrays oder einzelne Variablen/Konstanten sein.
TRIANGLE SAVE [#]n, x1,y1,x2,y2,x3,y3 TRIANGLE RESTORE [#]n	Speichert einen dreieckigen Bereich des Bildschirms im Puffer #n. Stellt einen gespeicherten dreieckigen Bereich des Bildschirms wieder her und löscht den gespeicherten Puffer.
TURTLE	Die Firmware hat eine komplette Turtle-Grafik-Engine. Sieh dazu Anhang H.
TYP struct-name Struktur-Element1, Struktur-Element2 … END TYPE	Definiere eine Struktur (auch als benutzerdefinierter Typ bezeichnet), mit der verwandte Variablen unterschiedlicher Typen unter einem einzigen Namen zusammengefasst werden können. Das wird ausführlich in der Datei *MMBasic_Structures_Manual.pdf* beschrieben, die im ZIP-Archiv zum Herunterladen der Firmware enthalten ist.
FIRMWARE AKTUALISIEREN	<u>NICHT BEI USB-VERSIONEN</u> Versetzt die PicoMite-Firmware in den Firmware-Update-Modus (entspricht dem Einschalten bei gedrückter BOOTSEL-Taste). Dieser Befehl ist nur an der Eingabeaufforderung verfügbar.
VAR SAVE var [, var]… oder VAR RESTORE	VAR SAVE speichert eine oder mehrere Variablen in einem nichtflüchtigen Flash-Speicher, wo sie später wiederhergestellt werden können (normalerweise nach einem Stromausfall). „var” kann eine beliebige Anzahl von numerischen oder

oder VAR CLEAR	Zeichenfolgenvariablen und/oder Arrays sein. Arrays werden durch leere Klammern angegeben. Beispiel: var() Der Befehl VAR SAVE kann mehrmals benutzt werden. Vorher gespeicherte Variablen werden mit ihrem neuen Wert aktualisiert, und alle neuen Variablen (die vorher nicht gespeichert wurden) werden zur späteren Wiederherstellung zur gespeicherten Liste hinzugefügt. VAR RESTORE ruft die zuvor gespeicherten Variablen ab und fügt sie (und ihre Werte) in die Variablentabelle ein. VAR CLEAR löscht alle gespeicherten Variablen. Dieser Befehl wird normalerweise zum Speichern von Kalibrierungsdaten, Optionen und anderen Daten verwendet, die sich nicht oft ändern, aber auch bei einem Stromausfall erhalten bleiben müssen. Normalerweise wird der Befehl VAR RESTORE am Anfang des Programms platziert, damit zuvor gespeicherte Variablen wiederhergestellt werden und dem Programm beim Start sofort zur Verfügung stehen. Hinweise: • Der für diesen Befehl verfügbare Speicherplatz beträgt 16 KB. • Die Verwendung von VAR RESTORE ohne vorherige Speicherung hat keine Auswirkung und erzeugt keinen Fehler. • Wenn bei der Verwendung von RESTORE bereits eine Variable mit demselben Namen vorhanden ist, wird deren Wert überschrieben. • Gespeicherte Arrays müssen (mit DIM) deklariert werden, bevor sie wiederhergestellt werden können. • Beachte, dass String-Arrays den diesem Befehl zugewiesenen Speicher schnell vollständig belegen können. Der Qualifizierer LENGTH kann bei der Deklaration eines String-Arrays verwendet werden, um die Größe des Arrays zu reduzieren (siehe Befehl DIM). Bei normalen String-Variablen ist dies nicht erforderlich. • Die gespeicherten Variablen werden automatisch durch ein Firmware-Upgrade, durch den Befehl NEW oder beim Laden eines neuen Programms über AUTOSAVE, XMODEM usw. gelöscht.
WATCHDOG-Zeitüberschreitung oder WATCHDOG OFF oder WATCHDOG HW-Timeout oder WATCHDOG HW AUS	Startet den Watchdog-Timer, der die Prozessoren automatisch neu startet, wenn die Zeit abgelaufen ist. Das kann genutzt werden, um nach einem Problem, das das laufende Programm gestoppt hat (wie eine Endlosschleife oder ein Programmier- oder anderer Fehler), wieder hochzufahren. Das kann bei unbeaufsichtigten Steuerungen wichtig sein. Die Zeitüberschreitung kann entweder im System-Timer-Interrupt (WATCHDOG-Befehl) oder als echter CPU-/Hardware-Watchdog (WATCHDOG HW-Befehl) verarbeitet werden. Wenn der Hardware-Watchdog verwendet wird, hat der Timer eine maximale Dauer von 8,3 Sekunden. Für den Software-Watchdog gibt's keine solche Begrenzung. „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 Null herunterzählt. Wenn der Timer auf Null läuft (vielleicht weil das BASIC-Programm nicht mehr läuft), wird die PicoMite-Firmware automatisch neu gestartet und die automatische Variable MM.WATCHDOG wird auf „true” (also 1) gesetzt, was bedeutet, dass ein Fehler aufgetreten ist. Bei einem normalen Start wird MM.WATCHDOG auf „false” (also 0) gesetzt. Beachte, dass OPTION AUTORUN angegeben werden muss, damit das Programm neu gestartet wird. Mit WATCHDOG OFF kann der Watchdog-Timer deaktiviert werden (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] WII [CLASSIC] CLOSE	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, was eine Voraussetzung ist. Open versucht, mit dem Wii Classic zu kommunizieren, und gibt einen Fehler zurück, wenn er nicht gefunden wird. Wenn er gefunden wird, tastet die Firmware die Wii-Daten im Hintergrund mit einer Rate von 50 Hz ab. Wenn eine optionale Benutzerunterbrechung angegeben ist, wird diese ausgelöst, wenn sich eine der Tasten ändert (sowohl ein- als auch ausgeschaltet). Informationen zum Auslesen von Daten aus dem Wii Classic findest du unter der Funktion DEVICE. CLOSE stoppt die Hintergrundabfrage und deaktiviert alle angegebenen Interrupts.
WII NUNCHUCK OPEN [,interrupt] WII NUNCHUCK CLOSE	Öffnet einen WII Nunchuck-Controller und führt eine Hintergrundabfrage des Geräts durch. Der Wii Nunchuck muss an die durch OPTION SYSTEM I2C angegebenen Pins angeschlossen sein, was eine Voraussetzung ist. Versucht, mit dem Wii Nunchuck zu reden, und gibt 'ne Fehlermeldung raus, wenn er nicht gefunden wird. Wenn er gefunden wird, checkt die Firmware die Wii-Daten im Hintergrund mit 'ner Rate von 50 Hz. Wenn 'ne optionale Benutzerunterbrechung angegeben ist, wird die ausgelöst, wenn sich einer der Knöpfe ändert (sowohl ein- als auch ausgeschaltet). Wie man Daten vom Wii Nunchuck liest, erfährst du in der DEVICE-Funktion. CLOSE stoppt die Hintergrundabfrage und deaktiviert alle angegebenen Unterbrechungen.
WEB WEB CONNECT [ssid$, passwd$, [name$] [,ipaddress$, mask$, gateway$]] WEB MQTT CONNECT addr$, port, user$, passwd$ [, interrupt] WEB MQTT PUBLISH topic$, msg$, [,qos] [,retain]	<u>NUR WEBMITE</u> Die WEB-Befehle werden verwendet, um die Internetfunktionen des WebMite zu verwalten. Dieser Befehl ohne optionale Parameter stellt, wenn möglich, eine Verbindung zum Standardnetzwerk her (wie zuvor mit OPTION WIFI festgelegt) oder stellt mit den optionalen Parametern eine Verbindung zum angegebenen Netzwerk her und richtet OPTION WIFI für die zukünftige Verwendung ein. Stellt eine Verbindung zu einem MQTT-Broker her. „addr$" ist die IP-Adresse, „port" ist die zu verwendende Portnummer, „user$" ist der Benutzername, „passwd$" ist das Passwort des Kontos und „interrupt" ist optional und, falls angegeben, die Subroutine, die beim Empfang einer Nachricht aufgerufen wird. WEB CONNECT trennt die Verbindung zu einem zuvor verbundenen Netzwerk nicht, sollte also nur verwendet werden, wenn vorher nichts eingerichtet wurde oder wenn ein zuvor konfiguriertes Netzwerk nicht aktiv ist oder wenn ein zuvor konfiguriertes Netzwerk beim Booten keine Verbindung herstellen konnte (keine Parameter). Veröffentlicht Inhalte in einem MQTT-Broker-Thema. „topic$" ist der Name des Themas und „msg$" ist die Nachricht/ „qos" ist die optionale Dienstgüte mit den Werten 0, 1 oder 2 (Standardwert ist 1).

WEB MQTT SUBSCRIBE topic$ [,qos]	Abonniere ein MQTT-Broker-Thema. „topic$“ ist der Name des Themas und „qos“ ist die optionale Dienstqualität mit den Werten 0, 1 oder 2 (Standard ist 1).
WEB MQTT UNSUBSCRIBE topic$	Abonnement eines MQTT-Broker-Themas kündigen. „topic$” ist der Name des Themas.
WEB MQTT CLOSE	Schließt eine dauerhafte MQTT-Verbindung.
WEB NTP [Zeitversatz [, NTPserver$]] [,Zeitlimit]]]	Hol dir das Datum/die Uhrzeit von einem NTP-Server und stell die interne WebMite-Uhr ein. „timeoffset” ist die lokale Zeitzone. Wenn das weggelassen wird, wird das Datum/die Uhrzeit auf GMT eingestellt. „NTPserver$” ist der zu verwendende Zeitserver. Wenn das weggelassen wird, wird standardmäßig ein internationaler Zeitserverpool verwendet. „timeout” ist die optionale Zeitüberschreitung in Millisekunden und ist standardmäßig auf 5000 eingestellt.
WEB OPEN TCP CLIENT address$, port	Öffnet eine TCP-Client-Verbindung zu einem WEB-Server. „address$“ ist eine Zeichenfolge und die Adresse des Servers, mit dem eine Verbindung hergestellt werden soll. Es kann sich entweder um eine URL (z. B. „api.openweathermap.org“) oder eine IP-Adresse (z. B. „192.168.1.111“) handeln. „port“ ist die Nummer des zu verwendenden Ports. Wird zusammen mit WEB TCP CLIENT REQUEST verwendet, um den Server abzufragen. Beachte, dass nur eine CLIENT-Verbindung erlaubt ist.
WEB OPEN TCP STREAM address$, port	Öffnet eine TCP-Client-Verbindung zu einem WEB-Server wie WEB OPEN TCP CLIENT, verbindet aber die WEB TCP CLIENT STREAM-Empfängerlogik statt der Logik für WEB TCP CLIENT REQUEST. „address$“ ist eine Zeichenfolge und die Adresse des Servers, mit dem eine Verbindung hergestellt werden soll. Es kann sich entweder um eine URL (z. B. „api.openweathermap.org“) oder eine IP-Adresse (z. B. „192.168.1.111“) handeln. „port“ ist die Nummer des zu verwendenden Ports. Beachte, dass eine CLIENT-Verbindung erlaubt ist.
WEB SCAN [array%()]	Sucht nach allen verfügbaren WLAN-Verbindungen. Wenn „array%()“ angegeben ist, wird die Ausgabe in einer Longstring gespeichert, andernfalls wird sie an die Konsole ausgegeben. Der Befehl kann unabhängig davon verwendet werden, ob bereits eine Netzwerkverbindung aktiv ist oder nicht.
WEB TCP CLIENT REQUEST request$, buff%() [,timeout]	Sendet eine Anfrage an den mit WEB OPEN TCP CLIENT geöffneten Remote-Server und wartet auf eine Antwort. „request$“ ist eine Zeichenfolge und die Anfrage, die an den Server gesendet werden soll. „buff%()“ ist ein Integer-Array, das die Antwort als LONGSTRING empfängt. Die Größe dieses Puffers begrenzt die vom Server empfangene Datenmenge. 'timeout' ist die optionale Zeitüberschreitung in Millisekunden und standardmäßig auf 5000 gesetzt. Wenn die Anfrage zeitlich begrenzt ist, kommt es zu einem Fehler, andernfalls werden die empfangenen Daten im LONGSTRING 'buff%()' gespeichert. Wenn es sich bei den empfangenen Daten um eine JSON-Zeichenkette handelt, kann die Funktion JSON$() zum Parsen verwendet werden.
WEB TCP CLIENT STREAM command$, buffer%(), readpointer%, writepointer%	Stellt eine Verbindung zu einem Server her, der zuvor mit WEB OPEN TCP STREAM geöffnet wurde. 'command$' ist eine Zeichenkette und die Anfrage, die an den Server gesendet

	werden soll. 'buffer%()' ist ein Integer-Array, das die laufenden Antworten empfängt und als zirkulärer Puffer für empfangene Bytes fungiert. Die Firmware verwaltet den Parameter „writepointer%", wenn die Daten vom Server eintreffen. „readpointer%" sollte vom Basic-Programm gepflegt werden, da es Daten aus dem Ringpuffer entfernt. Wenn „writepointer%" „readpointer%" einholt, wird „readpointer%" erhöht, um ein Byte voraus zu bleiben, und eingehende Daten gehen verloren. 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 Öffnungsversuch unternommen wird.
WEB TCP INTERRUPT InterruptSub	Startet den TCP-Server. „InterruptSub" ist die Subroutine, die aufgerufen wird, wenn eine Anfrage an den TCP-Server gestellt wird (d. h. eine Unterbrechung). Beachte, dass zuerst der Befehl OPTION WIFI und dann der Befehl OPTION TCP SERVER PORT verwendet werden muss, um den TCP-Server zu aktivieren.
WEB TCP READ cb%, buff%()	Lies die Daten von einer möglichen TCP-Verbindung 'cb%'. 'buff%()' ist ein Array, das alle Daten von dieser Verbindung als Longstring empfängt. Die Größe dieses Puffers bestimmt, wie viele Daten vom Remote-Client empfangen werden können. Wenn über diese Verbindung nichts empfangen wird, gibt das Programm eine leere Zeichenfolge zurück (d. h. `LLEN(buff%())=0`). Wenn Daten empfangen wurden, muss das BASIC-Programm mit einem der WEB TRANSMIT-Befehle antworten, um zu reagieren und die Verbindung zu schließen.
WEB TCP SEND cb%, data%() WEB TCP CLOSE cb%	Diese beiden Befehle bieten mehr Flexibilität bei der Verwendung des TCP-Servers. Im Gegensatz zu WEB TRANSMIT PAGE oder WEB TRANSMIT FILE erstellt WEB TCP SEND weder einen Header noch schließt es die TCP-Verbindung nach der Übertragung. Es sendet einfach genau das, was sich in LONGSTRING data%() befindet, und es ist Aufgabe des Basic-Programmierers, die Verbindung zum richtigen Zeitpunkt zu schließen.
WEB TRANSMIT CODE cb%, nnn%	Sendet eine numerische Antwort an die offene TCP-Verbindung „cb%" und schließt dann die Verbindung. Typisch wäre `TRANSMIT CODE cb%, 404,` um anzuzeigen, dass die Seite nicht gefunden wurde.
WEB TRANSMIT FILE cb%, Dateiname$, Inhaltstyp	Erstellt einen HTTP 1.1-Header mit dem angegebenen „content-type$", sendet ihn und sendet dann den Inhalt der Datei an die offene TCP-Verbindung cb%. Nach Abschluss wird die Verbindung geschlossen. „content-type$" ist ein MIME-Typ, der als Zeichenfolge ausgedrückt wird. Z. B. „image/jpeg".

WEB TRANSMIT PAGE cb %, Dateiname$ [,Puffergröße]	Erstellt einen HTTP 1.1-Header, sendet ihn und schickt dann den Inhalt der Datei an die offene TCP-Verbindung cb%. Nach Abschluss wird die Verbindung geschlossen. MMBasic ersetzt alle MMBasic-Variablen oder Ausdrücke, die in der Datei in geschweiften Klammern definiert sind, z. B. {myvar%}, durch aktuelle Werte. Variablen können einfach, Array-Elemente oder Ausdrücke sein. Eine öffnende geschweifte Klammer kann mit {{ in die Ausgabe aufgenommen werden. Standardmäßig weist der Befehl einen Puffer in der Größe der Datei + 4096 Bytes zu, um die zu übertragende Seite zu erstellen. Wenn die Seite jedoch komplex ist und viele MMBasic-Variablen enthält, die Text erzeugen, der größer als der Variablenname ist, kann es sein, dass der Puffer nicht groß genug ist. In diesem Fall kann der Benutzer den zusätzlich benötigten Speicherplatz angeben (Standardwert ist 4096, wenn nichts angegeben ist).
WEB UDP INTERRUPT intname	Richtet eine BASIC-Interrupt-Routine ein, die bei jedem Empfang eines UDP-Datagramms ausgelöst wird. Der Inhalt wird in MM.MESSAGE$ gespeichert. Die IP-Adresse des Absenders wird in MM.ADDRESS$ gespeichert.
WEB UDP SEND addr$, port, data	Wird verwendet, um ein Datagramm an einen entfernten Empfänger zu senden. In diesem Fall muss die IP-Adresse angegeben werden, die entweder eine numerische Adresse (z. B. „192.168.1.147“) oder eine normale Textadresse (z. B. „google.com“) sein kann. Die Portnummer des Empfängers muss ebenfalls angegeben werden, ebenso wie die Nachricht selbst. Der Befehl SEND kann als Antwort auf eine eingehende Nachricht oder eigenständig verwendet werden.
WS2812 Typ, Pin, Nr., Wert% [()]	Dieser Befehl steuert einen oder mehrere WS2812-LED-Chips, die an „Pin“ angeschlossen sind. Beachte, dass der Pin vor Verwendung dieses Befehls auf einen digitalen Ausgang eingestellt werden muss. „type” ist ein einzelnes Zeichen, das den Typ des angesteuerten Chips angibt: O = Original WS2812 B = WS2812B S = SK6812 W = SK6812W (RGBW) „nbr“ ist die Anzahl der LEDs in der Kette (1 bis 256). Das Array „value%()“ sollte ein Integer-Array sein, dessen Größe genau der Anzahl der anzusteuernden LEDs entspricht. Bei den ersten drei Varianten sollte jedes Element im Array die Farbe im normalen RGB888-Format enthalten (d. h. 0 bis &HFFFFFF). Für den Typ W nimm einen RGBW-Wert (0-&HFFFFFFFF). Wenn nur eine LED angeschlossen ist, sollte für „value%“ eine einzelne Ganzzahl verwendet werden (also kein Array).
XMODEM SEND oder XMODEM SEND file$ oder XMODEM RECEIVE oder XMODEM RECEIVE file$ oder XMODEM CRUNCH	Überträgt ein BASIC-Programm mit dem XModem-Protokoll zu oder von einem Remote-Computer. Die Übertragung läuft über die USB-Konsolenverbindung. XMODEM SEND sendet das aktuelle Programm, das im Programmspeicher des PicoMite gespeichert ist, an das Remote-Gerät. XMODEM RECEIVE nimmt ein vom Remote-Gerät gesendetes Programm an und speichert es im Programmspeicher des PicoMite, wobei das dort aktuell gespeicherte Programm überschrieben wird. 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 schon da ist, wird sie beim Empfang einer Datei

	überschrieben. Beachte, dass die Daten im RAM gepuffert werden, was die maximale Übertragungsgröße begrenzt. Dieser Befehl erstellt auch eine Sicherungskopie des Programms im Flash-Speicher, die automatisch abgerufen wird, wenn die CPU zurückgesetzt wird oder die Stromversorgung ausfällt. 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 verwendet werden, damit sie in den begrenzten Speicher passen. SEND, RECEIVE und CRUNCH können mit S, R und C abgekürzt werden. Das XModem-Protokoll braucht ein kooperierendes 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 Programm zu verwenden. Nachdem du den XMODEM-Befehl in MMBasic ausgeführt hast, wählst du: Datei -> Übertragen -> XMODEM -> Empfangen/Senden im Tera Term-Menü, um die Übertragung zu starten. Die Übertragung kann bis zu 15 Sekunden dauern, bis sie startet. Wenn der XMODEM-Befehl die Kommunikation nicht herstellen kann, kehrt er nach 60 Sekunden zur MMBasic-Eingabeaufforderung zurück und lässt den Programmspeicher unberührt. Lade Tera Term von http://ttssh2.sourceforge.jp/ herunter.
YMODEM SEND oder YMODEM SEND Datei$ oder YMODEM RECEIVE oder YMODEM RECEIVE Datei$ oder YMODEM CRUNCH	Nur für RP2350-Versionen ohne USB Dieser Befehl ist wie XMODEM, überträgt Dateien aber viel schneller (>10x) über eine serielle CDC-Verbindung. YMODEM unterscheidet sich von XMODEM dadurch, dass die Nachricht den Dateinamen enthält. Das heißt, wenn du eine Datei vom Pico an einen Computer sendest, verwendet das empfangende Programm automatisch denselben Namen, den du beim Senden angegeben hast. Wenn die Datei schon da ist, erhöht der Empfänger automatisch die Versionsnummer im Namen der empfangenen Datei. fred1.bas, fred2.bas usw. Beim Senden aus dem Speicher mit YMODEM S gibt es zwei Möglichkeiten. Wenn die Datei vom Laufwerk A: oder B: geladen wurde, hat sie einen versteckten Dateinamen, der dann benutzt wird. Wenn die Datei mit dem Editor erstellt oder automatisch gespeichert wurde, hat die empfangene Datei automatisch den Namen FILEn.DAT Es liegt in der Verantwortung des empfangenden Programms, anzugeben, wo die Datei gespeichert werden soll. Bei Teraterm ist das eine Option, die du im allgemeinen Einstellungsbildschirm unter „Dateiübertragungsordner” einstellen kannst. Wenn du eine Datei auf dem Pico empfängst, kannst du den Namen und das Verzeichnis ganz normal frei wählen. Hinweis: YMODEM reagiert sehr empfindlich auf die Qualität der Verbindung 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(Zahl)	Gibt den absoluten Wert des Arguments „Zahl" zurück (d. h. alle negativen Vorzeichen werden entfernt und eine positive Zahl zurückgegeben).
ACOS(Zahl)	Gibt den inversen Kosinus des Arguments „Zahl" in Radianten zurück.
ASC(Zeichenfolge$)	Gibt den ASCII-Code (also den Byte-Wert) für den ersten Buchstaben in „Zeichenfolge$" zurück.
ASIN(Zahl)	Gibt den inversen Sinuswert des Arguments „Zahl" in Radianten zurück.
ATN(Zahl)	Gibt den Arkustangens des Arguments „number" in Radianten zurück.
ATAN2(y, x)	Gibt den Arkustangens der beiden Zahlen x und y als Winkel in Radianten zurück. Radiant ausgedrückt. Das ist ähnlich wie die Berechnung des Arkustangens von y / x, nur dass die Vorzeichen beider Argumente verwendet werden, um den Quadranten des Ergebnisses zu bestimmen.
BIN$(Zahl [, Zeichen])	Gibt eine Zeichenfolge zurück, die den Binärwert (Basis 2) für die „Zahl" angibt. „Zeichen" ist optional und gibt die Anzahl der Zeichen in der Zeichenfolge an, wobei Null als führendes Auffüllzeichen verwendet wird.
BIN2STR$(Typ, Wert [,BIG])	Gibt eine Zeichenfolge zurück, die die binäre Darstellung von „Wert" enthält. „type" kann sein: INT64 vorzeichenbehaftete 64-Bit-Ganzzahl, die in eine 8-Byte-Zeichenkette umgewandelt wird UINT64 vorzeichenlose 64-Bit-Ganzzahl, die in eine 8-Byte-Zeichenkette umgewandelt wurde INT32 Vorzeichenbehaftete 32-Bit-Ganzzahl, die in eine 4-Byte-Zeichenkette umgewandelt wird UINT32 Vorzeichenlose 32-Bit-Ganzzahl, die in eine 4-Byte-Zeichenkette umgewandelt wird INT16 vorzeichenbehaftete 16-Bit-Ganzzahl, die in eine 2-Byte-Zeichenkette umgewandelt wird UINT16 Vorzeichenlose 16-Bit-Ganzzahl, die in eine 2-Byte-Zeichenkette umgewandelt wird INT8 vorzeichenbehaftete 8-Bit-Ganzzahl, die in eine 1-Byte-Zeichenkette umgewandelt wird UINT8 Vorzeichenlose 8-Bit-Ganzzahl, die in eine 1-Byte-Zeichenkette umgewandelt wird EINFACHE Gleitkommazahl mit einfacher Genauigkeit, die in eine 4-Byte-Zeichenkette umgewandelt wird DOUBLE Doppelt genaue Gleitkommazahl, die in eine 8-Byte-Zeichenkette umgewandelt wird 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 Umwandlung von Ganzzahlen wird ein Fehler ausgegeben, wenn der „Wert" nicht in den „Typ" passt (z. B. wenn versucht wird, den Wert 400 in einem INT8 zu speichern). Diese Funktion erleichtert die Vorbereitung von Daten für eine effiziente binäre Datei-E/A oder die Vorbereitung von Zahlen für die Ausgabe an Sensoren und das Speichern im Flash-Speicher. Siehe auch die Funktion STR2BIN
BIT(var%, bitno)	gibt den Wert eines bestimmten Bits (0-63) in einer Integer-Variablen (0 oder 1) zurück. Siehe auch den Befehl BIT
BOUND(array() [,dimension]	Gibt die Obergrenze des Arrays für die angeforderte Dimension zurück. Die Dimension ist standardmäßig eins, wenn nichts anderes angegeben wird. Wenn du einen Dimensionswert von 0 angibst, wird der aktuelle Wert von OPTION BASE zurückgegeben. Nicht verwendete Dimensionen geben den Wert Null zurück. Beispiel: DIM myarray(44,45) BOUND(myarray(),2) gibt 45 zurück
BYTE(var$, byteno)	Gibt den ganzzahligen Wert eines bestimmten Bytes in einer Zeichenfolge zurück (0-255). Dies entspricht ASC(MID$(var$,byteno,1)), arbeitet jedoch wesentlich schneller. Schau dir auch den Befehl BYTE an
CALL(userfunname$, [,userfunparameters,..])	Das ist eine coole Möglichkeit, 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 läuft viel schneller. „userfunname$" kann eine beliebige Zeichenfolge, Variable oder Funktion sein, die zum Namen einer normalen Benutzerfunktion (kein integrierter Befehl) aufgelöst wird. „userfunparameters" sind die gleichen Parameter, die zum direkten Aufruf der Funktion verwendet würden. Ein typischer Anwendungsfall für diesen Befehl könnte das Schreiben einer beliebigen Art von Emulator sein, bei dem eine von vielen Funktionen in Abhängigkeit von einer bestimmten Variablen aufgerufen werden soll. Er bietet auch eine Methode, um einen Funktionsnamen als Variable an eine andere Subroutine oder Funktion zu übergeben.
CHOICE(Bedingung, AusdruckWennWahr, AusdruckWennFalsch)	Mit dieser Funktion kannst du einfache Entweder-oder-Auswahlen effizienter und schneller machen als mit IF THEN ELSE ENDIF-Klauseln. Die Bedingung ist alles, was zu einem Wert ungleich Null (wahr) oder Null (falsch) führt. Die Ausdrücke können alles sein, was du normalerweise einer Variablen zuweisen oder in einem Befehl verwenden kannst, und können Ganzzahlen, Gleitkommazahlen oder Zeichenfolgen sein. Beispiele: PRINT CHOICE(1, "Hallo","Tschüss") gibt "Hallo" aus PRINT CHOICE (0, "hello","bye") gibt "Bye" aus a=1 : b=1 : PRINT CHOICE (a=b, 4, 5) gibt 4 aus
CHR$(Zahl)	Gibt eine einstellige Zeichenkette zurück, die aus dem Zeichen besteht, das

	dem durch das Argument „number" angegebenen ASCII-Code (d. h. Byte-Wert) entspricht.
CINT(Zahl)	Rundet Zahlen mit Bruchteilen auf oder ab auf die nächste ganze Zahl oder ganze Zahl. Zum Beispiel wird 45,47 auf 45 gerundet 45,57 wird auf 46 gerundet -34,45 wird auf -34 gerundet -34,55 wird auf -35 gerundet Schau dir auch INT() und FIX() an.
COS(Zahl)	Gibt den Kosinus des Arguments „Zahl" in Radianten zurück.
CWD$	Gibt das aktuelle Arbeitsverzeichnis auf dem Flash-Dateisystem oder der SD-Karte zurück. Für das exFAT-Format nicht gültig. Das Format ist: `A:/dir1/dir2`.
DATE$	Gibt das aktuelle Datum basierend auf der internen Uhr von MMBasic als Zeichenfolge im Format „TT-MM-JJJJ" zurück. Zum Beispiel „28-07-2012". Die interne Uhr/der interne Kalender verfolgt die Zeit und das Datum einschließlich Schaltjahren. Um das Datum festzulegen, benutze den Befehl DATE$ =.
DATETIME$(n)	Gibt das Datum und die Uhrzeit zurück, die der Epoche Nummer „n" entsprechen (Anzahl der Sekunden, die seit Mitternacht GMT am 1. Januar 1970 vergangen sind). Das Format der zurückgegebenen Zeichenfolge ist „dd-mm-yyyy hh:mm:ss". Verwende den Text NOW, um die aktuelle Datums- und Zeitzeichenfolge zu erhalten, d. h. DATETIME$(NOW)
DAY$(date$)	Gibt den Wochentag für ein bestimmtes Datum als Zeichenfolge zurück. Zum Beispiel „Montag", „Dienstag" usw. „date$" ist eine Zeichenfolge, deren Format DD-MM-YY, DD-MM-YYYY oder YYYY-MM-DD sein kann. Du kannst auch NOW verwenden, um den Tag für das aktuelle Datum zu erhalten, z. B. PRINT DAY$(NOW)
DEG(Radiant)	Rechnet „Bogenmaß" in Grad um.
DEVICE(GAMEPAD channel, funct)	Gibt Daten von einem USB-PS3- oder PS4-Controller zurück. „funct" ist ein 1- oder 2-Buchstaben-Code, der die zurückzugebenden Informationen wie folgt angibt: LX die Position der x-Achse des analogen linken Joysticks LY die Position der analogen linken Joystick-Y-Achse RX die Position der x-Achse des analogen rechten Joysticks RY die Position der analogen rechten Joystick-Y-Achse GX der Messwert vom X-Achsen-Gyroskop (wo unterstützt) GY der Messwert vom Y-Achsen-Gyroskop (sofern unterstützt) GZ der Messwert vom Z-Achsen-Gyroskop (wenn unterstützt) AX Der Messwert vom Beschleunigungsmesser der X-Achse (wenn unterstützt) AY der Messwert vom Y-Achsen-Beschleunigungsmesser (wenn unterstützt) AZ der Messwert vom Z-Achsen-Beschleunigungsmesser (wenn verfügbar) L die Position der analogen linken Taste R die Position der analogen rechten Taste

	B eine Bitmap des Status aller Tasten. Ein Bit wird auf 1 gesetzt, wenn die Taste gedrückt wird. T die ID-Nummer des Controllers Die Tasten-Bitmap sieht so aus: BIT 0 Taste R/R1 BIT 1 Taste Start/Optionen BIT 2 Taste Home BIT 3 Auswahl-/Teilen-Taste BIT 4 Taste L/L1 BIT 5 Taste Cursor nach unten BIT 6 Taste rechter Cursor BIT 7 Taste Cursor nach oben BIT 8 Taste links Cursor BIT 9 Rechte Schultertaste 2/R2 BIT 10 Taste X/Dreieck BIT 11 Taste a/Kreis BIT 12 Taste y/Quadrat BIT 13 Taste b/Kreuz BIT 14 Linker Schulterknopf 2/L2 BIT 15 Touchpad
DEVICE(MOUSE channel, funct)	Sendet Daten von einer Maus, die über den „Kanal" angeschlossen ist. Eine PS2-Maus wird immer Kanal 2 zugewiesen. Normalerweise wird auch eine USB-Maus Kanal 2 zugewiesen, aber das kann variieren. Weitere Infos findest du unter MM.INFO(USB n). „funct" ist ein einstelliger Code, der die zurückzugebenden Informationen wie folgt angibt: X die X-Koordinate (0 bis MM.HRES-1) Y die Y-Koordinate (0 bis MM.VRES-1) L der Status der linken Maustaste R der Status der rechten Maustaste M der Status der mittleren Maustaste (Scrollrad-Klick) D 1, wenn die linke Maustaste doppelt geklickt wurde W gibt die Veränderung der Radposition seit dem letzten Aufruf an (PS2-Maus oder USB-Maus mit eingestellter OPTION MOUSE SENSITIVITY) B Der Status aller Tasten als Bitmap (nur USB-Maus)
DEVICE(WII [CLASSIC] funct)	Gibt Daten von einem Wii Classic Controller zurück. „funct" ist ein 1- oder 2-Buchstaben-Code, der die zurückzugebenden Informationen wie folgt angibt: LX die Position der x-Achse des analogen linken Joysticks LY die Position der analogen linken Joystick-Y-Achse RX die Position der x-Achse des analogen rechten Joysticks RY die Position der analogen rechten Joystick-Y-Achse L die Position des analogen linken Knopfes R die Position des analogen rechten Knopfes B eine Bitmap des Status aller Tasten. Ein Bit wird auf 1 gesetzt, wenn die Taste gedrückt wird. T Der ID-Code des Controllers – sollte hex &HA4200101 sein Die Tasten-Bitmap sieht so aus: BIT 0 Taste R

	BIT 1 Taste Start BIT 2 Taste Home BIT 3 Auswahl-Taste BIT 4 Taste L BIT 5 Taste „Cursor nach unten“ BIT 6 Taste Cursor nach rechts BIT 7 Taste Cursor nach oben BIT 8 Taste links Cursor BIT 9 Taste ZR BIT 10 Taste x BIT 11 Taste a BIT 12 Taste y BIT 13 Taste b BIT 14 Taste ZL
DEVICE(NUNCHUCK-Funktion)	Gibt Daten von einem Nunchuck-Controller zurück. „funct” ist ein 1- oder 2-stelliger Code, der die zurückzugebenden Informationen wie folgt angibt: AX die Beschleunigung der x-Achse AY die Beschleunigung auf der y-Achse AZ die Beschleunigung auf der Z-Achse JX die Position der X-Achse des Joysticks JY die Position der Y-Achse des Joysticks C der Status der C-Taste Z der Status der Z-Taste T der ID-Code des Controllers – sollte hex &HA4200000 sein

DIR$(fspec, type)
oder
DIR$(fspec)
oder
DIR$( )
	Sucht im Standard-Flash-Dateisystem oder auf der SD-Karte nach Dateien und gibt die Namen der gefundenen Einträge zurück.
	„fspec” ist eine Dateispezifikation mit Platzhaltern, die genauso wie beim Befehl FILES verwendet werden. Zum Beispiel gibt „*.*” alle Einträge zurück, „*.TXT” gibt Textdateien zurück. Beachte, dass der Platzhalter *.* keine Dateien oder Ordner ohne Erweiterung findet.
	„type” ist der Typ des zurückzugebenden Eintrags und kann einer der folgenden sein:
	ALL Suche nach allen Dateien und Verzeichnissen
	DIR Nur nach Verzeichnissen suchen
	FILE Nur nach Dateien suchen (Standard, wenn „type” nicht angegeben ist)
	Die Funktion gibt den ersten gefundenen Eintrag zurück. Um weitere Einträge zu bekommen, benutze die Funktion ohne Argumente, also DIR$( ). Wenn eine leere Zeichenfolge zurückgegeben wird, gibt's keine weiteren Einträge mehr.
	In diesem Beispiel werden alle Dateien in einem Verzeichnis angezeigt:
	`f$ = DIR$("*.*", FILE)`
	`DO WHILE f$ <> ""`
	`PRINT f$`
	`f$ = DIR$()`
	`LOOP`
	Du musst in das gewünschte Verzeichnis wechseln, bevor du diesen Befehl ausführst.

DISTANCE(trigger, echo)	Misst den Abstand zu einem Ziel mit dem Ultraschall-Abstandssensor HC-

oder DISTANCE(trig-echo)	SR04. Vierpolige Sensoren haben 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. Dreipolige Sensoren haben einen kombinierten Trigger- und Echoanschluss. In diesem Fall musst du nur einen I/O-Pin für die Schnittstelle zum Sensor angeben. 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. Die I/O-Pins werden von dieser Funktion automatisch konfiguriert, und es können mehrere Sensoren an verschiedenen I/O-Pins verwendet werden. Der zurückgegebene Wert ist die Entfernung zum Ziel in Zentimetern oder -1, wenn kein Ziel erkannt wurde, oder -2, wenn ein Fehler aufgetreten ist (z. B. Sensor nicht angeschlossen).
Draw3D(arg)	NICHT IN DER WEBMITE-VERSION VERFÜGBAR Die Funktion DRAW3D kann verwendet werden, um die Grenzen eines 3D-Objekts zu bestimmen und einen Bereich zu definieren, der vor dem erneuten Zeichnen gelöscht werden soll. Eine vollständige Beschreibung findest du im Dokument *„3D_Graphics_User_Manual.*pdf" im PicoMite-Firmware-Download.
EOF([#]fnbr)	Gibt „true" zurück, wenn die zuvor im Flash-Dateisystem oder auf der SD-Karte für INPUT geöffnete Datei mit der Dateinummer „#fnbr" am Ende der Datei positioniert ist. Das # ist optional. Sieh dir auch die Befehle OPEN, INPUT und LINE INPUT sowie die Funktion INPUT$ an.
EPOCH(DATETIME$)	Gibt die Epochenzahl (Anzahl der Sekunden, die seit Mitternacht GMT am 1. Januar 1970 vergangen sind) für die angegebene DATETIME$-Zeichenfolge zurück. Das Format für DATETIME$ ist „dd-mm-yyyy hh:mm:ss", „dd-mm-yy hh:mm:ss" oder „yyyy-mm-dd hh:mm:ss". Verwende NOW, um die Epoche für das aktuelle Datum und die aktuelle Uhrzeit zu erhalten, z. B. PRINT EPOCH(NOW).
EVAL(string$)	Wertet 'string$' wie einen BASIC-Ausdruck aus und gibt das Ergebnis zurück. 'string$' kann eine Konstante, eine Variable oder ein String-Ausdruck sein. Der Ausdruck kann alle Operatoren, Funktionen, Variablen, Unterprogramme usw. verwenden, die zum Zeitpunkt der Ausführung bekannt sind. Der zurückgegebene Wert ist je nach Ergebnis der Auswertung eine Ganzzahl, eine Gleitkommazahl oder ein String. Beispiel: S$ = "COS(RAD(30)) * 100" : PRINT EVAL(S$) Zeigt an: 86,6025
EXP(Zahl)	Gibt den Exponentialwert von „number" zurück, also e^x, wobei x „number" ist.
FIELD$(string1, nbr, string2 [, string3])	Gibt ein bestimmtes Feld in einer Zeichenfolge zurück, wobei die Felder durch Trennzeichen getrennt sind. Beachte, dass ein Leerzeichen nicht als Trennzeichen verwendet werden kann. „nbr" ist das zurückzugebende Feld (das erste ist nbr 1). „Zeichenfolge1" ist die zu suchende Zeichenfolge und „Zeichenfolge2" ist eine Zeichenfolge, die die Trennzeichen enthält (es können mehrere verwendet werden). Das Leerzeichen darf nicht als Trennzeichen verwendet werden.

	'string3' ist optional und enthält, wenn angegeben, Zeichen, die zum Zitieren von Text in 'string1' verwendet werden (d. h., zitierter Text wird nicht nach einem Trennzeichen durchsucht). Beispiel: S$ = "foo, boo, zoo, doo" r$ = FIELD$(s$, 2, ",") ergibt r$ = „boo". Während: s$ = "foo, 'boo, zoo', doo" r$ = FIELD$(s$, 2, ",", "'") gibt das r$ = "boo, zoo".
FIX(Zahl)	Kürzt eine Zahl auf eine ganze Zahl, indem der Dezimalpunkt und alle Zeichen rechts davon entfernt werden. Zum Beispiel gibt 9,89 den Wert 9 zurück und -2,11 den Wert -2. Der Hauptunterschied zwischen FIX() und INT() besteht darin, dass FIX() eine echte Ganzzahlfunktion bietet (d. h. gibt bei negativen Zahlen nicht wie INT() die nächstkleinere Zahl zurück). Dieses Verhalten dient der Kompatibilität mit Microsoft. Siehe auch CINT().
FLAG(n%)	Gibt den Wert (0 oder 1) des Bits n% (0-63) im Systemflag-Register zurück. Siehe auch MM.FLAGS und die Befehle FLAG und FLAGS.
FORMAT$(nbr [, fmt$])	Gibt eine Zeichenfolge zurück, die „nbr" gemäß den Angaben in der Zeichenfolge „fmt$" formatiert. Die Formatangabe fängt mit einem %-Zeichen an und endet mit einem Buchstaben. Alles, was außerhalb dieser Konstruktion ist, wird so wie es ist in die Ausgabe kopiert. Die Struktur einer Formatvorgabe ist: % [Flags] [Breite] [.Genauigkeit] Typ Dabei kann „flags" sein: - Den Wert innerhalb einer bestimmten Feldbreite linksbündig ausrichten 0 Verwende 0 als Füllzeichen anstelle von Leerzeichen + Zwingt die Anzeige des Pluszeichens für positive Zahlen Leerzeichen Sorgt dafür, dass bei positiven Werten ein Leerzeichen für das Vorzeichen angezeigt wird. Negative Werte zeigen weiterhin das – Zeichen. „width" ist die Mindestanzahl der auszugebenden Zeichen. Bei weniger Zeichen wird die Zahl aufgefüllt, bei mehr Zeichen wird die Breite erweitert. „precision" gibt die Anzahl der zu generierenden Dezimalstellen mit einem e- oder f-Typ oder die maximale Anzahl der zu generierenden signifikanten Stellen mit einem g-Typ an und ist standardmäßig auf 4 Stellen eingestellt. Wenn angegeben, muss der Genauigkeit ein Punkt (.) vorangestellt werden. „type" kann einer der folgenden Werte sein: g Formatiert die Zahl automatisch für die beste Darstellung. f Formatiert die Zahl mit dem Dezimalpunkt und den folgenden Stellen. e Formatiere die Zahl im Exponentialformat. Wenn Großbuchstaben G oder F verwendet werden, wird für die exponentielle Ausgabe ein Großbuchstabe E verwendet. Wenn die Formatangabe nicht angegeben ist, wird „%g" angenommen.

	Beispiele: format$(45) gibt 45 zurück format$(45, „%g”) gibt 45 zurück
GETSCANLINE	NUR VGA- UND HDMI-VERSIONEN Das zeigt die Zeile an, die gerade auf dem VGA/HDMI-Monitor im Bereich von 0 bis 525 gezeichnet wird. Das ist unabhängig vom aktuellen MODUS. Wenn du das nutzt, um Updates auf dem Bildschirm zu timen, kannst du Timing-Effekte vermeiden, die durch Updates entstehen, während der Bildschirm aktualisiert wird. Die erste sichtbare Zeile gibt den Wert 0 zurück. Alle Zeilennummern über 479 liegen im Bildausblendungszeitraum.
GPS()	Die GPS-Funktion gibt Daten von einem seriellen Kommunikationskanal zurück, der mit den Befehlen OPEN GPS oder OPTION GPS als GPS geöffnet wurde. Diese Funktion wird ausführlich in der Datei *Option_GPS_User_Manual.pdf* beschrieben, die im ZIP-Archiv zum Herunterladen der Firmware enthalten ist. Die Funktion GPS(VALID) sollte vor dem Zugriff auf die Daten überprüft werden, um sicherzustellen, dass der zurückgegebene Wert gültig ist.
HEX$(Zahl [, Zeichen])	Gibt eine Zeichenfolge zurück, die den Hexadezimalwert (Basis 16) für die „Zahl” angibt. „Zeichen” ist optional und gibt die Anzahl der Zeichen in der Zeichenfolge an, wobei Null als führendes Auffüllzeichen verwendet wird.
INKEY	Überprüft den Eingabepuffer der Konsole und entfernt, wenn ein oder mehrere Zeichen in der Warteschlange stehen, das erste Zeichen und gibt es als einzelnes Zeichen in einer Zeichenfolge zurück. Wenn der Eingabepuffer leer ist, gibt diese Funktion sofort eine leere Zeichenkette (d. h. „“) zurück.
INPUT$(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 ist. Diese Funktion gibt so viele Zeichen zurück, wie in der Datei oder im Empfangspuffer bis zu „nbr“ vorhanden sind. Wenn keine Zeichen verfügbar sind, wird sofort eine leere Zeichenkette zurückgegeben. #0 kann verwendet werden, um auf den Eingabepuffer der Konsole zu verweisen. Das # ist optional. Sieh dir auch den Befehl OPEN an.
INSTR([Startposition,] gesuchter-String$, String-Muster$ [,Größe])	Gibt die Position zurück, an der „string-pattern$” in „string-searched$” vorkommt, beginnend bei „start-position”. Wenn „start-position” nicht angegeben wird, wird standardmäßig 1 verwendet. Sowohl die zurückgegebene Position als auch „start-position” verwenden 1 für das erste Zeichen, 2 für das zweite usw. Die Funktion gibt Null zurück, wenn 'string-pattern$' nicht gefunden wird. Wenn der optionale Parameter „size” angegeben wird, wird „string-pattern” wie ein regulärer Ausdruck behandelt. Details dazu findest du in *Anhang E*.
INT(Zahl)	Kürzt einen Ausdruck auf die nächste ganze Zahl, die kleiner oder gleich dem Argument ist. Beispielsweise gibt 9,89 den Wert 9 und -2,11 den Wert -3 zurück. Dieses Verhalten dient der Kompatibilität mit Microsoft. Die Funktion FIX() bietet eine echte Ganzzahlfunktion. Siehe auch CINT().

JSON$(array%(), string$)	Gibt eine Zeichenfolge zurück, die ein bestimmtes Element aus der JSON-Eingabe darstellt, die im Longstring-Array %() gespeichert ist. Beachte, dass viele JSON-Datensätze ziemlich groß sind und möglicherweise zu groß sind, um mit dem verfügbaren Speicherplatz analysiert zu werden. Beispiele (aus api.openweathermap.org): JSON$(a%(), „name") JSON$(a%(), „coord.lat") JSON$(a%(), „weather[0].description") JSON$(a%(),"list[4].weather[0].description
KEYDOWN(n)	Gibt den dezimalen ASCII-Wert der gerade gedrückten Taste der USB-Tastatur zurück oder Null, wenn keine Taste gedrückt ist. Die Dezimalwerte für die Funktions- und Pfeiltasten findest du in Anhang I. Diese Funktion meldet mehrere gleichzeitig gedrückte Tasten, wobei der Parameter „n" die Nummer der zu meldenden Tastenbetätigung ist. KEYDOWN(0) gibt die Anzahl der gedrückten Tasten zurück. Wenn zum Beispiel „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 gemeldet, in der sie gedrückt wurden. Wenn man einen Finger von einer Taste nimmt, wird die nächste gedrückte Taste zu #1. Die erste Taste („n" = 1) wird in den Tastaturpuffer eingegeben (zugänglich über INKEY$), während auf die Tasten 2 bis 6 nur über diese Funktion zugegriffen werden kann. Durch die Verwendung dieser Funktion wird der Konsoleneingabepuffer gelöscht. KEYDOWN(7) gibt alle gedrückten Modifikatortasten zurück. Diese Tasten werden nicht zur Zählung in keydown(0) hinzugefügt. Der Rückgabewert ist eine Bitmaske wie folgt: lalt = 1, lctrl = 2, lgui = 4, lshift = 8, ralt = 16, rctrl = 32, rgui = 64, rshift = 128 KEYDOWN(8) gibt den aktuellen Status der Sperrtasten zurück. Diese Tasten werden nicht zur Zählung in keydown(0) hinzugefügt. Der Rückgabewert ist eine Bitmaske wie folgt: caps_lock = 1, num_lock = 2, scroll_lock = 4 Beachte, dass manche Tastaturen die Anzahl der aktiven Tasten, die sie melden können, begrenzen.
LCASE$(string$)	Gibt „string$" in Kleinbuchstaben zurück.
LCOMPARE(array1%(), array2%())	Vergleiche den Inhalt von zwei langen String-Variablen „array1%()" und „array2%()". Die Rückgabe ist eine ganze Zahl und ist -1, wenn „array1%()" kleiner als „array2%()" ist. Sie ist Null, wenn sie in Länge und Inhalt gleich sind, und +1, wenn „array1%()" größer als „array2%()" ist. Der Vergleich nutzt den ASCII-Zeichensatz und unterscheidet zwischen Groß- und Kleinschreibung.
LEFT$(string$, nbr)	Gibt eine Teilzeichenfolge von „string$" mit „nbr" Zeichen vom Anfang der Zeichenfolge zurück.
LEN(string$)	Gibt die Anzahl der Zeichen in 'string$' zurück.
LGETBYTE(array%(), n)	Gibt den numerischen Wert des n-ten Bytes in der LONGSTRING zurück, die in „array%()" gespeichert ist. Diese Funktion berücksichtigt die Einstellung von OPTION BASE bei der Bestimmung des zurückzugebenden Bytes.

LGETSTR$(array%(), start, length)	Gibt einen Teil einer langen Zeichenfolge zurück, die in „array%()" als normale MMBasic-Zeichenfolge gespeichert ist. Die Parameter start und length legen fest, welcher Teil der Zeichenfolge zurückgegeben wird.
LINSTR(array%(), search$ [,start] [,size]))	Gibt die Position einer Suchzeichenfolge in einer langen Zeichenfolge zurück. Der zurückgegebene Wert ist eine ganze Zahl und ist Null, wenn die Teilzeichenfolge nicht gefunden werden kann. „array%()" ist die zu durchsuchende Zeichenfolge und muss eine lange Zeichenfolgenvariable sein. „search$" ist die zu suchende Teilzeichenfolge und muss eine normale MMBasic-Zeichenfolge oder ein Ausdruck sein (keine lange Zeichenfolge). Bei der Suche wird zwischen Groß- und Kleinschreibung unterschieden. Normalerweise fängt die Suche beim ersten Zeichen in 'array%()' an, aber mit dem optionalen dritten Parameter kann man die Startposition der Suche festlegen. Wenn der optionale Parameter „size" angegeben wird, wird „search$" wie ein regulärer Ausdruck behandelt. Details dazu findest du in *Anhang E*.
LLEN(array%())	Gibt die Länge einer langen Zeichenfolge zurück, die in „array%()" gespeichert ist.
LINPUT(array%(),fnbr,nbr)	Liest „nbr" Bytes aus einer als „fnbr" geöffneten Datei in die LONGSTRING „array%()". Die Funktion gibt die Anzahl der tatsächlich gelesenen Bytes zurück. Wenn du dich also am Ende der Datei befindest, kann die Anzahl kleiner sein als die angeforderte. Ende der Datei bist, kann die Zahl kleiner sein als die angeforderte. Das funktioniert nur mit Datei-E/A und nicht mit serieller oder Konsolen-E/A.
LOC([#]fnbr)	Für eine Datei im Flash-Dateisystem oder auf einer SD-Karte, die als „fnbr" geöffnet ist, gibt das die aktuelle Position des Lese-/Schreibzeigers in der Datei zurück. Beachte, dass das erste Byte in einer Datei die Nummer 1 hat. Für einen seriellen Kommunikationsport, der als „fnbr" geöffnet ist, gibt diese Funktion die Anzahl der empfangenen Bytes zurück, die im Empfangspuffer zum Lesen bereitstehen. #0 kann verwendet werden und bezieht sich auf den Eingabepuffer der Konsole. Das # ist optional.
LOF([#]fnbr)	Für eine Datei auf dem Flash-Dateisystem oder der SD-Karte, die als „fnbr" geöffnet ist, gibt das die aktuelle Länge der Datei in Bytes zurück. Für einen als „fnbr" geöffneten seriellen Kommunikationsport gibt diese Funktion den im Sendepuffer verbleibenden Platz (in Zeichen) zurück. Beachte, dass MMBasic bei vollem Puffer beim Hinzufügen eines neuen Zeichens pausiert und wartet, bis Platz verfügbar wird. Diese Funktion kann verwendet werden, um dies zu vermeiden. Das # ist optional.
LOG(Zahl)	Gibt den natürlichen Logarithmus des Arguments „number" zurück.

MAP(n)	NUR FÜR HDMI-, VGA- UND PICOMITE RP2350-PUFFERTreiber Gibt den 24-Bit-RGB-Wert für den Index „n" in der Farbtabelle zurück. Siehe den Befehl MAP. Damit kann der Basic-Programmierer eine durch den Befehl MAP festgelegte Farbe verwenden. z. B. MAP(8) = RGB(100,100,100) MAP SET Pixel x,y,map(8) Hinweis: Bei VGA werden alle mit dem Befehl „map" festgelegten Farben in die nächstgelegene RGB121-Farbe umgewandelt, die durch das VGA-Widerstandsnetzwerk bestimmt wird. Bei HDMI-Bildschirmen werden die Farben in die nächstgelegene RGB555-Farbe (Auflösung 640 x 480) oder RGB332-Farbe (Auflösung 1024 x 768 oder 1280 x 720) umgewandelt. Bei PIcoMite RP2350-gepufferten Treibern werden die Farben in die nächstgelegene RGB332-Farbe umgewandelt.
MATH **Einfache Funktionen** MATH(ATAN3 x,y) MATH(COSH a) MATH(LOG10 a) MATH(SINH a) MATH(TANH a) MATH(CRCn Daten [,Länge] [,Polynom] [,Startmaske] [,endmask] [,reverseIn] [,reverseOut] MATH(RAND) **Einfache Statistik** MATH(CHI a()) MATH(CHI_p a())	Die mathematische Funktion macht viele einfache Berechnungen, die man in Basic programmieren kann, aber es ist schneller, Schleifenstrukturen in C zu programmieren, und es ist auch gut, dass sie nach dem Debuggen für alle da sind, ohne dass man das Rad neu erfinden muss. Gibt ATAN3 von x und y zurück Gibt den hyperbolischen Kosinus von a zurück Gibt den Logarithmus zur Basis 10 von a zurück Gibt den hyperbolischen Sinus von a zurück Gibt den hyperbolischen Tangens von a zurück Berechnet den CRC auf n Bits (8, 12, 16, 32) von „data". „data" kann ein ganzzahliges oder ein Gleitkomma-Array oder eine String-Variable sein. „Length" ist optional und wenn es nicht angegeben wird, wird die Größe des Arrays oder die String-Länge 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 polynomes sind CRC8=&H07, CRC12=&H80D, CRC16=&H1021, crc32=&H04C11DB7 Beispiel: Für crc16_CCITT verwenden Sie MATH(CRC16 array(), n,, &HFFFF) Gibt eine Zufallszahl 0,0 <= n < 1,0 unter Verwendung des „Mersenne-Twister-Algorithmus" zurück. Wenn nicht mit MATH RANDOMIZE initialisiert, wird bei der ersten Verwendung die Zeit in Mikrosekunden seit dem Start als Startwert verwendet. Hinweis: Der RP2350 enthält einen H/W-Zufallszahlengenerator.

	Gibt den Pearson-Chi-Quadrat-Wert des zweidimensionalen Arrays a()) zurück.
MATH(CROSSING array() [,level] [,direction]	Gibt die zugehörige Wahrscheinlichkeit in % des Pearson-Chi-Quadrat-Werts des zweidimensionalen Arrays a()) zurück.
MATH(CORREL a(), a())	Gibt den Array-Index zurück, bei dem die Werte im Array den „Level" in der angegebenen Richtung überschreiten. Der Standardwert für „level" ist 0. Der Standardwert für „direction" ist 1 (gültige Werte sind -1 oder 1).
MATH(MAX a() [,index%])	Gibt den Pearson-Korrelationskoeffizienten zwischen den Arrays a() und b() zurück.
MATH(MEAN a())	Gibt das Maximum aller Werte im Array a() zurück, wobei a() beliebig viele Dimensionen haben kann. Wenn die Ganzzahlvariable angegeben ist, wird sie mit dem Index des Maximalwerts im Array aktualisiert. Dies ist nur für eindimensionale Arrays verfügbar.
MATH(MEDIAN a())	Gibt den Durchschnitt aller Werte im Array a() zurück, wobei a() beliebig viele Dimensionen haben kann.
MATH(MIN a(), [index%])	Gibt den Median aller Werte im Array a() zurück, wobei a() beliebig viele Dimensionen haben kann
	Gibt den kleinsten Wert aller Werte im Array a() zurück, wobei a() beliebig viele Dimensionen haben kann. Wenn die Ganzzahlvariable angegeben ist, wird sie mit dem Index des kleinsten Werts im Array aktualisiert. Dies ist nur bei eindimensionalen Arrays verfügbar.
MATH(SD a())	
MATH(SUM a())	Gibt die Standardabweichung aller Werte im Array a() zurück, wobei a() beliebig viele Dimensionen haben kann.
Vektorarithmetik	
MATH(MAGNITUDE v())	Gibt die Summe aller Werte im Array a() zurück, wobei a() beliebig viele Dimensionen haben kann.
MATH(DOTPRODUCT v1(), v2())	
	Gibt die Größe des Vektors v() zurück. Der Vektor kann beliebig viele Elemente haben.
Matrix-Arithmetik	
MATH(M_DETERMINANT array!())	Gibt das Skalarprodukt der beiden Vektoren v1() und v2() zurück. Die Vektoren können beliebig viele Elemente haben, müssen aber die gleiche Kardinalität haben.
	Gibt die Determinante des Arrays zurück. Das Array muss quadratisch sein.

Erstellung complex% = MATH(C_CPLX r!, i!) complex% = MATH(C_POLAR radius!, angle!) **Gleitkommazahlen** real! = MATH(C_REAL complex%) imag! = MATH(C_IMAG complex%) arg! = MATH(C_ARG complex%) mod! = MATH(C_MOD komplex%) phase! = MATH(C_PHASE complex%) **Unäre Funktionen** complex1% = MATH(C_CONJ complex2%) complex1% = MATH(C_SIN complex2%) complex1% = MATH(C_COS complex2%) complex1% = MATH(C_TAN complex2%) complex1% = MATH(C_ASIN complex2%) komplex1% = MATH(C_ACOS komplex2%) komplex1% = MATH(C_ATAN komplex2%) komplex1% = MATH(C_SINH komplex2%) komplex1% = MATH(C_COSH komplex2%) komplex1% = MATH(C_TANH komplex2%) komplex1% = MATH(C_ASINH komplex2%) komplex1% = MATH(C_ACOSH komplex2%) komplex1% = MATH(C_ATANH komplex2%) komplex1% = MATH(C_PROJ komplex2%) **Grundlegende Arithmetik** komplex1% = MATH(C_ADD komplex2%,komplex3%) komplex1% = MATH(C_SUB komplex2%,komplex3%) complex1% = MATH(C_MUL complex2%,complex3%) komplex1% = MATH(C_DIV komplex2%,komplex3%) komplex1% = MATH(C_POW komplex2%,komplex3%) komplex1% = MATH(C_AND komplex2%,komplex3%) komplex1% = MATH(C_OR komplex2%,komplex3%) komplex1% = MATH(C_XOR komplex2%,komplex3%)	MMBasic hat alle Funktionen, die du brauchst, um mit komplexen Zahlen rumzuspielen. In dieser Version haben komplexe Zahlen einen 32-Bit-Realteil und einen 32-Bit-Imaginärteil. Damit das in MMBasic klappt, werden dafür Ganzzahlen (64 Bit) benutzt.
MATH(PID-Kanal, Sollwert!, Messwert))	Diese Funktion muss in der PID-Callback-Subroutine für den angegebenen „Kanal" aufgerufen werden und gibt die Ausgabe der Reglerfunktion zurück. Der Wert „setpoint" ist der gewünschte Zustand, den der Regler erreichen will. „measurement" ist der aktuelle Wert in der realen Welt. https://www.thebackshed.com/forum/ViewTopic.php?FID=16&TID=17263 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. Diese Base64-Kodierung oder -Dekodierung kodiert oder dekodiert die Daten in „in" 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 [, …]]) oder MIN(arg1 [, arg2 [, …]])	Gibt die größte oder kleinste Zahl in der Argumentliste zurück. Beachte, dass der Vergleich ein Fließkomma-Vergleich ist (ganzzahlige Argumente werden in Fließkommazahlen umgewandelt) und eine Fließkommazahl zurückgegeben wird.

MID$(string$, start) oder MID$(string$, start, nbr)	Gibt eine Teilzeichenfolge von „string$“ zurück, die bei „start“ beginnt und sich über „nbr“ Zeichen erstreckt. Das erste Zeichen in der Zeichenfolge ist die Zahl 1. Wenn „nbr” weggelassen wird, geht die zurückgegebene Zeichenkette bis zum Ende von „string$”.
OCT$(number [, chars])	Gibt eine Zeichenkette zurück, die die oktale (Basis 8) Darstellung von „number“ angibt. „Zeichen” ist optional und gibt an, wie viele Zeichen die Zeichenkette hat, wobei führende Zeichen durch Nullen aufgefüllt werden.
PEEK(BYTE addr%) oder PEEK(SHORT addr%) oder PEEK(WORD addr%) oder PEEK(INTEGER addr%) oder PEEK(FLOAT addr%) oder PEEK(VARADDR var) oder PEEK(VARHEADER var) oder PEEK(CFUNADDR cfun) oder PEEK(VAR var, ±offset) oder PEEK(VARTBL, ±Offset) oder PEEK(PROGMEM, ±Offset) PEEK(BP n%) PEEK(SP n%) PEEK(WP n%)	Hinweis: Adressen werden abgerundet, um dem angeforderten Datentyp zu entsprechen (z. B. PEEK(SHORT) oder es kommt zu einem Fehler, wenn sie nicht ausgerichtet sind, z. B. PEEK(SP Gibt ein Byte oder ein Wort innerhalb des virtuellen Speicherbereichs des Prozessors zurück. BYTE gibt das Byte (8 Bit) zurück, das sich an „addr%” befindet. SHORT gibt die kurze Ganzzahl (16 Bit) an der Adresse „addr%” zurück WORD gibt das Wort (32 Bit) zurück, das sich an „addr%” befindet INTEGER gibt die Ganzzahl (64 Bit) an der Adresse „addr%“ zurück. FLOAT gibt die Gleitkommazahl (64 Bit) an der Adresse „addr%“ zurück. VARADDR gibt die Adresse (32 Bit) der Variablen „var” im Speicher zurück. Ein Array wird als var() angegeben. VARHEADER gibt die Adresse (32 Bit) des Headers der Variablen „var” im Speicher zurück. Ein Array wird als var() angegeben. Dies ist die Adresse des ersten Bytes des Variablennamens. CFUNADDR gibt die Adresse (32 Bit) der CFunktion „cfun” im Speicher zurück. Diese Adresse kann an eine andere CFunktion übergeben werden, die sie dann aufrufen kann, um einen gemeinsamen Prozess auszuführen. VAR gibt ein Byte im Speicher zurück, das 'var' zugewiesen ist. Ein Array wird als var() angegeben. VARTBL gibt ein Byte im Speicher zurück, das der von MMBasic verwalteten Variablentabelle zugewiesen ist. Beachte, dass nach dem Schlüsselwort VARTBL ein Komma steht. PROGMEM gibt ein Byte im Speicher zurück, das dem Programm zugewiesen ist. Beachte, dass nach dem Schlüsselwort PROGMEM ein Komma steht. Beachte, dass „addr%” eine ganze Zahl sein sollte. PEEK(bp n%) ' gibt das Byte an der Adresse n% zurück und erhöht n%, um auf das nächste Byte zu zeigen. PEEK(sp n%) gibt das Short an der Adresse n% zurück und erhöht n%, um auf das nächste Short zu zeigen. 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 E/A-Pin zurück. Null heißt digital niedrig, 1 heißt digital hoch, und bei analogen Eingängen wird die gemessene Spannung als Fließkommazahl zurückgegeben. Frequenzeingänge geben die Frequenz in Hz zurück. Ein Periodeneingang gibt die Periode in Millisekunden zurück, während ein Zähleingang die Anzahl seit dem Zurücksetzen zurückgibt (die Zählung erfolgt an der positiven Flanke). Der Zähleingang kann durch Zurücksetzen des Pins auf Zähleingang auf Null zurückgesetzt werden (auch wenn er bereits so konfiguriert ist). Wenn ein Pin als Ausgang konfiguriert ist, gibt diese Funktion den Wert der Ausgangseinstellung zurück (d. h. hoch oder niedrig). Siehe auch die Befehle SETPIN und PIN() =. Eine allgemeine Beschreibung der Ein-/Ausgabefunktionen des PicoMite findest du im Kapitel *„ Verwendung der E/A-Pins*“.
PIN(BOOTSEL)	Gibt den Status des Boot-Auswahlschalters zurück, sodass er als Benutzereingabe in einem Programm verwendet werden kann.
PIN(TEMP)	Gibt die Temperatur des RP2040/RP2350-Chips zurück (Details findest du im Datenblatt).
PIO(DMA RX POINTER) 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)	Aus einem PIO-FIFO lesen „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 den angegebenen PIO zurück

PIO (FLEVEL pio)	gibt den Wert des FLEVEL-Registers für das angegebene pio zurück PIO(FLEVEL pio)
PIO(FLEVEL pio ,sm, DIR)	dir kann RX oder TX sein. Gibt den Pegel des bestimmten FIFO zurück
PIO(.WRAP) PIO(.WRAP TARGET)	Gibt die Position der .wrap-Direktive in PIO ASSEMBLE zurück Gibt die Position der .wrap-Zielanweisung in PIO ASSEMBLE zurück. Diese können in der Funktion PIO(EXECCTRL wie folgt verwendet werden: PIO (EXECCTRL jmp_pin PIO(.WRAP TARGET), PIO(.WRAP) [,side_pindir] [,side_en])
PIO(NEXT LINE)	Gibt den nächsten ungenutzten PIO-Befehlsslot nach einem mit END PROGRAM beendeten PIO-Befehlsblock 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. 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 E/A-Pins in einem Vorgang zurück. „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 der Bits „nbr". Die verwendeten E/A-Pins müssen fortlaufend nummeriert sein, und jeder E/A-Pin, der ungültig oder nicht als Eingang konfiguriert ist, führt zu einem Fehler. Das Start-/Anzahl-Paar kann bis zu 25 Mal wiederholt werden, wenn zusätzliche Gruppen von Eingangspins hinzugefügt werden müssen. Diese Funktion gibt auch den Ausgangsstatus eines als Ausgang konfigurierten Pins zurück. Dies kann zur bequemen Kommunikation mit parallelen Geräten wie Speicherchips verwendet werden. Es kann eine beliebige Anzahl von E/A-Pins (und damit Bits) von 1 bis zur Anzahl der E/A-Pins auf dem Chip verwendet werden. Hinweis: Wenn die Pins mit der GPn-Syntax definiert sind, ignoriert die Firmware ungültige Pins, sodass `PORT(GP0, 8)` acht I/O-Pins liest: GP0, GP1, GP2, GP3, GP4, GP5, GP6 und GP7. Siehe den Befehl PORT, um gleichzeitig an mehrere Pins auszugeben.
PULSIN(Pin, Polarität) oder PULSIN(Pin, Polarität, t1) oder PULSIN(Pin, Polarität, t1, t2)	Misst die Breite eines Eingangsimpulses von 1 µs bis 1 Sekunde mit einer Auflösung von 0,1 µs. „Pin" ist der für die Messung zu verwendende E/A-Pin, der zuvor als digitaler Eingang konfiguriert werden muss. „Polarität" ist der zu messende Impulstyp. Ist der Wert Null, gibt die Funktion die Breite des nächsten negativen Impulses zurück, ist er ungleich Null, wird der nächste positive Impuls gemessen. „t1" ist die Zeitüberschreitung, die beim Warten auf den Impuls angewendet wird, „t2" ist die Zeitüberschreitung, die beim Messen des Impulses verwendet wird. Beide sind in Mikrosekunden (µs) angegeben und optional. Wenn „t2" weggelassen wird, wird der Wert von „t1" für beide Zeitüberschreitungen verwendet. Wenn sowohl „t1" als auch „t2" weggelassen werden, werden die Zeitüberschreitungen auf 100000 (d. h. 100 ms) gesetzt. Diese Funktion gibt die Breite des Impulses in Mikrosekunden (µs) zurück oder -1, wenn eine Zeitüberschreitung aufgetreten ist. Die Messung ist auf ±0,5 % und ±0,5 µs genau.

	Beachte, dass diese Funktion das laufende Programm während der Messung pausiert und Interrupts während dieser Zeit ignoriert werden.
RAD(Grad)	Rechnet „Grad" in Bogenmaß um.
RGB (Rot, Grün, Blau) oder RGB(Shortcut)	Macht einen echten RGB-Farbwert. „Rot", „Blau" und „Grün" zeigen die Intensität jeder Farbe an. Der Wert Null steht für Schwarz und 255 für volle Intensität. Mit „Abkürzung" kannst du gängige Farben durch ihre Bezeichnung angeben. Die Farben, die du benennen kannst, sind Weiß, Schwarz, Blau, Grün, Cyan, Rot, Magenta, Gelb, Braun, Weiß, Orange, Rosa, Gold, Lachs, Beige, Hellgrau und Grau (oder die US-amerikanische Schreibweise Gray/Lightgray). Zum Beispiel RGB(rot) oder RGB(cyan).
RIGHT$(string$, Anzahl-der-Zeichen)	Gibt eine Teilzeichenfolge von „string$" mit „number-of-chars" Zeichen von rechts (Ende) der Zeichenfolge zurück.
RND(Zahl) oder RND	Gibt eine Zufallszahl (RP2350) oder Pseudozufallszahl (RP2040) im Bereich von 0 bis 0,999999 zurück. Der Wert „number" wird ignoriert, wenn er angegeben wird. Schau dir auch den Befehl RANDOMIZE an (nur RP2040).
SGN(Zahl)	Gibt das Vorzeichen des Arguments „number" zurück: +1 für positive Zahlen, 0 für 0 und -1 für negative Zahlen.
SIN(Zahl)	Gibt den Sinus des Arguments „number" in Radianten zurück.
SPACE$(Zahl)	Gibt eine Zeichenfolge mit Leerzeichen zurück, die „Zahl" Zeichen lang ist.
SPI (data) oder SPI2 (data)	Sendet und empfängt Daten über einen SPI-Kanal. Bei einer einzelnen SPI-Transaktion werden Daten gesendet und gleichzeitig Daten vom Slave empfangen. „data" sind die zu sendenden Daten, und die Funktion gibt die während der Transaktion empfangenen Daten zurück. „data" können eine Ganzzahl, eine Gleitkommavariable oder eine Konstante sein.
SPRITE()	NUR VGA- UND HDMI-VERSIONEN Die SPRITE-Funktionen geben Infos zu Sprites zurück, das sind kleine Grafiken auf dem VGA/HDMI-Bildschirm. Die sind nützlich, wenn du Spiele programmierst. Sieh dir auch die SPRITE-Befehle an.
SPRITE(C, [#]n)	Gibt die Anzahl der aktuell aktiven Kollisionen für Sprite n zurück. Wenn n=0 ist, gibt es die Anzahl der Sprites zurück, die nach einem SPRITE SCROLL-Befehl eine aktuell aktive Kollision haben.
SPRITE(C, [#]n, m)	Gibt die Nummer des Sprites zurück, das die m-te Kollision von Sprite n verursacht hat. Wenn n=0 ist, gibt es die Sprite-Nummer des m-ten Sprites zurück, das nach einem SPRITE SCROLL-Befehl eine aktive Kollision hat. Wenn die Kollision mit dem Rand des Bildschirms passiert ist, ist der Rückgabewert: &HF1 Kollision mit der linken Bildschirmseite &HF2 Kollision mit dem oberen Bildschirmrand &HF4 Kollision mit der rechten Seite des Bildschirms &HF8 Kollision mit dem unteren Bildschirmrand

SPRITE(D ,[#]s1, [#]s2)	Gibt den Abstand zwischen den Mittelpunkten der Sprites „s1" und „s2" zurück (gibt -1 zurück, wenn eines der Sprites nicht aktiv ist).
SPRITE(E, [#]n)	Gibt eine Bitmap zurück, die alle Kanten des Bildschirms anzeigt, mit denen das Sprite in Kontakt ist: 1 = linke Bildschirmseite, 2 = obere Bildschirmseite, 4 = rechte Bildschirmseite, 8 = untere Bildschirmseite
SPRITE(H,[#]n)	Gibt die Höhe von Sprite n zurück. Diese Funktion ist aktiv, egal ob das Sprite gerade angezeigt wird (aktiv) oder nicht.
SPRITE(L, [#]n)	Gibt die Layernummer des aktiven Sprites Nummer n zurück.
SPRITE(N)	Gibt die Anzahl der angezeigten (aktiven) Sprites zurück.
SPRITE(N,n)	Zeigt die Anzahl der Sprites an, die auf Ebene n angezeigt werden.
SPRITE(S)	Gibt die Nummer des Sprites zurück, das zuletzt eine Kollision verursacht hat. Hinweis: Wenn die zurückgegebene Nummer Null ist, dann ist die Kollision das Ergebnis eines SPRITE SCROLL-Befehls und die Funktion SPRITE(C…) sollte verwendet werden, um herauszufinden, wie viele und welche Sprites kollidiert sind.
SPRITE(T, [#]n)	Gibt eine Bitmap zurück, die alle Sprites zeigt, die gerade das angeforderte Sprite berühren. Die Bits 0-63 in der zurückgegebenen Ganzzahl zeigen eine aktuelle Kollision mit den Sprites 1 bis 64 an.
SPRITE(V,[#]s1, [#]s2)	Gibt den Vektor vom Sprite „s1" zum Sprite „s2" in Radianten zurück. Der Winkel basiert auf der Uhr, wenn also „s2" auf dem Bildschirm über „s1" liegt, ist das Ergebnis Null. Das kann für jedes Paar sichtbarer Sprites verwendet werden. Wenn eines der Sprites nicht sichtbar ist, gibt die Funktion -1 zurück. Das ist besonders nützlich nach einer Kollision, wenn der Programmierer eine differenzierte Entscheidung treffen will, je nachdem, wo die Kollision passiert ist. Der Winkel wird zwischen den Mittelpunkten der Sprites berechnet, die natürlich unterschiedlich groß sein können.
SPRITE(W, [#]n)	Gibt die Breite von Sprite n zurück. Diese Funktion ist aktiv, unabhängig davon, ob das Sprite gerade angezeigt wird (aktiv) oder nicht.
SPRITE(X, [#]n)	Gibt die X-Koordinate von Sprite n zurück. Diese Funktion ist nur aktiv, wenn das Sprite gerade angezeigt wird (aktiv). Andernfalls gibt sie 10000 zurück.
SPRITE(Y, [#]n)	Gibt die Y-Koordinate von Sprite n zurück. Diese Funktion ist nur aktiv, wenn das Sprite gerade angezeigt wird (aktiv). Sonst gibt sie 10000 zurück.
SQR(Zahl)	Gibt die Quadratwurzel des Arguments „number" zurück.

STR$(Zahl) oder STR$(Zahl, m) oder STR$(Zahl, m, n) oder STR$(Zahl, m, n, c$)	Gibt eine Zeichenfolge in der Dezimaldarstellung (Basis 10) von „number“ zurück. Wenn 'm' angegeben ist, werden am Anfang der Zahl genügend Leerzeichen eingefügt, damit die Anzahl der Zeichen vor dem Dezimalpunkt (einschließlich des Vorzeichens) mindestens 'm' Zeichen beträgt. Wenn 'm' gleich Null ist oder die Zahl mehr als 'm' signifikante Stellen hat, werden keine Leerzeichen eingefügt. Wenn 'm' negativ ist, werden positive Zahlen mit einem Pluszeichen und negative Zahlen mit einem Minuszeichen versehen. Wenn 'm' positiv ist, wird nur das Minuszeichen verwendet. „n” ist die Anzahl der Stellen, die nach dem Dezimalpunkt stehen müssen. Wenn es Null ist, wird die Zeichenfolge ohne Dezimalpunkt zurückgegeben. Wenn es negativ ist, wird die Ausgabe immer das Exponentialformat mit einer Auflösung von „n” Stellen verwenden. Wenn „n” nicht angegeben ist, variieren die Anzahl der Dezimalstellen und das Ausgabeformat automatisch je nach Zahl. „c$“ ist eine Zeichenfolge. Wenn sie angegeben wird, wird das erste Zeichen dieser Zeichenfolge anstelle eines Leerzeichens als Auffüllzeichen verwendet (siehe Argument „m“). Beispiele für „ “: STR$(123.456) gibt „123.456” zurück STR$(-123.456) gibt „-123,456“ zurück STR$(123.456, 1) gibt „123,456” zurück STR$(123,456, -1) gibt „+123,456” zurück STR$(123,456, 6) gibt „ 123,456” zurück STR$(123,456, -6) gibt „+123,456” zurück STR$(-123,456, 6) gibt „ -123,456” zurück STR$(-123,456, 6, 5) gibt „ -123,45600” zurück STR$(-123,456, 6, -5) gibt „ -1,23456e+02” zurück STR$(53, 6) gibt „ 53” zurück STR$(53, 6, 2) gibt „ 53,00” STR$(53, 6, 2, "*") gibt „****53.00” zurück
STR2BIN(Typ, Zeichenfolge$ [,BIG])	Gibt eine Zahl zurück, die der binären Darstellung in „string$” entspricht. „type” kann sein: INT64 macht aus einer 8-Byte-Zeichenkette, die eine vorzeichenbehaftete 64-Bit-Ganzzahl darstellt, eine Ganzzahl UINT64 macht aus einer 8-Byte-Zeichenkette, die eine vorzeichenlose 64-Bit-Ganzzahl ist, eine Ganzzahl INT32 macht aus einer 4-Byte-Zeichenkette, die eine vorzeichenbehaftete 32-Bit-Ganzzahl ist, eine Ganzzahl UINT32 macht aus einer 4-Byte-Zeichenkette, die eine vorzeichenlose 32-Bit-Ganzzahl ist, eine Ganzzahl INT16 macht aus einer 2-Byte-Zeichenkette, die eine vorzeichenbehaftete 16-Bit-Ganzzahl ist, eine Ganzzahl UINT16 macht aus einer 2-Byte-Zeichenkette, die eine vorzeichenlose 16-Bit-Ganzzahl ist, eine Ganzzahl INT8 macht aus einer 1-Byte-Zeichenkette, die eine vorzeichenbehaftete 8-Bit-Ganzzahl ist, eine Ganzzahl UINT8 macht aus einer 1-Byte-Zeichenkette, die eine vorzeichenlose 8-Bit-Ganzzahl ist, eine Ganzzahl SINGLE macht aus einer 4-Byte-Zeichenkette, die eine Gleitkommazahl mit einfacher Genauigkeit ist, eine Gleitkommazahl DOUBLE macht aus einer 8-Byte-Zeichenkette, die eine Gleitkommazahl mit einfacher Genauigkeit ist, eine Gleitkommazahl

	Standardmäßig muss die Zeichenfolge die Zahl im Little-Endian-Format enthalten (d. h. das niedrigstwertige Byte ist das erste in der Zeichenfolge). Wenn du den dritten Parameter auf „BIG" setzt, wird die Zeichenfolge im Big-Endian-Format interpretiert (d. h. das Byte das erste in der Zeichenkette ist). Diese Funktion macht es einfach, Daten aus Binärdateien zu lesen, Zahlen von Sensoren zu verstehen Zahlen von Sensoren zu interpretieren oder Binärdaten effizient aus Flash-Speicherchips zu lesen . Wenn die Zeichenfolge nicht die richtige Länge für die gewünschte Konvertierung hat, wird ein Fehler ausgegeben. gewünschte Konvertierung Siehe auch die Funktion BIN2STR$
STRING$(nbr, ascii) oder STRING$(nbr, string$)	Gibt eine Zeichenfolge mit einer Länge von „nbr" Bytes zurück, die entweder aus dem ersten Zeichen von string$ oder aus dem Zeichen besteht, das den ASCII-Wert „ascii" darstellt, bei dem es sich um eine ganze Zahl oder eine Gleitkommazahl im Bereich von 0 bis 255 handelt.
STRUCT(arg, …)	Gibt Daten aus einer Struktur zurück (auch als benutzerdefinierter Typ bekannt), die es ermöglicht, zusammengehörige Variablen unterschiedlicher Typen unter einem einzigen Namen zu gruppieren. Das wird ausführlich in der Datei *„MMBasic_Structures_Manual.pdf"* beschrieben, die im ZIP-Archiv zum Herunterladen der Firmware enthalten ist.
TAB(Nummer)	Gibt Leerzeichen aus, bis die durch „Nummer" angegebene Spalte in der Konsolenausgabe erreicht ist.
TAN(Zahl)	Gibt den Tangens des Arguments „number" in Radianten zurück.
TEMPR(Pin [,Timeout])	Gibt die Temperatur zurück, die von einem DS18B20-Temperatursensor gemessen wurde, der an „Pin" angeschlossen ist (der nicht konfiguriert werden muss). Der zurückgegebene Wert ist in Grad Celsius mit einer Standardauflösung von 0,25 °C. Wenn während der Messung ein Fehler auftritt, ist der zurückgegebene Wert 1000. Die für die gesamte Messung benötigte Zeit beträgt 200 ms, und Interrupts werden während dieses Zeitraums ignoriert. Der optionale Parameter „timeout" kann verwendet werden, um den Standardwert (200 ms) zu überschreiben und langsamere Geräte zu berücksichtigen. Alternativ kann der Befehl TEMPR START verwendet werden, um die Messung zu starten, und dein Programm kann während der Konvertierung andere Aufgaben ausführen. Wenn diese Funktion aufgerufen wird, wird der Wert sofort zurückgegeben, vorausgesetzt, die Konvertierungszeit ist abgelaufen. Ist dies nicht der Fall, wartet diese Funktion die restliche Konvertierungszeit ab, bevor sie den Wert zurückgibt. Der DS18B20 kann separat mit einer 3-V- bis 5-V-Stromversorgung betrieben werden oder mit parasitärer Energie vom Raspberry Pi Pico. Weitere Infos findest du im Kapitel *„Spezielle Hardwaregeräte*".

TIME$	Gibt die aktuelle Uhrzeit basierend auf der internen Uhr von MMBasic als Zeichenfolge im Format „HH:MM:SS" in 24-Stunden-Notation zurück. Zum Beispiel „14:30:00". Um die aktuelle Uhrzeit einzustellen, benutze den Befehl TIME$ = .
TIMER	Gibt die seit dem Zurücksetzen verstrichene Zeit in Millisekunden (z. B. 1/1000 einer Sekunde) zurück. Der Timer wird beim Einschalten oder bei einem Neustart der CPU auf Null zurückgesetzt. Du kannst ihn auch mit dem Befehl TIMER zurücksetzen. Wenn er nicht explizit zurückgesetzt wird, zählt er unendlich weiter (es handelt sich um eine 64-Bit-Zahl, die erst nach 200 Millionen Jahren wieder auf Null zurücksetzt).
TOUCH(X) oder TOUCH(Y oder nur FT6336 TOUCH(X2) oder TOUCH(Y2)	Gibt die X- oder Y-Koordinate der Stelle zurück, die gerade auf einem LCD-Bildschirm berührt wird. Wenn der Bildschirm nicht berührt wird, gibt die Funktion -1 zurück. Für den FT6336 geben TOUCH(X2) und TOUCH(Y2) die Position einer zweiten Berührungsstelle zurück oder -1, wenn keine zweite Stelle berührt wird.
TRIM$(Quelle$ [,Maske$] [,wo/wo$])	Diese Funktion kann Zeichen am Anfang oder Ende einer Zeichenfolge oder an beiden Stellen entfernen. „source$" ist die Eingabezeichenfolge „mask$" ist eine Zeichenkette, die eine Liste der zu entfernenden Zeichen enthält. Wenn sie weggelassen wird, ist der Standardwert ein Leerzeichen. „where/where$" kann L, R oder B sein oder eine Zeichenkette, die mit L, R oder B anfängt, um angeben, welche Zeichen links von der Quelle, rechts von der Quelle oder an beiden Stellen entfernt werden sollen. Wenn das weggelassen wird, ist die Standardeinstellung L.
UCASE$(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. Diese Funktion erkennt das Präfix &H für eine Hexadezimalzahl, &O für eine Oktalzahl und &B für eine Binärzahl.

Veraltete Befehle und Funktionen

Diese Befehle und Funktionen sind hauptsächlich dazu da, um bei der Konvertierung von Programmen zu helfen, 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 zum Befehl GAMEPAD geändert.
DEVICE HUMID	In den Befehl HUMID geändert
DEVICE KEYPAD	In den Befehl KEYPAD geändert
DEVICE MOUSE	Jetzt heißt es MOUSE
GERÄT LCD	In den Befehl LCD geändert
DEVICE WII	In den Befehl WII geändert
DEVICE WS2812	Geändert zu WS2812-Befehl
GOSUB-Ziel	Startet einen Unterprogrammaufruf zum Ziel, das eine Zeilennummer oder eine Bezeichnung sein kann. Das Unterprogramm muss mit RETURN enden. Neue Programme sollten definierte Unterprogramme verwenden (d. h. SUB… END SUB).
IF-Bedingung THEN Zeilenumbruch	Aus Gründen der Kompatibilität mit Microsoft wird ein GOTO angenommen, wenn auf die THEN-Anweisung eine Zahl folgt. Ein Label ist in dieser Konstruktion ungültig. Neue Programme sollten Folgendes verwenden: IF-Bedingung THEN GOTO Zeilenumbruch | Label
IRETURN	Kehrt von einer Unterbrechung zurück, wenn das Ziel der Unterbrechung eine Zeilennummer oder ein Label war. 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 Unterroutine auf (GOSUB), basierend auf dem gerundeten Wert von „nbr”; wenn dieser 1 ist, wird das erste Ziel aufgerufen, wenn 2, das zweite Ziel usw. Das Ziel kann eine Zeilennummer oder eine Bezeichnung sein. Neue Programme sollten SELECT CASE verwenden.
POS	Gibt für die Konsole die aktuelle Cursorposition in der Zeile in Zeichen zurück.
RETURN	RETURN beendet eine mit GOSUB aufgerufene Unterroutine 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.

E/A-Pins

Bevor eine serielle Schnittstelle genutzt 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 ist Standard für Geräte, die mit TTL-Spannungen laufen. Im Leerlauf ist die Spannung hoch, das Startbit ist eine niedrige Spannung, Daten verwenden eine hohe Spannung für Logik 1 und das Stoppbit ist eine hohe Spannung. Mit diesen Signalpegeln kannst du Geräte wie GPS-Module (die normalerweise TTL-Spannungspegel verwenden) direkt anschließen.

Befehle

Nach dem Öffnen hat die serielle Schnittstelle eine zugeordnete Dateinummer, und du kannst alle Befehle verwenden, die mit einer Dateinummer arbeiten, um Daten zu lesen und zu schreiben. Eine serielle Schnittstelle kann mit dem Befehl CLOSE geschlossen werden.

Hier ein Beispiel:

```
SETPIN GP13, GP16, COM1     ' Weisen Sie die E/A-Pins für die erste serielle
Schnittstelle zu.
OPEN „COM1:4800” AS #5      ' Öffne den ersten seriellen Anschluss mit einer
Geschwindigkeit von 4800 Baud
PRINT #5, „Hallo”           ' Schick die Zeichenfolge „Hallo” über den seriellen
Port raus
dat$ = INPUT$(20, #5)       ' bis zu 20 Zeichen vom seriellen Port holen
CLOSE #5                    ' Schließ die serielle Schnittstelle
```

Der Befehl OPEN

Eine serielle Schnittstelle wird mit dem Befehl geöffnet:

```
OPEN comspec$ AS #fnbr
```

„fnbr” ist die zu verwendende Dateinummer. Sie muss zwischen 1 und 10 liegen. Das # ist optional.

„comspec$“ ist die Kommunikationsspezifikation und eine Zeichenfolge (kann eine Zeichenfolgenvariable sein), die die zu öffnende serielle Schnittstelle 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"` Dabei gilt:

- „n” die Nummer der seriellen Schnittstelle für entweder COM1: oder COM2: ist.
- „baud” die Baudrate ist. Diese kann zwischen 1200 und 921600 liegen. Der Standardwert ist 9600.
- „buf” die Größe des Empfangspuffers in Byte ist (Standardgröße ist 256). Der Sendepuffer ist auf 256 Byte festgelegt.
- „int“ ist die Interrupt-Subroutine, die aufgerufen wird, wenn der serielle Anschluss 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 angewendet wird, was zu einer 9-Bit-Übertragung führt, sofern nicht 7BIT gesetzt ist.
- ODD legt fest, dass ein ungerades Paritätsbit angewendet wird, was zu einer 9-Bit-Übertragung führt, es sei denn, 7BIT ist gesetzt.
- 7BIT legt fest, dass 7 Datenbits vorhanden sind. Dies wird normalerweise zusammen mit EVEN oder ODD verwendet.
- INV bedeutet, dass die Ausgangssignale invertiert werden und die Eingabe als invertiert angenommen wird.

Beispiele

Öffnen einer seriellen Schnittstelle mit allen Standardeinstellungen:

```
ÖFFNE „COM1:“ ALS #2
```

Öffnen einer seriellen Schnittstelle, wobei nur die Baudrate (4800 Bit pro Sekunde) angegeben wird:

```
OPEN „COM1:4800” AS #1
```

Öffnen eines seriellen Anschlusses mit Angabe der Baudrate (9600 Bit pro Sekunde) und der Empfangspuffergröße (1 KB):

```
OPEN "COM2:9600, 1024" AS #8
```

Wie oben, aber mit zwei Stoppbits:

```
OPEN "COM2:9600, 1024, S2" AS #8
```

Ein Beispiel, bei dem alles angegeben wird, einschließlich einer Unterbrechung, einer Unterbrechungsstufe und zwei Stoppbits:

```
OPEN "COM2:19200, 1024, ComIntLabel, 256, S2" AS #5
```

Lesen und Schreiben

Sobald eine serielle Schnittstelle geöffnet ist, kannst du jeden Befehl oder jede Funktion verwenden, die eine Dateinummer zum Lesen und Schreiben auf die Schnittstelle nutzt. Daten, die über die serielle Schnittstelle empfangen werden, werden von MMBasic automatisch im Speicher gepuffert, bis sie vom Programm gelesen werden. Die Funktion INPUT$() ist dafür am besten geeignet. Bei Verwendung der Funktion INPUT$() entspricht die angegebene Zeichenanzahl der maximalen Anzahl der zurückgegebenen Zeichen, kann jedoch geringer sein, wenn sich weniger Zeichen im Empfangspuffer befinden. Tatsächlich gibt die Funktion INPUT$ () sofort eine leere Zeichenfolge zurück, wenn im Empfangspuffer keine Zeichen verfügbar sind.

Die Funktion LOC() ist auch praktisch; sie gibt die Anzahl der Zeichen zurück, die im Empfangspuffer warten (d. h. die maximale Anzahl von Zeichen, die von der Funktion INPUT$() abgerufen werden können). Beachte, dass der serielle Port automatisch die ältesten Daten löscht, um Platz für neue Daten zu schaffen, wenn der Empfangspuffer mit eingehenden Daten überläuft.

Der Befehl PRINT wird für die Ausgabe an eine serielle Schnittstelle verwendet, und alle zu sendenden Daten werden in einem Speicherpuffer gehalten, während die serielle Schnittstelle sie sendet. Das heißt, 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 Kapazität im Sendepuffer zurück. Damit kannst du vermeiden, dass das Programm ins Stocken gerät, während es darauf wartet, dass im Puffer wieder Platz verfügbar wird.

Wenn du sicher sein willst, dass alle Daten gesendet wurden (vielleicht weil du die Antwort vom Remote-Gerät lesen willst), 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 kannst du mit dem Befehl CLOSE schließen. Dadurch wird gewartet, bis der Sendepuffer leer ist, dann wird der von den Puffern belegte Speicher freigegeben und der Interrupt (falls gesetzt) abgebrochen. 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 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 eingegangen sein können, insbesondere bei hohen Baudraten. Wenn du zum Beispiel die Interrupt-Stufe 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 solltest du den Puffer auf 512 Zeichen oder mehr erhöhen.

Anhang B
I²C-Kommunikation

Es gibt zwei I²C-Kanäle. Die können im Master- oder Slave-Modus laufen.

I/O-Pins

Bevor die I²C-Schnittstelle genutzt werden kann, musst du die I/O-Pins mit dem folgenden Befehl für den ersten Kanal (als I2C bezeichnet) definieren:

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 auf GP28 nicht verfügbar ist

Und der folgende Befehl für den zweiten Kanal (als I2C2 bezeichnet):

SETPIN sda, scl, I2C2

Gültige Pins sind SDA: GP2, GP6, GP10, GP14, GP18, GP22 oder GP26
SCL: GP3, GP7, GP11, GP15, GP19 oder GP27

Wenn der I²C-Bus mit mehr als 100 kHz läuft, ist die Verkabelung zwischen den Geräten echt wichtig. Am besten sollten die Kabel so kurz wie möglich sein (um die Kapazität zu verringern) und die Daten- und Taktleitungen sollten nicht nebeneinander verlaufen, sondern durch ein Erdungskabel voneinander getrennt sein (um Übersprechen zu reduzieren).

Wenn die Datenleitung bei hohem Takt nicht stabil ist oder die Taktleitung schwankend ist, können die I²C-Peripheriegeräte „verwirrt" werden und den Bus sperren (normalerweise durch Halten der Taktleitung auf niedrigem Niveau). Wenn du keine höheren Geschwindigkeiten benötigst, ist der Betrieb mit 100 kHz die sicherste Wahl.

Mit dem Befehl I2C CHECK addr kannst du überprüfen, ob ein Gerät an der Adresse „addr" 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 verwendet werden können, wie folgt.

Die Befehle für den zweiten Kanal (I2C2) sind identisch, außer dass der Befehl I2C2 lautet.

I2C OPEN Geschwindigkeit, Zeitüberschreitung

Aktiviert das I²C-Modul im Master-Modus. Der Befehl I2C bezieht sich auf Kanal 1, während der Befehl I2C2 sich mit derselben Syntax auf Kanal 2 bezieht.

„speed" ist die zu verwendende Taktrate (in KHz) und muss entweder 100, 400 oder 1000 sein.

„timeout" ist ein Wert in Millisekunden, nach dem die Master-Sende- und Empfangskommandos unterbrochen werden, wenn sie nicht abgeschlossen sind. Der Mindestwert ist 100. Ein Wert von Null deaktiviert das Timeout (dies wird jedoch nicht empfohlen).

I2C WRITE addr, option, sendlen, senddata [,sendata ..]

Sendet Daten an das I²C-Slave-Gerät. 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 die Kontrolle über den Bus nach dem Befehl zu behalten (eine Stoppbedingung wird nach Abschluss des Befehls nicht gesendet).

„sendlen" ist die Anzahl der zu sendenden Bytes.

„senddata" sind die zu sendenden Daten – diese können auf verschiedene Weise angegeben werden (alle 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 angegeben werden, das mit leeren Klammern (d. h. ohne Dimensionen) angegeben wird. „sendlen“ Bytes des Arrays werden beginnend mit dem ersten Element gesendet. Beispiel: I2C WRITE &H6F, 0, 3, ARRAY()
- Die Daten können eine String-Variable sein (keine Konstante). Beispiel: I2C WRITE &H6F, 0, 3, STRING$

I2C READ addr, option, rcvlen, rcvbuf

Ruft Daten vom I²C-Slave-Gerät ab. 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 die Kontrolle über den Bus nach dem Befehl zu behalten (eine Stoppbedingung wird nach Abschluss des Befehls nicht gesendet).

„rcvlen” ist die Anzahl der zu empfangenden Bytes.

„rcvbuf“ ist die Variable oder das Array, das zum Speichern der empfangenen Daten verwendet wird – dies kann sein:

- Eine Zeichenfolgenvariable. Die Bytes werden als aufeinanderfolgende Zeichen in der Zeichenfolge gespeichert.
- Ein eindimensionales Array von Zahlen, das mit leeren Klammern angegeben wird. 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 E/A-Pins wieder in den Zustand „nicht konfiguriert”. Dieser Befehl sendet auch ein Stopp-Signal, wenn der Bus noch gehalten wird.

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 Slave-I²C-Adresse.

„send_int” ist die Subroutine, die aufgerufen wird, wenn das Modul erkennt, 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 und der Befehl aus der Hauptprogrammschleife aufgerufen werden, wenn das Flag gesetzt ist.

„sendlen” ist die Anzahl der zu sendenden Bytes.

„senddata” sind die zu sendenden Daten. Diese können auf verschiedene Weise angegeben werden, siehe die I2C-WRITE-Befehle für Details.

I2C SLAVE READ rcvlen, rcvbuf, rcvd	Empfängt 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. 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 und der Befehl aus der Hauptprogrammschleife aufgerufen werden, wenn das Flag gesetzt ist. „rcvlen” ist die maximale Anzahl von Bytes, die empfangen werden sollen. „rcvbuf” ist die Variable, die die Daten empfängt. Diese kann auf verschiedene Weise angegeben werden, siehe die I2C-READ-Befehle für Details. „rcvd” ist eine Variable, die nach Abschluss des Befehls die tatsächlich empfangene Anzahl von Bytes enthält (die von „rcvlen” abweichen kann).
I2C SLAVE CLOSE	Deaktiviert das Slave-I²-C-Modul und setzt die externen E/A-Pins wieder auf „nicht konfiguriert”. Sie können dann mit SETPIN konfiguriert werden.

Fehler

Nach einem I²C-Schreib- oder Lesevorgang wird die automatische Variable MM.I2C gesetzt, um das Ergebnis wie folgt anzuzeigen:

0 = Der Befehl wurde ohne Fehler abgeschlossen.
1 = NACK-Antwort erhalten
2 = Befehl abgelaufen

7-Bit-Adressierung

Die Standardadressen, die in diesen Befehlen benutzt werden, sind 7-Bit-Adressen (ohne Lese-/Schreibbit). MMBasic fügt das Lese-/Schreibbit hinzu und passt es während der Übertragungen entsprechend an.

Einige Anbieter stellen 8-Bit-Adressen zur Verfügung, die das Lese-/Schreibbit enthalten. Du kannst feststellen, ob das der Fall ist, weil sie eine Adresse zum Schreiben auf das Slave-Gerät und eine andere zum Lesen vom Slave bereitstellen. In diesen Fällen solltest du nur die oberen sieben Bits der Adresse verwenden. Beispiel: Wenn die Leseadresse 9B (hex) und die Schreibadresse 9A (hex) ist, erhältst du durch Verwendung nur der oberen sieben Bits die Adresse 4D (hex).

Ein weiterer Hinweis darauf, dass ein Anbieter 8-Bit-Adressen anstelle von 7-Bit-Adressen verwendet, ist die Überprüfung des Adressbereichs. Alle 7-Bit-Adressen sollten im Bereich von 08 bis 77 (hex) liegen. Wenn deine Slave-Adresse größer als dieser Bereich ist, hat dein Anbieter wahrscheinlich eine 8-Bit-Adresse bereitgestellt.

Beispiele

Als Beispiel für eine einfache Kommunikation, bei der der Raspberry Pi Pico der Master ist, liest und zeigt das folgende Programm die aktuelle Uhrzeit (Stunden und Minuten) an, die von einem PCF8563-Echtzeituhr-Chip gespeichert wird, der an den zweiten I2C-Kanal angeschlossen ist:

```
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                 ' Öffne den I2C-Kanal
I2C2 WRITE &H51, 0, 1, 3            ' Setze das erste Register auf 3
I2C2 READ &H51, 0, 2, RData()       ' zwei Register lesen
I2C2 CLOSE                          ' I2C-Kanal schließen
PRINT "Die 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 Anfrage an den Slave und gibt die Antwort aus. Der Slave liest das vom Master gesendete Byte und gibt es aus. Als Antwort auf die Anfrage 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$
  a$ ausgeben
  PAUSE 500
NECT i
```

Dann der Slave:

```
SETPIN GP2, GP3, I2C2
I2C2 SLAVE OPEN &H50, tint, rint
DO : LOOP

SUB rint
  LOCAL count, a$
  I2C2 SLAVE READ 10, a$, count
  PRINT LEFT$(a$, count)
END SUB

SUB tint
  LOKAL a$ = Zeit$
  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…]	Schickt eine Anzahl von Bytes
ONEWIRE READ Pin, Flag, Länge, Daten [, Daten…]	Ein paar Bytes holen

Wo:

Pin – Der zu verwendende E/A-Pin. Das kann jeder Pin sein, der digitale E/A unterstützt.

Flag – Eine Kombination der folgenden Optionen:

1 – Reset vor Befehl senden

2 – Reset nach Befehl senden

4 – Nur ein Bit statt eines Bytes an Daten senden/empfangen

8 – Starke Pullup-Funktion nach dem Befehl ausführen (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 Parameter „length" übereinstimmen.

Die automatische Variable MM.ONEWIRE gibt „true" zurück, wenn ein Gerät gefunden wurde

Nach der Ausführung des Befehls wird der I/O-Pin auf den nicht konfigurierten Zustand gesetzt, es sei denn, die Flag-Option 8 wird verwendet.

Wenn ein Reset angefordert wird, gibt die automatische Variable MM.ONEWIRE „true" zurück, wenn ein Gerät gefunden wurde. Das passiert beim Befehl ONEWIRE RESET und den Befehlen ONEWIRE READ und ONEWIRE WRITE, wenn ein Reset angefordert wurde (Flag = 1 oder 2).

Das 1-Wire-Protokoll wird oft für die Kommunikation mit dem Temperaturmesssensor DS18B20 verwendet. Um das zu unterstützen, hat MMBasic die Funktion TEMPR(), die eine bequeme Methode bietet, um die Temperatur eines DS18B20 direkt zu lesen, ohne diese Funktionen zu benutzen.

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 generiert 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 (bezeichnet als SPI) 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 (bezeichnet als SPI2) 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, musst du zuerst den SPI-Kanal öffnen.

Die Syntax zum Öffnen des ersten SPI-Kanals ist (für den zweiten Kanal SPI2 verwenden):

```
SPI OPEN Geschwindigkeit, Modus, Bits
```

Dabei gilt:

- „speed" die gewünschte Geschwindigkeit für den SPI-Takt ist. Es kann jeder Wert angefordert werden, die Firmware wählt dann die nächste erreichbare Geschwindigkeit, die gleich oder langsamer als die angeforderte Geschwindigkeit ist. Die tatsächlich eingestellte Geschwindigkeit ist CPU-Geschwindigkeit/2 geteilt durch 1 bis 256.
- „mode" ist eine einzelne Ziffer, die den Übertragungsmodus angibt – siehe Übertragungsformat 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 manipulieren.

Übertragungsformat

Das höchstwertige Bit wird zuerst gesendet und empfangen. Das Format der Übertragung kann durch den „mode" wie unten gezeigt festgelegt werden. Mode 0 ist das gängigste Format.

Modus	Beschreibung	CPOL	CPHA
0	Der Takt ist hochaktiv, die Daten werden an der steigenden Flanke erfasst und an der fallenden Flanke ausgegeben.	0	0
1	Der Takt ist hochaktiv, die Daten werden an der fallenden Flanke erfasst und an der steigenden Flanke ausgegeben.	0	1
2	Der Takt ist aktiv Low, die Daten werden an der fallenden Flanke erfasst und an der steigenden Flanke ausgegeben.	1	0
3	Der Takt ist aktiv niedrig, die 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

Standardmäßiges Senden/Empfangen

Wenn der erste SPI-Kanal offen ist, kannst du Daten mit der SPI-Funktion senden und empfangen (für den zweiten Kanal nimmst du SPI2). Die Syntax ist:

```
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 Gleitkommavariable oder eine Konstante sein.

Wenn du keine Daten senden möchtest (d. h. nur empfangen möchtest), kannst du eine beliebige Zahl (z. B. Null) für die zu sendenden Daten verwenden. Wenn du die empfangenen Daten nicht verwenden möchtest, kannst du sie einer Variablen zuweisen und ignorieren.

Massenversand/-empfang

Daten können auch massenhaft gesendet werden (verwende SPI2 für den zweiten Kanal):

```
SPI WRITE nbr, data1, data2, data3, … etc
```

oder

```
SPI WRITE nbr, string$
```

oder

```
SPI WRITE nbr, array()
```

Bei der ersten Methode ist „nbr" die Anzahl der zu sendenden Datenelemente und die Daten sind die Ausdrücke in der Argumentliste (also „data1", „data2" usw.). Die Daten können eine Ganzzahl, eine Gleitkommavariable oder eine Konstante sein.

Bei der zweiten oder dritten Methode, die oben aufgeführt sind, sind die zu sendenden Daten in „string$" oder im Inhalt von „array()" (das ein eindimensionales Array aus Ganzzahlen oder Gleitkommazahlen sein muss) enthalten. Die Länge der Zeichenfolge oder die Größe des Arrays muss gleich oder größer als nbr sein. Alle vom Slave zurückgegebenen Daten werden verworfen.

Daten können auch in großen Mengen 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 Array mit Ganzzahlen, 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 man den SPI-Port für allgemeine E/A nutzen kann. Es sendet einen Befehl 80 (hex) und empfängt zwei Bytes vom SPI-Slave-Gerät mit der Standard-Sende-/Empfangsfunktion:

```
  PIN(10) = 1 : SETPIN 10, DOUT     ' Pin 10 wird als Freigabesignal verwendet
  SETPIN GP20, GP3, GP2, SPI        ' I/O-Pins zuweisen
  SPI OPEN 5000000, 3, 8            ' Geschwindigkeit ist 5 MHz und die Datengröße
ist 8 Bit
  PIN(10) = 0                       ' Freigabeleitung aktivieren (aktiv niedrig)
  junk = SPI(&H80)                  ' Befehl senden und Rückgabe ignorieren
  byte1 = SPI(0)                    ' hol das erste Byte vom Slave
  byte2 = SPI(0)                    ' hol das zweite Byte vom Slave
  PIN(10) = 1                       ' Slave abwählen
  SPI CLOSE                         ' und Kanal schließen
```

Das Folgende ähnelt dem oben genannten Beispiel, aber diesmal wird die Übertragung mit den Befehlen für Massen-Senden/Empfangen gemacht:

```
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       ' Weisen Sie die E/A-Pins zu
PIN(10) = 1 : SETPIN 10, DOUT    ' Pin 10 wird als Freigabesignal genutzt
SPI OPEN 5000000, 3, 8           ' Geschwindigkeit ist 5 MHz, 8 Bit Daten
PIN(10) = 0                      ' Freigabeleitung aktivieren (aktiv niedrig)
SPI WRITE 1, &H80                ' Befehl senden
SPI READ 2, data%()              ' zwei Bytes vom Slave holen
PIN(10) = 1                      ' Slave abwählen
SPI CLOSE                        ' und Kanal schließen
```

Anhang E
Regex-Syntax

Die alternativen Formen der Funktionen INSTR() und LINSTR() können einen regulären Ausdruck als Suchmuster verwenden.

Die alternativen Formen der Funktionen lauten:

INSTR([start],text$, search$ [,size])

LINSTR(text%(),search$ [,start] [,size]

In beiden Fällen führt die Angabe des Größenparameters dazu, dass die Firmware die Suchzeichenfolge als regulären Ausdruck interpretiert. Der Größenparameter ist eine Gleitkomma- oder Ganzzahlvariable, die von der Firmware verwendet wird, um die Größe einer übereinstimmenden Zeichenfolge zurückzugeben. Wie in MMBasic implementiert, musst du die zurückgegebenen ***Start-*** und Größenwerte auf die MID$-Funktion 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 regulärer Ausdrücke mit OPTION ESCAPE sollte möglichst vermieden werden, da es sonst leicht zu Verwirrung kommen kann!

Anker

^ Beginn der Zeichenfolge
$ Ende der Zeichenfolge
\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-Alphabete
\s Leerzeichen \t \f \r \n \v Leerzeichen
\S Nicht-Leerzeichen
\d Ziffern
\D Nicht-Ziffern
\xXX hexadezimal codiertes Byte

Escape-Zeichen, die du brauchst, um normale Zeichen zu finden

\^ für ^ (Caret)
\. für die Übereinstimmung mit . (Punkt)
* für * (Sternchen)
\$ für $ (Dollarzeichen)
\[passt zu [(linke Klammer)
\\ passt auf \ (Backslash)
\? passt auf ? (Fragezeichen)
\{ für die Übereinstimmung mit { (linke geschweifte Klammer)
\} passt auf } (rechte geschweifte Klammer)
\| passt auf | (Vertikalstrich)
\(passt auf ((linke Klammer)
\) passt auf) (rechte Klammer)
\ + entspricht + (Pluszeichen)

Übereinstimmungsregeln

Nicht-Sonderzeichen passen zu sich selbst

. passt zu jedem Zeichen

Eine Wortgrenze ist am Anfang oder Ende einer

einer Zeichenfolge oder dort, wo ein \w-Zeichen ein

\W-Zeichen daneben steht.

Einschränkungen

Anker innerhalb einer Gruppe werden nicht unterstützt.

Beispielsweise werden (^hello) oder (hello$)
nicht wie erwartet mit „hello” am Anfang oder
Ende der Zeile abgeglichen.

Anker außerhalb der Gruppe sind aber okay.
Zum Beispiel passen ^(hello) oder (hello)$

Beispielausdruck, der mit einer IP-Adresse übereinstimmt.

```
„[\d]+\.[\d]+\.[\d]+\.[\d]+“
```

Verwendung regulärer Ausdrücke mit OPTION ESCAPE

Wenn ein regulärer Ausdruck direkt in die Funktion INSTR oder LINSTR eingebettet ist, wird jede Syntax, die das Escape-Zeichen „\“ verwendet, korrekt verarbeitet, ohne dass der Backslash weiter escaped werden muss, egal ob OPTION ESCAPE verwendet wird oder nicht. z. B.

```
 ? instr("test123","\d",size)
```

Die Funktionen INSTR / LINSTR deaktivieren OPTION ESCAPE vorübergehend, während der reguläre Ausdruck gelesen wird.

Wenn der reguläre Ausdruck aber als String-Variable übergeben wird und OPTION ESCAPE aktiviert ist, wenn du den regulären Ausdruck der Variable zuweist, musst du jeden Backslash im regulären Ausdruck mit einem Escape-Zeichen versehen. z. B.

```
s$=”\\d”
? instr("test123",s$,size)
```

Anhang F
Das PIO-Programmierpaket

Einführung in das PIO

Der RP2040 und der RP2350 haben viele eingebaute Peripheriegeräte wie PWM, UART, ADC und SPI. Mit PIOs kann man spezielle Funktionen/Peripheriegeräte wie schnelle serielle Datenschnittstellen und Bitströme hinzufügen.

PIOs kann man sich als abgespeckte, hochspezialisierte CPU-Kerne vorstellen. Der RP2040 hat zwei PIO-Blöcke, während der RP2350 drei Blöcke hat. MMBasic nennt sie PIO0, PIO1 und PIO2, genau wie in der Raspberry Pi-Dokumentation. Die PIOs laufen komplett unabhängig vom Hauptsystem und voneinander und sind extrem schnell, mit einem Durchsatz von bis zu 32 Bit pro Taktzyklus.

PIOs setzen Zustandsmaschinen um. Bevor eine Zustandsmaschine ihr Programm ausführen kann, muss das Programm in den PIO-Speicher geschrieben und die Zustandsmaschine konfiguriert werden.

Dieser Anhang beschreibt die Unterstützung, die MMBasic bei der Verwendung von PIOs bieten kann. Er enthält keine Erklärung, wie man PIO-Zustandsmaschinenprogramme schreibt. Um besser zu verstehen, wie diese funktionieren, lies den folgenden Thread „PIO explained PICOMITE" im Forum von thebackshed.com: 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

Überblick über PIO

Ein einzelner PIO-Block hat vier unabhängige Zustandsmaschinen. Alle vier Zustandsmaschinen teilen sich einen einzigen 32-Befehls-Programmbereich des Flash-Speichers. Dieser Speicher kann vom Hauptsystem nur beschrieben werden, hat aber vier Leseports, einen für jede Zustandsmaschine, sodass jede unabhängig mit ihrer eigenen Geschwindigkeit darauf zugreifen kann. Jede Zustandsmaschine hat ihren eigenen Programmzähler.

Jede Zustandsmaschine hat außerdem zwei 32-Bit-„Scratchpad"-Register, X und Y, die als temporäre Datenspeicher genutzt werden können.

Der Zugriff auf die E/A-Pins erfolgt über ein Eingabe-/Ausgabe-Zuordnungsmodul, das auf 32 Pins zugreifen kann (bei RP2040 jedoch auf 30 beschränkt). Alle Zustandsmaschinen können unabhängig voneinander und gleichzeitig auf alle Pins zugreifen.

MMBasic kann Daten in das Eingangsende eines 4-Wort-32-Bit-breiten TX-FIFO-Puffers schreiben. Die Zustandsmaschine kann dann PULL verwenden, um das Ausgangswort des FIFO in das OSR (Output Shift Register) zu verschieben. Sie kann auch OUT verwenden, um jeweils 1-32 Bits vom OSR in das Ausgangs-Mapping-Modul oder andere Ziele zu verschieben. Mit AUTOPULL können Daten automatisch gezogen werden, bis der TX-FIFO leer ist oder einen voreingestellten Pegel erreicht.

MMBasic kann Daten vom Ausgangsende eines 4-Wort-RX-FIFO-Puffers mit einer Breite von 32 Bit lesen. Die Zustandsmaschine kann dann IN verwenden, um jeweils 1 bis 32 Datenbits vom Eingangsabbildungsmodul in das ISR (Input Shift Register) zu verschieben. Anschließend kann sie PUSH verwenden, um den Inhalt des ISR in den FIFO zu verschieben. Mit AUTOPUSH können Daten automatisch verschoben werden, bis der RX-FIFO voll ist oder einen voreingestellten Pegel erreicht.

Die FIFO-Puffer können neu konfiguriert werden, um einen unidirektionalen 8-Wort-32-Bit-FIFO in einer Richtung zu bilden. Die Puffer ermöglichen die Übertragung von Daten zu und von den Zustandsmaschinen, ohne dass das System oder die Zustandsmaschine aufeinander warten müssen.

Jede der vier Zustandsmaschinen im PIO hat vier zugehörige Register:

- CLKDIV ist der Taktteiler, der über einen 16-Bit-Ganzzahlteiler und einen 8-Bit-Bruchteilteiler 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 Verwendung der Schieberegister.
- PINCTRL steuert, welche und wie die GPIO-Pins 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 sie setzen, zurücksetzen oder auf ihre Änderung warten. Auf diese Weise können sie bei Bedarf synchron laufen. Die unteren vier Flags sind auch für das Hauptsystem zugänglich, sodass der PIO von MMBasic aus gesteuert werden kann oder Interrupts zurückgeben kann.

DMA kann verwendet werden, um Informationen über seinen FIFO vom Speicher des RP2040 zum und vom PIO-Block zu übertragen.

Ein PIO hat neun mögliche Programmierbefehle, aber es kann viele Variationen für jeden einzelnen geben. Zum Beispiel kann Mov bis zu 8 Quellen, 8 Ziele und 3 Prozessoperationen während des Kopiervorgangs haben, mit optionalen Verzögerungs- und/oder Side-Set-Operationen!

- Jmp Springe zu einer absoluten Adresse im Programmspeicher, wenn eine Bedingung wahr ist (oder sofort).
- Warten Verzögert den Betrieb der Zustandsmaschine, bis eine Bedingung erfüllt ist.
- In Verschiebt eine Anzahl von Bits aus einer Quelle in den ISR.
- Aus Verschiebt eine Anzahl von Bits aus dem OSR an ein Ziel.
- Push Schiebt 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 Daten sofort an einen Zielort schreiben.

Alle Befehle sind 16-Bit-Befehle und enthalten sowohl den Befehl 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.

Außerdem gibt's eine Funktion namens „Side-Set”, mit der ein Wert in einige vordefinierte Ausgangspins geschrieben werden kann, während ein Befehl aus dem Speicher gelesen wird. Das ist für das Programm nicht sichtbar.

Programmierung von PIO

Die PicoMite-Firmware programmiert den PIO-Zustandsmaschinen-Speicher mit einer von drei Methoden. Jede Methode wird anhand eines Beispiels mit genau demselben Programm erklärt, das einen der GPIO-Pins des Raspberry Pi Pico umschaltet. Welcher GPIO-Pin umgeschaltet wird, hängt von der Konfiguration ab, nicht vom Programm selbst.

PIO ASSEMBLE

Dieser Befehl wird verwendet, um mit dem integrierten Assembler das Programm aus Mnemoniken zu generieren und es dann direkt in den PIO-Speicher zu schreiben.

```
PIO ASSEMBLE 1,".program test"          'Ein Programm muss einen Namen haben
PIO ASSEMBLE 1,".line 0"                'das Programm bei Zeile 0 starten
PIO ASSEMBLE 1,"SET PINDIRS,1"          'GPIO-Zeile auf Ausgabe setzen
PIO ASSEMBLE 1, „label:"                'Label namens „label" definieren
PIO ASSEMBLE 1,"SET PINS,1"             'GPIO-Pin auf High setzen
PIO ASSEMBLE 1,"SET PINS,0"             'GPIO-Pin auf niedrig setzen
PIO ASSEMBLE 1, „JMP label"             'Springe zu „label"
PIO ASSEMBLE 1, „.end program list"     'Programm beenden, list=Ergebnis
anzeigen
```

Der Assembler lässt auch ein besser lesbares Format wie dieses zu

```
PIO ASSEMBLE 1, „.program test"   'Ein Programm muss einen Namen haben
.Zeile 0                          'das Programm beginnt bei Zeile 0
   PINDIRS,1 einstellen                 'GPIO-Leitung auf Ausgang setzen
.LABEL label:                     'Definiere eine Bezeichnung namens „label"
   SET PINS,1                     'GPIO-Pin auf High setzen
   PINS,0 SET                     'GPIO-Pin auf niedrig setzen
   JMP label                      'Springe zu „label"
.end program list                 'Programm beenden, list=Ergebnis anzeigen
```

PIO-PROGRAMMZEILE

Mit diesem Befehl kannst du 16-Bit-Werte für einzelne Zeilen (Speicherplätze) im PIO-Speicher programmieren.

```
PIO PROGRAM LINE 1,0,&hE081             'Pin-Ausgang festlegen
PIO PROGRAM LINE 1,1,&hE001             'Pin hoch setzen
PIO PROGRAM LINE 1,2,&hE000             'Pin niedrig setzen
PIO PROGRAM LINE 1,3,&h0001             'JMP zu Zeile 1
```

PIO PROGRAM

Dieser Befehl schreibt alle 32 Zeilen in einem PIO aus einem Array. Das ist nützlich, wenn ein PIO-Programm debuggt wurde. Es ist super kompakt.

```
DIM a%(7)=(&h0001E0000E001E081,0,0,0,0,0,0,0)
PIO-PROGRAMM 1,a%()
```

PIO konfigurieren

Die PicoMite-Firmware kann jede Zustandsmaschine einzeln konfigurieren. Die Konfiguration ermöglicht es, dass zwei Zustandsmaschinen genau dieselben Programmzeilen ausführen (z. B. eine SPI-Schnittstelle), aber mit unterschiedlichen GPIO-Pins und unterschiedlichen Geschwindigkeiten arbeiten. Es gibt mehrere Konfigurationsfelder.

FREQUENZ

Die PicoMite-Firmware enthält für jedes Konfigurationsfeld eine Standardkonfiguration, außer für die Frequenz. Die Frequenz wird durch einen 16-Bit-CLKDIV-Teiler vom ARM-Taktgeber eingestellt. Beispiel: Wenn OPTION CPUSPEED 126000 eingestellt ist, kann die PIO mit Geschwindigkeiten zwischen 126 MHz und 1,922 kHz (126000000 / 65536) laufen. 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 die PIO die Kontrolle über einen GPIO-Pin übernehmen kann, muss MMBasic ihn wie unten gezeigt der PIO zuweisen.

SETPIN GPxx,PIOx (z. B. SETPIN gp0,pio1)

Eine Zustandsmaschine kann den Zustand eines Pins SETZEN (SET ist eine Zustandsmaschinenanweisung), aber auch serielle Daten mit der OUT-Anweisung an einen oder mehrere GPIO-Pins ausgeben. Oder sie kann serielle Daten mit der IN-Anweisung lesen. Und GPIO-Pins können als Nebeneffekt einer beliebigen Zustandsmaschinenanweisung (SIDE SET) gesetzt werden. Für jede Schnittstellenmethode können der Zustandsmaschine unterschiedliche Pins zugeordnet werden.

Wichtig zu wissen ist, dass diese Befehle auf aufeinanderfolgende Pins wirken. Das heißt, es gibt einen Bereich von Pins, die gesteuert werden können, beginnend mit der niedrigsten GPx-Pin-Nummer (z. B. GP0), und die benachbarten Pins können mit einbezogen werden (insgesamt bis zu 5 Pins). GP0, GP1, GP2 ist also ein gültiger Satz von IO-Pins. GP0, GP1, GP6 ist es nicht. Beachte das 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 schreiben
- b/ die Anzahl der SET-Pins (0...5), SET kann 5 Pins gleichzeitig schreiben
- c/ die Anzahl der OUT-Pins (0...31), OUT kann 32 Pins gleichzeitig schreiben
- 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, und Felder, die die Zeilenadresse des Hauptprogrammschleifenbeginns (.WRAP TARGET) und -endes (.WRAP) enthalten.

Wenn wir wollen, dass sich der Programmablauf in Abhängigkeit vom 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 hoch ist.

Das Zustandsmaschinenprogramm startet am Anfang und läuft bis zum Ende. Im obigen Demoprogramm läuft das Programm „ ” mit einer (bedingungslosen) JMP-Anweisung vom Ende zum Anfang. Eine Alternative zur Verwendung der JMP-Anweisung 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 Anweisungen ausführt. Der JMP-Befehl in Zeile 3 wird 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 IN- und OUT-Befehle 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 die Daten im FIFO ab, der PIO liest sie aus. Dabei wird der TX-FIFO verwendet. Umgekehrt wird der RX-FIFO genutzt. Die FIFOs sind normalerweise 4 Wörter tief, können aber auch als einzelner 8 Wörter tiefer 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 der FIFO aufgefüllt wird.

Im RP2350 können die FIFOs auch als einzelne Register genutzt werden, was eine flexiblere Kommunikation zwischen MMBasic und den Zustandsmaschinen ermöglicht. Das wird durch FJOIN_RX_GET und FJOIN_RX_PUT im SHIFTCTRL-Register gemacht.

PIO(SHIFTCTRL a,b,c,d,e,f,g,h)

a/ Push-Schwelle	(1..31 Bits werden vor autoPUSH in ISR verschoben)
b/ Pull-Schwelle	(1..31 Bits werden vor autoPULL aus OSR verschoben)
c/ AutoPush	(1 = automatisches Drücken zulassen)
d/ autopull	(1 = automatisches Ziehen 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 verbinden)
h/ fjoin_rx	(TX- und RX-FIFO zu 1 TX-FIFO verbinden)
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

Die Konfiguration der Zustandsmaschine wird mit dem folgenden Befehl geschrieben:

PIO INIT MACHINE a,b,c,d,e,f,g

a/ die PIO (0,1 oder 2(nur 2350))
b/ die Zustandsmaschinen-Nummer (0...3)
c/ Frequenz (CPUSPEED/65536...CPUSPEED in Hz)
d/ Pinsteuerungswert (PIO(PINCTRL .....))
e/ Exekutionssteuerungswert (PIO(EXECCTRL.....))
f/ Schaltsteuerungswert (PIO(SHIFCTRL.....))
g/ Startadresse (0....31, die Zeile, in der die Zustandsmaschine mit der Ausführung beginnt, kann eine Bezeichnung 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/ Frequenz (CPUSPEED/65535 bis zu CPUSPEED in Hz)
d/ Startadresse (0..31)
e/ Seiteneinstellungsbasis (SIDE SET-Bereich beginnt mit GPx)
f/ Seiten-Set-Nr. (0..5...Auswirkung auf DELAY-Bereich)
g/ sidesetout (1 = oben definierte Pins automatisch als Ausgang festlegen)
h/ setbase (SET-Bereich beginnt mit GPy)
i/ setno (0...5)
j/ Ausrichtung (1=die oben definierten Pins automatisch als Ausgang festlegen)
k/ outbase (OUT/MOV-Bereich beginnt mit GPz)
l/ outno (0...5)
m/ outout (1 = oben definierte Pins automatisch als Ausgang 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 = nur Pins explizit aktualisieren, wenn side set in der Anweisung steht)
s/ sidepindir (1 = Side-Set ändert die Pin-Richtung, nicht den Pin-Status)
t/ pushthreshold (1..31 Bits werden vor autoPUSH in ISR verschoben)
u/ pullthreshold (1..31 Bits werden vor autoPULL aus OSR herausgeschoben)
v/ autopush (1 = automatisches Drücken zulassen)
w/ autopull (1 = automatisches Ziehen zulassen)
x/ inshiftdir (1 = Richtung MSB→LSB verschieben, 0 = Richtung LSB→MSB verschieben)
y/ outshiftdir (1 = MSB→LSB verschieben, 0 = LSB→MSB verschieben)
z/ joinrxfifo (1 = RX-FIFO ist 8 tief, TX-FIFO = 0
aa/ jointxfifo (1 = TX-FIFO ist 8 tief, RX-FIFO = 0
bb/ joinrxfifoget (1 = RX-FIFO ändert sich zu 4 lesbaren Registern (nur 2350)
cc/ joinrxfifoput (1 = RX-FIFO ändert sich zu 4 beschreibbaren Registern (nur 2350)

STARTEN UND STOPPEN EINER ZUSTANDSMASCHINE

Sobald die PIO konfiguriert ist, kannst du die Zustandsmaschine mit folgenden Befehlen 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 Stoppen genau an der Stelle stoppt, an der sie sich gerade befindet. Um die Zustandsmaschine neu zu starten, solltest du zuerst PIO INIT MACHINE ausführen.

BEISPIELPROGRAMM 1

Eine komplette PIO-Implementierung, die einen GPIO-Pin umschaltet, kann in MMBasic wie unten gezeigt

gemacht werden. Schließ einen Summer an GP0 an und hör dir den von der PIO erzeugten Ton an.

```
'ARM von GP0 trennen
setpin gp0,pio1                  'GP0 als Ausgangspin für PIO 1 verwenden

'PIO-Programm verwendet
'0 E081   'Pin-Ausgang einstellen
'1 E001   'Pin hoch setzen
'2 E000   'Pin niedrig setzen
'3 0001   'jmp 1

'Programm pio 1 mit einem Array, um das Programm in den PIO-Speicher zu
schreiben, und starten
Dim a%(7)=(&h0001E000E001E081,0,0,0,0,0,0,0)
PIO-Programm 1,a%()

'PIO 1-Zustandsmaschine 0 konfigurieren
p=Pio(pinctrl 0,1,,,,gp0,)   'SET definiert die Verwendung von 1 Pin, und zwar
GP0
f=3100                       '3051 Hz ist die niedrigste Frequenz CPUSPEED
200000
PIO-Initialisierungsmaschine 1,0,f,p     „Standardwerte für execctrl, shiftctrl,
                            start verwenden                    'Adresse (=0)

'PIO 1-Zustandsmaschine starten 0
PIO starten 1,0
```

Beachte, dass das MMBasic-Programm beendet wird, der Summer aber weiterläuft. PIO ist unabhängig von der ARM-CPU und läuft weiter, bis es gestoppt wird. Durch Aufrufen des MMBasic-Editors wird PIO gestoppt.

FIFOs

MMBasic und PIO tauschen Infos über FIFOs aus. Die PIOs schieben Daten in den RX-FIFO (MMBasic ist der Empfänger) oder ziehen Daten aus dem TX-FIFO (MMBasic ist der Sender).

Wenn die PIO Daten aus dem FIFO abruft, werden die Daten an das OSR (Output Shift Register) übertragen, von wo aus sie verarbeitet werden können. Die PIO kann die Daten aus dem ISR (Input Shift Register) in den FIFO schieben. Zusätzlich verfügt die PIO über zwei Register X und Y, die zum Speichern oder Zählen verwendet werden können. Die PIO kann nicht addieren, subtrahieren oder vergleichen.

Datenfluss:

MMBasic -> FIFO -> OSR -> PIO (oder Pins)
PIO (oder Pins) -> ISR -> FIFO -> MMBasic

MMBasic kann Daten in den TX-FIFO schreiben und Daten aus dem RX-FIFO lesen, indem es Folgendes verwendet:

PIO READ a,b,c,d
PIO WRITE a,b,c,d
- a/ PIO-Nummer (0,1 oder 2 (nur 2350))
- b/ Zustandsnummer (0...3)
- c/ Anzahl der 32-Bit-Wörter (1...4)
- d/ Name der ganzzahligen Variablen (z. B. variable% oder array%())

PIO CLEAR löscht alle PIO-FIFOs, genauso wie PIO START und PIO INIT MACHINE.

Das MMBAsic-Programm muss nicht auf das Erscheinen von Daten im FIFO warten, da dem RX-FIFO ein Interrupt zugewiesen werden kann. Die MMBasic-Interrupt-Routine kann die Daten aus dem FIFO holen.

Ähnlich verhält es sich mit dem TX-Interrupt, bei dem MMBasic einen Interrupt erhält, wenn Daten für den 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 erklärt viele der oben vorgestellten MMbasic-Funktionen und -Befehle. Das Programm liest einen NES-Controller (SPI), der an den Raspberry Pi Pico angeschlossen ist. Der NES-Controller besteht aus einem HEF4021-Schieberegister, das mit 8 Druckschaltern verbunden ist.

Das Programm nutzt: **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 Verkabelung ist wie im Code definiert:

```
'ARM von GP0/1/2 trennen
  setpin gp0,pio1                'Takt ausgeben
  setpin gp1,pio1                'Auslesen
  Pin GP2, PIO1 setzen                 'Daten rein

'PIO-Programm

PIO assemble 1,".program NES"     'Ein Programm braucht einen Namen
.Seite setzen 2                        '2 Bits für Seitensatz, 3 für
                                       Verzögerung verwenden
Zeile 0                           'Startcode in Zeile 0
   SET pindirs,&b11               'GP0,GP1-Ausgang einstellen, Seite GP0,GP1
                                       niedrig
.Ziel umschließen                      'Ziel umschließen = Anfang der Schleife
   IN null,32 Seite 2             'set ISR auf 0, GP1 hoch (laden), GP0 niedrig
   SET X,7 Seite 0                'X-Zähler auf 7 setzen, GP0,GP1 niedrig
.label Schleife:                       'innere Schleifenbezeichnung
   IN Pins,1 Seite 0              '1 Datenbit verschieben, GP0,GP1 niedrig
                                       halten
   JMP X-- Schleifenseite 1            'jmp zu Schleife, dec. X, GP0
                                       'hoch(Takt)
   PUSH Seite 0 [7]               'jetzt X=0, Ergebnis in FIFO schieben,
                                       Verzögerung 7
.wrap"                            'äußere Schleife beenden, wiederholen
.end Programmliste                'Ende des Programms, Ergebnis auflisten

'pio1 einrichten
  p=Pio(pinctrl 2,2,,gp2,gp0,gp0,)'GP0,GP1 aus (SET und SIDESET), GP2 IN
  f=1e5                           '100 kHz
  s=PIO(shiftctrl 0,0,0,0,0,0)    'Verschiebung von LSB für IN (und OUT)
  e=PIO(execctrl gp0,PIO(.wrap target),PIO(.wrap))  'Wrap und Wrap-Ziel

'Konfiguration schreiben
  PIO init machine 1,0,f,p,e,s,0  'bei Zeile 0 starten

'PIO1-Code starten
  PIO start 1,0


'Die gelesenen Daten in MMBasic überprüfen und ausgeben
  dim d%
  do
    pio lesen 1,0,1,d%
    Ausgabe bin$(d%)
    pause 200
  Schleife
  END
```

DMA zu und von den FIFOs

So funktioniert DMA:

Beim Lesen aus dem FIFO wartet der DMA-Controller, bis Daten im FIFO sind, und wenn sie da sind, schickt er sie in den Prozessorspeicher. Bei jedem Lesevorgang verschiebt er den Zeiger im Prozessorspeicher weiter, damit er zum Beispiel ein Array nach und nach füllen kann, sobald die Daten verfügbar sind.

Beim Schreiben in den FIFO schreibt der DMA-Controller Daten aus dem Prozessorspeicher automatisch 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, wie es das PIO-Programm braucht, an den PIO-FIFO.

DMA kann ein 32-Bit-Wort, ein 16-Bit-Short oder ein 8-Bit-Byte übertragen. Beim Einrichten von DMA musst du die Größe der Übertragung und die Anzahl der durchzuführenden Übertragungen angeben. Da jede Übertragung den Speicherzeiger um 1, 2 oder 4 Bytes erhöht, muss MMBasic mit den im Speicher gepackten Daten arbeiten und nicht mit den 64 Bit, die für MMbasic-Ganzzahlen und -Gleitkommazahlen verwendet werden. Zum Glück hat MMBasic zwei Befehle, MEMORY PACK und MEMORY UNPACK, die das sehr effizient machen, aber es geht auch mit Standard-BASIC-Arithmetik.

Der DMA kann so konfiguriert werden, dass er Daten wiederholt in einen Speicherbereich (einen Ringpuffer) ein- oder aus diesem ausliest.

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, und es ist kein PIO-START-Befehl nötig. Bevor du die Übertragung startest, solltest du aber sicherstellen, dass ein neues PIO INIT MACHINE durchgeführt wird, damit die Zustandsmaschine an der gewünschten Startadresse startet.

Wenn ein Ringpuffer verwendet wird, sind spezielle Vorbereitungen erforderlich:

PIO MAKE RING BUFFER a, b

Wobei:	a = Name des Integer-Puffers	
	b = Größe des Arrays in Bytes	

Beispiel:

```
DIM packed%                              Achtung: KEINE Klammern()
PIO MAKE RING BUFFER packed%,4096
```

packed% ist dann ein Integer-Array, das 4096/8 = 512 Integer-Werte enthält

Das kann dann vom DMA für einen Loopback-Zähler mit DMA von 1024 32-Bit-Wörtern, 2048 16-Bit-Shorts oder 4096 8-Bit-Bytes verwendet werden.

BEISPIELPROGRAMM 3

Dieses Programm bringt 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 Logikanalysatorprogramm demonstriert auch PIO DMA RX, MEMORY UNPACK und die Verwendung von Puffern. Es verwendet PWMs, um zu Demonstrationszwecken ein Testsignal auf gp0..gp5 zu erzeugen. Dieselben Pins werden vom Logikanalysator gelesen und an die Konsole ausgegeben.

Um diesen Logikanalysator zu nutzen, musst du die ersten 14 Zeilen auskommentieren.

```
'Erzeugt ein 50-Hz-3-Phasen-Testsignal, um den DMA auf 6 GPIO-Pins zu zeigen.
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-Synchronisation 0, 100/3, 200/3

------------------------------ LA-Code PIO -------------------------
'PIO-Code zum Abtasten von GP0..GP6 als elementarer Logikanalysator
PIO löschen 1

'In diesem Programm liest der PIO GP0..GP5 mit Brute-Force
'und schiebt die Daten in den FIFO. Die Taktrate bestimmt die
'Abtastrate. Es gibt 2 Befehle pro Zyklus
'mit 10000/2 / 50 = 100 Abtastungen pro 50-Hz-Zyklus.

PIO assemble 1, „.program push"
   .Zeile 0
   .wrap Ziel
   IN-Pins,6                „6 Bits von GP0..GP5 holen
   PUSH-Block               'Daten schieben, wenn FIFO Platz hat
   .wrap
.end Programmliste

'Konfiguration
f=1e4                                   'PIO läuft mit 10 kHz
p=Pio(pinctrl 0,0,0,gp0,,,)             'IN-Basis = GP0
e=Pio(execctrl gp0,PIO(.wrap target),PIO(.wrap)) 'Wrap 1 bis 0, gp0 ist
                                                  „Standard
s=Pio(shiftctrl 0,0,0,0,0,0)      'Verschiebung durch LSB, Ausgang wird nicht
benutzt

'Konfiguration schreiben, Geschwindigkeit 10 kHz (Daten in FIFO 10 us nach
Flanke GP0)
PIO init machine 1,0,f,p,e,s,0          'Startadresse = 0

'-------------------------- LA-Code MMBasic ----------------------------
'Speicherpuffer definieren
Dim a$(1)=("_","-")                     'Zeichen für den Ausdruck
Länge%=64                          'Größe des gepackten Arrays
Dim data%(2*length%-1)                  'Array zum Speichern der 32-Bit-Samples
                                        'FIFO-Format
Dim gepackt%(länge%-1)                  'DMA-Array zum Packen von 32-Bit-Samples
                                        'in 64-Bit-Ganzzahlen

'DMA-Maschine laufen lassen und nach Belieben wiederholen
Do
  PIO DMA RX 1,0,2*Länge%,gepackt%(),ReadyInt
  print "Drück irgendeine Taste, um die Abtastung neu zu starten"
  do:loop while inkey$=""
Schleife
Ende

'--------------------------------SUBS MMBasic -------------------------
Sub ReadyInt
'PIO stoppen und für nächsten Durchlauf neu starten
PIO stop 1,0
PIO init machine 1,0,f,p,e,s,0          'Startadresse = 0

'Hol die Daten aus dem gepackten DMA-Puffer und pack sie wieder in das
ursprüngliche 32-Bit-Format
Speicher entpacken packed%(),data%(),2*length%,32

'Serielle Ausgabe wie bei Logikanalysator-Traces
Für j=0 bis 5
  Maske%=2^j
  Für i=0 bis 2*Länge%-1
    Wenn i<106 Dann Drucken a$(((Daten%(i) Und Maske%)=Maske%));
  Nächstes i
  Drucken: Drucken
Nächstes j
Ende Sub
```

BEISPIELPROGRAMM 4

Dieses Programm läuft nur auf RP2350 und zeigt, wie man die FIFOs als einzelne Register benutzt. Der PIO des RP2350 hat spezielle Befehle, die das unterstützen: MOV RXFIFO[y],ISR und MOV OSR,RXFIFO[y].

MMBasic kann die 4 einzelnen FIFO-Register mit folgendem Befehl lesen/schreiben:

PIO WRITEFIFO a, b, c, d
PIO READFIFO(a, b, c)

a/ pio	(0,1 oder 2(nur 2350))
b/ Zustandsmaschine	(0...3)
c/ Anzahl	(FIFO-Register 0…3)
d/ Daten%	(32-Bit-Ganzzahlwert)

Das Programm macht das FIFO für einzelne Lesevorgänge klar und schreibt dann ein paar Werte in diese Register. Es startet das PIO, das 2 der 4 einzelnen FIFO-Register aktualisiert. Dann liest MMBasic die Werte, um zu zeigen, dass sich nur 2 Register geändert haben und keine Daten verschoben wurden (wie es beim RP2040 FIFO passieren würde).

```
Nur für RP2350-Assembler. Das funktioniert nicht auf RP2040.

pio clear 1

pio assemble 1,".program test"
.line 0
set y,4                 'nur ein Wert 4 in Y
mov isr,y               'Y nach isr kopieren
jmp y--,next            'y=y-1 immer zum nächsten
.label next             'Bezeichnung
mov rxfifo[y],isr       'sollte Programm 4 in FIFO [3]
mov isr,y               'Y nach isr kopieren
mov rxfifo[2],isr       'sollte 3 in FIFO [2] programmieren
jmp 0                   'wiederholen
.Ende Programm

f=1e6 '1 MHz

'PIO(EXECCTRL a,b,c)
e=pio(execctrl gp0,0,31)            'Standardwert, wird nicht wirklich geändert

'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) 'Einzelnen RX lesen, RX=4 tief
sw=pio(shiftctrl 0,0,0,0,0,0,0,0,1,0) 'einzelnes RX schreiben, RX=4 tief

'PIO(PINCTRL a,b,c,d,e,f,g)
p=pio(pinctrl 0,0,0,gp0,gp0,gp0,gp0) 'Standardwert, wird nicht wirklich geändert

'FIFO als einzelne Register testen
'FIFO mit vorab festgelegten Werten füllen
pio init machine 1,0,f,p,e,sw,0   'Maschine zum Schreiben in RX-FIFO
initialisieren
for i=0 to &h3                    'die 4 RXFIFO-Register schreiben
 pio writefifo 1,0,i,&h100*(i+1)  'Werte &h100, &h200, &h300, &h400 schreiben
next

'Überprüfe, ob die Werte richtig geschrieben wurden
print "3 RXFIFO-Register vor dem Ausführen des Programms"
pio init machine 1,0,f,p,e,sr,0   'Maschine zum Lesen des RX-FIFO initialisieren
für i=0 bis 3                     'Lies die 4 RXFIFO-Register
   print i,hex$(pio(readfifo 1,0,i))  'Überprüfe, ob sie richtig geschrieben sind
nächstes
aus

'PIO-Programm ausführen. Das sollte (kontinuierlich) in die Register 2 und 3 des
FIFO schreiben, aber die Register 0 und 1 nicht verändern
```

```
pio start 1,0

'Zeige die aktualisierten FIFO-Register
print „3 RXFIFO-Register nach dem Ausführen des Programms”
für i=0 bis 3     'die 4 FIFO-Register lesen, um zu sehen, ob das Programm
funktioniert
   print i,hex$(pio(readfifo 1,0,i))
next
```

Die erwartete Ausgabe ist:

```
3 RXFIFO-Register vor dem Ausführen des Programms
0      100
1      200
2      300
3      400

3 RXFIFO-Register nach dem Ausführen des Programms
0      100
1      200
2      3
3      4
```

BEISPIELPROGRAMM 5

Zum Schluss noch 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) zur Bestimmung des Startpunkts des zweiten Programms.

IRQ 1 setzt und IRQ. Wenn die andere Zustandsmaschine auf WAIT 1 IRQ 1 wartet, fährt sie fort, nachdem IRQ 1 als gesetzt erkannt wurde, und löscht gleichzeitig IRQ 1.

```
'Test für PIO-IRQs auf den PIO 0-Zustandsmaschinen 0 und 1

  'verwendet GP0 und GP1
  setpin gp0,pio0
  setpin gp1,pio0
  pio clear 0

  'Zielfrequenz für PIO
  f=1e5 '100 kHz

pio assemble 0
  .Programm sm0
  .Zeile 0
    pindirs,1 festlegen                  „GP0 auf Ausgang setzen, siehe pinctrl
  .Ziel umschließen
    Pins setzen, 1 [31]             GP0 auf High setzen, 31 Zyklen warten
    Pins setzen, 0 [31]             GP0 auf niedrig setzen, 31 Zyklen warten
    irq 0                           „IRQ 0 setzen
    1 IRQ 1 warten                        ‘Warte, bis sm1 IRQ 1 setzt
  .wrap
.end programmliste

  ln=pio(nächste Zeile)             „Zeile im Programm merken
  p0=pio(pinctrl 0,1,,,,gp0)        „GP0 ist Pin für SET
  e0=pio(execctrl gp0,pio(.wrap target),pio(.wrap))

pio assemble 0
  .Programm sm1
  .Zeile weiter
    pindirs,1 setzen                     „GP1-Ausgang festlegen
  .Ziel umschließen
    warte 1 irq 0                   ‘auf IRQ 0 warten
```

```
    Pins setzen, 1 [31]             GP1 auf High setzen und 31 Zyklen warten
    Pins,0 [31] setzen              „GP1 auf niedrig setzen und 31 Zyklen warten
    irq 1                           „IRQ 1 setzen
  .wrap
.end Programmliste

  p1=pio(pinctrl 0,1,,,,gp1)
  e1=pio(execctrl gp0,pio(.wrap target),pio(.wrap))

  pio init Maschine 0,1,f,p1,e1,,ln ‘sm1 bei Zeile ln starten
  pio init Maschine 0,0,f,p0,e0,,0  „sm0 bei Zeile 0 starten

  pio start 0,1
  pio start 0,0

do:loop 'nichts tun, PIO die Signale generieren lassen

* Ende *
```

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 kommt, wenn du das Sprite anzeigst. In diesem Fall speichert die Firmware beim ersten Mal den Speicherbereich (oder den Bildschirmbereich), 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 Grenzen von Sprites, die sich berühren, um einen Interrupt zu erzeugen, oder nach Sprites, die den Rand des Rahmens berühren.

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 einfach Sprite 1 verschieben würdest, würde sein Hintergrund Teile von 2 und 3 überschreiben – was nicht in unserem Sinne ist. SPRITE SHOW SAFE löst den LIFO auf, indem es jedes Sprite in umgekehrter Reihenfolge entfernt, Sprite 1 verschiebt und dann zuerst 2 und dann 3 darüber wiederherstellt. Schließlich gibt es noch das Konzept der Ebenen (dies ist der vierte Parameter in SPRITE SHOW).

Das Sprite-Konzept sieht so aus:

- Sprites sind vollfarbig und können jede beliebige Größe haben. Die Kollisionsgrenze ist das umschließende Rechteck.
- Sprites werden bis zu einer bestimmten Anzahl (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, Ebene 0 oder dem 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 unbewegt.
- Sprites auf Ebene 0 scrollen mit dem Hintergrund, 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() kann der Benutzer die Details einer Kollision vollständig abfragen.
- 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 Schreibweise 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, um 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 zum Erkennen aller Kollisionen, die entweder durch einen SPRITE SHOW-Befehl oder einen SCROLL-Befehl verursacht wurden

```
'
' Diese Routine zeigt eine vollständige Abfrage von Kollisionen
'
SUB collision
```

```
  LOCAL INTEGER i
' Zuerst mit der SPRITE(S)-Funktion schauen, was die Unterbrechung verursacht hat
  IF SPRITE(S) <> 0 THEN 'Kollision eines bestimmten einzelnen Sprites
    'SPRITE(S) gibt das Sprite zurück, das sich bewegt hat und die Kollision
verursacht hat
    PRINT „Kollision bei Sprite “, SPRITE(S)
    process_collision(SPRITE(S))
    PRINT
  SONST '0 bedeutet, dass die Kollision von einem oder mehreren Sprites durch eine
Bewegung im Hintergrund verursacht wurde
          ' SPRITE(C, 0) zeigt uns, wie viele Sprites eine Kollision hatten
    PRINT "Durch das Scrollen sind insgesamt ", SPRITE(C,0)," Sprites
zusammengestoßen"
    FOR I = 1 TO SPRITE(C, 0)
      ' SPRITE(C, 0, i) zeigt uns die Sprite-Nummer des „I”-ten Sprites
      PRINT "Sprite ", SPRITE(C, 0, i)
      process_collision(SPRITE(C, 0, i))
    NEXT i
    PRINT
  ENDIF
END SUB

' Details zu den spezifischen Kollisionen für ein bestimmtes Sprite abrufen
SUB process_collision(S AS INTEGER)
  LOKAL INTEGER i, j
  ' SPRITE(C, #n) gibt die Anzahl der aktuellen Kollisionen für Sprite n zurück
  PRINT „Insgesamt „SPRITE(C, S) “ Kollisionen”
   FOR I = 1 TO SPRITE(C, S)
    ' SPRITE(C, S, i) gibt uns die Sprite-Nummer des „I“-ten Sprites
    j = SPRITE(C, S, i)
    WENN j = &HF1 DANN
      DRUCKE „Kollision mit der linken Bildschirmseite”
    SONST WENN j = &HF2 DANN
      PRINT "Kollision mit dem oberen Bildschirmrand"
    SONST WENN j = &HF4 DANN
      DRUCKE „Kollision mit der rechten Bildschirmseite”
    SONST WENN j = &HF8 DANN
      PRINT "Kollision mit dem unteren Bildschirmrand"
    SONST
      ' SPRITE(C, #n, #m) gibt Details zur m-ten Kollision zurück
      PRINT "Kollision mit Sprite ", SPRITE(C, S, i)
    ENDIF
  NEXT i
END SUB
```

Anhang H
Turtle-Grafik

Diese Version hat eine echt umfassende Turtle-Grafik-Implementierung für alle Versionen, außer für WebMite RP2040 und WebMite RP2350.

Die unterstützten Befehle mit dem Präfix „TURTLE “ sind:

Bewegungsbefehle

FORWARD Entfernung	(FD) – Um die angegebene Pixelanzahl vorwärts bewegen
BACK distance	(BK) – Um distance Pixel rückwärts bewegen
LINKS [Winkel]	(LT) – Nach links um Winkelgrad drehen, 90 Grad, wenn nichts angegeben ist
RECHTS [Winkel]	(RT) - Nach rechts um Winkel Grad drehen, 90 Grad, wenn nicht anders angegeben

Positionsbefehle

SET XY x, y	- Zur absoluten Position (x,y) gehen
SET X x	- X-Koordinate einstellen, Y beibehalten
SET Y y	- Y-Koordinate einstellen, X behalten
SET HEADING Winkel	(SETH) - Absolute Richtung einstellen (0=nach oben, 90=nach rechts)
HOME	- Zurück zur Mitte (MM.HRES\2,MM.VRES\2) Richtung 0

Stiftsteuerungsbefehle

STIFT HOCH	(PU) – Stift anheben (Zeichnen beenden)
PEN DOWN	(PD) - Stift absenken (Zeichnen starten)
PEN COLOUR Farbe	(PC) - Stiftfarbe einstellen
STIFTBREITE Breite	(PW) - Strichstärke einstellen

Befehle für Bögen und Kurven

BOGEN Radius Winkel	– Zeichne einen Bogen mit dem angegebenen Radius und Winkel
ARCLEFT Radius, Winkel	(ARCL) - Zeichne einen nach links gerichteten Bogen
ARCRIGHT Radius,Winkel	(ARCR) - Rechtsdrehenden Bogen zeichnen
BEZIER cp1 , cp1winkel, cp2, cp2winkel, ende, endekwinkel	- Zeichne eine Bezierkurve mit Kontrollpunkten

Grundlegende Formbefehle

CIRCLE Radius	– Zeichne einen Kreis an der aktuellen Position
DOT Größe	- Gefüllten Punkt zeichnen (Standardgröße = 5)
FCIRCLE radius	- Gefüllten Kreis zeichnen
FRECTANGLE Breite, Höhe	(FRECT) - Gefülltes Rechteck zeichnen
WEDGE Radius Anfang Ende	– Gefüllten Keil/Tortenstück zeichnen

Füllbefehle

FILL COLOUR Farbe	(FC) – Füllfarbe festlegen und Füllung aktivieren
FILL PATTERN Muster	(FP) - Füllmuster festlegen (0-31)
KEINE FÜLLUNG	– Füllung deaktivieren
FÜLLEN	– An der aktuellen Position flächig füllen
FÜLLUNG STARTEN	(BF) - Polygon für Füllung aufnehmen
END FILL	(EF) - Polygonaufzeichnung beenden und füllen

Cursor-Befehle

SCHILDKRÖTE ANZEIGEN	(ST) - Schildkrötencursor anzeigen
SCHILDKRÖTE VERSTECKEN	(HT) - Schildkröten-Cursor ausblenden
CURSORGRÖSSE Größe	(CS) - Cursorgröße einstellen

CURSORFARBE Farbe	CC) - Cursorfarbe einstellen
STEMPEL	– Zeichne eine Schildkröte an der aktuellen x,y-Position

Befehle zur Statusverwaltung

RESET [anzeigen] show = 1	- Bildschirm löschen und alles zurücksetzen, Schildkröte anzeigen, wenn
PUSH	- Aktuelle Position und Richtung im Stapel speichern
POP	– Position und Richtung aus dem Stapel wiederherstellen

Der Code kann Kreise, Rechtecke und Polygone mit einer strukturierten Füllung ausfüllen. Lass ihn im Modus 2 auf einem RP2040-VGA-System oder im Modus 3 auf einem RP2350-VGA- oder HDMI-System laufen.

Füllmuster

0: Vollflächige Füllung
1: Schachbrettmuster
2: Vertikale Linien
3: Horizontale Linien
4: Diagonales Kreuz
5: Diagonale Streifen
6: Kreuzschraffur
7: Feine Diagonale
8: Dichtes Schachbrettmuster
9: Diagonale rechts mittel
10: Diagonal links mittel
11: Vertikale Linien mittel
12: Horizontale Linien, mittel
13: Großes Schachbrettmuster
14: Vertikal gepunktet
15: Enge horizontale Streifen
16: Gitter
17: Webmuster
18: Rautenmuster
19: Diagonaler Farbverlauf
20: Diagonaler Farbverlauf umgekehrt
21: Rand/Rahmen
22: Vertikale Teilung
23: Gewebt
24: Spärliche Punkte
25: Diagonal sehr fein
26: Pfeil nach oben
27: Dichte Punkte
28: Chevron
29: Hohlraute
30: Kreis
31: Gefüllter Kreis

Anhang I
Spezielle Tastaturtasten

MMBasic macht ein einziges Zeichen für die Funktionstasten und andere Sondertasten auf der Tastatur.
Hinweis: Die DE-USB-Tastatur gibt die Codes 200-209 für Tastenzeichen mit Akzenten
bei normaler Eingabe (input$/inkey$)

Die Codes sind in dieser Tabelle als Hexadezimal- und Dezimalzahlen angegeben:

Tastaturtaste	Tastencode (Hex)	Tastencode (dezimal)
DEL	7F	127
Pfeil nach oben	80	128
Pfeil nach unten	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 runter	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
SHIFT_TAB	9F	159
SHIFT_DEL	A0	160
SHIFT_DOWN_ARROW	A1	161
SHIFT_RIGHT_ARROW	A3	163

** funktioniert auch für VT100-Emulatoren

Bei angeschlossenen PS2- und USB-Tastaturen wird, wenn die Umschalttaste gleichzeitig mit den Funktionstasten F1 bis F12 gedrückt wird, 20 (hex) zum Code hinzugefügt (dies entspricht der Einstellung von Bit 5). Beispielsweise erzeugt Umschalt-F10 BA (hex).

Der Umschaltmodifikator funktioniert mit den Funktionstasten F1 bis F12; er wird für die anderen Tasten außer TAB, DEL, DOWN_ARROW und RIGHT_ARROW, wie oben angegeben, ignoriert. MMBasic übersetzt die meisten VT100-Escape-Codes, die von Terminalemulatoren wie TeraTerm und Putty erzeugt werden, in diese Codes (der Umschaltmodifikator funktioniert nur für F3-F8). Das heißt, dass ein Terminalemulator, der über einen USB- oder seriellen Anschluss als Konsole läuft, die gleichen Tastencodes erzeugt wie eine direkt angeschlossene Tastatur.

Anhang J
Programmieren in BASIC – Ein Tutorial

Die Sprache BASIC wurde 1964 vom Dartmouth College in den USA als Computersprache für den Programmierunterricht eingeführt und ist daher einfach zu verwenden und zu erlernen. Gleichzeitig hat sie sich als kompetente und leistungsfähige Programmiersprache bewährt und war daher in den späten 70er und frühen 80er Jahren sehr beliebt. Auch heute noch sind einige große kommerzielle Datensysteme in der Sprache BASIC (hauptsächlich Pick Basic) geschrieben.

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 grob nachahmt.

Für Programmierer ist der größte Vorteil von BASIC, dass es so einfach zu benutzen ist. Einige modernere Sprachen wie C und C++ können echt kompliziert sein, aber mit BASIC kannst du mit einem einzeiligen Programm anfangen und schon was Sinnvolles damit machen. MMBasic ist auch cool, weil du damit anspruchsvolle Grafiken zeichnen, die externen E/A-Pins für die Steuerung anderer Geräte manipulieren und über eine Reihe integrierter Kommunikationsprotokolle mit anderen Geräten kommunizieren kannst.

Befehlszeile

Die Interaktion mit der MMBasic- erfolgt über die Konsole an der Eingabeaufforderung (d. h. dem Größer-als-Zeichen (>) auf der Konsole). Beim Start zeigt MMBasic die Eingabeaufforderung an und wartet darauf, dass ein Befehl eingegeben wird. Es kehrt auch zur Eingabeaufforderung zurück, wenn dein Programm beendet wird oder eine Fehlermeldung ausgibt.

Wenn die Befehlszeile angezeigt wird, kannst du eine Vielzahl von Befehlen eingeben und ausführen. In der Regel werden damit das im Speicher befindliche Programm aufgelistet (LIST) oder bearbeitet (EDIT) oder vielleicht einige Optionen festgelegt (Befehl OPTION). Meistens lautet der Befehl einfach RUN, wodurch MMBasic angewiesen wird, das im Programmspeicher befindliche Programm auszuführen.

Fast jeder Befehl kann an der Eingabeaufforderung eingegeben werden, was oft genutzt wird, 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
```

Es überrascht nicht, dass MMBasic die Zahl 4 ausgibt, bevor es zur Befehlszeile zurückkehrt.

Diese Möglichkeit, einen Befehl an der Befehlszeile zu testen, ist nützlich, wenn du das Programmieren in BASIC lernst. Daher lohnt es sich, einen Raspberry Pi Pico mit der PicoMite-Firmware zur Hand zu haben, um während der Arbeit mit diesem Tutorial gelegentlich Tests durchzuführen.

Struktur eines BASIC-Programms

Ein BASIC-Programm beginnt an der ersten Zeile und läuft bis zum Ende oder bis es auf einen END-Befehl trifft – an diesem Punkt zeigt MMBasic die Befehlszeile (>) auf der Konsole an und wartet auf eine Eingabe.

Ein Programm besteht aus einer Reihe von Anweisungen oder Befehlen, von denen jeder den BASIC-Interpreter dazu veranlasst, etwas zu tun (die Begriffe *„Anweisung”* und *„Befehl”* haben im Allgemeinen dieselbe Bedeutung 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. Zeilennummern waren in den frühen BASIC-Interpretern obligatorisch, moderne Implementierungen (wie MMBasic) benötigen sie jedoch nicht. Du kannst sie weiterhin verwenden, wenn du möchtest, aber sie haben keinen Vorteil und überladen in der Regel nur deine Programme.

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 Befehl GOTO verwendet werden kann. Dies wird näher erläutert, wenn wir den Befehl GOTO behandeln, aber hier ist ein Beispiel (der Labelname 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 eingefügt werden und erstreckt sich bis zum Ende der Zeile. 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 Personen, die mit dem Programm nicht vertraut sind, allgemein darüber zu informieren, wie es funktioniert und was es tut. Denken Sie daran, dass Sie sich nach nur wenigen Monaten nicht mehr an ein von Ihnen geschriebenes Programm erinnern können und es Ihnen seltsam vorkommen wird, wenn Sie es wieder zur Hand nehmen. Aus diesem Grund werden Sie sich später dafür bedanken, wenn Sie viele Kommentare verwenden.

Hier sind ein paar Beispiele für Kommentare:

```
' berechne die Hypotenuse
PRINT SQR(a * a + b * b)
```

oder

```
INPUT var    ' Temperatur abfragen
```

Ältere BASIC-Programme haben den Befehl REM benutzt, um einen Kommentar zu starten, und du kannst den auch benutzen, wenn du willst, aber das einfache Anführungszeichen ist einfacher und praktischer.

Der Befehl PRINT

Es gibt eine Reihe von grundlegenden Befehlen, die wir in diesem Tutorial behandeln werden, aber der wohl nützlichste ist der Befehl PRINT. Seine Aufgabe ist einfach: etwas auf der Konsole ausgeben. Er wird hauptsächlich verwendet, um Daten anzuzeigen (z. B. das Ergebnis von Berechnungen) oder informative Meldungen auszugeben.

PRINT ist auch nützlich, wenn du einen Fehler in deinem Programm suchst. Du kannst damit die Werte von Variablen ausgeben und Meldungen an wichtigen Stellen während der Ausführung des Programms anzeigen.

In seiner einfachsten Form gibt der Befehl einfach alles aus, was in der Befehlszeile steht. Zum Beispiel:

```
PRINT 54
```

zeigt auf der Konsole die Zahl 54 gefolgt von einer neuen Zeile an.

Die auszugebenden Daten können einfach wie hier sein oder ein Ausdruck, also etwas, das berechnet werden muss. Wir werden später noch genauer auf Ausdrücke eingehen, 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 – du musst es nicht eingeben.

Weitere Beispiele für den Befehl PRINT 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 „Die erste Zahl von  ist“ 20+25 „und die zweite ist“
18/3
Die erste Zahl ist 45 und die zweite ist 6
>
```

Normalerweise werden die Werte durch ein Leerzeichen getrennt, wie im vorherigen Beispiel gezeigt, aber du kannst die Werte auch durch ein Komma (,) trennen. Durch das Komma wird zwischen den beiden Werten ein Tabulator eingefügt. In MMBasic sind die Tabulatoren im PRINT-Befehl acht Zeichen voneinander entfernt.

Um die Tabulatorfunktion zu veranschaulichen, gibt der folgende Befehl eine mit Tabulatoren getrennte 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 (-), falls 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 nach dem Drucken des gesamten Textes in eine neue Zeile springt. Zum Beispiel:

```
PRINT "Das wird";
PRINT „ in einer einzigen Zeile gedruckt.“
```

Das ergibt diese 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 erklären, was eine „Variable" ist, da sie für die BASIC-Sprache (und eigentlich für die meisten Programmiersprachen) echt wichtig ist. Eine Variable ist einfach ein Ort, an dem Daten (also ihr „Wert") gespeichert werden. Dieser Wert kann sich während der Ausführung des Programms ändern, deshalb heißt sie „Variable".

Variablen in MMBasic können drei verschiedene Typen haben. Am häufigsten wird der Typ „Gleitkomma" verwendet, der automatisch angenommen wird, wenn der Typ der Variablen nicht angegeben wird. Die beiden anderen Typen sind „Ganzzahl" und „Zeichenkette", auf die wir später noch eingehen werden. Eine Gleitkommazahl ist eine normale Zahl, die einen Dezimalpunkt enthalten kann. Beispielsweise sind 3,45, -0,023 oder 100,00 allesamt Gleitkommazahlen.

Eine Variable kann zum Speichern einer Zahl verwendet werden und dann genauso wie die Zahl selbst genutzt werden. In diesem Fall repräsentiert sie den Wert der letzten Zahl, die ihr zugewiesen wurde.

Ein einfaches Beispiel:

```
A = 3
B = 4
PRINT A + B
```

zeigt die Zahl 7 an. In diesem Fall sind sowohl A als auch B Variablen, und MMBasic hat ihre aktuellen Werte in der PRINT-Anweisung verwendet. MMBasic erstellt automatisch eine Variable, wenn es zum ersten Mal auf sie trifft. 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, Unterstrichen oder Punkten bestehen kann. Der Name kann bis zu 31 Zeichen lang sein, wobei die Groß- und Kleinschreibung keine Rolle spielt. Hier sind einige Beispiele:

```
Total_Count
ForeColour
temp3
count
x
DasIstEinSehrLangerVariablenname
Inkrement.Wert
```

Du kannst den Wert einer Variablen überall in deinem Programm ändern, indem du den Zuweisungsbefehl benutzt, z. B.:

```
variable = Ausdruck
```

Zum Beispiel:

```
temp3 = 24,6
count = 5
CTemp = (FTemp - 32) * 0,5556
```

Im letzten Beispiel sind sowohl CTemp als auch FTemp Variablen. Diese Zeile wandelt den Wert von FTemp (in Grad Fahrenheit) in Grad Celsius um und speichert das Ergebnis in der Variablen CTemp.

Ausdrücke

Wir haben den Begriff „Ausdruck" schon mal in diesem Tutorial gesehen, und in der Programmierung hat er eine bestimmte Bedeutung. Es ist eine Formel, die vom BASIC-Interpreter in eine einzelne Zahl oder einen Wert umgewandelt werden kann.

MMBasic wertet numerische Ausdrücke nach den gleichen Regeln aus, die wir in der Schule gelernt haben. Zum Beispiel werden Multiplikation und Division zuerst ausgeführt, gefolgt von Addition und Subtraktion. Diese Regeln werden als Vorrangregeln bezeichnet und sind zuvor in diesem Handbuch ausführlich beschrieben worden (siehe Kapitel *„Ausdrücke und Operatoren*").

Das heißt, MMBasic berechnet 2 + 3 * 6, indem es zuerst 3 mit 6 multipliziert, was 18 ergibt, und dann 2 addiert, was einen Endwert von 20 ergibt. Ebenso werden sowohl 5 * 4 als auch 10 + 4 * 3 – 2 zu 20 berechnet.

Wenn du den Interpreter dazu bringen willst, Teile des Ausdrucks zuerst zu berechnen, kannst du diesen Teil des Ausdrucks in Klammern setzen. Zum Beispiel ergibt (10 + 4) * (3 – 2) 14 und nicht 20, wie es ohne Klammern der Fall gewesen wäre. 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 schon erwähnt, kannst du Variablen in einem Ausdruck genauso wie Zahlen verwenden. Zum Beispiel erhöht das den Wert der Variablen `temp` um eins:

```
temp = temp + 1
```

Du kannst auch Funktionen in Ausdrücken verwenden. Das sind spezielle Operationen, die MMBasic anbietet, zum Beispiel um trigonometrische Werte zu berechnen.

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 berechnet diesen Ausdruck, indem es zuerst `a` mit `a` multipliziert, dann `b` mit `b` multipliziert und schließlich 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 zur Anzeige an den Befehl PRINT zurückgibt.

Einige andere mathematische Funktionen, die MMBasic bietet, sind:

SIN(r) – der Sinus von r
COS(r) – der Kosinus von r
TAN(r) – der Tangens von r

Es gibt noch viele weitere Funktionen, die dir zur Verfügung stehen und 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).

Eine weitere Eigenschaft der meisten Programmiersprachen (einschließlich BASIC) ist, dass du Funktionsaufrufe ineinander verschachteln kannst. Wenn du zum Beispiel den Winkel in Grad (also „d") hast, kannst du den Sinus dieses Winkels mit diesem Ausdruck berechnen:

```
PRINT SIN(RAD(d))
```

In diesem Fall nimmt MMBasic zuerst den Wert von d und wandelt ihn mit der Funktion RAD() in Radianten um. Das Ergebnis dieser Funktion wird dann in die Funktion SIN() eingegeben.

Die IF-Anweisung

Entscheidungen zu treffen ist ein zentraler Bestandteil 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 meistens 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 Tests, 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 anfängliche Bedingung falsch ist:

```
IF Bedingung THEN wahre Aktion ELSE falsche Aktion
```

Das führt zum Beispiel zu unterschiedlichen Aktionen, wenn die Temperatur unter 25 oder 25 oder mehr ist:

```
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
```

oder

```
WENN Bedingung DANN
    true-action
    true-action
SONST
    falsche Aktion
    false-action
ENDIF
```

Im Gegensatz zur einzeiligen IF-Anweisung kannst du viele wahre Aktionen haben, die jeweils in einer eigenen Zeile stehen, und genauso viele falsche Aktionen. Im Allgemeinen ist die einzeilige IF-Anweisung praktisch, wenn du eine einfache Aktion ausführen musst, während die mehrzeilige Version viel einfacher zu verstehen ist, wenn die Aktionen zahlreich und komplizierter sind.

Ein Beispiel für eine mehrzeilige IF-Anweisung mit mehr als einer Aktion ist:

```
IF Betrag < 100 THEN
    PRINT „Zu niedrig”
    PRINT „Mindestwert ist 100”
ELSE
    PRINT „Eingabe akzeptiert”
    Auf SD-Karte speichern
    DRUCKEN „Zweiten Betrag eingeben“
ENDIF
```

Beachte, dass im obigen Beispiel jede Aktion eingerückt ist, um zu zeigen, zu welchem Teil der IF-Struktur sie gehört. Das Einrücken ist nicht zwingend erforderlich, macht das Programm aber für jemanden, der damit nicht vertraut ist, viel verständlicher und ist daher sehr empfehlenswert.

In einer mehrzeiligen IF-Anweisung kannst du mit dem Befehl ELSE IF zusätzliche Tests machen. Das lässt sich am besten anhand eines Beispiels erklären (die Temperaturen sind alle in °C angegeben):

```
IF Temp < 0 THEN
    PRINT „Frost“
ELSE IF Temp < 20 THEN
    PRINT „Kalt”
ELSE IF Temp < 35 THEN
    DRUCKE „Warm”
SONST
    DRUCKE „Heiß”
ENDIF
```

ELSE IF nutzt die gleichen Tests wie ein normales IF (also <, <= usw.), aber der Test wird nur gemacht, wenn der vorherige Test falsch war. Du bekommst also zum Beispiel nur die Meldung *„Warm”*, wenn `Temp < 0` nicht stimmt und `Temp < 20` nicht stimmt, aber `Temp < 35` stimmt. Das letzte ELSE fängt den Fall ab, in dem alle Tests falsch waren.

Ein Ausdruck wie `Temp < 20` wird von MMBasic entweder als wahr oder falsch ausgewertet, wobei wahr den Wert eins und falsch den Wert null 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.

Ähnlich wird das Folgende `0` ausgeben, 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, sondern wertet die Bedingung einfach aus und nimmt bei einem Ergebnis von Null „falsch“ und bei einem Ergebnis ungleich Null „wahr“ an.

Dies ermöglicht einige praktische Abkürzungen. Wenn beispielsweise `BalanceCorrect` eine Variable ist, die wahr (ungleich Null) ist, wenn eine bestimmte Funktion des Programms korrekt ist, kann Folgendes verwendet werden, um eine Entscheidung auf der Grundlage dieses Werts zu treffen:

```
IF BalanceCorrect THEN …etwas tun…
```

FOR-Schleifen

Eine weitere häufige Anforderung in der Programmierung ist die Wiederholung einer Reihe von Aktionen. Du möchtest vielleicht alle sieben Tage der Woche durchlaufen und für jeden Tag die gleiche Funktion ausführen. BASIC bietet für diese Art von Aufgabe die FOR-Schleifenkonstruktion, die wie folgt funktioniert:

```
FOR day = 1 TO 7
  Mach was basierend auf dem Wert von „day”
NEXT day
```

Zuerst wird die Variable „`day`” erstellt und mit dem Wert 1 belegt. Das Programm führt dann die folgenden Anweisungen aus, bis es zur NEXT-Anweisung kommt. 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 nach der NEXT-Anweisung fort.

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 Drucken der Zahl zur nächsten Tabulator-Spalte zu springen, und das Semikolon lässt den Cursor in dieser Zeile stehen, anstatt automatisch zur nächsten Zeile zu springen. Dadurch werden die Zahlen in ordentlichen Spalten auf der Seite gedruckt.

Das Ergebnis sieht dann so aus:

```
1       2       3       4       5       6       7       8       9       10
```

Die FOR-Schleife hat noch ein paar zusätzliche Tricks auf Lager. Mit dem Schlüsselwort STEP kannst du den Wert ändern, um den die Variable erhöht wird. So werden beispielsweise mit dem folgenden Code nur die ungeraden Zahlen gedruckt:

```
FOR nbr = 1 TO 10 STEP 2
  PRINT nbr,;
NEXT nbr
```

Der Wert des Schritts (oder Inkrementwerts) ist standardmäßig eins, wenn das Schlüsselwort STEP nicht verwendet wird, aber du kannst ihn auf eine beliebige Zahl setzen.

Wenn MMBasic die Variable erhöht, checkt es, ob die Variable 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` elf, und an diesem Punkt verlässt die Ausführung die Schleife. Dieser Test wird auch am Anfang der Schleife gemacht. Wenn zum Beispiel der Wert der Variablen am Anfang den TO-Wert überschreitet, wird die Schleife nie ausgeführt, nicht einmal.

Wenn du den STEP-Wert auf eine negative Zahl setzt, kannst du die FOR-Schleife verwenden, um von einer hohen Zahl zu einer niedrigen zu gelangen. In diesem Fall muss die Startzahl größer als die TO-Zahl sein.

Das folgende Beispiel gibt die Zahlen von 1 bis 10 in umgekehrter Reihenfolge aus:

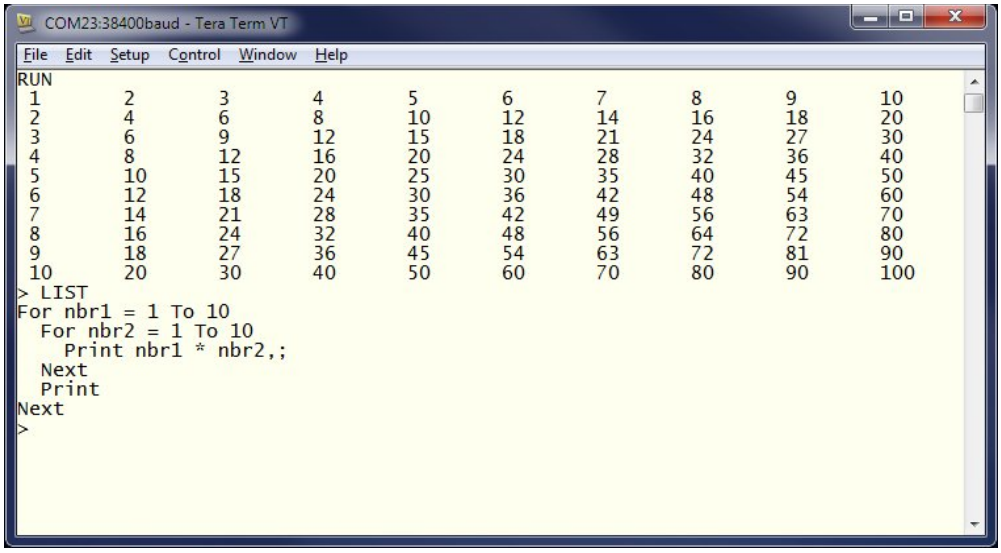
```
FOR nbr = 10 TO 1 STEP -1
  PRINT nbr,;
NEXT nbr
```

Multiplikationstabelle

Um noch besser zu zeigen, wie Schleifen funktionieren und wie nützlich sie sein können, benutzt das folgende kurze Programm zwei FOR-Schleifen, um die Multiplikationstabelle auszugeben, die wir alle in der Schule gelernt haben. Das Programm dafür ist nicht kompliziert:

```
FOR nbr1 = 1 to 10
  FOR nbr2 = 1 bis 10
    PRINT nbr1 * nbr2,;
  NEXT nbr2
  PRINT
NÄCHSTE nbr1
```

Die Ausgabe siehst du im folgenden Screenshot, wo auch das Programm aufgelistet ist.



Du musst die Logik dieses Beispiels Zeile für Zeile durchgehen, um zu verstehen, was es macht. Im Wesentlichen besteht es aus einer Schleife innerhalb einer anderen. Die innere Schleife, die die Variable `nbr2` erhöht, druckt eine horizontale Zeile der Tabelle. Wenn diese Schleife beendet ist, führt sie den folgenden PRINT-Befehl aus, der nichts zu drucken hat – also gibt sie einfach eine neue Zeile aus (d. h. beendet die von der inneren Schleife gedruckte Zeile).

Das Programm führt dann eine weitere Iteration der äußeren Schleife aus, indem es `nbr1` erhöht und die innere Schleife erneut ausführt. Wenn die äußere Schleife schließlich beendet ist (wenn `nbr1` größer als 10 ist), erreicht das Programm das Ende und wird beendet.

Ein letzter Hinweis: Du kannst den Variablennamen in der NEXT-Anweisung weglassen, und MMBasic errät, 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 beenden, indem du eine durch Kommas getrennte Liste von Variablen in der NEXT-Anweisung verwendest. Zum Beispiel:

```
FOR var1 = 1 TO 5
  FOR var2 = 10 to 13
    PRINT var1 * var2
NEXT var1, var2
```

DO-Schleifen

Eine andere Art von Schleife ist die DO…LOOP-Struktur, die so aussieht:

```
DO WHILE Bedingung
   <Anweisung>
   <Anweisung>
LOOP
```

Zuerst wird die *Bedingung* geprüft. Wenn sie wahr ist, werden die Anweisungen ausgeführt, bis der Befehl LOOP erreicht wird. Dann springt das Programm zurück zur DO-Anweisung und die *Bedingung* wird nochmal geprüft. Wenn sie immer noch wahr ist, wird die Schleife nochmal ausgeführt. Die Bedingung ist die gleiche wie beim IF-Befehl (z. B. `X < Y`).

Das folgende Beispiel druckt beispielsweise 4 Sekunden lang das Wort „Hello” auf der Konsole 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 zu `Timer`

(wie oben) oder beim Einschalten des PicoMite.

Eine Variante der DO-LOOP-Struktur ist die folgende:

```
DO
   <Anweisung>
   <Anweisung>
LOOP UNTIL Bedingung
```

Hier wird die Schleife erst mal durchlaufen, dann wird die *Bedingung* geprüft. Wenn die Bedingung nicht stimmt, wird die Schleife so oft wiederholt, bis die *Bedingung* stimmt. Denk dran, dass die Prüfung in LOOP UNTIL das Gegenteil von DO WHILE ist.

Ähnlich wie im vorherigen Beispiel wird auch das Folgende vier Sekunden lang „Hallo“ ausgeben:

```
Timer = 0
DO
  PRINT „Hallo“
LOOP UNTIL Timer >= 4000
```

Beide Formen der DO-LOOP-Schleife machen im Grunde dasselbe, also kannst du die Struktur nehmen, die am besten zu dem passt, was du machen willst.

Schließlich ist es auch möglich, eine DO-Schleife ohne Bedingungen zu verwenden, z. B.

```
DO
   <Anweisung>
   <Anweisung>
LOOP
```

Diese Konstruktion wird die Schleife endlos fortsetzen, und du als Programmierer musst eine Möglichkeit bereitstellen, die Schleife explizit zu verlassen (dies geschieht mit dem Befehl EXIT DO). Beispiel:

```
Timer = 0
DO
  PRINT „Hallo“
  IF Timer >= 4000 THEN EXIT DO
LOOP
```

Konsoleneingabe

Deine Programme sollen nicht nur Daten für den Benutzer ausgeben, sondern auch Eingaben vom Benutzer entgegennehmen. Dazu musst du Tastenanschläge von der Konsole erfassen, was mit dem Befehl INPUT möglich ist. In seiner einfachsten Form lautet der Befehl:

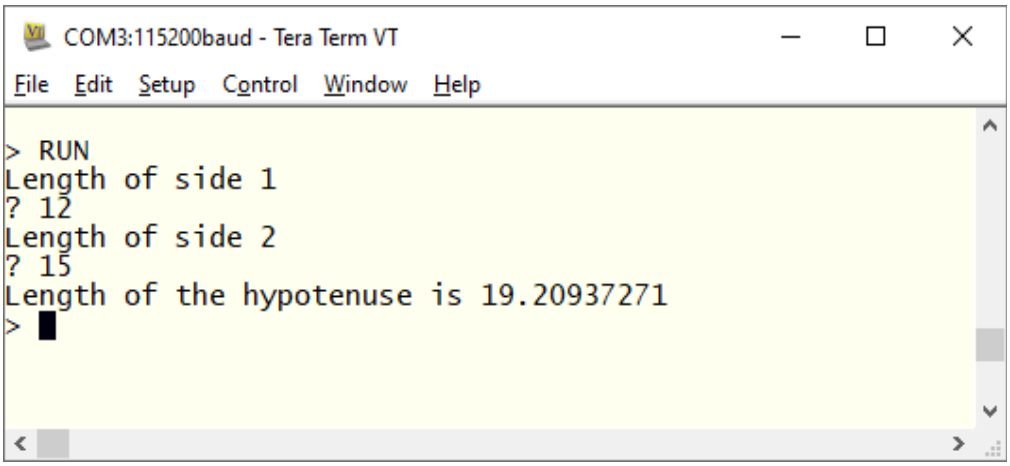
```
INPUT var
```

Dieser Befehl zeigt ein Fragezeichen auf dem Konsolenbildschirm an und wartet darauf, dass eine Zahl eingegeben und die Eingabetaste gedrückt wird. Diese Zahl wird dann der Variablen `var` zugewiesen.

Das folgende Programm erweitert zum Beispiel die Formel zur Berechnung der Hypotenuse eines Dreiecks, indem es dem Benutzer ermöglicht, die Längen der anderen Seiten über die Konsole einzugeben.

```
PRINT „Länge der Seite 1”
INPUT a
PRINT „Länge der Seite 2”
INPUT b
PRINT „Länge der Hypotenuse ist“ SQR(a * a + b * b)
```

Hier ist ein Screenshot einer typischen Sitzung:



Der Befehl INPUT kann auch deine Eingabeaufforderung ausgeben, sodass du keinen separaten PRINT-Befehl brauchst. Das funktioniert zum Beispiel genauso wie das obige Programm:

```
INPUT „Länge der Seite 1”; a
INPUT "Länge der Seite 2"; b
PRINT "Die Länge der Hypotenuse ist" SQR(a * a + b * b)
```

Mit dem Befehl INPUT kannst du schließlich eine Reihe von durch Kommas getrennten Zahlen eingeben, wobei jede Zahl in einer anderen Variablen gespeichert wird.

Zum Beispiel:

```
INPUT "Gib die Länge der beiden Seiten ein: ", a, b
PRINT "Die Länge der Hypotenuse ist" SQR(a * a + b * b)
```

Wenn der Benutzer `12,15` eingibt, wird die Zahl `12` in der Variablen `a` und `15` in `der Variablen b` gespeichert.

Eine andere Möglichkeit, Eingaben von der Konsole zu bekommen, ist der Befehl LINE INPUT. Damit wird die ganze Zeile, die der Benutzer eingegeben hat, in eine String-Variable gespeichert. Wie beim Befehl INPUT kannst du auch hier eine Eingabeaufforderung festlegen. Hier ein einfaches Beispiel:

```
LINE INPUT „Wie heißt du?“, s$
PRINT „Hallo “ s$
```

Wir werden später in diesem Tutorial auf Zeichenfolgenvariablen eingehen, aber im Moment kannst du sie dir als Variablen vorstellen, die eine Folge von Zeichen enthalten. Wenn du das obige Programm ausführst und bei der Eingabeaufforderung `„John“` eingibst, antwortet das Programm mit `„Hallo John“`.

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 Zeichenfolge mit nur einem Zeichen oder als leere Zeichenfolge (d. h. ohne Zeichen) zurückgibt, wenn nichts eingegeben wurde.

GOTO und Labels

Eine Methode zur Steuerung des Programmablaufs ist der Befehl GOTO. Dieser weist MMBasic im Wesentlichen an, zu einem anderen Teil des Programms zu springen und die Ausführung von dort aus fortzusetzen. Das Ziel von GOTO ist ein Label, das zunächst erklärt werden muss.

Ein Label ist ein Bezeichner, der einen Teil des Programms markiert. Er muss an erster Stelle in der Zeile stehen und mit einem 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 mit dem Befehl GOTO zu diesem bestimmten Teil des Programms springen willst, würdest du den Befehl wie folgt verwenden:

```
GOTO LoopBack
```

Um das Ganze in einen Zusammenhang zu bringen, gibt das folgende Programm alle Zahlen von 1 bis 10 aus:

```
z = 0
LoopBack:  z = z + 1
PRINT z
IF z < 10 THEN GOTO LoopBack
```

Das Programm fängt damit an, die Variable `z` auf Null zu setzen und sie dann in der nächsten Zeile auf 1 zu erhöhen. Der Wert von `z` wird ausgegeben und dann geprüft, ob er kleiner als 10 ist. Wenn er kleiner als 10 ist, springt die Programmausführung zurück zum Label `LoopBack`, wo der Vorgang wiederholt wird. Irgendwann erreicht der Wert von `z` den Wert 10, und das Programm läuft bis zum Ende durch und wird beendet.

Beachte, dass eine FOR-Schleife das Gleiche machen kann (und einfacher ist), also ist dieses Beispiel nur dazu da, um zu zeigen, was der GOTO-Befehl kann.

Früher hatte der GOTO-Befehl einen schlechten Ruf. 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 solche Programme sind für andere Programmierer fast unmöglich zu verstehen. Mit Konstrukten wie mehrzeiligen 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.

Prüfung auf Primzahlen

Das folgende einfache Programm vereint viele der zuvor besprochenen Programmierfunktionen.

```
DO
  InpErr:
  PRINT
  INPUT „Gib eine Zahl ein: "; a
  IF a < 2 THEN
    PRINT „Die Zahl muss mindestens 2 sein"
    GOTO InpErr
  ENDIF

  Divs = 0
  FÜR x = 2 BIS SQR(a)
    r = a/x
    WENN r = FIX(r) DANN Divs = Divs + 1
  NÄCHSTES x

  DRUCKE a " ist ";
  WENN Divs > 0 DANN DRUCKE "nicht ";
  DRUCKE „eine Primzahl."
LOOP
```

Zuerst wird (auf der Konsole) nach einer Zahl gefragt. Wenn sie eingegeben wurde, wird geprüft, ob es sich um eine Primzahl handelt, und eine entsprechende Meldung angezeigt.

Es fängt mit einer DO-Schleife ohne Bedingung an – also läuft es immer weiter. Das ist genau das, was wir wollen. Das heißt, wenn der Benutzer eine Zahl eingegeben hat, wird gemeldet, ob es sich um eine Primzahl handelt oder nicht, und dann wird eine weitere Zahl abgefragt. Der Benutzer kann das Programm beenden (wenn er will), indem er das Abbruchzeichen (normalerweise STRG-C) eingibt.

Das Programm zeigt dann eine Eingabeaufforderung für den Benutzer an, die mit einem Semikolon beendet wird. Das heißt, der Cursor bleibt am Ende der Eingabeaufforderung für den Befehl INPUT stehen, der die Zahl abfragt und in der Variablen a speichert.

Danach wird die Zahl geprüft. Wenn sie kleiner als 2 ist, wird eine Fehlermeldung angezeigt und das Programm springt zurück und fragt erneut nach der Zahl.

Jetzt können wir prüfen, ob die Zahl eine Primzahl ist. Das Programm nutzt eine FOR-Schleife, um die möglichen Teiler durchzugehen und zu prüfen, ob jeder einzelne die eingegebene Zahl gleichmäßig teilen kann. Jedes Mal, wenn das passiert, erhöht das Programm die Variable `Divs`.

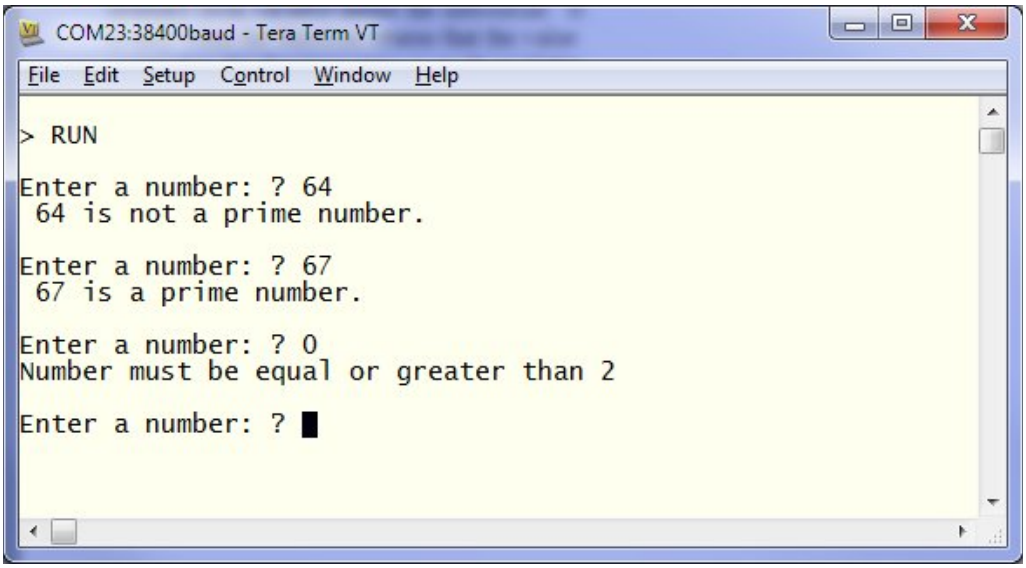
Beachte, dass die Prüfung mit der Funktion FIX(r) durchgeführt wird, die einfach alle Ziffern nach dem Dezimalpunkt entfernt. Die Bedingung `r = FIX(r)` ist also wahr, wenn `r` eine ganze Zahl ist (d. h. keine Ziffern nach dem Dezimalpunkt hat).

Schließlich erstellt das Programm die Meldung für den Benutzer. Wichtig ist, dass, wenn die Variable `Divs` größer als Null ist, eine oder mehrere Zahlen gefunden wurden, die die Testzahl gleichmäßig teilen können. In diesem Fall fügt die IF-Anweisung das Wort „nicht" in die Ausgabemeldung ein.

Wenn die eingegebene Zahl zum Beispiel 21 war, sieht der Benutzer diese Antwort:

```
21 ist keine Primzahl.
```

Dies ist das Ergebnis der Ausführung des Programms und ein Teil der Ausgabe:



Du kannst dieses Programm testen, indem du es mit dem Editor (dem Befehl EDIT) eingibst.

Mit deinen neu erworbenen Kenntnissen kannst 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, um 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. Dazu kannst du den Befehl GOTO verwenden oder den Befehl EXIT FOR, der die FOR-Schleife sofort beendet.

Weitere Effizienzsteigerungen sind möglich, indem man nur die Division mit ungeraden Zahlen testet (indem man zunächst eine gerade Zahl testet, dann die FOR-Schleife bei 3 startet und STEP 2 verwendet) oder indem man nur Primzahlen für den Test verwendet (was allerdings viel komplizierter wäre).

Arrays

Arrays sind etwas, das du auf den ersten Blick wahrscheinlich nicht für nützlich hältst, aber wenn du sie brauchst, wirst du sie tatsächlich als sehr praktisch empfinden.

Ein Array lässt sich am besten als eine Reihe von Briefkästen für einen Block von Wohnungen oder Eigentumswohnungen vorstellen, wie rechts dargestellt. Die Briefkästen befinden sich alle an derselben Adresse, und jeder Kasten steht für eine Wohnung oder Eigentumswohnung an dieser Adresse. Du kannst einen Brief in den Kasten für Wohnung eins, Wohnung zwei usw. legen.

Ähnlich ist ein Array in BASIC eine einzelne Variable mit mehreren Untereinheiten (in BASIC als *Elemente* bezeichnet), die nummeriert sind. Du kannst Daten in Element eins, Element zwei usw. legen. 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 zum Beispiel das Element Nummer 100 in diesem Array auf (sagen wir) die Zahl 876 setzen willst, machst du das so:

```
numarr(100) = 876
```

Normalerweise ist der Index eines Arrays keine konstante Zahl wie in diesem Beispiel (also 100), sondern eine Variable, die geändert werden kann, um auf verschiedene Array-Elemente zuzugreifen.

Als Beispiel für die Verwendung eines Arrays stell dir vor, du möchtest die Höchsttemperatur für jeden Tag des Jahres aufzeichnen und am Ende des Jahres den Gesamtdurchschnitt berechnen. Du könntest normale Variablen verwenden, um die Temperatur für jeden Tag aufzuzeichnen, aber du würdest 365 davon benötigen, was dein Programm ziemlich unhandlich machen würde. 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 Nummer des Tages 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 einfach, den Durchschnitt für das Jahr zu berechnen.
Zum Beispiel:

```
Gesamt = 0
FÜR i = 1 bis 365
  Gesamt = Gesamt + Tage(i)
NEXT i
PRINT „Durchschnitt ist:“  Gesamt / 365
```

Das ist viel einfacher, als 365 einzelne Variablen zu addieren und zu mitteln.

Das obige Array war eindimensional, aber du kannst auch mehrere Dimensionen haben. Wenn wir wieder zu unserem Beispiel mit den Briefkästen zurückkommen, könnte man sich ein zweidimensionales Array wie ein mehrstöckiges Wohnhaus vorstellen. Ein Block könnte eine Reihe

mit vier Briefkästen für die erste Etage haben, eine weitere Reihe mit vier Briefkästen für die zweite Etage und so weiter. Um einen Brief in einen Briefkasten zu werfen, musst du die Etagennummer und die Nummer der Einheit auf dieser Etage angeben.

In BASIC wird so ein Array mit zwei durch ein Komma getrennten Indizes angegeben. Zum Beispiel:

```
LetterBox(Etage, Einheit)
```

Als praktisches Beispiel nehmen wir an, du musstest 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 ist eine Zahl, die das Jahr angibt. Wenn du den Tag 100 im Jahr 3 auf 24 Grad setzen 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 und mit dem RP2350-Prozessor bis zu fünf Dimensionen verwenden. Die Größe eines Arrays ist nur durch die Menge des verfügbaren freien RAM begrenzt.

Ganzzahlen

Bisher waren alle Zahlen und Variablen, die wir verwendet haben, Gleitkommazahlen. Wie schon gesagt, sind Gleitkommazahlen praktisch, weil sie die Stellen nach dem Komma verfolgen und bei Divisionen ein sinnvolles Ergebnis liefern. Wenn du also einfach nur was erledigen willst und dich nicht um die Details kümmerst, solltest du bei Gleitkommazahlen bleiben.

Die Einschränkung von Gleitkommazahlen besteht jedoch darin, dass sie Zahlen als Annäherung mit einer Genauigkeit von 14 Stellen in der PicoMite-Firmware speichern. In den meisten Fällen ist diese Eigenschaft von Gleitkommazahlen kein Problem, aber es gibt einige Fälle, in denen du größere Zahlen genau speichern musst.

Nehmen wir zum Beispiel an, du möchtest die Zeit auf die Mikrosekunde genau manipulieren , damit du zwei verschiedene Datums-/Zeitangaben vergleichen und herausfinden kannst, welche davon früher ist. Der einfachste Weg, dies zu tun, besteht darin, die Datums-/Zeitangabe in die Anzahl der Mikrosekunden seit einem bestimmten Datum (z. B. dem 1. Januar des Jahres Null) umzuwandeln – dann ist es nur noch eine Frage der arithmetischen Vergleichung in einer IF-Anweisung, um das frühere der beiden Daten zu ermitteln.

Das Problem ist, dass die Anzahl der Mikrosekunden seit diesem Datum den Genauigkeitsbereich von Gleitkommavariablen überschreitet, und hier kommen Integer-Variablen ins Spiel. Eine Integer-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 Brüche (d. h. Zahlen nach dem Komma) speichern kann. Jede Berechnung, die ein gebrochenes Ergebnis liefert, wird bei der Zuweisung zu einer Ganzzahl auf die nächste ganze Zahl auf- oder abgerundet. Diese Eigenschaft kann jedoch bei der Arbeit mit Geld nützlich sein – zum Beispiel möchtest du niemandem eine Rechnung über 100,13333333333 $ schicken.

Eine Ganzzahlvariable lässt sich ganz einfach erstellen, indem man das Prozentzeichen (%) als Suffix an einen Variablennamen anhängt. `sec%` ist zum Beispiel eine Ganzzahlvariable. Innerhalb eines Programms kann man Ganzzahlen und Gleitkommazahlen mischen, und MMBasic nimmt die erforderlichen Umrechnungen vor. Wenn man jedoch die volle Genauigkeit von Ganzzahlen beibehalten möchte, sollte man eine Vermischung der beiden vermeiden.

Genau wie bei Gleitkommazahlen kannst du auch Arrays von Ganzzahlen erstellen. Dazu musst du lediglich das Prozentzeichen als Suffix an den Array-Namen anhängen. Ein Beispiel: `days%(365, 5)`.

Anfänger sind oft verwirrt, wann sie Gleitkommazahlen und wann Ganzzahlen verwenden sollen. 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.

Zeichenfolgen

Zeichenfolgen sind ein weiterer Variablentyp (wie Gleitkommazahlen und Ganzzahlen). Zeichenfolgen werden verwendet, um eine Folge von Zeichen zu speichern. Zum Beispiel im Befehl:

```
PRINT "Hallo"
```

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.

Zeichenfolgenvariablennamen verwenden das Dollarzeichen ($) als Suffix, um sie als Zeichenfolge statt als normale Gleitkommavariable zu kennzeichnen, und du kannst ihre Werte mit einer normalen Zuweisung festlegen. Hier sind ein paar Beispiele (beachte, dass das zweite Beispiel ein Array von Zeichenfolgen verwendet):

```
Car$ = "Holden"
Country$(12) = „India"
Name$ = "Fred"
```

Du kannst Zeichenfolgen auch mit dem Pluszeichen verbinden:

```
Wort1$ = "Hallo"
Wort2$ = „Welt"
Begrüßung$ = Wort1$ + " " + Wort2
```

In diesem Fall ist der Wert von `Greeting$ „Hallo Welt"`.

Zeichenfolgen kannst du auch mit Operatoren wie = (gleich), <> (ungleich), < (kleiner als) usw. vergleichen. Zum Beispiel:

```
IF Car$ = "Holden" THEN PRINT "War ein in Australien
hergestelltes Auto"
```

Der Vergleich wird mit dem vollständigen ASCII-Zeichensatz gemacht, sodass ein Leerzeichen vor einem druckbaren Zeichen steht. Außerdem wird beim Vergleich zwischen Groß- und Kleinschreibung unterschieden, sodass „holden" nicht gleich „Holden" ist. Mit der Funktion UCASE() kannst du die Zeichenfolge in Großbuchstaben umwandeln und dann einen Vergleich ohne Berücksichtigung der Groß-/Kleinschreibung durchführen. Zum Beispiel:

```
IF UCASE$(Car$) = "HOLDEN" THEN PRINT "War ein in Australien
hergestelltes Auto"
```

Du kannst Zeichenfolgen-Arrays verwenden, musst aber bei ihrer 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 belegt beispielsweise standardmäßig 25 KB RAM.

Um das zu vermeiden, kannst du den LENGTH-Qualifizierer benutzen, um die maximale Größe jedes Elements zu begrenzen. Wenn du zum Beispiel weißt, dass die maximale Länge jeder Zeichenfolge, die im Array gespeichert wird, weniger als 20 Zeichen beträgt, kannst du die folgende Deklaration verwenden, um nur 20 Byte für jedes Element zuzuweisen:

```
DIM MyArray$(100) LENGTH 20
```

Das resultierende Array belegt dann nur 2 KB RAM.

Du kannst den LENGTH-Qualifizierer auch verwenden, wenn du eine normale (nicht als Array definierte) Zeichenfolgenvariable deklarierst. Dadurch sparst du 256 Byte RAM, wenn die Länge 9

oder weniger (RP2040) bzw. 15 oder weniger (RP2350) beträgt.

Manipulieren von Zeichenfolgen

Die Bearbeitung von Zeichenfolgen ist eine der Stärken von MMBasic. 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 vom Anfang der Zeichenfolge zurück.

RIGHT$(Zeichenfolge$, nbr) Wie oben, gibt aber *nbr* Zeichen vom rechten Ende der Zeichenkette zurück.

MID$(Zeichenfolge$, 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).

Wenn zum Beispiel `S$ = „Dies ist eine Zeichenfolge“`
dann: `R$ = LEFT$(S$, 7)` würde der Wert von R$ auf „This is” gesetzt werden
und: `R$ = RIGHT$(S$, 8)` würde dazu führen, dass der Wert von R$ auf „eine Zeichenfolge” gesetzt wird
und schließlich: `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(Zeichenfolge$, Muster$) Gibt eine Zahl zurück, die die Position angibt, an der *pattern$* in *string*$ vorkommt.

Das kann man nutzen, um eine Zeichenkette innerhalb einer anderen Zeichenkette zu suchen. Die zurückgegebene Zahl ist die Position der Teilzeichenkette innerhalb der Hauptzeichenkette. Wie bei MID$() ist der Anfang der Zeichenkette die Position 1.

Wenn zum Beispiel S$ = „This is a string“
Dann: `pos = INSTR(S$, " ")`
würde dazu führen, dass `pos` auf die Position des ersten Leerzeichens in S$ (d. h. 5) gesetzt wird.

INSTR() kann mit anderen Funktionen kombiniert werden, sodass das erste **Wort** in S$ zurückgegeben wird:

```
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)
```

Das letzte Beispiel ist ziemlich kompliziert, daher lohnt es sich, 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 anhält), wenn sie als

Zeichenposition verwendet wird. In einem praktischen Programm würdest du also zuerst prüfen, ob INSTR() die Zahl Null zurückgibt, bevor du diesen Wert verwendest.

Zum Beispiel:

```
r$ = ""
pos1 = INSTR(S$, " ")
wenn pos1 > 0 DANN
  pos2 = INSTR(pos1 + 1, S$, " ")
  wenn pos2 größer als 0 ist, dann r$ = MID$(S$, pos1 + 1, pos2
- pos1)
ENDIF
```

Wissenschaftliche Notation

Bevor wir das Thema Datentypen abschließen, müssen wir noch auf Fließkommazahlen und wissenschaftliche Notation eingehen.

Die meisten Zahlen können ganz normal geschrieben werden, zum Beispiel 11 oder 24,5, aber sehr große oder kleine Zahlen sind schwieriger. Zum Beispiel wird geschätzt, dass es auf der Erde 7500000000000000000 Sandkörner gibt. Das Problem bei dieser Zahl ist, dass man leicht den Überblick darüber verliert, wie viele Nullen sie enthält, und es daher schwierig ist, sie mit einer ähnlich großen Zahl zu vergleichen.

Ein Wissenschaftler würde diese Zahl als 7,5 x 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 Notation, wenn sehr große oder kleine Gleitkommazahlen ausgegeben werden. Wenn die oben genannte Zahl beispielsweise in einer Gleitkommavariablen gespeichert wäre, würde der Befehl PRINT sie als `7,5E+18` anzeigen (dies ist die BASIC-Darstellung für 7,5 x 10^{18}). Ein weiteres Beispiel: Die Zahl 0,0000000456 würde als `4,56E-8` angezeigt werden, was 4,56 x 10^{-8}entspricht.

Du kannst auch die wissenschaftliche Schreibweise verwenden, wenn du Konstanten in MMBasic eingibst. Zum Beispiel:

```
SandGrains = 7,5E+18
```

MMBasic nutzt die wissenschaftliche Schreibweise nur für Gleitkommazahlen (nicht für ganze Zahlen). Wenn du zum Beispiel die Anzahl der Sandkörner einer ganzzahligen Variablen zuweist, zeigt der Befehl PRINT sie als normale Zahl an (mit vielen Nullen).

DIM-Befehl

Wir haben den DIM-Befehl schon mal zum Definieren von Arrays benutzt, aber er kann auch zum Erstellen von normalen Variablen verwendet werden. Du kannst zum Beispiel gleichzeitig vier String-Variablen wie folgt erstellen:

```
DIM STRING Auto, Name, Straße, Stadt
```

Beachte, dass diese Variablen mit dem Befehl DIM als Zeichenfolgen definiert wurden und wir daher das Suffix $ nicht brauchen. Die Definition allein reicht aus, damit MMBasic ihren Typ erkennt. Wenn du diese Variablen in einem Ausdruck verwendest, brauchst du ebenfalls kein Typ-Suffix. 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 ebenfalls zur Definition von Arrays verwendet werden.

Zum Beispiel:

```
DIM INTEGER seconds(200)
```

Eine andere Möglichkeit, den Typ von Variablen zu definieren, ist das Schlüsselwort AS. Zum Beispiel:

```
DIM Car AS STRING, Name AS STRING, Street AS STRING
```

Diese Methode wird von Microsoft verwendet (MMBasic versucht, die Kompatibilität mit Microsoft aufrechtzuerhalten) und ist nützlich, wenn die Variablen unterschiedliche Typen haben. Beispiel:

```
DIM Auto AS STRING, Alter AS INTEGER, Wert AS FLOAT
```

Du kannst jede dieser Methoden zum Definieren des Variablentyps verwenden, sie funktionieren alle gleich.

Der Vorteil der Definition von Variablen mit dem Befehl DIM besteht darin, dass sie klar definiert sind (vorzugsweise am Anfang des Programms) und ihr Typ (Float, Integer oder String) nicht falsch interpretiert werden kann.

Du kannst das noch verstärken, indem du die folgenden Befehle ganz oben in deinem Programm einfügst:

```
OPTION EXPLICIT
OPTION DEFAULT NONE
```

Der erste Befehl sagt MMBasic, dass alle Variablen explizit mit DIM definiert werden müssen, bevor sie benutzt werden können. Der zweite Befehl sagt, 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` gespeichert haben, aber an einer Stelle schreibst du diese versehentlich falsch als `Tmp`. Dies führt dazu, dass MMBasic automatisch eine Variable namens `Tmp` erstellt und ihren 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 du weißt, dass etwas nicht stimmt. Wenn du dagegen den Befehl OPTION EXPLICIT am Anfang deines Programms verwendest, würde MMBasic die automatische Erstellung der Variablen verweigern und stattdessen eine Fehlermeldung anzeigen, wodurch dir wahrscheinlich Kopfzerbrechen erspart bleibt.

Der Befehl OPTION DEFAULT NONE ist auch hilfreich, weil er MMBasic sagt, dass der Programmierer den Typ jeder Variablen bei der Deklaration genau angeben muss. Man vergisst leicht, den Typ anzugeben, und wenn man MMBasic den Typ automatisch zuweisen lässt, kann das zu unerwarteten Problemen führen.

Bei kleinen, schnellen und einfachen Programmen ist es okay, MMBasic die Variablen automatisch erstellen zu lassen, aber bei größeren Programmen solltest du diese Funktion immer mit OPTION EXPLICIT deaktivieren und mit OPTION DEFAULT NONE verstärken.

Wenn eine Variable erstellt wird, wird sie für Float- und Integer-Variablen auf Null gesetzt und für String-Variablen auf eine leere Zeichenfolge (d. h. sie enthält keine Zeichen) . Du kannst ihren Anfangswert bei der Erstellung 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, sodass dieses Array tatsächlich drei Elemente mit den Indexnummern 0, 1 und 2 hat. Deshalb brauchten wir drei String-Konstanten, um es zu

initialisieren.

Konstanten

Eine häufige Anforderung in der Programmierung ist es, einen Bezeichner zu definieren, der einen Wert repräsentiert, ohne dass die Gefahr besteht, dass der 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 usw. repräsentieren.

Mit dem Befehl CONST kannst du eine Konstante erstellen. Damit wird ein Bezeichner definiert, der wie eine Variable funktioniert, aber auf einen Wert gesetzt ist, der nicht geändert werden kann.

Wenn du zum Beispiel 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 kannst du dann im Programm verwenden, wo sie für den Leser 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 leicht zu finden sind und bequem von anderen Programmierern angepasst werden können (falls nötig).

Unterprogramme

Eine Unterprogramm ist ein Block von Programmcode, der in sich geschlossen ist (wie ein Modul) und von überall innerhalb deines Programms aufgerufen werden kann. Für dein Programm sieht es wie ein integrierter MMBasic-Befehl aus und kann genauso verwendet werden. Angenommen, du brauchst einen Befehl, der einen Fehler durch Ausgeben einer Meldung auf der Konsole signalisiert. Du könntest das Unterprogramm wie folgt definieren:

```
SUB ErrMsg
  PRINT "Fehler erkannt"
END SUB
```

Wenn diese Unterprogramm in dein Programm eingebettet ist, musst du nur den Befehl ErrMsg verwenden, wenn du die Meldung anzeigen möchtest. Zum Beispiel:

```
IF A < B THEN ErrMsg
```

Die Definition einer Subroutine kann an beliebiger Stelle im Programm stehen, normalerweise steht sie aber am Ende. Wenn MMBasic während der Ausführung deines Programms auf die Definition stößt, überspringt es sie einfach.

Das obige Beispiel ist in Ordnung, aber es wäre besser, wenn eine nützlichere Meldung angezeigt werden könnte, die bei jedem Aufruf der Unterroutine angepasst werden kann. Dies kann erreicht werden, indem eine Zeichenfolge als Argument (manchmal auch als Parameter bezeichnet) an die Unterroutine übergeben wird.

In diesem Fall würde die Definition der Unterroutine so aussehen:

```
SUB ErrMsg  Msg$
  PRINT "Fehler: " + Msg$
END SUB
```

Wenn du dann die Subroutine aufrufst, kannst du die auszugebende Zeichenfolge 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 verkettet, um die vollständige Fehlermeldung zu bilden.

Eine Subroutine kann beliebig viele Argumente haben, die Float-, Integer- oder String-Werte sein können, wobei jedes Argument durch ein Komma getrennt wird.

Innerhalb der Subroutine verhalten sich die Argumente wie normale Variablen, aber sie existieren nur innerhalb der Subroutine und verschwinden, wenn die Subroutine beendet wird. Du kannst Variablen mit dem gleichen Namen im Hauptprogramm haben, die dann innerhalb der Subroutine ausgeblendet werden und sich von den für die Subroutine definierten Argumenten unterscheiden.

Der Typ des zu übergebenden Arguments kann mit einem Typ-Suffix angegeben werden (d. h. $, % oder ! für Zeichenfolge, Ganzzahl und Gleitkomma). Im folgenden Beispiel muss das erste Argument eine Zeichenfolge und das zweite eine Ganzzahl sein:

```
SUB MySub Msg$, Nbr%
  …
END SUB
```

MMBasic konvertiert die übergebenen Werte, wenn möglich. Wenn dein Programm also einen Gleitkommawert als zweites Argument übergibt, konvertiert MMBasic diesen in eine Ganzzahl. Wenn MMBasic den Wert nicht konvertieren kann, zeigt es eine Fehlermeldung an und kehrt zur Eingabeaufforderung zurück. Wenn du beispielsweise eine Zeichenkette als zweites Argument übergeben hast, wird dein Programm mit einer Fehlermeldung beendet.

Du musst die Typ-Suffixe nicht verwenden, sondern kannst den Typ der Argumente stattdessen mit dem Schlüsselwort AS definieren, ähnlich wie es im Befehl DIM verwendet wird.

Das folgende Beispiel ist zum Beispiel identisch mit dem obigen Beispiel:

```
SUB MySub Msg AS STRING, Nbr AS INTEGER
  …
END SUB
```

Wenn du im ganzen Programm nur einen Variablentyp benutzt und diesen Typ mit OPTION DEFAULT festgelegt hast, kannst du die Frage nach den Variablentypen natürlich komplett ignorieren.

Wenn eine Subroutine mit einem Argument aufgerufen wird, das eine Variable ist (also keine Konstante oder kein Ausdruck), erstellt MMBasic eine entsprechende Variable innerhalb der Subroutine, *die auf diese Variable zurückverweist.* Jede Änderung an der Variable, die das Argument innerhalb der Subroutine darstellt, ändert auch die im Aufruf verwendete Variable. Dies wird als Übergabe von Argumenten durch Referenz bezeichnet.

Am besten lässt sich das anhand eines Beispiels erklären:

```
DIM MyNumber = 5       ‘ setze die Variable auf 5
CalcSquare MyNumber    ' Die Unterroutine berechnet den
Quadratwert
PRINT MyNumber         ‘ hiermit wird die Zahl 25 ausgegeben
END

SUB CalcSquare nbr
  nbr = nbr * nbr      ' das Argument wird quadriert und
zurückgegeben
END SUB
```

Die Subroutine CalcSquare nimmt ihr Argument, quadriert es und schreibt es zurück in die Variable, die das Argument (`nbr`) darstellt. Weil die Subroutine mit einer Variablen (`MyNumber`) aufgerufen

wurde, zeigt die Variable `nbr` wieder auf `MyNumber`, und jede Änderung an `nbr` ändert auch `MyNumber` entsprechend. Deshalb gibt die PRINT-Anweisung 25 aus.

Die Übergabe von Argumenten per Referenz ist praktisch, weil sie es einer Subroutine ermöglicht, Werte an den Code zurückzugeben, der sie aufgerufen hat. Es könnte aber zu Problemen führen, wenn eine Subroutine die Variable, die ein Argument darstellt, 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 das zu vermeiden, solltest du ihrer Definition das Schlüsselwort BYVAL voranstellen. Dadurch wird MMBasic angewiesen, immer den Wert des Arguments zu verwenden, auch wenn es sich um eine Variable handelt, und niemals auf die im Aufruf verwendete Variable zurückzuweisen.

Wenn du eine Subroutine aufrufst, kannst du einige (oder alle) Parameter weglassen, und sie nehmen den Wert Null (für Fließkommazahlen oder Ganzzahlen) oder eine leere Zeichenfolge 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 ohne Angabe einer Fehlermeldung als Parameter verwendet werden:

```
SUB ErrMsg  Msg$
  IF Msg$ = "" THEN
    PRINT "Fehler erkannt"
  ELSE
    PRINT "Fehler: " + Msg$
  ENDIF
END SUB
```

In einer Subroutine kannst du die meisten Funktionen von MMBasic nutzen, zum Beispiel andere Subroutinen aufrufen, IF…THEN-Befehle, FOR…NEXT-Schleifen und so weiter. Eine Sache geht aber nicht: Du kannst nicht mit GOTO aus einer Subroutine springen (wenn du das versuchst, wird das Ergebnis unklar und du bekommst vielleicht graue Haare).

Normalerweise wird die Unterprogrammierung beendet, wenn der Befehl END SUB erreicht wird, aber du kannst die Unterprogrammierung auch vorzeitig mit dem Befehl EXIT SUB beenden.

Funktionen

Funktionen sind ähnlich wie Unterprogramme, mit dem Hauptunterschied, dass eine Funktion dazu dient, einen Wert in einem Ausdruck zurückzugeben. Wenn du zum Beispiel 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 kannst du sie in einem Ausdruck verwenden:

```
Input "Gib eine Temperatur in Celsius ein: ", t
PRINT „Das entspricht“ Fahrenheit(t) „F“
```

Oder ein anderes Beispiel:

```
WENN Fahrenheit(temp) <= 32 DANN DRUCKE „Gefrierpunkt“
```

Du kannst auch das Gegenteil festlegen:

```
FUNKTION Celsius(F)
  Celsius = (F - 32) * 0,5556
END FUNCTION
```

Wie du siehst, wird der Funktionsname als normale lokale Variable innerhalb der Unterroutine 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 sind ähnlich wie bei Unterprogrammen. Der einzige Unterschied besteht darin, dass beim Aufruf einer Funktion immer 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 man dem Namen der Funktion innerhalb der Funktion einen Wert zu. Wenn der Name der Funktion mit einem Typ-Suffix endet (d. h. $, % oder !), gibt die Funktion diesen Typ (Zeichenkette, Ganzzahl oder Gleitkommazahl) 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 brauchst du kein Typ-Suffix (ähnlich wie beim Definieren einer Variablen mit DIM).

Hier ist das obige Beispiel, das umgeschrieben wurde, 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 Zeichenfolge.

Wie bei Unterprogrammen kannst du die meisten Funktionen von MMBasic auch in Funktionen nutzen. Dazu gehören FOR…NEXT-Schleifen, das Aufrufen 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 erstellt werden, werden als *globale* Variablen bezeichnet. Das heißt, sie können überall im Programm gesehen und verwendet werden, auch in Unterprogrammen und Funktionen. Innerhalb eines Unterprogramms oder einer Funktion musst du jedoch häufig Variablen für verschiedene Aufgaben verwenden, die innerhalb des Unterprogramms/der Funktion liegen. In portierbarem Code solltest du vermeiden, dass der Name, den du für eine solche Variable gewählt hast, mit einer globalen Variablen gleichen Namens kollidiert. Zu diesem Zweck kannst du eine Variable mit dem Befehl LOCAL innerhalb der Unterprogramm/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 bestimmten Wert initialisieren.

Hier ist zum Beispiel unsere Unterprogramm-Funktion ErrMsg, aber diesmal wurde sie erweitert, um eine lokale Variable zum Verbinden der Fehlermeldungszeichenfolgen 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 (wie die Argumentliste) nur innerhalb der Subroutine existiert und beim Beenden der Subroutine verschwindet. Du kannst eine globale Variable namens `tstr` in deinem Hauptprogramm haben, die sich von der Variable `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 diese dazu beitragen, das Modul wesentlich eigenständiger und portabler zu machen.

Statische Variablen

Lokale Variablen werden bei jedem Start der Unterroutine oder Funktion auf ihre Anfangswerte (normalerweise Null oder eine leere Zeichenfolge) zurückgesetzt. Manchmal möchtest du aber, 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 Unterprogramm-Funktion ErrMsg erweitern, um zu verhindern, dass doppelte Aufrufe der Unterprogramm-Funktion wiederholt dieselbe Meldung anzeigen. Unser Programm könnte diese Unterprogramm-Funktion beispielsweise an mehreren Stellen aufrufen, aber wenn die Meldung in mehreren aufeinanderfolgenden Aufrufen identisch 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
    letzteMeldung = Msg
  ENDIF
END SUB
```

Um die zuletzt angezeigte Meldung zu speichern, benutzen 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. Dadurch würde bei jedem Aufruf mit einem doppelten Meldungstext nur eine Meldung ausgegeben werden.

Der Befehl STATIC hat genau die gleiche Syntax wie DIM. Das heißt, du kannst verschiedene Arten von statischen Variablen definieren, einschließlich Arrays, und sie auch mit einem bestimmten Wert initialisieren.

Die statische Variable wird beim ersten Auftreten des Befehls STATIC erstellt und automatisch auf Null (bei einem Float- oder Integer-Wert) oder eine leere Zeichenfolge 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 auch eine statische Variable auf einen bestimmten Wert initialisieren. Zum Beispiel:

```
STATIC INTEGER var = 123
```

Beim ersten Aufruf (wenn die Variable erstellt wird) wird sie auf 123 initialisiert, aber bei späteren Aufrufen behält sie den zuvor festgelegten Wert bei.

Meistens werden statische Variablen verwendet, um den *Status* von etwas innerhalb einer Subroutine oder Funktion zu verfolgen. Ein *Status* ist eine Aufzeichnung von etwas, das zuvor passiert ist.

Beispiele hierfür sind:

- Wurde der COM-Port schon geöffnet?
- Welche Schritte einer Sequenz haben wir schon gemacht?
- Welcher Text wurde schon angezeigt?

Normalerweise benutzt man globale Variablen (die mit DIM erstellt werden), um einen *Status* zu verfolgen, aber manchmal will man, dass dieser in einem Modul enthalten ist, und hier sind statische Variablen nützlich. Genau wie LOCAL hilft die Verwendung von STATIC dabei, Unterprogramme und Funktionen eigenständiger und portabler zu machen.

Tage berechnen

Wir haben in diesem Tutorial bisher viele Programmierbefehle und -techniken behandelt, und bevor wir es beenden, 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:

```
' Beispielprogramm zur Berechnung der Anzahl der Tage zwischen zwei Daten

OPTION EXPLICIT
OPTION DEFAULT NONE

DIM STRING s
DIM FLOAT d1, d2

DO
  ' Hauptprogrammschleife
  PRINT : PRINT „Gib das Datum im Format TT.MM.JJJJ ein”
  PRINT „Erstes Datum”;
  INPUT s
  d1 = GetDays(s)
  WENN d1 = 0 DANN PRINT „Ungültiges Datum!“ : WEITER
  DRUCKEN „Zweites Datum“;
  INPUT s
  d2 = GetDays(s)
  WENN d2 = 0 DANN DRUCKE „Ungültiges Datum!” : WEITER
  DRUCKE „Die Differenz beträgt“ ABS(d2 - d1) „Tage“
LOOP

' Anzahl der Tage seit dem 1.1.1900 berechnen
FUNCTION GetDays(d$) AS FLOAT
  LOKALER STRING Monat(11) =
("Jan","Feb","März","Apr","Mai","Jun","Jul","Aug","Sep","Okt","Nov","Dez")
  LOKAL FLOAT Tage(11) = (0,31,59,90,120,151,181,212,243,273,304,334)
  LOCAL FLOAT Tag, Monat, Jahr, s1, s2

  ' Suche das Trennzeichen innerhalb eines Datums
  s1 = INSTR(d$, " ")
  WENN s1 = 0 DANN FUNKTION BEENDEN
  s2 = INSTR(s1 + 1, d$, " ")
  WENN s2 = 0 DANN FUNKTION BEENDEN

  ' Hol dir den Tag, den Monat und das Jahr als Zahlen
  day = VAL(MID$(d$, 1, s2 - 1)) - 1
  WENN Tag < 0 ODER Tag > 30 DANN FUNKTION BEENDEN
  FÜR mth = 0 BIS 11
    WENN LCASE$(MID$(d$, s1 + 1, 3)) = Monat(mth) DANN BEENDE FOR
  NEXT mth
  WENN Monat > 11 DANN FUNKTION BEENDEN
  yr = VAL(MID$(d$, s2 + 1)) - 1900
  WENN Jahr < 1 ODER Jahr >= 200 DANN FUNKTION BEENDEN

  ' Anzahl der Tage inklusive Anpassung für Schaltjahre berechnen
  GetDays = (Jahr * 365) + FIX((Jahr - 1) / 4)
  WENN Jahr MOD 4 = 0 UND Monat >= 2 DANN GetDays = GetDays + 1
  GetDays = GetDays + Days(mth) + day
END FUNCTION
```

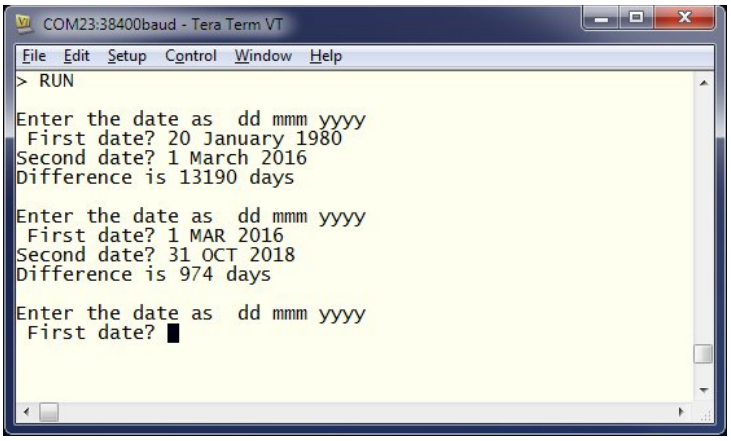
Beachte, dass die Zeile, die mit `LOCAL STRING Month(11)` beginnt, wegen der begrenzten Seitenbreite umgebrochen wurde – sie besteht aus einer Zeile wie folgt:

```
LOCAL STRING Month(11) = ("jan","feb","mar","apr","may","jun","jul","aug","sep","oct","nov","dec")
```

Dieses Programm fragt den Benutzer an der Konsole nach zwei Daten und rechnet sie dann in die Anzahl der Tage seit 1900 um. Mit diesen beiden Zahlen ergibt eine einfache Subtraktion die Anzahl der Tage zwischen ihnen.

Wenn dieses Programm ausgeführt wird, werden Sie aufgefordert, die beiden Daten einzugeben. Sie müssen dabei das Format `dd mmm yyyy` verwenden.

Der Screenshot rechts zeigt, wie das Programm aussieht, wenn es läuft.



Das Hauptmerkmal des Programms ist die definierte Funktion `GetDays()`, die eine in der Konsole eingegebene Zeichenfolge nimmt, sie in ihre Bestandteile 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 dienen 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 sind nur eine praktische Methode, um die Namen der Monate und die kumulative Anzahl der Tage für jeden Monat nachzuschlagen. Später in der Funktion (der FOR-Schleife) kannst du sehen, wie sie den Umgang mit zwölf verschiedenen Monaten ziemlich effizient machen.

Eine weitere Funktion, die in diesem Programm hervorgehoben wird, sind die Zeichenfolgenverarbeitungsfunktionen von MMBasic. Die Funktion INSTR() wird verwendet, um die beiden Leerzeichen in der Datumszeichenfolge zu finden, und später verwendet die Funktion MID$() diese, um die Tages-, Monats- und Jahreskomponenten 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 dass sie einen Wert von Null zurückgibt, wenn ihr kein Wert zugewiesen wurde. Das macht die Fehlerbehandlung einfach, 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 anzeigt.

Dieses Programm zeigt einen der Vorteile der Verwendung von Unterprogrammen und Funktionen: Wenn sie geschrieben und vollständig getestet sind, können sie wie eine 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 Sie einen Fehler hinzufügen).

Dieses Programm hat ein paar Funktionen, 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 zum Beispiel 4 durch 15 teilst, bekommst du einen Rest von 3, was genau dem Ergebnis des Ausdrucks 15 MOD 4 entspricht. 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 zu Ende ist, EXIT FUNCTION beendet eine Funktion sofort, auch wenn die Ausführung noch nicht am Ende der

Funktion angekommen ist, und CONTINUE DO sorgt dafür, dass das Programm sofort zum Ende einer DO-Schleife springt und sie erneut ausführt.

Wozu ist dieses Programm gut? Manche Leute zählen ihr Alter gerne in Tagen, dann ist jeder Tag ein Geburtstag! Du kannst dein Alter in Tagen berechnen, indem du einfach dein Geburtsdatum und das heutige Datum eingibst. Das ist zwar nicht besonders nützlich, aber das Programm selbst ist wertvoll, da es viele Eigenschaften der Programmierung in MMBasic demonstriert. Arbeite dich also durch das Programm und gehe jeden Abschnitt durch, bis du ihn verstanden hast – es sollte eine lohnende Erfahrung sein.