CoreBASIC Reference Guide

Version: 3.1

Contents

CrossWorks CoreBASIC Library

About the CrossWorks CoreBASIC Library

The *CrossWorks CoreBASIC Library* is an application that makes extensive use of the software components in the CrossWorks Target Library.

The components that CoreBASIC Library uses are:

- *CrossWorks Platform Library*: provides base platform services.
- *CrossWorks Device Library*: provides drivers for common digital sensors, such as accelerometers, gyroscopes, magnetometers, and so on.
- *CrossWorks Shield Library*: provides drivers for a range of Arduino-style shields.
- *CrossWorks Graphics Library*: is a library of simple graphics functions for readily-available LCD controllers.

Architecture

The *CrossWorks CoreBASIC Library* is one part of the *CrossWorks Target Library*. Many of the low-level functions provided by the target library are built using features of the *CrossWorks Tasking Library* for multi-threaded operation.

Delivery format

The *CrossWorks CoreBASIC Library* is delivered in source form.

License

The source files in this package are not public domain and are not open source. They represent a substantial investment undertaken by Rowley Associates to assist CrossWorks customers in prototyping solutions using well-written, tested drivers.

Should you wish to incorporate CoreBASIC in a product, you will need to purchase a Commercial Use license for CoreBASIC.

Feedback

This facility is a work in progress and may undergo rapid change. If you have comments, observations, suggestions, or problems, please feel free to air them on the **CrossWorks Target and Platform API** discussion forum.

Setting up a SolderCore

The SolderCore is a small printed circuit board containing a programmable microcontroller. After unpacking your SolderCore and attaching it to your network, you can be up and programming in a few minutes.

However, the SolderCore Project is much broader than this one product. The project's objective is to simplify the construction of physical computing systems, to excite hackers and makers, and to rekindle the enthusiasm for computing, especially in children, that the 1980s microcontroller revolution started.

Set up SolderCore

Unpacking your SolderCore Sounds boring, but be careful not to zap your shiny new SolderCore before using it!	**SolderCore anatomy** Hammering square pegs into round holes is bad. Get familiar with what goes where on SolderCore.	**Power up!** There's more to this than plugging in and crossing fingers. Keep the smoke in the chips, where it belongs…
Contact SolderCore Get SolderCore onto your LAN and set sail for the Internet!	**Fire up CoreBASIC** Start interacting with CoreBASIC over the network!	**Who did all this?** Learn about the team behind SolderCore and CoreBASIC.

Unpacking your SolderCore

Your SolderCore is packaged inside a cardboard box to protect it from knocks during shipping. On opening the box you'll find that the SolderCore is enclosed in a static shielding bag to prevent damage from electrostatic discharges (ESD).

Much like the shock you receive when walking over a carpet and touching something metal, or taking a polyester fleece off over a cotton shirt, ESD can happen when you pick up a component when working at your bench or desk.

ESD may cause components you touch to no longer work properly. ESD can happen without you even feeling a shock! It might damage a small part of your SolderCore, and in the extreme, could cause the whole of the your SolderCore to malfunction.

Preventing ESD

You should take precautions to make sure that you don't inadvertently damage the SolderCore by ESD. The best method of preventing ESD is to use an ESD wrist strap or use a grounding mat or table. However, because most customers don't have these, here are some hints on reducing the chances of ESD damage:

- *sitting down*: Be aware that sitting on a chair can build up electrostatic charge from rubbing together your clothing and seat cover.
- *clothes*: Don't wear any clothing that conducts a lot of electrical charge, such as a wool sweater, when handling electronics.
- *weather*: During dry weather, static charge does not dissipate very easily, so on dry days it's more likely that you'll damage your SolderCore by ESD.
- *jewelry*: It's a good idea to remove rings, watches, and chains when tinkering with electronics. Gold and silver are excellent conductors of electricity and wearing such items only increases the chances of shorting contacts when handling SolderCore and its accessories.

Next...

SolderCore anatomy

SolderCore anatomy

The first thing to notice about the SolderCore is that it has no keyboard and no display, just some connectors. This is because the SolderCore is designed to control external hardware which you plug into it. You can add LCD displays, sensors, motor drivers, whatever you want.

The SolderCore has the same physical layout as the very popular Arduino range of boards.

SolderCore is fitted with the following connectors:

- a barrel jack to provide power to the SolderCore using an external supply.
- an RJ45 socket that uses standard twisted pair cable to connect SolderCore to a local area network (LAN) and, through a router, to the Internet.
- Single in-line connectors along two edges of the PCB for interfacing other digital and analog electronics.
- A push-eject microSD connector for mass storage on readily available microSD cards.
- a microUSB connector to provide auxiliary power to SolderCore and for interfacing USB devices.
- a button to reset the SolderCore, and anthing else attached to the reset signal.

In addition, SolderCore has some unpopulated connectors and sites:

- An 10-way Cortex JTAG header which you can use to completely erase CoreBASIC and program the SolderCore on-the-metal using C.
- a two-pin connector for a secondary I2C bus.
- on the reverse, two SOIC-8 sites for additional storage devices.

These features are not directly supported by CoreBASIC; however, using C you have complete control over devices attached to these sites.

Next...

The following section will walk you through setting up the SolderCore so you can start programming!

Power up!

Power up!

To power SolderCore, you have two options:

- Connect 6V DC, center positive, to the barrel jack;
- Connect a standard phone charging cable to the microUSB connector.

Which one you choose usually comes down to convenience and environment. If you're using a laptop, powering SolderCore using a USB port and a charging cable is pretty convenient; if you have SolderCore on the bench, powering it from a bench power supply or wall wart makes sense.

Apply power...

Before attaching any power, make sure that your SolderCore is on a flat surface and is not in contact with anything metal to avoid shorts, sparks, and fireworks.

When you apply power, two green LEDs close to the barrel jack indicate the health of the 3V3 and 5V supplies. If you power SolderCore using the the barrel jack and the power LEDs do not illuminate when you plug in the connector, unplug and use a voltmeter to check the polarity of your supply: the center pin of the connector must be positive. If you are using a wall wart, the polarity of the connector is usually shown on the plastic case—this Wikipedia page has an excellent explanation of the symbols:

http://en.wikipedia.org/wiki/Polarity_symbols

A SolderCore with no accessories or shields installed requires approximately 200 mA, so your external supply must be capable of providing that. If it isn't, the LEDs will not illuminate and the regulators will not power up the SolderCore.

Next...

Once you have the power on and the supplies are healthy, it's time to move on to networking!

Contact SolderCore

Contact SolderCore

You interact with the SolderCore over a LAN using the standard twisted pair RJ45 connector. The first thing you should do is connect your SolderCore to a router with a network cable.

If you look at the SolderCore's network socket, you will see that it has a green and a yellow LED. The green LED indicates a good link and will illuminate when both ends of the cable are connected to compatible network equipment. The yellow LED indicates network activity: it momentarily pulses when network packets arrive at the SolderCore.

Finding SolderCore on the LAN

Using your SolderCore requires that you know a little about your network and the ability to run a few tools from the command line.

On the back of your SolderCore you will find a label inscribed with six letters and numbers, like this:

This is the *serial number* and *network name* of your SolderCore. Each SolderCore's serial number is unique and forms part of the default network name for the SolderCore.

By default, the SolderCore is set up to request its IP address using DHCP. That is, it will communicate with your router to determine how to set itself up on your LAN. DHCP is the modern way to assign addresses to network equipment on a LAN and home routers have this capability already built into them.

So, let's say that the SolderCore's serial number is `0000be`. The leading zeros are important here: the serial number is *exactly* six characters long. When the SolderCore registers itself on the network, it will use the network name `core-`*xxxxxx*. *xxxxxx* is the serial number. So, in this case, the SolderCore's network name will be `core-0000be` as printed on the label.

First, we must open a command line window to test out the network connection.

To open a command window using Windows 7:

- Choose **Start** > **All Programs** > **Accessories** > **Command Prompt**

Now you should be be able to ping the SolderCore and see that it responds:

```
C:\Users\Paul> ping core-0000be

Pinging core-0000be.rowley.co.uk [10.0.0.25] with 32 bytes of data:
Reply from 10.0.0.25: bytes=32 time<1ms TTL=64
Reply from 10.0.0.25: bytes=32 time<1ms TTL=64
Reply from 10.0.0.25: bytes=32 time<1ms TTL=64
Reply from 10.0.0.25: bytes=32 time<1ms TTL=64

Ping statistics for 10.0.0.25:
```

```
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Paul>
```

To open a command window using Mac OS X:

- In the finder, choose **Go** > **Utilities**.
- Double-click the **Terminal** application.

Now you should be be able to ping the SolderCore and see that it responds:

```
pauls-macbook-pro:~ plc$ ping core-0000be
PING core-0000be.rowley.co.uk (10.0.0.25): 56 data bytes
64 bytes from 10.0.0.25: icmp_seq=0 ttl=64 time=0.695 ms
64 bytes from 10.0.0.25: icmp_seq=1 ttl=64 time=0.866 ms
64 bytes from 10.0.0.25: icmp_seq=2 ttl=64 time=0.840 ms
64 bytes from 10.0.0.25: icmp_seq=3 ttl=64 time=0.854 ms
64 bytes from 10.0.0.25: icmp_seq=4 ttl=64 time=0.847 ms
64 bytes from 10.0.0.25: icmp_seq=5 ttl=64 time=0.754 ms ^C
--- core-0000be.rowley.co.uk ping statistics ---
6 packets transmitted, 6 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.695/0.809/0.866/0.063 ms
pauls-macbook-pro:~ plc$
```

The ping request will continue to be sent until you type **Ctrl+C**.

Next...

Once you have this working, your SolderCore is registered on the network and is successfully communicating with the PC. Now you need to start a Telnet session.

Fire up CoreBASIC

Fire up CoreBASIC

On Windows 7, Telnet is not automatically installed. To make Telnet available:

- Choose **Start** > **Control Panel**.
- Click **Programs**.
- Click **Turn Windows Features on or off**.

Make sure that the Telnet Client feature is checked:

Start telnet!

Now, start **telnet** from the command line using `telnet Core-0000be` and substituting your serial number in the command line. You should see something like this:

Next...

Selecting and preparing microSD cards

Selecting and preparing microSD cards

SolderCore uses the microSD cards to store programs and data permanently. Before you can use a microSD card with SolderCore, you must make sure that it is formatted correctly. This section advises you about the SD card to select and how to prepare it for use with SolderCore.

Manufacturers

Always purchase your microSD cards from a reputable distributor. Fake SD cards are scattered around the Internet on bargain-value sites, and they *do not work very well!*

During testing of SolderCore and its mass storage drivers, we collected nearly 100 microSD cards and for testing from suppliers far and wide. Of those, we had a few obvious fakes and destruction testing reveals that they are not very durable and will lose their contents easily. A few just didn't work at all.

We would highly recommend that you purchase a SanDisk microSD card from a reputable supplier with a storage capacity of 2 GB. These cards are robust and compliant and we have yet to find a genuine SanDisk card that does not work in SolderCore.

Preparing you card

To prepare your card to work with SolderCore, place it into a microSD to SD adapter, and then insert that in your PC. It shows up in the Windows Explorer:

Right-click on removable media and select **Format**.

Note: Make sure to select "FAT" for the file system, *not* "FAT32".

Go ahead and format the card.

Now create a folder named SYS at the root of the file system. This will hold system files that SolderCore will write.

Cards that don't work

Some microSD cards do not comply with the SD specification as published by the SD Card Association (SDA). That is, they may well work in cameras and mobile devices, but do not when used with SolderCore. These cards should offer features for small, embedded devices, but to cut costs such features are simply not implemented. We have a number of cards which fail to comply in different ways, and they simply will not work with SolderCore.

We cannot emphasize strongly enough: purchase a card that is known to work! A SanDisk 2 GB card is an excellent choice.

If you do purchase a card that doesn't work, please let us know the manufacturer make a model: we can then try to source the card and diagnose why it doesn't work.

Setting up a Raspberry Pi

In this section you'll prepare your Raspberry Pi to run CoreBASIC.

What you will need

To use Arduino shields on your Raspberry Pi, you'll need to connect a Libelium **Raspberry Pi to Arduino shields connection bridge**. This takes the signals from the Raspberry Pi's GPIO connector and converts them into an Arduino footprint. Note that not *all* shields are compatible with the bridge installed: some shields work at the wrong voltage, some shields require digital I/O on the analog connectors, and some require general-purpose I/O on the I2C connector.

Installing CoreBASIC

…to write.

CoreBASIC User Guide

Starting out with CoreBASIC

In this section we'll take a look at some of the features of CoreBASIC. We'll assume that you've read through the previous sections and have set up your CoreBASIC device ready for use.

The building blocks of CoreBASIC are *statements*, *operators*, and *identifiers*:

- *statements* command CoreBASIC to do something, such as print a number or load a program.
- *operators* work on numbers and strings of characters to perform calculations or manipulate strings.
- *identifiers* are defined by you, the programmer, to store the data that you need in your program.

Calculator mode

When CoreBASIC starts, it signs on with a banner and waits for commands from you. CoreBASIC asks you for a command with a simple *greater than* prompt:

```
> _
```

Here, the underscore indicates the position of the *insertion point*, which is usually a flashing vertical line or block on the screen. When you type on the keyboard, what you type appears at the insertion point. So, why not try it out? Type `PRINT 12+6` and press Enter or Return (visually indicated by '#'):

```
> PRINT 12+7
19
> _
```

This commands CoreBASIC to print out the result of a calculation, in this case adding 12 and 7 together. The prompt reappears immediately and CoreBASIC is ready for another command.

You don't need to type in capitals, you can use lowercase letters:

```
> print 12+7
19
> _
```

All examples in this manual use lowercase letters for user input because it's easier to read than all-capitals, and will most likely be the letter case you use.

You might like to try some more calculations with CoreBASIC. For instance, CoreBASIC understands `-` to mean *subtract*, `*` to mean *multiply*, and `/` to mean divide:

```
> print 12*7
84
> print 12/7
1.71429
> _
```

When CoreBASIC is waiting for a command, it is in *calculator mode*; in some literature you might also find this called *immediate mode* because everything you type is executed immediately when you press Enter.

CoreBASIC understands the standard mathematical rules that you learn at school: multiplication is done before addition, for example. (See **Wikipedia: Order of operators**.) Testing this out:

```
> print 5+3*4
17
> _
```

CoreBASIC worked this out by calculating 3×4 first, which is 12, and then calculating 5+12 to arrive at the answer 17.

When you become proficient with CoreBASIC, you'll find that you rely heavily on PRINT. Because PRINT is so often used in CoreBASIC programs, there's a shorthand for it: you can use ? wherever you'd use a PRINT command:

```
> ? 5+3*4
17
> _
```

You can print things other than numbers: CoreBASIC knows about strings made from a sequence of characters:

```
> print "I'm sorry, Dave. I'm afraid I can't do that."
I'm sorry, Dave. I'm afraid I can't do that.
> _
```

The string to print is enclosed by quotation marks. If you want to include a quotation mark in the string, you need to *double them up*:

```
> print """More human than human"" is our motto."
"More human than human" is our motto.
> _
```

You might wonder why you'd want to enter a command that simply echoes what you type? While this might not seem immediately useful in calculator mode, you'll discover that you use this again and again when starting to write programs in CoreBASIC.

To finish off this section, there are some other features of PRINT that are worth mentioning. You can print more than one item at a time if you separate them with a comma:

```
> print 1/2, 1/3
0.5     0.333333
> _
```

The comma starts printing in a new *field*, and each field spans eight characters. Each comma advances the print position to the next field, even if there is nothing to print:

```
> print 1/2,, 1/3
0.5             0.333333
> _
```

To print items next to each other, separate them with a semicolon:

```
> print 1/2; 1/3
0.50.333333
> _
```

Unfortunately, the two numbers run together and the result is rather confusing. You might like to print a space between the two numbers, using a string, like this:

```
> print 1/2; " "; 1/3
0.5 0.333333
> _
```

This section touches a few of the ways that `PRINT` can format its output. There are additional features in CoreBASIC to exercise precise control of output format, but for many programs these simple formatting controls are enough.

Making mistakes

You will make mistakes when typing commands into CoreBASIC. Most likely you'll misspell a command or forget some punctuation, and when you do, CoreBASIC tells you about it:

```
> prunt 1/2
?expected '='
> _
```

CoreBASIC responds to problems by printing an error, starting with a question mark, and continuing with CoreBASIC's best diagnosis of the problem. Sometimes this message is spot-on, and at other times it might be a rather vague "syntax error" when CoreBASIC can't understand what you're asking it to do. If you happen to catch the right bracket on your keyboard when reaching for Enter, you'll get this:

```
> print 1/2]
?syntax error
> _
```

Don't worry about making mistakes like this, they're not going to do any harm, and CoreBASIC simply forgets the mistakes you make and moves on. However, if you spot a mistake you've made while typing, you can correct it using the cursor keys. CoreBASIC's command line editor understands the following keystrokes:

Completion	
Enter	Send line to CoreBASIC.
Moving	
Left	Move insertion point back one position.
Right	Move insertion point forward one position.
Home or Ctrl+A	Move insertion point to start of the line.
End or Ctrl+E	Move insertion point to end of the line.
Editing	
Backspace or Ctrl+H	Delete character before insertion point.
Delete or Ctrl+G	Delete character after insertion point.
Ctrl+K	Delete from insertion point to end of line.
Ctrl+U	Delete entire line.

Typing a printable character (letter, digit, punctuation, or symbol) inserts that character at the insertion point, moving the remainder of the line forward. Pressing Enter will send the entire line to CoreBASIC, irrespective of where the insertion point is on the line.

Your first program

Up to now you have typed in commands that CoreBASIC executes immediately. The main feature of a computer is its ability to run *stored programs* at high speed. So, let's get down and try it out! Type in:

```
> 10 print "This is my first program!"
> _
```

Notice that CoreBASIC doesn't do any printing after pressing Enter. Instead, the `PRINT` instruction is stored in CoreBASIC's memory, ready to be executed when commanded to do so—the `10` is a line number that you can ignore for the moment.

Finish off the program by entering:

```
> 20 end
> _
```

This stores a second program instruction telling CoreBASIC to end the program.

To have CoreBASIC show the program you've just typed in, type `LIST`:

```
> list
   10 PRINT "This is my first program!"
   20 END
> _
```

If your program doesn't look like this, don't worry, see **More on mistakes** to correct it.

The listing displays the stored program in line-number order. Line numbers are simply a way of setting the order of program instructions when entering them from the command line and are part of the heritage of the BASIC language. (Note that CoreBASIC has a built-in full screen editor to make editing programs much simpler, just like using a text editor or word processor on a personal computer, which we'll cover later.)

To run the program you've just entered, type RUN:

```
> run
This is my first program!
> _
```

Excellent! CoreBASIC executed the stored instructions when you said RUN. You can run the program a second time using another RUN:

```
> run
This is my first program!
> _
```

What CoreBASIC does, when instructed to run the program, is to execute each program line in sequence, starting at the lowest-numbered program line, and continuing until told to stop. The small program you entered, when run, starts at line `10` and prints to the screen. Once that program line is complete, execution continues

to program line 20, and this line instructs CoreBASIC to stop executing (that's what END does) and return to calculator mode.

In fact, it's not strictly necessary to store an END instruction as "running off the end of a program", without further program lines to execute, returns CoreBASIC to calculator mode.

See also

PRINT, **LIST**, **RUN**

More on mistakes

We touched on how to edit lines as you type them in, but what happens if you've already entered a program line and you know it's wrong? How do you go about correcting it?

Let's start out with a badly-typed program:

```
> list
   20 END
   29 END
   100 PRUNT "This is my first program!"
> _
```

The first option you have is to use NEW, which erases the entire program, and start typing it in again with the hope that you enter it perfectly on the second, or perhaps third, attempt. If you had to enter a long program without any mistakes each time you'd simply give up programming as worthless exercise!

There is a much better way to proceed, and that is to correct the mistakes that you've made by editing the program in memory.

In the program, line 29 shouldn't be there. To delete a program line from memory, type the line number only and press Return:

```
> 29
> list
   20 END
  100 PRUNT "This is my first program!"
> _
```

Fantastic, we're starting to clean up the program. Now, line 100 should really be line 10, but typing all that again is a chore. To recall the line so you can edit it with standard editing keys, use EDIT and the program line number you'd like to edit, in this case, 100:

```
> edit 100
100 PRUNT "This is my first program!"
```

After using EDIT, the line is recalled from memory and is presented for editing with the insertion point on the instruction. Now use the the standard editing keys to correct the program line number from 100 to 10 and change PRUNT to PRINT:

```
> edit 100
10 PRINT "This is my first program!"
```

Press Return to finish editing. It doesn't matter where the insertion point is on the line, the entire line of text, with everything beyond the insertion point, is passed to CoreBASIC for processing.

Use LIST to see how we're getting along:

```
> list
   10 PRINT "This is my first program!"
```

```
   20 END
  100 PRUNT "This is my first program!"
> _
```

What? Line `100` is still there! Why is that, wasn't it edited to be line `10`? Well, no, it is simply because `EDIT` brings in the line to be edited, but doesn't immediately delete it from memory. When you edited the line number, you didn't edit it in memory, it was just like typing in a new line.

To finalize the corrections, kill line `100` and, behold, a perfect program:

```
> 100
> list
   10 PRINT "This is my first program!"
   20 END
> _
```

There are other utility commands for editing a program, such as `DELETE` and `RENUMBER`, and also a full screen editor which you can bring up with `EDIT` on its own. If you're curious, take a moment to look at the reference material for the editing commands to see what else you can do.

See also

EDIT

Using variables

Now you've experienced a little of CoreBASIC, it's time to move on to how to store data. You've already seen some data that CoreBASIC can work with: you've used numbers for calculation and strings of characters to print messages. This data was use momentarily and then thrown away. To store data inside CoreBASIC, you *assign* it to a *variable*:

```
> let x = 3.14
> print x
3.14
> _
```

In this example, the LET statement assigns the number `3.14` to the variable *x*. Once assigned, CoreBASIC remembers the value and you can recall it at any time using the assigned variable name. Variable names are made using a combination of letters, digits, and underscores; creating meaningful names for variables is something that you'll get the hang of over time. These examples confine themselves to using single letters for names.

You can assign the results of calculations to variables, just like simple numbers:

```
> let r = 12
> let a = x * r * r
> print a
452.16
> _
```

Because assignment is so common in CoreBASIC programs, you can drop the LET from assignments and CoreBASIC will assume it's there:

```
> v = 4/3 * 3.14 * r * r * r
> print v
7234.56
> _
```

So, it's time to put what you've seen in these last few sections to use! We've almost seen enough of CoreBASIC to make a useful program, and the one missing element follows naturally.

Let's start with a new program. If you have followed along, then CoreBASIC still contains your stored program:

```
> list
   10 PRINT "This is my first program!"
   20 END
> _
```

To clear out that program and start a new one, use NEW:

```
> new
> list
> _
```

Here's the program you should type in:

```
> 10 print "Please type in the radius"
> 20 input r
> 30 p = 3.14159
> 40 print "Circumferenace of a circle: "; 2 * p * r
> 50 print "Area of a circle: "; pi * r * r
> 60 print "Volume of a sphere: "; 4/3 * p * r * r * r
> 70 end
> _
```

You've met all of this before apart from INPUT which, as it suggests, asks you for input.

Before running the program, check for potential problems using CHECK.

```
> check
> _
```

If all is well, CHECK is silent and the program contains no syntax errors that CoreBASIC can spot. The program may not do what you want it to do when you run it, of course, and RUN automatically performs a CHECK before running a program.

If you're unlucky enough for CHECK to tell you that there is a mistake, you can use the program editing commands covered earlier to fix it up.

Now, run the program:

```
> run
Please type in the radius
R? _
```

CoreBASIC is now running the INPUT statement and asking you to type in a number to store into the variable R. Let's say that the radius is 10:

```
> run
Please type in the radius
R? 10
Circumferenace of a circle: 62.8318
Area of a circle: 314.159
Volume of a sphere: 4188.79
> _
```

You can run this program again and provide another radius measurement to try out.

See also

LET, INPUT, CHECK

Saving your programs

Now that you've typed in your program, it would be a shame to lose it. The SolderCore has a microSD card slot that you can use for permanent storage of your programs; see **Selecting and preparing microSD cards** for details of how to prepare a microSD card for SolderCore to use.

When you reset SolderCore, any microSD card inserted into the holder is mounted as removable device "/c". To save your program to the microSD card, use SAVE and a file name:

```
> save "circle.bas"
> _
```

If the file does not exist, it is created. Any existing file with the same name is overwritten. CoreBASIC does check that folder names you use are correct and exist, that you can create the file, and that the permissions of any existing file allow you to overwrite it.

You load a saved program with LOAD:

```
> new
> list
> load "circle.bas"
> list
> _
```

See also

SAVE, **LOAD**

Loading examples

This manual is packed with examples. All nontrivial examples are stored online and are accessible through a network connection to the Internet. In the manual you'll see references to loading these examples, like this:

```
You can load this into CoreBASIC using EXAMPLE "welcome" or |welcome.
```

This tells you how to load the example from the Internet. So, try loading the Welcome application:

```
> example "welcome"
Connecting to www.soldercore.com (192.232.216.121)...
Loading welcome.bas from network...
Program loaded and ready. Type RUN to execute.
> run
Welcome to CoreBASIC on the SolderCore!
For more information, visit http://www.soldercore.com/
> list
   10 ' Welcome program for CoreBASIC.
   20 '
   30 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
   40 PRINT "For more information, visit http://www.soldercore.com/"
   50 '
   60 END
> _
```

If you're curious about the examples that are stored online, you can list them using EXAMPLE CATALOG, which you can abbreviate to EXAMPLE:

```
> example
Connecting to www.soldercore.com (192.232.216.121)...
Requesting /examples/ from network...

Index of /examples

  * Parent Directory
  * 3d-cube-1.bas
  * 3d-function-plot.bas
  * bouncing-lines.bas
  * charlcd.bas
  * colors-shield-message.bas
  * compass-demo.bas
  ?
  * welcome.bas

Apache Server at www.soldercore.com Port 80
> _
```

See also

EXAMPLE LOAD, EXAMPLE CATALOG

Listing directories

You can list the contents of a mounted microSD card using `DIR`:

```
> dir
Directory of: /c/*.*

12/05/12   11:26         <DIR>   sys
01/01/12   00:00           144   work.bas
01/01/12   00:00         1,913   crc.bas
01/01/12   00:00         2,499   union.bas
01/01/12   00:00           194   analog.bas
01/01/12   00:00        13,291   trek.bas
01/01/12   00:00         1,986   calib.bas
01/01/12   00:00           227   blinky.bas
01/01/12   00:00           157   !run.bas
01/01/12   00:00         2,580   flag.bas
01/01/12   00:00           538   air.bas
01/01/12   00:00            88   ansi.bas
01/01/12   00:00           102   test.bas
01/01/12   00:00           910   cam.bas
01/01/12   00:00         <DIR>   www

> _
```

If you want to see the contents of a specific folder, use `DIR` with a folder name:

```
> dir /c/sys

Directory of: /c/sys/*.*

12/05/12   11:26         <DIR>   .
12/05/12   11:26         <DIR>   ..
01/01/12   00:00           896   history.log
01/01/12   00:00       491,520   core.fw
01/01/12   00:00           113   !network.bas
01/01/12   00:00            84   !boot.bas
> _
```

Watchdog protection

Many microcontrollers provide an integrated watchdog to recover from software (and some hardware) faults. If you deploy your application in the field, over long periods, the probability is that at some point your application will stop responding and may require a hardware reboot.

CoreBASIC supports integrated hardware watchdogs with the `WATCHDOG` keyword.

A typical scenario

Most applications have a main loop that cycles doing some work, such as gather and log sensor measurements, and then pause before the next measurement. If anything happens to the software such that it locks up, you will stop gathering and logging data which may well be critical in an application that needs to log over months, or even years.

To protect against this, you can configure the watchdog to reboot the microcontroller and start running your program again if a lock-up is detected.

Preparing the watchdog

Assume your application logs data every ten seconds by repeatedly measuring sensor data and writing that to a microSD card or to a cloud-based service such as Xively. To protect against failure, you could configure the watchdog to time out after 20 seconds, and in your main loop, service (or 'kick') the watchdog to reset it. If your main loop fails to service the watchdog within 20 seconds, the microcontroller will reset.

`WATCHDOG TIMER` sets the watchdog timeout in seconds—see **WATCHDOG TIMER** for exact details. `WATCHDOG TIMER` does not start the watchdog running, you do that separately.

Ensuring the watchdog times out

When testing your code, it's not that convenient if the watchdog times out and resets the microcontroller, so CoreBASIC enables you to start the watchdog in *debug mode* using `WATCHDOG THROW`. In debug mode, the hardware watchdog is still configured but rather than resetting the microcontroller, it generates an interrupt that's passed through the CoreBASIC interpreter and causes your program to stop with a *watchdog timeout* error. If this happens, you know that your timeout is too short, or that your application has locked up, and the microcontroller would have been reset.

Here's an example of setting up the watchdog in debug mode, not servicing it, and waiting for it to time out:

```
***../examples/watchdog-timeout-demo.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "watchdog-timeout-demo"` or `|watchdog-timeout-demo`.

Running this, you see:

```
> run

Watchdog configured to go kaboom in five seconds.

ten:    4.9987 seconds remaining
nine:   3.98871 seconds remaining
eight:  2.97875 seconds remaining
seven:  1.96874 seconds remaining
six:    0.958748 seconds remaining

?watchdog timeout in 210: PAUSE 1
> _
```

Servicing the watchdog

To service the watchdog and reset the interval, you use `WATCHDOG RESTORE`. You'd normally use this as part of the main loop, resetting the watchdog when the scheduled task is complete. A modification of the previous program services the watchdog to prevent it from timing out:

```
***../examples/watchdog-service-demo.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "watchdog-service-demo"` or `|watchdog-service-demo`.

Running this, you see:

```
> run

Watchdog configured to time out after five seconds.

ten:    4.99868 seconds remaining
nine:   3.99211 seconds remaining
eight:  2.98211 seconds remaining
seven:  4.99965 seconds remaining
six:    3.98963 seconds remaining
five:   2.97967 seconds remaining
four:   4.99966 seconds remaining
three:  3.98967 seconds remaining
two:    2.97967 seconds remaining
one:    4.99967 seconds remaining

Watchdog is still active!
> _
```

This program does not stop with a watchdog timeout, it continues to run because the watchdog is services. When the watchdog is serviced, the timer is reset, and you can see that the remaining time remaining is reset in the output above.

Deploying

Once you are satisfied your application meets its deadlines, you can deploy it to the field with the watchdog configured in protection mode. You replace `WATCHDOG THROW` with `WATCHDOG REBOOT` to configure the watchdog to reboot, rather than throw an error, when it times out.

Using the first example again, with WATCHDOG REBOOT:

```
> run

Watchdog configured to go kaboom in five seconds.

ten:    4.9987 seconds remaining
nine:   3.98871 seconds remaining
eight:  2.97875 seconds remaining
seven:  1.96874 seconds remaining
six:    0.958748 seconds remaining

Connection to host lost.

C:\Users\Paul> _
```

Time and date

CoreBASIC has a number of functions to manipulate time and date as well as functions that read and update the current time and date maintained by CoreBASIC.

At its simplest, the current time and date are accessible using `TIME$` and `DATE$`:

```
> print date$, time$
2013/01/23     23:54:33
> _
```

`TIME$` and `DATE$` make it easy to print the time and date for log messages, or to confirm the time and date set on the SolderCore.

How the date gets set

When CoreBASIC starts, it has no knowledge of the current calendar time, so initializes its time and date to 1 January 1970. (We'll see why this is so a little later.) When a network connection becomes available, because a LAN cable is plugged in, CoreBASIC sends a request to the network to ask an available Internet time server for the current time and date. When the time server responds, and the response is valid, the current time and date is set from the server's response. So, as long as you have a network connection which can access a correctly configured time server, you should always be able to have the core time set correctly.

If you're running CoreBASIC on something other than a SolderCore, such as a Raspberry Pi or on a PC under Windows or Linux, CoreBASIC's time and date is queried from the operating system rather than by pinging a time server directly.

How the time is stored

The time of day clock `CORE.TIME` increments once per second. The format of `CORE.TIME` happens to be *Unix* or *POSIX* time, which is the number of elapsed seconds since midnight UTC, 1 January 1970 and is the standard way of representing time in CoreBASIC. See **Unix time - Wikipedia, the free encyclopedia**.

Other sources for time and date

There are a number of sources that you can use to initialize CoreBASIC's time of day:

- Manually setting the time and date by direct assignment to `CORE.TIME`. Manually setting the core time this way enables you to test your application with a known time and date.
- From the time delivered by a network time server using NTP. This is the default way of initializing the time when you have the CoreBASIC system connected to the Internet.
- From the time of day stored in an external real time clock chip (RTC). Usually, real time clock ICs are battery-backed so that they continue to mark time even when the CoreBASIC system is powered off. This is an ideal way of maintaining time and date for a system that is not connected to the Internet.
- From the time broadcast by the Global Positioning System (GPS). GPS time is highly accurate, but the drawback is that the receiver's antenna needs to be in view of a set of satellites and usually positioned near a window, or even outside, for this to work reliably.

Debunking common misconceptions

There will always be language wars in computer science. BASIC was designed to help beginners start to program, but BASIC is the foundation of the Microsoft empire.

Here we'll deal with a few common misconceptions about BASIC and its implementation.

BASIC is slow

BASIC isn't slow; an implementation of BASIC may be slow, but there are many BASIC compilers in the world and they produce top-notch code which out-performs C in some cases. And there is no reason for a BASIC interpreter to be any slower than a interpreted eLua, Python, or Perl: it all comes down to the design and implementation of the interpreter, not the source language.

For concrete proof of SolderCore's performance, see **Benchmarking CoreBASIC**.

BASIC is bad: syntax errors are caught at run time

Not in CoreBASIC: the CoreBASIC interpreter checks the structure and syntax of your program before execution begins. For some less-capable BASIC interpreters, syntax checking happens when executing a program, and it's really frustrating finding syntax errors only when your program runs! Because CoreBASIC pre-flight checks your program before running, the execution engine doesn't syntax check each statement every time it's executed, over and over again. The benefit? Your program runs at blistering speed!

Short variable names speed up your program

Not in CoreBASIC: all variables are accessed in constant time, irrespective of the length of their name. For some less-capable BASIC interpreters, it *is* true that shorter variable names lead to faster programs.

It's faster to GOTO a line at the start of the program than at the end

Not in CoreBASIC: all GOTO and GOSUB statements execute in constant time, irrespective of whether the target line is at the start or end of a program. For some less-capable BASIC interpreters, it *is* true that using GOTO takes more time to find a line at the end of a program that at the beginning of a program.

BASIC programmers learned to put all their subroutines at the start of a program so that their code ran faster. You just don't need to do this in CoreBASIC—and besides, you should be using named procedures anyway!

String variables can't hold more than 255 characters

Not in CoreBASIC: all strings can hold up to 65,535 characters. Some BASIC interpreters do limit the number of characters in a string and that may well be a problem for some applications.

A FOR loop executes at least once

Not in CoreBASIC: if the loop should not execute because the initial value exceeds the final value, the loop body is skipped. For some less-capable BASIC interpreters, it is true that the loop body is executed at least once, even if the initial value exceeds the final value.

BASIC freezes for ages when garbage collection kicks in

Not in CoreBASIC: the CoreBASIC garbage collector is extremely fast, taking but a moment to scavenge through memory. Historically, the simple garbage collector in Microsoft's early interpreted BASIC did suffer this problem, but Microsoft improved their garbage collector over time.

CoreBASIC offers an excellent two-stage garbage collector which you just won't notice.

Line numbers? Seriously?

Well, yes and no. You can use line numbers to type in old programs and use `GOTO` and `GOSUB`, but BASIC has moved on. CoreBASIC is fully structured and includes user-defined procedures, so you never need to use a line number.

CoreBASIC Language Reference

This is the online documentation set for **CoreBASIC**. CoreBASIC is a programming language for embedded microcontrollers. It's easy to use, it's powerful, and most of all it's *interactive*.

Get familiar with CoreBASIC

Change History Please take a moment to read the changes in this version of CoreBASIC.	**Keywords by function** The nuts and bolts of the CoreBASIC language, organized by task.	**Keywords, A to Z** The complete reference to the CoreBASIC language, from A to Z.
Drivers by function An introduction to the capabilities of the drivers built into CoreBASIC.	**Drivers by vendor** A complete reference to the hardware capabilities programmed into CoreBASIC.	**Example programs** A collection of examples that demonstrate how to use CoreBASIC to develop your own projects.

Copyright

Disclaimer

The information contained in this manual is subject to change and does not represent a commitment on the part of the copyright holder. While the information contained herein is assumed to be accurate, Rowley

Associates assumes no responsibility for any errors or omissions. In no event shall Rowley Associates, its employees, its contractors, or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees, or expenses of any nature or kind.

Trademarks

Change history

While we have done all we can to test CoreBASIC, this is early in the development cycle and, as such, please be warned that there will be the inevitable bug. We encourage you to report any bugs to us so they can be corrected for all CoreBASIC users.

Release 1.3.6 changes

New items:

- Added `BOSCH-SENSORTEC-BMA250` driver. See **Bosch Sensortec BMA250 Driver**.
- Added `BOSCH-SENSORTEC-BMM150` driver. See **Bosch Sensortec BMM150 Driver**.
- Added `STMICROELECTRONICS-LIS3DSH` driver. See **STMicroelectronics LIS3DSH Driver**.
- Added `STMICROELECTRONICS-LIS3LV02DL` driver. See **STMicroelectronics LIS3LV02DL Driver**.

Release 1.3.5 changes

Major items:

- FAT32 support is preliminary. This needs some real testing on more cards. During the work for FAT32, I benchmarked a selection of other cards in 1-bit SPI and 4-bit bus mode, and the results are surprising. I will reiterate what my testing confirms: *Please select SanDisk cards for your SolderCore*. Genuine Samsung cards are also pretty good, from the small sample I have.

New items:

- Added READ$. See **READ$**.
- Added `INPUT$`. See **INPUT$**.

Usability improvements:

- Added `APPEND` mode to `OPEN`. Although CoreBASIC already supports appending to a stream *s* by setting `POS` *s* = `EXT` *s* after opening, it seems that many users expect some form of integrated "open for append" mode. So, `APPEND` is now a valid open mode. See **OPEN**.

General improvements:

- Fixed long-standing bug where out-of-band data is sometimes parsed using a stale packet interrupt handler.
- Extended `streams` system command to dump more information for socket streams.

Release 1.3.4 changes

Usability improvements:

- Added paging option to `LIST`. See **LIST USING**.

- **Ctrl+C** can interrupt `FIRMWARE DOWNLOAD`.
- Reinstated automatic flush on `PRINT` for socket streams. Fair warning, I will revisit auto-flush on streams over the coming weeks and may well require users to manually flush their streams to guarantee pushing data to the network.

Release 1.3.3 changes

New items:

- Added RENAME to rename files. See **RENAME**.

Usability improvements:

- Extended `EDIT` to load and edit files. See **EDIT**.
- Improved TCP performance on noisy and busy networks.

Release 1.3.2 changes

Usability improvements:

- Increased default DNS name lookup timeout to three seconds. This prevents timeout errors with slow DNS lookups where the DNS records are not cached close to the SolderCore.

New items:

- Added `ANSI-GRAPHICS` driver to provide 8-color graphics capability on a terminal. If no graphics driver is installed and you request graphics by using a graphics keyword, `ANSI-GRAPHICS` is installed by default. This means that all graphics-related demonstrations work out of the box on any CoreBASIC target. The bouncing line demo, `EXAMPLE "bouncing-lines"`, is particularly nice when run with the ANSI graphics driver. See **ANSI Graphics Driver**.
- Added `XTERM-GRAPHICS` driver to provide 216-color graphics capability on a terminal. This is similar to the `ANSI-GRAPHICS` except that `XTERM-GRAPHICS` uses the 256-color mode of xterm (supporting a 6x6x6 color cube). This driver requires an xterm-compatible terminal emulator, such as Tera Term. See **Xterm Graphics Driver**.

Release 1.3.1 changes

Major items:

- Added hardware watchdog protection capability with `WATCHDOG` commands. See **WATCHDOG** and **Watchdog protection**.

Usability improvements:

- Added `NEWS` to display the CoreBASIC release history. See **NEWS**.
- Enhanced `HELP`, `WEB`, `NEWS`, and `EXAMPLE CATALOG` to display content one page at a time with the option to continue or quit.

- Extended IF to allow cascaded ELSE IF. See IF ... THEN.
- Improved debugger experience when stepping IF statements: the debugger automatically advances over ELSE and ENDIF tokens rather than requiring an additional step.

Release 1.3.0 changes

Major items:

- Added HTTP-SERVER to serve HTTP requests from the SolderCore. SolderCore gets a web server! See HTTP Server.
- Added FTP-SERVER which enables upload and download of files with an FTP client such as FileZilla or Windows FTP. See FTP Server.
- Added EXTENDED-USER-MEMORY driver to extend long variable name storage so you can use lots of meaningful variable names. See Extended User Memory.

Usability improvements:

- Improved loading speed for large programs.
- Fully implemented a "current folder" capability.
- Added CHDIR and CD. See CHDIR and CD.
- Modified the way DIR works. See DIR.
- Added $CWD. See $CWD.
- Improved CATALOG directory listing. See CATALOG.
- Extended MEMORY to dump more information. See MEMORY.
- Extended INSTALL LIST to show high memory use. See INSTALL LIST.
- Extended PIN LIST to show pin mode, pin speed, and drive strength.
- Extended "DOS wedge" shortcuts.

New hardware support:

- Added FREESCALE-MPL3115A2 driver. See Freescale MPL3115A2 Driver.
- Added FREESCALE-MMA8491Q driver. See Freescale MMA8491Q Driver.
- Added CISECO-LED-MATRIX-SHIELD driver.
- Added AIRSENSOR-128GLCD driver. See AirSensor 128GLCD.

Release 1.2.30 changes (beta)

- Added LINEAR-TECHNOLOGY-LTC2309 driver.
- Added AIRSENSOR-192GLCD driver.
- Added DIRECTION properties for parallel bus drivers.

Release 1.2.29 changes (beta)

- Added CORE.Y and CORE.G properties on the SolderCore to allow configuration and control of the LAN connector's yellow and green LEDs.

- Reverted SolderCore LCD driver: some releases shipped with an LCD driver with left-over debugging code in.

Release 1.2.28 changes (beta)

- Added FREESCALE-MPL115A2 driver.
- Added LOCK and UNLOCK statements.
- Corrected SENSECORE driver to match production variant (thanks to Arthur for bringing this to our attention). The production SenseCore uses a PCA9655 port expander and the preproduction SenseCore uses an PCF8575 port expander. You can install a driver for preproduction SenseCores using the specific driver name SOLDERCORE-SENSECORE-PCF8575 or the common name SENSECORE-PCF8575. The production SenseCore has specific name SOLDERCORE-SENSECORE-PCA9655 and common names SENSECORE-PCA9655 and simply SENSECORE.
- Added automatic translation of // to ' for Iain, who finds it hard transitioning from C++ to CoreBASIC.

Release 1.2.27 changes (beta)

- Added much-requested HISTORY command set to record command line history. By default, history recording is not turned on, you can turn it on in your boot file if needed. Commands are recorded to /c/sys/history.log so you will need a mounted SD card for command line history to work. Because the SolderCore keeps appending to the history, if your history grows long, recalling and appending to the history file may well take some time and reduce interactivity. If you find this happens, just use HISTORY KILL to clear the history file and restore quick responses.
- When invoking EDIT immediately after an error is reported, the full screen editor places the cursor on the offending line rather than at the first program line.
- Added $ATTR.

Release 1.2.26 changes (beta)

- Added MICROCHIP-MCP4725 driver.
- ADC resolution increased to 12 bits for greater precision on analog inputs.
- Reverted to use the internal AREF for analog channels.

Release 1.2.25 changes (beta)

- The analog voltage reference defaults to AREF on the header and is not configurable in software. Previous releases used the internal 3.0V reference, independent of AREF. Note that the SolderCore is shipped with AREF on the header connected to the analog input reference voltage; you can select the 3V3 rail as as the reference voltage by moving solder jumper JP4.

Release 1.2.24 changes (beta)

- Added capability of reading A4 and A5 in analog mode when JP1 and JP3 select analog mode for the header pins. Use pin numbers 20 and 21 when JP1 and JP3 select analog mode.

- Added $BLINK, $STEADY, $NEGATIVE, $POSITIVE, $BOLD, $REGULAR, and $RESET.

Release 1.2.23 changes (beta)

- Additional User Guide material.
- Ran PC-lint over CoreBASIC sources to clean up any potential issues.
- Added SILICON-LABS-SI7005 driver.

Release 1.2.22 changes (beta)

- Changed the definition of the RESOLUTION property of the BOSCH-SENSORTEC-BMP085 driver.

Release 1.2.21 changes (beta)

- Added ITEAD-STUDIO-ITDB02-4.3 and ITEAD-STUDIO-ITDB02-5.0 drivers.

Release 1.2.20 changes (beta)

- Added ANALOG-DEVICES-ADXL362 driver.
- Added HONEYWELL-HIH6130 driver.
- Added FILL LINE USING capability.

Release 1.2.19 changes (beta)

- Rationalized internal temperature sensor interfaces.
- Added TEXAS-INSTRUMENTS-TMP100 driver.
- Added TEXAS-INSTRUMENTS-TMP102 driver.
- Added ANALOG-DEVICES-ADT7410 driver.
- All temperature sensor drivers now implement a RESOLUTION property.

Release 1.2.18 changes (beta)

- Added STMICROELECTRONICS-LIS331HH driver.
- Added STMICROELECTRONICS-L3G4200D driver.

Release 1.2.17 changes (beta)

- Added ATMEL-ATAVRSBIN1 driver.
- Added ATMEL-ATAVRSBIN2 driver.
- Added BOSCH-SENSORTEC-BMA150 driver.
- EXAMPLE will not automatically append .bas if an extension is already present.

Release 1.2.16 changes (beta)

- Added SOFTWARE-SPI driver.

- CoreBASIC will now automatically close files and recycle file handles if files are not closed by a user program. File handles are recycled when the program in memory is changed or saved.
- Added SENSIRION-SHT2X driver. This is Iain's first crack at writing a CoreBASIC driver! But Paul did clean it up a bit…
- Added KIONIX-KXTF9 driver.
- Added STMICROELECTRONICS-LPS331AP driver.

Release 1.2.15 changes (beta)

- More testing carried out for next release.
- Added MAXIM-MAX6675 driver.

Release 1.2.14 changes (beta)

- Exposed internal formatting modes that CoreBASIC supports by adding LIST USING. Noteworthy is LIST USING mode two which dumps syntax-highlighted HTML listings for direct inclusion into web pages.
- Added FREESCALE-MPL115A1 driver.

Release 1.2.13 changes (beta)

- Added BGET and BGET$ to read a stream in binary mode.
- Added a CoreBASIC example that replicates the function of FIRMWARE GET.
- Reduced time between socket creation and connection associated with firmware download such that the socket is not recycled prematurely.
- Updated DS1340 driver so that DAY property computes the day from the date held in the RTC registers and does not read the internal RTC day register.

Release 1.2.12 changes (beta)

- Added RAVEL function to ravel an array into a vector.
- Added DFT and IDFT functions from the CoreBASIC DSP module.

Release 1.2.11 changes (beta)

- Added MAXDETECT-DHT11 and MAXDETECT-DHT21 humidity sensor drivers. This will work for RHT03-compatible devices (DHT21, DHT22, RHT02, AM2301 devices) which use the same protocol but come in different packaging.
- Added LEFT property to the UART to inquire number of characters remaining to be sent from the output buffer.

Release 1.2.10 changes (beta)

- Ironed out the final kinks, I hope, with the KS0108 LCD driver started in 1.2.6.

- Added TIME property to DS1340 driver to read and write the current time and date in a single transaction.
- Added DAY%, DATE%, MONTH%, YEAR%, HOUR%, MINUTE%, and SECOND% to breakdown time and date of a standard CoreBASIC time (seconds elapsed since 1 January 1970).

Release 1.2.9 changes (beta)

- Improved CATALOG to color-code files and directories and to inhibit listing system and hidden files.
- Fixed reset of working filename when loading; loading sets the working filename to the file loaded.
- Refined fix for PTR positioning issued in 1.2.8.

Release 1.2.8 changes (beta)

- Improved CoreBASIC start-up time when no SD card is present.
- Fixed bug in mass storage code when moving the current file position by assigning to PTR.
- If the system time is not set, newly created files and directories are given a date of 1 January 2012.

Release 1.2.7 changes (beta)

- Fixed bug introduced into PAUSE by 1.2.6 enhancement.

Release 1.2.6 changes (beta)

- Added SAVE AUTO capability.
- Added CORE.TEMP property to measure SolderCore's die temperature.
- Added CORE.I2C(n) property to access hardware I2C bus *n*.
- Added CORE.SPI(n) property to access hardware SPI bus *n*.
- Added USING clause to I2C statement to specify I2C bus to run transaction on.
- Added MOUNT and EJECT to quickly mount and eject SD cards from CoreBASIC.
- Added MICROCHIP-MCP342X ADC driver. Untested at present!
- Added MICROCHIP-MCP23017 port expander driver to support the AirSensor KS0108 LCD on an MCP23017 port expander.
- NEW now sets the working filename to /c/work.bas so that it is harder to inadvertently save over your last-saved file.
- PAUSE no longer imposes a maximum wait time; additionally, it now checks for user break-ins every 500 ms.
- Added SOFTWARE-I2C driver to create an I2C bus from a pair of digital I/O signals. The software I2C bus allows a Sensirion SHT1x humidity sensor to share the bus with I2C-compliant devices.
- All I2C-based drivers which were bound to using the primary I2C bus for communication are now extended with an optional USING clause. The optional USING can specify the I2C bus that the device is attached to. Shields where I2C is fixed to the primary I2C bus continue to use the primary I2C bus and do not support this form of USING.
- Documented additional PIN capabilities that have been in CoreBASIC for some time.

Release 1.2.5 changes (beta)

- Documented and released NMEA-PARSER driver.
- Added CVF.
- Added $CLS.
- Added MEMORY at Iain's request.
- Added MODULES.
- Removed DATA qualifier from CRUNCH: all uses of CRUNCH now report program size reduction.

Release 1.2.4 changes (beta)

- Added PORT option to SOLDERCORE-UART driver which enables use of both hardware UARTs in CoreBASIC on the SolderCore.

Release 1.2.3 changes (beta)

- Added SENSIRION-SHT1X driver.
- Added LINEAR-TECHNOLOGY-LTC6904 driver.
- Documented AMS-TSL2561 driver.

Release 1.2.2 changes (beta)

- Added PIN LIST and PIN CATALOG.
- Added INSTALL LIST.
- Added CVI and CVU.

Release 1.2.1 changes (beta)

- Added FREEDOM-ACCELEROMETER driver for the Freedom Board.
- Added MMA8451Q driver.

Release 1.2.0 changes

- CoreBASIC rough-cut release for the Freescale Freedom Board. The Freedom Board has 128 KB of flash and 12 KB of RAM, but in that space we managed to squeeze full support for the CoreBASIC language, I2C, SPI, graphics, on-chip editor, and a selection of sensor and shield drivers. Because of space constraints, we have limited support to the SolderCore shields and sensors. Connect a terminal to the OpenSDA CDC UART at 115200 baud and press reset for the CoreBASIC experience!
- A known problem on the Freedom Board is that CRUNCH MAX will crash.
- The Freedom Board does not support ANALOG OUTPUT mode yet.
- The Freedom Board does not support ANALOG INPUT mode yet.
- Added FREEDOM-CPU driver for the Freedom Board.

Release 1.1.0 changes

- This is an Internal Testing release.

- Core internal platform APIs rewritten to make porting CoreBASIC to other hardware easier. A benefit of this rewrite is that all core GPIO is faster and therefore all GPIO-intensive drivers are faster.
- CoreBASIC port to Raspberry Pi and Panda Board using SDL is now minimally functional. Excellent work, Jon! The existing port of CoreBASIC running on Windows using Qt will be retired.

Release 1.0.0 changes

- Improved TCP/IP buffer handling.
- Added `%WIDTH` and `%HEIGHT`.

Release 0.9.16 changes

- Added `%COLOR` and `COLOR$`.
- Added `REBOOT`.
- Added `INSTR`.
- Enhanced VDU can now output to channels.
- TCP ports that are not open will simply drop connection requests rather than refuse connections.
- Fixed excess NTP requests being made to the local network with no explicit time server set.

Release 0.9.15 changes

- Added `WEB` keyword.
- Fixed an issue with the DHCP client when DHCP servers issued 10-year leases to an IP address. Thanks for the bug report Andreas! **http://www.watterott.com/**
- Added support for InvenSense MPU-6000 Evaluation Board with AK8975 magnetometer. The MPU-6000EVB driver integrates the AK8975 which is fully isolated on the auxiliary I2C bus. Electrical connection does not require anything special, i.e. I2C pass-through is not required for initialization.
- Documented `$LEFT, $RIGHT, $UP, $DOWN`.
- Added `%IN` and `%OUT`.
- Extra documentation for UART and additional polling capabilities.

Release 0.9.14 changes

- Added device support for MPU-6150 and MPU-9150. The type of device, MPU-6050, MPU-6150, or MPU-9150, and its revision is detected by the `INVENSENSE-MPU-6050` driver and automatically set up.
- Added `INVENSENSE-MPU-EVB` driver to support the InvenSense MPU-6050EVB and MPU-9150EVB evaluation boards with AK8975 magnetometer.

Release 0.9.13 changes

- Added `ITDB02-2.2` driver.
- Added `$OFF` and `$ON`.
- Automatically turn the cursor on when returning to the visual editor.
- Corrected a problem in the AHRS driver when fusing gyroscopes and accelerometers on separate devices (introduced in 0.9.6 firmware).

- Added VERSION to display the installed firmware version The version number continues to be provided by the CORE.VERSION property.
- Added Conway's Game of Life demos for running on a standard console (no extra hardware required) and the Jimmie Rodgers LoL Shield. These are easily adapted for other bitmap displays, and it runs brilliantly on an Arcade Shield or LCD Shield.
- Improved performance of LET when assigning to a multi-dimensional array.

Release 0.9.12 changes

- Added HIGH to retrieve the high bound of a vector.
- Added DOT and CROSS for vector dot and cross products.
- Added ITDB02-2.4E driver which works on both the ITDB02-2.4E LCD module on an ITDB02 shield or the ITDB02-2.4E shield.

Release 0.9.11 changes

- HMC5883L driver now initializes the magnetometer bandwidth to 75 Hz on installation.
- Added SPI-based MPU-6000 driver. This was tested on an InvenSense MPU-6000 Evaluation Board connected to a CoreProto rivet plugged into a SenseCore.
- Documented MORSE$.
- Fixed an editor lock-up and prevented Page Down advancing beyond the end of the source text.
- Added alpha ITDB02-4.3 and ITDB02-5.0 drivers; these panels seem to suffer problems when mounted on the ITDB02 v1.2 shield and must not be considered stable at this time. We are working to resolve this with ITead Studio.

Release 0.9.10 changes

- Added statement TRY and ERROR for user-supplied error handling.
- Documented SHA1$.
- Streamlined ITDB02 drivers to recover some space in flash.
- ITDB02 2.8" LCD Touch Shield tested and confirmed working with the ITDB02-2.8 driver.

Release 0.9.9 changes

- Added FLUSH to allow programmed push of partial TCP segments.
- CLOSE does not throw an exception on a close of closed channel.
- CLOSE on a socket channel initiates a full close, rather than a half-close, of the open socket.
- Documented URI$, BASE64$, and JUSTIFY$.

Release 0.9.8 changes

- Added @ to index an array or string by a unary expression. X@I is identical to X(I) in all contexts where I is a unary expression. X@I is a shorthand inspired by the ? and ! operators of BBC Basic. Using @ takes slightly less program space and executes marginally faster than X(I).

- Extended QUAT to accept two arguments, a scalar part and a vector part, to construct a quaternion.
- Fixed a bug in WRITE which can cause the token pointer to be off by one in some cases.

Release 0.9.7 changes

- Renamed RPT to REPEAT$ for compatibility with PowerBASIC programs.
- STRING$ added for compatibility with Microsoft BASIC programs.
- Extended CHR to accept string arguments.
- The default I2C address of the CORE-MPU driver is changed from 0xD0 to 0xD2 to match the CoreMPU production schematics. The default I2C address for the MPU-6050 remains at 0xD0.
- The AT function is renamed to TAB.
- Added property selectors I, J, K to directly access the vector parts of a quaternion, and R to directly access the scalar part of a quaternion. Assignment to a component part of a quaternion is not supported.
- Added property selector V which returns the vector part of a quaternion a three-element vector, as IM does.

Release 0.9.6 changes

- SOLDERCORE-GRAPHICS-SHIELD driver added which dynamically installs either the SolderCore Arcade Shield driver or the SolderCore LCD Shield Driver, depending upon which device is detected as attached to the SolderCore.
- The default I2C address of the bus expander used by the SOLDERCORE-SENSECORE driver is changed from 0x46 to 0x42 to match the SenseCore production schematics.
- The SOLDERCORE-LCD-SHIELD and SOLDERCORE-ARCADE-SHIELD drivers have been updated to work only with firmware versions 133 and later. If you have versions of these shields running pre-production firmware, please use an XTAG-2 programmer to upgrade the firmware over JTAG rather than over SPI.
- More microSD cards are supported in CoreBASIC. We still recommend that you use genuine SanDisk cards in your SolderCore.
- The DELAY statement is removed and all examples are updated to use the equivalent PAUSE statement.
- HELP has been extended to display help for drivers when followed by a string containing a valid driver name.
- Improved performance of the AHRS driver by using advertised internal interfaces to accelerometers, gyroscopes, and magnetometers.
- Fixed a problem where CoreBASIC would potentially crash after installing a driver that throws a CoreBASIC exception.

Keywords by function

Task	Example keywords
Write and edit programs	NEW, LIST, EDIT…
Load and save programs	LOAD, SAVE, NAME, EXAMPLE…
Build loops and decision structures	FOR ... NEXT, WHILE ... WEND, IF ... THEN, CASE ... ENDCASE…
Define and call procedures and subroutines	DEFPROC ... ENDPROC, CALL, GOSUB, RETURN…
Device input and output	PRINT, INPUT, GET$…
Manipulate and transform strings	LEFT, RIGHT, MID…
Calculate with numbers	+, -, *, /, ^ …
Display graphics and text	LINE, CIRCLE, DRAW…
Mathematical functions	SQR, SIN, COS, TAN, EXP, LOG, ATN2…
Manipulate and transform arrays	LEN, SORT, SHUFFLE, GEN…
Linear systems and matrices	DIM, MAT, TRN, INV…
Complex numbers	%I, CMPLX, RE, IM…

Write and edit programs

Basics	
NEW	Start a new program
LIST	List program, or part of program, to screen
AUTO	Automatically number a program
EDIT	Edit part, or all of, your program
Development	
DELETE	Delete part of a program
RENUMBER	Renumber the program
CHECK	Check program syntax
MEMORY	See how much space your program takes
Transforming programs	
CRUNCH	Crunch program to reduce memory requirement

Load and save programs

When developing programs you will want to manage the external media that programs are stored on. You can do that with the following keywords:

Loading and saving programs	
LOAD	Load a program from external storage
SAVE	Save a program to external storage
NAME	Name your program
EXAMPLE	Load example programs from the network
MERGE	Merge a program from external storage
Listing the contents of your disks	
CATALOG	Display folder contents
DIR	Shorthand for interactive CATALOG.
Managing your storage	
KILL	Remove a file from external storage
RENAME	Rename an existing file
CHDIR	Change working folder
MKDIR	Create a folder
RMDIR	Remove a folder
MOUNT	Mount a volume
EJECT	Eject a volume

Build loops and decision structures

Making decisions	
IF ... THEN	Conditional execution
CASE ... ENDCASE	Multi-way branch
IFF	Expression "if-else"
Loops	
FOR ... NEXT	Iterate a fixed number of times
FOR EACH ... NEXT	Iterate over an array
REPEAT ... UNTIL	Repeat statements while a condition is false
WHILE ... WEND	Repeat statements while a condition is false

Define and call procedures and subroutines

Procedures	
DEFPROC	Define a named procedure
CALL	Call a procedure
Classic subroutines	
GOSUB	Call a subroutine
RETURN	Return from a subroutines

Device input and output

Simple input and output	
PRINT	Write formatted data to a file or stream
INPUT	Get user input or read from a file or stream
GET, GET$	Wait for a single keystroke
File access	
OPEN	Open a file for reading or writing
CLOSE	Close a file or stream
BGET, BGET$	Read a byte from a file.
File operations	
EOF	Inquire if at end of file
EXT	Set extent of file
EXT()	Find extent of file
PTR	Set position within file
PTR()	Find position within file

Calculate with numbers

Simple operations	
+	Add numbers
-	Subtract numbers
*	Multiply numbers
/	Divide numbers
^	Exponentiate numbers
MOD	Compute remainder after division
\	Integer division
Comparing numbers	
=, <>	Compare numbers for equality
<, <=, >, >=,	Order numbers
MAX	Maximum of two numbers
MIN	Minimum of two numbers
Operating on numbers	
ABS	Compute absolute value
SQR	Compute square root
SGN	Compute sign of a number
INV	Reciprocal of a number
Special functions	
CEIL	Compute smallest integer not greater than…
FIX	Floor
FLT	Convert to floating
INT	Integer part of a number
Miscellaneous functions	
RND	Generate a random number

Manipulate and transform strings

Getting part of a string	
LEFT	Return a number of characters from the left part of a string
MID	Return a number of characters from the middle of a string

Display graphics and text

Setting up graphics	
ORIGIN	Move graphics origin
MOVE	Move drawing position
COLOR	Select color to draw shapes and text
CLG	Clear and initialize graphics display
Simple drawing	
LINE	Draw lines
CIRCLE	Draw or fill a circle
PLOT	Plot points
Drawing text	
FONT	Select font to draw text
DRAW	Draw text on graphics display
Colors	
RGB	Construct a color
RED%	Extract red component of a color
GREEN%	Extract green component of a color
BLUE%	Extract blue component of a color
Predefined colors	
%BLACK	Black color
%WHITE	White color
%RED	Red color
%GREEN	Green color
%BLUE	Blue color
%CYAN	Cyan color
%MAGENTA	Magenta color
%YELLOW	Yellow color
Miscellaneous	
GFX	Inquire graphics capabilities

Manipulate and transform arrays

Constructing arrays	
[...]	Construct array
GEN	Create arithmetic progression
CON	Create unitary array
ZER	Create zero array
DIM	Assign zero matrix
Properties of an array	
LEN	Number of elements in an array
HIGH	Upper bound for an array.
Getting part of an array	
LEFT	Return a number of elements from the left part of an array
MID	Return a number of elements from the middle of an array
RIGHT	Return a number of elements from the right part of an array
Accessing matrices	
COL	Extract column of matrix
ROW	Extract row of matrix
PICK	Pick elements of an array by predicate
SELECT	Select elements of array by index
RAVEL	Ravel an array to a vector
Inquiring arrays	
MAX	Return maximum element of an array
MIN	Return minimum element of an array
Operating on arrays	
SUM	Sum all elements of an array
SORT	Sort array into ascending order
SHUFFLE	Shuffle an array into random order
INNER	Compute inner product
REDUCE	Reduce an array using a binary operator
MERGE	Merge an array into a single string
Generating arrays	
EXPAND	Expand string to array of ASCII codes

SAMPLE	Sample properties into an array
SPLIT	Split string into an array of strings

Complex numbers

Create complex numbers	
%I	Imaginary unit and pure imaginary constructor
CMPLX	Construct complex number
Extract parts	
RE	Extract real part
IM	Extract imaginary part
Operate on complex numbers	
ABS	Compute magnitude
SGN	Compute signum
ARG	Compute argument
CNJ	Compute complex conjugate
Conversions	
ROT	Convert to rotation matrix

Mathematical functions

Conversion functions	
DEG	Convert radians to degrees
RAD	Convert degrees to radians
Trigonometric functions	
SIN	Compute sine
COS	Compute cosine
TAN	Compute tangent
CIS	Compute sine and cosine
Inverse trigonometric functions	
ASN	Compute inverse sine
ACS	Compute inverse cosine
ATN	Compute inverse tangent
ATN2	Compute inverse tangent with two arguments
Roots, exponentials, and logarithms	
SQR	Compute square root
LOG	Compute natural logarithm
EXP	Compute natural exponential
LOG10	Compute base 10 logarithm
LOG2	Compute base 2 logarithm
Hyperbolic functions	
SINH	Compute hyperbolic sine
COSH	Compute hyperbolic cosine
TANH	Compute hyperbolic tangent
Inverse hyperbolic functions	
ASNH	Compute inverse hyperbolic sine
ACSH	Compute inverse hyperbolic cosine
ATNH	Compute inverse hyperbolic tangent

Linear systems and matrices

Creating matrices	
[...]	Construct matrix
IDN	Create identity matrix
CON	Create unitary matrix
ZER	Create zero matrix
Assigning matrices	
MAT LET	Assign matrix
DIM	Assign zero matrix
Accessing matrices	
COL	Extract column of matrix
ROW	Extract row of matrix
Basic matrix functions	
+	Add matrices
-	Subtract matrices
*	Multiply matrices
/	Right divide matrices
\	Left divide matrices
Operating on matrices	
TRN	Transpose matrix
DET	Compute determinant
INV	Compute matrix inverse
ROT	Compute rotation matrix

Keywords, A to Z

This section contains a description of the commands and operators that CoreBASIC understands.

Symbols

'	+	-	*
/	:	<	<=
=	<>	>	>=
[	\	]	^
|			

$

$BEL	$BLACK	$BLINK	$BLUE
$BOLD	$BOOT	$BS	$CLS
$CRLF	$CR	$CSI	$CWD
$DOWN	$EOF	$ESC	$FF
$GREEN	$LEFT	$LF	$NL
$MAGENTA	$NEGATIVE	$NET	$NUL
$OFF	$ON	$POSITIVE	$RED
$REGULAR	$RESET	$RIGHT	$RUN
$STEADY	$TAB	$UP	$VT
$WHITE	$WORK	$WS	$YELLOW

%

%BLACK	%BLUE	%COLOR	%CYAN
%E	%FALSE	%IN	%GREEN
%HEIGHT	%I	%MAGENTA	%OUT
%RED	%WHITE	%WIDTH	%YELLOW
%PI	%TRUE		

A

ABS	ACS	ACSH	AND
AND THEN	ARG	AS	ASC
ASN	ASNH	ATN2	ATNH
AUTO			

B

BASE64$	BGET	BGET$	BLUE%
BYE			

C

CALL	CASE	CATALOG	CD
CEIL	CHAIN	CHDIR	CHECK
CHR	CINT	CIRCLE	CIS
CLG	CLOSE	CLS	CMPLX
CNJ	CON	COL	COLOR
COLOR$	CORE	COS	COSH
CREDITS	CROSS	CRUNCH	

D

DATA	DATE$	DATE%	DAY%
DEBUG	DEFPROC	DEG	DELETE
DELETE$	DET	DFT	DIM
DIR	DNS	DOT	DRAW
DUMP			

E

EDIT	ELSE	EJECT	END
ENDCASE	ENDIF	ENDPROC	EOF
EQV	ERROR	EXAMPLE CATALOG	EXAMPLE LOAD
EXIT FOR	EXIT REPEAT	EXIT WHILE	EXP
EXPAND	EXT	EXT()	

F

FALSE	FILL	FIRMWARE CATALOG	FIRMWARE CHECK
FIRMWARE GET	FIRMWARE KILL	FIRMWARE RUN	FIRMWARE SAVE
FIX	FLUSH	FLT	FOR
FONT	FONT CATALOG		

G

GEN	GET	GET$	GOTO
GFX	GOSUB	GREEN%	

H

HELP	HEX	HIGH	HISTORY LIST
HISTORY KILL	HISTORY OFF	HISTORY ON	HISTORY PICK
HOUR%			

I

I2C	IDFT	IDN	IF
IFF	IM	IMP	IN
INF	INK	INNER	INPUT
INPUT$	INSERT$	INSTALL	INSTALL CATALOG
INSTALL LIST	INSTR	INT	INV

J

JOIN	JUSTIFY$		

K

KILL			

L

LCASE	LEFT	LEN	LET
LIST	LIST USING	LOCK	LOG
LOG10	LOG2	LOAD	LTRIM

M

MAIL	MAT	MAT LET	MAT PRINT
MAT()	MATCH	MAX()	MAX
MEMORY	MERGE	MERGE()	MID
MIN	MIN()	MINUTE%	MKDIR
MKDIR()	MKF	MKI	MOD
MODULES	MONTH%	MORE	MOUNT
MOVE			

N

NAN	$NAME	NAME	NET
NEW	NEWS	NEXT	NUMBER$

O

OPEN	OR	OR ELSE	ORIGIN

OTHERWISE			

P

PAPER	PAUSE	PI	PICK
PIN	PIN CATALOG	PIN LIST	PIN()
PLOT	PRINT		

Q

QUAT			

R

RAD	RANDOMIZE	RAVEL	RE
READ	READ$	REBOOT	RECTANGLE
RECYCLE	RED%	REDUCE	REM
RENAME	RENUMBER	REPEAT	REPEAT$
REPORT	REPORT()	RESTORE	RETURN
REVERSE	RGB	RMDIR	RMDIR()
RND	ROT	ROW	RTRIM
RUN	RUN()		

S

SAMPLE	SAVE	SAVE AUTO	SECOND%
SELECT	SGN	SHA1$	SHUFFLE
SIN	SINH	SOCKET	SORT
SPLIT	SPC	SPOKEN$	SQR
STEP	STOP	STR	STRING$
SUBST$	SUM	SYSTEM	

T

TAB	TAN	TANH	THEN
TIME$	TIMER	TO	TRIM
TRN	TRUE	TRUTH	TRY

U

UCASE	UNLOCK	UNTIL	URI$
UTF			

V

VAL	VDU	VERSION	

W

WAIT	WATCHDOG()	WATCHDOG REBOOT	WATCHDOG RESTORE
WATCHDOG THROW	WATCHDOG TIMER	WEND	WHEN
WHILE			

X

XOR			

Y

YEAR%			

Z

ZER			

$constant

Synopsis

$*name*

Predefined string.

Description

$ provides convenient access to some predefined strings. Although these strings can be constructed using `INK` and `COLOR`, for instance, using a predefined name has benefits: it's faster, your stored program is smaller, and your program reads better.

Predefined ASCII characters

The following provide a convenient way to access single ASCII characters as strings.

Name	Description	Equivalent to
`$NUL`	null	`CHR 0`
`$BEL`	bell	`CHR 7`
`$BS`	backspace	`CHR 8`
`$TAB`	horizontal tab	`CHR 9`
`$LF` or `$NL`	line feed	`CHR 10`
`$VT`	vertical tab	`CHR 11`
`$FF`	form feed	`CHR 12`
`$CR`	carriage return	`CHR 13`
`$EOF`	end of file	`CHR 26`
`$ESC`	escape	`CHR 27`

Predefined strings

The following provide names for common strings:

Name	Description	Equivalent to
`$CRLF`	carriage return, line feed	`$CR + $LF`
`$CSI`	control sequence introducer	`$ESC + "["`
`$CLS`	clear screen, home cursor	`$CSI + "2J" + $CSI + "H"`
`$WS`	whitespace characters	`" " + $TAB + $CR + $LF + $VT + $FF`

Predefined inks

The following provide a convenient way to change `INK`:

Name	Description	Equivalent to
$BLACK	Black ink	INK 0
$RED	Red ink	INK 1
$GREEN	Green ink	INK 2
$YELLOW	Yellow ink	INK 3
$BLUE	Blue ink	INK 4
$MAGENTA	Magenta ink	INK 5
$CYAN	Cyan ink	INK 6
$WHITE	White ink	INK 7

Predefined attributes

The following provide a convenient way to change display attributes:

Name	Description	Equivalent to
$NEGATIVE	Negative display	$ATTR(7)
$POSITIVE	Positive display	$ATTR(27)
$BLINK	Blink on	$ATTR(5)
$STEADY	Blink off	$ATTR(25)
$BOLD	Bolding on	$ATTR(1)
$REGULAR	Bolding off	$ATTR(22)
$RESET	Reset all attributes	$ATTR(0)

Note that not all terminal emulators support every attribute, and the way that attributes are rendered may not always be intuitive. For instance, bold characters are usually displayed not by the type of bolding you see in print, but are displayed brighter to make them stand out. Blink is particularly difficult for some bitmap-based terminal emulators, and isn't generally supported by them.

Predefined actions

The following provide a convenient way to control the cursor:

Name	Description	Equivalent to
$OFF	Turn cursor off	$CSI + "?25l"
$ON	Turn cursor on	$CSI + "?25h"

Predefined file names

The following provide a quick way of referring to system files:

Name	Expands to	Description

$WORK	/c/work.bas	User work file name.
$RUN	/c/!run.bas	CoreBASIC "autoexec" file.
$NET	/c/sys/!network.bas	Network configuration file.
$BOOT	/c/sys/!boot.bas	Boot configuration file.

See also

$ATTR

$ATTR

Syntax

`$ATTR(`*n*`)`

Construct attribute change sequence.

Description

`$ATTR(n)` returns a string containing VT220 control codes that select attribute *n*. The string return for `$ATTR(n)` is identical to the following CoreBASIC statement:

```
$CSI + STR(n) + "m"
```

The attributes supported by a terminal emulator vary; you will need to consult the documentation of the terminal emulator you use to see which attributes are supported.

$CWD

Synopsis

$CWD

Current working folder name.

Description

$CWD returns the name of the current working folder.

Example

```
> print $cwd
/c
> chdir "www"
> print $cwd
/c/www
> chdir "/c/sys/backups"
> print $cwd
/c/sys/backups
> chdir ".."
> print $cwd
/c/sys
> _
```

See also

CHDIR

$DOWN

Syntax

$DOWN $DOWN(*n*)

Move cursor down.

Description

$DOWN returns a string containing control codes to move the cursor down one line. $DOWN(*n*) returns a string containing control codes to move the cursor down *n* lines, where *n* is constrained to lie in the interval [0, 100].

$LEFT

Syntax

$LEFT $LEFT(*n*)

Move cursor left.

Description

$LEFT returns a string containing control codes to move the cursor left one position. $LEFT(*n*) returns a string containing control codes to move the cursor left *n* positions, where *n* is constrained to lie in the interval [0, 100].

$RIGHT

Syntax

`$RIGHT` `$RIGHT(`*n*`)`

Move cursor right.

Description

`$RIGHT` returns a string containing control codes to move the cursor right one position. `$RIGHT(`*n*`)` returns a string containing control codes to move the cursor right *n* positions, where *n* is constrained to lie in the interval [0, 100].

$UP

Syntax

$UP $UP(*n*)

Move cursor up.

Description

$UP returns a string containing control codes to move the cursor up one line. $UP(*n*) returns a string containing control codes to move the cursor up *n* lines, where *n* is constrained to lie in the interval [0, 100].

%constant

Syntax

%name
%name (*index*)

Predefined constant.

Description

% provides convenient access to the following constants:

Name	Description	Equivalent to
%E	Euler's number, *e*	2.718281828…
%PI	Pi, π.	3.141592653…
%I	Imaginary unit, *i*	$1 \times i$
%FALSE	Logical false	0
%TRUE	Logical true	1

Corresponding to the predefined character strings, CoreBASIC provides predefined ASCII codes:

Name	Description	Equivalent to
%NUL	null	0
%BEL	bell	7
%BS	backspace	8
%TAB	horizontal tab	9
%LF	line feed	10
%VT	vertical tab	11
%FF	form feed	12
%CR	carriage return	13
%EOF	end of file	26
%ESC	escape	27

% also provides a way to encode colors:

Name	Description	Equivalent to
%BLACK	Black color	RGB(0, 0, 0) or 0x000000
%WHITE	White color	RGB(1, 1, 1) or 0xffffff
%RED	Red color	RGB(1, 0, 0) or 0xff0000
%GREEN	Green color	RGB(0, 1, 0) or 0x00ff00

%BLUE	Blue color	RGB(0, 0, 1) or 0x0000ff
%CYAN	Cyan color	RGB(0, 1, 1) or 0x00ffff
%MAGENTA	Magenta color	RGB(1, 0, 1) or 0xff00ff
%YELLOW	Yellow color	RGB(1, 1, 0) or 0xffff00

You can select the saturation of the color by using an argument, for example %RED(0.5).1 specifies that the color is fully saturated and 0 that it is fully desaturated to black. Hence, RED(0.5) is a dull red at 50% saturation:

```
> print hex %red(0.5)
800000
> _
```

%COLOR

Synopsis

%COLOR

Current drawing color.

Description

%COLOR returns the drawing color set by the last COLOR statement. If no graphics device is selected, an error is thrown.

See also

COLOR

%E

Synopsis

%E

Euler's number.

Description

%E is Euler's number, also knows as Napier's constant.

```
> print %e
2.71828
> _
```

See also

EXP

%FALSE

Synopsis

%FALSE

Boolean false.

Description

%FALSE is a synonym for zero as Boolean values are represented as integers in CoreBASIC.

```
> print %false | truth %false
0
False
> _
```

See also

%TRUE

%HEIGHT

Synopsis

`%HEIGHT`

Height of Telnet NVT in rows.

Description

`%HEIGHT` returns the height of the Network Virtual Terminal (NVT) in rows. CoreBASIC negotiates with the Telnet client using NAWS and indicates the number of negotiated rows in `%HEIGHT`. If the client does not support NAWS, the height is defaulted to 25 rows.

%I

Synopsis

`%I`

`%I(`*b*`)`

Imaginary unit.

Description

`%I` evaluates to the complex number *i*.

```
> print %i
0+1j
> _
```

This form is more natural when expressing complex numbers in your program:

```
> print 4 - 7 * %i
4-7j
> print (3 - 4 * %i) * (-2 + 4 * %i)
10+20j
> print %i * %i
-1
> _
```

Using `%I` in this way is marginally more expensive than using `CMPLX` to construct a complex number.

When `%I` is used as a function, `%I(N)`, the result is the number N×*i*:

```
> print %i(%pi)
0+3.14159j
> print %e ^ %i(%pi) + 1
0+0j
> _
```

See also

CMPLX

%IN

Synopsis

```
%IN
```

Standard input channel.

Description

`%IN` evaluates to an integer that is the current input channel.

%OUT

Synopsis

%OUT

Standard output channel.

Description

%OUT evaluates to an integer that is the current output channel.

To force data remaining in a TCP/IP buffer to be flushed to the network, rather than waiting for a full packet to be completed, you can use FLUSH %OUT .

See also

FLUSH

%PI

Synopsis

`%PI`

Approximation for π.

Description

`%PI` evaluates to an approximation to π.

```
> print %pi
3.14159
> _
```

See also

PI

%TRUE

Synopsis

%TRUE

Boolean true.

Description

%TRUE is a synonym for 1 as Boolean values are represented as integers in CoreBASIC.

```
> print %true | truth %true
1
True
> _
```

See also

%FALSE

%WIDTH

Synopsis

`%WIDTH`

Width of Telnet NVT in columns.

Description

`%WIDTH` returns the width of the Network Virtual Terminal (NVT) in columns. CoreBASIC negotiates with the Telnet client using NAWS and indicates the number of negotiated columns in `%WIDTH`. If the client does not support NAWS, the width is defaulted to 80 columns.

'

Synopsis

' *comment*

Comment.

Description

See **REM**.

+

Synopsis

x + *y*

Add.

Description

+ adds *y* to *x*. *x* and *y* must be compatible; if they are incompatible, CoreBASIC throws an exception. For instance, CoreBASIC cannot add numbers and strings directly:

```
> print 1 + "3"
?type mismatch
> _
```

Numbers

Real and complex numbers are added using standard mathematical rules:

```
> print 1 + 2
3
> print 2 + cmplx(3, 4)
5+4j
> print cmplx(3, 4) + cmplx(5, 12)
8+16j
> _
```

Quaternion addition adds corresponding components:

```
> print quat(1, 2, 3, 4) + quat(3, 4, 5, 6)
4+6i+8j+10k
> _
```

Strings

CoreBASIC defines addition of strings as concatenation.

```
> print "Core" + "BASIC"
CoreBASIC
> _
```

Arrays

Arrays are added element-by-element and a new array is created containing the sum of each pair:

```
> print [1, 2, 3] + [4, 5, 6]
[5, 7, 9]
```

```
> _
```

When adding an array and a scalar value, each value in the array is added to the scalar value and a new array is created containing the sum of each element with the scalar value:

```
> print [1, 2, 3] + 2
[3, 4, 5]
```

You can use + to add prefixes or suffixes to an array containing strings:

```
> mat print "|" + ["Core", "BASIC"] + "|"
|Core|
|BASIC|
> _
```

-

Synopsis

x - *y*

Subtract.

Description

- subtracts *y* from *x*. Both *x* and *y* must be compatible; if they are incompatible, CoreBASIC throws an exception. For instance, CoreBASIC cannot subtract strings form numbers directly:

```
> print 1 - "3"
?type mismatch
> _
```

Numbers

Real and complex numbers are subtracted using standard mathematical rules:

```
> print 2 - 1
1
> print 2 - cmplx(3, 4)
-1-2j
> print cmplx(3, 4) - cmplx(5, 12)
-2-8j
> _
```

Quaternion subtraction subtracts corresponding components:

```
> print quat(1, 2, 3, 4) - quat(6, 5, 4, 3)
-5-3i-1j+1k
> _
```

The - operator also works as a prefix to negate a value:

```
> print - 1
-1
> _
```

Arrays

Arrays are subtracted element-by-element and a new array is created containing the difference of each pair:

```
> print [1, 2, 3] - [2, 3, 1]
[-1, -1, 2]
> _
```

You can subtract a scalar value from an array or an array form a scalar:

```
> print [1, 2, 3] - 2
[-1, 0, 1]
> print 2 - [1, 2, 3]
[1, 0, -1]
> _
```

If *arg* is an array, negation threads recursively over the elements of the array:

```
> print -[1, 2, 3]
[-1, -2, -3]
> _
```

*

Synopsis

x * *y*

Multiply.

Description

* multiplies *x* by *y*. Both *x* and *y* must be compatible; if they are incompatible, CoreBASIC throws an exception. For instance, CoreBASIC cannot multiply strings by numbers directly:

```
> print "3" * 1
?type mismatch
> _
```

Numbers

Real and complex numbers are multiplied using standard mathematical rules:

```
> print 2 * 7
14
> print 2 * cmplx(3, 4)
6+8j
> _
```

Arrays

Arrays are multiplied element-by-element and a new array is created containing the product of each pair.

```
> print [1, 2, 3] * [4, 5, 6]
[4, 10, 18]
> _
```

You can multiply a scalar value by an array:

```
> print [1, 2, 3] * 2
[2, 4, 6]
> print 2 * [1, 2, 3]
[2, 4, 6]
> _
```

Matrix mode

In matrix mode, if *x* and *y* are both matrices, * computes the matrix product of *x* and *y*.

```
> a = [[1, 2], [3, 4]]            ' 2x2 matrix
> mat print a
```

```
1       2
3       4
> p = a*a
> mat print p                    ' each element is squared
1       4
9       16
> mat p = a*a                    ' multiplication in matrix mode
> mat print p                    ' uses standard mathematical matrix multiply
7      10
15     22
> _
```

See also

DOT, CROSS

/

Synopsis

x / *y*

Divide.

Description

/ divides *x* by *y*. Both *x* and *y* must be compatible; if they are incompatible, CoreBASIC throws an exception. For instance, CoreBASIC cannot divide strings by numbers directly:

```
> print "3" / 2
?type mismatch
> _
```

Numbers

Real and complex numbers are divided using standard mathematical rules:

```
> print 2 / 7
0.285714
> print 10 / cmplx(3, 4)
1.2-1.6j
> _
```

Arrays

Arrays are divided element-by-element and a new array is created containing the quotient of each pair.

```
> print [1, 2, 3] / [4, 5, 6]
[0.25, 0.4, 0.5]
> _
```

You can divide a scalar value by an array, or an array by a scalar:

```
> print [1, 2, 3] / 2
[0.5, 1, 1.5]
> print 2 / [1, 2, 3]
[2, 1, 0.666667]
> _
```

Matrix mode

In matrix mode, if *x* and *y* are both matrices, / computes the right matrix quotient of *x* and *y* using right matrix division, x * INV *y*.

```
> a = [[1, 2], [3, 4]]
```

```
> b = [[5, 6], [7, 8]]
> mat print a
1       2
3       4
> mat print b
5       6
7       8
> mat print a/b
3       -2
2       -1
> mat print a * inv b     ' a/b is a*INV(b)
3       -2
2       -1
> mat print (a/b) * b
1       2
3       4
> mat print a/a           ' a matrix divided by itself is the identity matrix
1       0
0       1
> _
```

See also

\

\

Synopsis

x \ *y*

Integer divide; left matrix divide.

Description

\ divides *x* by *y*. If *x* and *y* are integers, *x* \ *y* is the integer part of the quotient. If either *x* or *y* are not integers, \ is equivalent to */*.

Matrix mode

In matrix mode, if *x* and *y* are both matrices, \ computes the left matrix quotient of *x* and *y* using left matrix division, `INV` x * *y*.

```
> a = [[1, 2], [3, 4]]
> b = [[5, 6], [7, 8]]
> mat print a
1       2
3       4
> mat print b
5       6
7       8
> mat print a \ b
-3      -4
4       5
> mat print inv a * b      ' a\b is INV(a) * b
-3      -4
4       5
> mat print a \ a          ' a matrix divided by itself is the identity matrix
1       0
0       1
> _
```

^

Synopsis

x ^ *y*

Exponentiate.

Description

^ raises *x* to the power of *y*. Both *x* and *y* must be compatible; if they are incompatible, CoreBASIC throws an exception. For instance, CoreBASIC cannot exponentiate strings by numbers:

```
> print "3" ^ 2
?type mismatch
> _
```

Numbers

Real and complex numbers are exponentiated using standard mathematical rules:

```
> print 10 ^ -3
0.001
> print -10 ^ -3
-0.001
> _
```

Arrays

Arrays are exponentiated element-by-element and a new array is created containing the power of each pair.

```
> print [1, 2, 3] ^ [4, 5, 6]
[1, 32, 729]
> _
```

You can exponentiate a scalar value by an array, or an array by a scalar:

```
> print [1, 2, 3] ^ 2
[1, 4, 9]
> print 2 ^ [1, 2, 3]
[1, 4, 8]
> _
```

Matrix mode

Exponentiation in matrix mode is not supported.

See also

LOG, EXP

&

Synopsis

x & *y*

Join.

Description

& creates a new array by joining the arrays *x* and *y* together:

```
> print [1, 2, 3] & ["Four", "Five", "Six"]
[1, 2, 3, "Four", "Five", "Six"]
> _
```

You can use this to add an extra row to a matrix:

```
> list
   10 M = INT(100 * RND CON(4, 3))        ' create a random 4x3 matrix
   20 MAT PRINT "Original matrix:"; M
   30 M = M & [ZER(LEN M(0))]             ' add a zero row
   40 MAT PRINT "New matrix:"; M
   50 END
> run
Original matrix:
80      64      80
71      19      92
14      31      98
17      12      45
New matrix:
80      64      80
71      19      92
14      31      98
17      12      45
0       0       0
> _
```

You can use & to augment a matrix with an additional column by using TRN twice:

```
> list
   10 M = INT(100 * RND CON(4, 3))        ' create a random 4x3 matrix
   20 MAT PRINT "Original matrix: "; M
   30 M = TRN(TRN M & [ZER(LEN M)])     ' add a zero column
   40 MAT PRINT "New matrix:"; M
   50 END
> run
Original matrix:
64      0       42
6       37      53
87      75      52
76      8       6
New matrix:
64      0       42      0
6       37      53      0
```

```
87      75      52      0
76      8       6       0
```

See also

TRN, ZER

:

Synopsis

:

Multi-statement separator.

Description

: enables you to place more than one statement on a line.

```
> print "Hello" : print "Again"
Hello
Again
> _
```

<

Synopsis

x < *y*

Less than.

Description

< evaluates to true if *x* is less than *y*. *x* and *y* must be compatible; if they are incompatible, CoreBASIC throws an exception. For instance, CoreBASIC cannot compare numbers and strings directly:

```
> print 1 < "3"
?type mismatch
> _
```

Real numbers

Real numbers are graded as you would expect:

```
> print 1 < 2 | 2 < 2 | 3 < 2
1
0
0
> _
```

Complex numbers

Complex numbers are graded according to their complex modulus (magnitude): the number with the greatest magnitude is graded highest. If both have the same magnitude, the one with the greatest phase angle is graded highest.

```
> a = cmplx(3, 4)
> b = cmplx(5, 12)
> print abs a, abs b
5       13
> print a < b
1
> b = cmplx(3, -4)
> print abs a, abs b
5       5
> print arg a, arg b
0.927295        -0.927295
> print a < b
0
> _
```

Quaternions

An ordering for quaternions isn't defined by CoreBASIC: comparing them will result in a type mismatch error.

Strings

Strings are graded on a character-by-character basis using each character's ASCII code, and case is significant.

```
> print "aardvark" < "abacus"
1
> print "aardvark" < "Abacus"
0
> _
```

Arrays

Arrays are graded element-by-element and a new array is created containing the elementwise comparison of each pair.

```
> print [1, 2, 3] < [2, 1, 0]
[1, 0, 0]
> _
```

When comparing an array with a scalar value, each value in the array is compared to the scalar value and a new array is created containing the elementwise comparison of each element with the scalar value:

```
> print [1, 2, 3] < 2
[1, 0, 0]
> print 2 < [1, 2, 3]
[0, 0, 1]
> print "aardvark" < ["abacus", "Abacus"]
[1, 0]
> _
```

<=

Synopsis

x <= *y*

Less than or equal to.

Description

<= evaluates to true if *x* is less than or equal to *y*. *x* and *y* must be compatible; if they are incompatible, CoreBASIC throws an exception. For instance, CoreBASIC cannot compare numbers and strings directly:

```
> print 1 <= "3"
?type mismatch
> _
```

Real numbers

Real numbers are graded as you would expect:

```
> print 1 <= 2 | 2 <= 2 | 3 <= 2
1
1
0
> _
```

Complex numbers

Complex numbers are graded according to their complex modulus (magnitude): the number with the greatest magnitude is graded highest. If both have the same magnitude, the one with the greatest phase angle is graded highest.

```
> a = cmplx(3, 4)
> b = cmplx(5, 12)
> print abs a, abs b
5       13
> print a <= b
1
> b = cmplx(3, -4)
> print abs a, abs b
5       5
> print arg a, arg b
0.927295        -0.927295
> print a <= b
0
> _
```

Quaternions

An ordering for quaternions isn't defined by CoreBASIC: comparing them will result in a type mismatch error.

Strings

Strings are graded on a character-by-character basis using each character's ASCII code, and case is significant.

```
> print "aardvark" <= "abacus"
1
> print "aardvark" <= "Abacus"
0
> _
```

Arrays

Arrays are graded element-by-element and a new array is created containing the elementwise comparison of each pair.

```
> print [1, 2, 3] <= [2, 2, 2]
[1, 1, 0]
> _
```

When comparing an array with a scalar value, each value in the array is compared to the scalar value and a new array is created containing the elementwise comparison of each element with the scalar value:

```
> print [1, 2, 3] <= 2
[1, 1, 0]
> print 2 <= [1, 2, 3]
[0, 1, 1]
> print "aardvark" <= ["abacus", "Abacus"]
[1, 0]
> _
```

<>

Synopsis

x <> *y*

Inequality.

Description

<> evaluates to true if *x* is not identical to *y*. *x* and *y* must be compatible; if they are incompatible, CoreBASIC throws an exception. For instance, CoreBASIC cannot compare numbers and strings directly:

```
> print 1 <> "3"
?type mismatch
> _
```

Real numbers

Real numbers are compares as you would expect:

```
> print 1 <> 1 | 1 <> 2
0
1
> _
```

Complex numbers

Complex numbers are identical if and only if their corresponding real and imaginary parts are identical:

```
> print cmplx(1, 2) <> cmplx(1, 2)
0
> print cmplx(1, 2) <> cmplx(1, 999)
1
> print cmplx(1, 2) <> cmplx(999, 2)
1
> _
```

Strings

Strings are identical if they are the of different lengths or contain different corresponding characters.

```
> print "aardvark" <> "aardvark"
0
> print "aardvark" <> "abacus"
1
> print "aardvark" <> "aardvark "
1
> _
```

Arrays

Arrays are graded element-by-element and a new array is created containing the elementwise comparison of each pair.

```
> print [1, 2, 3] <> [2, 2, 2]
[1, 0, 1]
> _
```

When comparing an array with a scalar value, each value in the array is compared to the scalar value and a new array is created containing the elementwise comparison of each element with the scalar value:

```
> print [1, 2, 3] <> 2
[1, 0, 1]
> print 2 <> [1, 2, 3]
[1, 0, 1]
> print "aardvark" <> ["abacus", "Abacus", "aardvark"]
[1, 1, 0]
> _
```

=

Synopsis

x = *y*

Equality.

Description

= evaluates to true if *x* is identical to *y*. *x* and *y* must be compatible; if they are incompatible, CoreBASIC throws an exception. For instance, CoreBASIC cannot compare numbers and strings directly:

```
> print 1 = "3"
?type mismatch
> _
```

Real numbers

Real numbers are compares as you would expect:

```
> print 1 = 1 | 1 = 2
1
0
> _
```

Complex numbers

Complex numbers are identical if and only if their corresponding real and imaginary parts are identical:

```
> print cmplx(1, 2) = cmplx(1, 2)
1
> print cmplx(1, 2) = cmplx(1, 999)
0
> print cmplx(1, 2) = cmplx(999, 2)
0
> _
```

Strings

Strings are identical if they are the same length and contain the same characters.

```
> print "aardvark" = "aardvark"
1
> print "aardvark" = "abacus"
0
> print "aardvark" = "aardvark "
0
> _
```

Arrays

Arrays are graded element-by-element and a new array is created containing the elementwise comparison of each pair.

```
> print [1, 2, 3] = [2, 2, 2]
[0, 1, 0]
> _
```

When comparing an array with a scalar value, each value in the array is compared to the scalar value and a new array is created containing the elementwise comparison of each element with the scalar value:

```
> print [1, 2, 3] = 2
[0, 1, 0]
> print 2 = [1, 2, 3]
[0, 1, 0]
> print "aardvark" = ["abacus", "Abacus", "aardvark"]
[0, 0, 1]
> _
```

>

Synopsis

x > *y*

Greater than.

Description

> evaluates to true if *x* is greater than *y*. *x* and *y* must be compatible; if they are incompatible, CoreBASIC throws an exception. For instance, CoreBASIC cannot compare numbers and strings directly:

```
> print 1 > "3"
?type mismatch
> _
```

Real numbers

Real numbers are graded as you would expect:

```
> print 1 > 2 | 2 > 2 | 3 > 2
0
0
1
> _
```

Complex numbers

Complex numbers are graded according to their complex modulus (magnitude): the number with the greatest magnitude is graded highest. If both have the same magnitude, the one with the greatest phase angle is graded highest.

```
> a = cmplx(3, 4)
> b = cmplx(5, 12)
> print abs a, abs b
5       13
> print a > b
0
> b = cmplx(3, -4)
> print abs a, abs b
5       5
> print arg a, arg b
0.927295        -0.927295
> print a > b
1
> _
```

Quaternions

An ordering for quaternions isn't defined by CoreBASIC: comparing them will result in a type mismatch error.

Strings

Strings are graded on a character-by-character basis using each character's ASCII code, and case is significant.

```
> print "aardvark" > "abacus"
0
> print "aardvark" > "Abacus"
1
> _
```

Arrays

Arrays are graded element-by-element and a new array is created containing the elementwise comparison of each pair.

```
> print [1, 2, 3] > [2, 2, 2]
[0, 0, 1]
> _
```

When comparing an array with a scalar value, each value in the array is compared to the scalar value and a new array is created containing the elementwise comparison of each element with the scalar value:

```
> print [1, 2, 3] > 2
[0, 0, 1]
> print 2 > [1, 2, 3]
[1, 0, 0]
> print "aardvark" > ["abacus", "Abacus"]
[0, 1]
> _
```

>=

Synopsis

x >= *y*

Greater than or equal to.

Description

> evaluates to true if *x* is greater than or equal to *y*. *x* and *y* must be compatible; if they are incompatible, CoreBASIC throws an exception. For instance, CoreBASIC cannot compare numbers and strings directly:

```
> print 1 >= "3"
?type mismatch
> _
```

Real numbers

Real numbers are graded as you would expect:

```
> print 1 >= 2 | 2 >= 2 | 3 >= 2
0
1
1
> _
```

Complex numbers

Complex numbers are graded according to their complex modulus (magnitude): the number with the greatest magnitude is graded highest. If both have the same magnitude, the one with the greatest phase angle is graded highest.

```
> a = cmplx(3, 4)
> b = cmplx(5, 12)
> print abs a, abs b
5       13
> print a >= b
0
> b = cmplx(3, -4)
> print abs a, abs b
5       5
> print arg a, arg b
0.927295        -0.927295
> print a >= b
1
> _
```

Quaternions

An ordering for quaternions isn't defined by CoreBASIC: comparing them will result in a type mismatch error.

Strings

Strings are graded on a character-by-character basis using each character's ASCII code, and case is significant.

```
> print "aardvark" >= "abacus"
0
> print "aardvark" >= "Abacus"
1
> _
```

Arrays

Arrays are graded element-by-element and a new array is created containing the elementwise comparison of each pair.

```
> print [1, 2, 3] >= [2, 2, 2]
[0, 1, 1]
> _
```

When comparing an array with a scalar value, each value in the array is compared to the scalar value and a new array is created containing the elementwise comparison of each element with the scalar value:

```
> print [1, 2, 3] >= 2
[0, 1, 1]
> print 2 >= [1, 2, 3]
[1, 1, 0]
> print "aardvark" >= ["abacus", "Abacus"]
[0, 1]
> _
```

[...]

Synopsis

[*expression*, *expression*...]

Construct array.

Description

[...] creates an array initialized with the comma-separated values.

```
> print [1, 1/2, 3]
[1, 0.5, 3]
> _
```

Each value can be any expression, and the array can hold values of mixed type:

```
> x = "Aardvark"
> print [1, ucase x, 37]
[1, "AARDVARK", 37]
> _
```

|

Description

The | symbol is used as a separator in a PRINT statement to indicate a new line.

See **PRINT**.

ABS

Synopsis

ABS *arg*

Compute absolute value.

Description

ABS computes the magnitude (absolute value) of *arg*.

Real numbers

The magnitude of a positive real number *arg* is simply *arg*, and the magnitude of a negative number *arg* is –*arg*:

```
> print abs 3, abs -3
3       3
> _
```

Complex numbers

The absolute value of a complex number *arg* is the distance from the origin to *arg*:

```
> print abs cmplx(3, 4)
5
> _
```

Quaternions

The absolute value of a quaternion *arg* is the distance from the origin to *arg*:

```
> print abs quat(2, 3, 4, 5)
7.34847
> _
```

Arrays

If *arg* is an array, ABS threads recursively over the elements of the array:

```
> print abs [-0.5, 1, -1.5]
[0.5, 1, 1.5]
> _
```

See also

SGN

ACS

Synopsis

ACS *arg*

Compute inverse circular cosine.

Description

ACS computes the inverse cosine of *arg* in radian measure. For real, complex, and quaternion types, ACS returns a number of the same type as its operand if it can:

```
> print acs 1
0
> print acs cmplx(1, 2)
1.14372-1.52857j
> _
```

If the argument is real but not in the interval [−1, +1], CoreBASIC cannot return a real result—it must return a complex number:

```
> print acs 2
0+1.31696j
> _
```

If *arg* is an array, ACS threads recursively over the elements of the array:

```
> print acs [0.5, 1, 1.5]
[1.0472, 0, 0+0.962424j]
> _
```

See also

COS, **ASN**, **ATN**

ACSH

Synopsis

ACSH *arg*

Compute inverse hyperbolic cosine.

Description

ACSH computes the inverse hyperbolic cosine of *arg*. For real, complex, and quaternion types, ACSH returns a number of the same type as its operand if it can:

```
> print acsh 1
0
> print acsh cmplx(1, 2)
1.46406+0.738411j
> print acsh quat(1, 2, 3, 4)
2.40145+0.516276i+0.774414j+1.03255k
> _
```

If the argument is real but does not lie in the interval [−1, +1], CoreBASIC cannot return a real result—it must return a complex number:

```
> print acsh 2
0+1.0472j
> _
```

If *arg* is an array, ACSH threads recursively over the elements of the array:

```
> print acsh [0.5, 1, 1.5]
[0+1.0472j, 0, 0.962424]
> _
```

See also

SINH, ACSH, ATNH

AND

Synopsis

x AND *y*

Bitwise conjunction.

Description

AND computes the bitwise conjunction of *x* and *y*. A bit is set in the result only if the corresponding bit is set in both *x* and *y*:

```
> print hex(0x1234 and 0xfedc)
1214
> _
```

If *x* and *y* are logical expressions, AND will compute the logical conjunction of *x* and *y* according to the standard truth table:

```
> list
   10 FOR X = 0 TO 1
   20   FOR Y = 0 TO 1
   30     PRINT X; " AND "; Y; " = "; X AND Y
   40   NEXT Y
   50 NEXT X
   60 END
> run
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1
> _
```

Both *x* and *y* are converted to integers before applying AND:

```
> print 1.5 and 2.2 | 1.5 and 0.2
1
0
> _
```

Arrays are processed element-by-element and a new array is created containing the elementwise conjunction of each pair:

```
> print [0, 1, 0, 1] and [0, 0, 1, 1]
[0, 0, 0, 1]
> _
```

See also

AND THEN

AND THEN

Synopsis

x AND THEN *y*

Short-circuit logical conjunction.

Description

AND THEN computes the short-circuit logical conjunction of *x* and *y*. The result AND THEN is *x* if *x* is zero, otherwise it is *y*. If *x* is zero, *y* is not evaluated:

```
> list
   10 FOR X = FALSE TO TRUE
   20   FOR Y = FALSE TO TRUE
   30     PRINT TRUTH X; " AND THEN "; TRUTH Y; " = "; TRUTH(X AND THEN Y)
   40   NEXT Y
   50 NEXT X
   60 END
> run
False AND THEN False = False
False AND THEN True = False
True AND THEN False = False
True AND THEN True = True
> _
```

Note that *x* AND THEN *y* is equivalent to `IFF(`*x*`,`*y*`,`*x*`)` with *x* evaluated only once:

```
> print 0 and then "Surprising"
0
> print 1 and then "Surprising"
Surprising
> _
```

Note

For bitwise operations, use AND.

See also

AND, IFF, OR ELSE

ARG

Synopsis

ARG *arg*

Compute argument of complex number.

Description

ARG computes the argument, or phase angle, of the complex number *arg*. The argument of a complex number is counterclockwise angle, in radians, from the positive real axis.

The argument of a real number is zero because all real numbers lie on the real axis of the complex plane:

```
> print arg 3
0
> _
```

The argument of a complex number `x` is defined as `ATN2(IM x, RE x)`:

```
> print arg cmplx(0, 1)
1.5708
> _
```

ARG is not defined for quaternions:

```
> print arg quat(1, 2, 3, 4)
?type mismatch
> _
```

If *arg* is an array, ARG threads recursively over the elements of the array:

```
> print arg [cmplx(1, 1), cmplx(-1, -1)]
[0.785398, -2.35619]
> print deg arg [cmplx(1, 1), cmplx(-1, -1)]
[45, -135]
> _
```

AS

Synopsis

`INSTALL` *driver* [AS *var*]

Install driver.

See **INSTALL**.

Synopsis

`INPUT` ["*string*`;`] *var* [AS *type*]…

Read from console.

See **INPUT**.

Synopsis

`PIN` *x* AS *mode*

Configure pin.

See **PIN**.

ASC

Synopsis

ASC *arg*

Extract ASCII code.

Description

ASC returns the ASCII code of the first character of the string *arg*:

```
> print asc "Aardvark"
65
> _
```

The string *arg* must not be empty:

```
> print asc ""
?argument error
> _
```

If *arg* is an array, ASC threads recursively over the elements of the array:

```
> lyric = ["Delerious", "Incredible", "Superficial", "Complicated", "Oh oh oh"]
> print asc lyric
[68, 73, 83, 67, 79]
> print merge asc lyric
DISCO
> _
```

Notes

To convert an entire string to its component ASCII codes, use EXPAND.

See also

EXPAND

ASN

Synopsis

ASN *arg*

Compute inverse circular sine.

Description

ASN computes the inverse sine of *arg* in radian measure. For real, complex, and quaternion types, ASN returns a number of the same type as its operand if it can:

```
> print asn 1
1.5708
> print asn cmplx(1, 2)
0.427079+1.52857j
> _
```

If the argument is real but does not lie in the interval [−1, +1], CoreBASIC cannot return a real result—it must return a complex number:

```
> print asn 2
1.5708-1.31696j
> _
```

If *arg* is an array, ASN threads recursively over the elements of the array:

```
> print asn [0.5, 1, 1.5]
[0.523599, 1.5708, 1.5708-0.962424j]
> _
```

See also

SIN, ACS, ATN

ASNH

Synopsis

ASNH *arg*

Compute inverse hyperbolic sine.

Description

ASNH computes the inverse hyperbolic sine of *arg*. For real, complex, and quaternion types, ASN returns a number of the same type as its operand:

```
> print asnh 1
0.881374
> print asnh cmplx(1, 2)
1.46935+1.06344j
> _
```

If *arg* is an array, ASNH threads recursively over the elements of the array:

```
> print asnh [0.5, 1, 1.5]
[0.481212, 0.881374, 1.19476]
> _
```

See also

SINH, ACSH, ATNH

ATN

Synopsis

ATN *arg*

Compute inverse circular tangent.

Description

ATN computes the inverse tangent of *arg* in radian measure. For real, complex, and quaternion types, ATN returns a number of the same type as its operand:

```
> print atn 1
0.785398
> print atn cmplx(1, 2)
1.33897+0.402359j
> _
```

If *arg* is an array, ATN threads recursively over the elements of the array:

```
> print atn [0.5, 1, 1.5]
[0.463648, 0.785398, 0.982794]
> _
```

See also

TAN, **ACS**, **ASC**

ATN2

Synopsis

ATN2(*y*, *x*)

Compute inverse circular tangent.

Description

ATN2 computes the inverse tangent of *y* divided by *x* using the signs of *x* and **y** to compute the quadrant of the return value. The principal value lies in the interval [–½π, +½π] radians.

```
> print atn2(1, 1) | atn2(-1, -1)
0.785398
-2.35619
> _
```

Note that the signs of *x* and *y* do matter—comparing ATN2 with ATN:

```
> print deg atn2(1, 1) | deg atn(1 / 1)
45
45
> print deg atn2(-1, -1) | deg atn(-1 / -1)
-135
45
> _
```

See also

ATN, **TAN**

ATNH

Synopsis

ATNH *arg*

Compute inverse hyperbolic tangent.

Description

ATNH computes the inverse hyperbolic tangent of *arg*. For real, complex, and quaternion types, ATNH returns a number of the same type as its operand if it can:

```
> print atnh 0.5
0.549306
> print atnh cmplx(1, 2)
0.173287+1.1781j
> print atnh quat(1, 2, 3, 4)
0.0323029+0.517345i+0.776018j+1.03469k
> _
```

If the argument is real but does not lie in the interval [−1, +1], CoreBASIC cannot return a real result—it must return a complex number:

```
> print atnh 2
0.549306+1.5708j
> _
```

If *arg* is an array, ATNH threads recursively over the elements of the array:

```
> print atnh [0.25, 0.5, 0.75]
[0.255413, 0.549306, 0.972955]
> _
```

See also

TANH

AUTO

Synopsis

AUTO
AUTO *start* [, *increment*]

Automatically number lines.

Description

AUTO instructs CoreBASIC to start generating line numbers for program entry. AUTO begins numbering at *start* and increments each subsequent line by *increment*. If *start* and *increment* are omitted, CoreBASIC starts numbering at 10 in increments of 10:

```
> new
> auto
> 10 print "'Twas brillig, and the slithy toves"
> 20 print "  Did gyre and gimble in the wabe:"
> 30 print "All mimsy were the borogoves,"
> 40 print "  And the mome raths outgrabe."
> 50 end
> 60
> _
```

To exit AUTO mode, press Enter when the next line number appears. In this case, any existing line with that number is not deleted and AUTO mode is canceled.

```
> list
   10 PRINT "'Twas brillig, and the slithy toves"
   20 PRINT "  Did gyre and gimble in the wabe:"
   30 PRINT "All mimsy were the borogoves,"
   40 PRINT "  And the mome raths outgrabe."
   50 END
> auto 8, 1
> 8 ' A poem by Lewis Carroll.
> 9 '
> 10
> list
    8 ' A poem by Lewis Carroll.
    9 '
   10 PRINT "'Twas brillig, and the slithy toves"
   20 PRINT "  Did gyre and gimble in the wabe:"
   30 PRINT "All mimsy were the borogoves,"
   40 PRINT "  And the mome raths outgrabe."
   50 END
> _
```

You can also exit AUTO mode by deleting the line number using Backspace or Ctrl+U, and typing another command, such as LIST or RUN.

See also

RENUMBER

BASE64$

Synopsis

BASE64$ *arg*

Encode string using Base64.

Description

BASE64$ returns a string containing the Base64-encoded representation of *arg* according to RFC4648 section 4 (and RFC 1421 section 4.3.2.4). Base64 encoding replaces the source octets with characters that will pass through a network connection without further modification.

```
> list
   10 ' Program to replicate Base64 examples from Wikipedia.
   20 '
   30 ' See http://en.wikipedia.org/wiki/Base64
   40 '
   50 S = "any carnal pleasure."
   60 FOR I = 1 TO 5
   70   PRINT "Input ends with: "; JUSTIFY$(S, -20); "  ";
   80   PRINT "Output ends with: "; BASE64$(S)
   90   S = LEFT(S, -1)
  100 NEXT I
  110 END
> run
Input ends with: any carnal pleasure.  Output ends with: YW55IGNhcm5hbCBwbGVhc3VyZS4=
Input ends with: any carnal pleasure   Output ends with: YW55IGNhcm5hbCBwbGVhc3VyZQ==
Input ends with: any carnal pleasur    Output ends with: YW55IGNhcm5hbCBwbGVhc3Vy
Input ends with: any carnal pleasu     Output ends with: YW55IGNhcm5hbCBwbGVhc3U=
Input ends with: any carnal pleas      Output ends with: YW55IGNhcm5hbCBwbGVhcw==
> _
```

See also

RFC4648 - The Base16, Base32, and Base64 Data Encodings

BGET

Synopsis

BGET *arg*

Read byte from stream.

Description

BGET reads the the next byte on the stream *arg*. The value returned is the ASCII code of the character read, or if there was an error reading the stream, a negative error code.

See also

BGET$

BGET$

Synopsis

BGET$ *arg*

Read byte from stream.

Description

BGET$ reads the next byte on the stream *arg*. The value returned is a string containing the single byte read. If there is an error reading the stream, CoreBASIC throws an appropriate error code.

See also

GET

BLUE%

Synopsis

BLUE% *arg*

Extract blue component of a color.

Description

BLUE% extracts the blue component of the 24-bit RGB color value *arg* and returns it as a number between 0 (fully desaturated) and 1 (fully saturated).

```
> x = rgb(0.7, 0.2, 0.1)
> print blue% x
0.1
> _
```

See also

GREEN%, **RED%**, **RGB**

BYE

Synopsis

BYE

Exit CoreBASIC.

Description

BYE exits the CoreBASIC interpreter by closing the established network connection. For the SolderCore, the current program remains in memory and, if connection is re-established, you can resume your CoreBASIC session. On the SolderCore Emulator, BYE will exit the CoreBASIC interpreter and any program in memory is lost.

```
> bye

Connection to host lost.

C:\Users\Paul> _
```

Note

Pressing `Ctrl`+`D` at the CoreBASIC command line will execute BYE.

CALL

Synopsis

CALL *name*(*arg*, *arg*, …)

Call a procedure.

Description

CALL calls the procedure *name* with the arguments in the list. The number of arguments in the list must match the number of arguments in the procedure's parameter list.

```
> list
   10 CALL CENTER_STRING("*************************")
   20 CALL CENTER_STRING("Welcome to CoreBASIC")
   30 CALL CENTER_STRING("*************************")
   40 END
  100 DEFPROC CENTER_STRING(TXT)
  110   PRINT "|"; SPC(20 - LEN(TXT)/2); TXT; SPC(20 - LEN(TXT)/2); "|"
  120 ENDPROC
> run
|       *************************       |
|          Welcome to CoreBASIC          |
|       *************************       |
> _
```

CASE ... ENDCASE

Synopsis

CASE *value*
 WHEN *expr*, *expr*, *expr*…
  statements
 WHEN *relation expr*…
  statements
 ?
 [OTHERWISE
  statements]
ENDCASE

Multi-way branch.

Description

CASE selects a sequence of statements to execute depending upon the value of *value*. If *value* matches any of the expressions after a WHEN, control is passed to the statements following matching WHEN, and run up to the following WHEN, OTHERWISE, or ENDCASE. Control then passes to the following ENDCASE.

If no match is found in the WHEN lists, control passes to the statements following OTHERWISE, if present, or ENDCASE if there is no OTHERWISE.

CASE statements are usually found testing the values of numbers:

```
> list
  10 FOR I = 1 TO 10
  20   PRINT I; " - ";
  30   CASE I
  40   WHEN 1, 2, 3, 5, 7
  50     PRINT "Prime"
  60   WHEN 2, 4, 6, 8, 10
  70     PRINT "Even"
  80   OTHERWISE
  90     PRINT "Odd"
 100   ENDCASE
 110 NEXT I
 120 END
> run
1 - Prime
2 - Prime
3 - Prime
4 - Even
5 - Prime
6 - Even
7 - Prime
8 - Even
9 - Odd
10 - Even
> _
```

Note that WHEN statements are evaluated in top-to-bottom fashion. In the example above, the value 2 appears in two WHEN statements. CoreBASIC matches 2 in the first WHEN, executes the statements following the matching WHEN and prints Prime. Control then transfers to ENDCASE, bypassing subsequent WHEN and OTHERWISE statements.

CASE statements can test strings too:

```
> list
  10 TEXT = "Hello, world"
  20 FOR EACH C IN CHR EXPAND TEXT
  30   CASE C
  40   WHEN "a", "A" : PRINT "Vowel - A"
  50   WHEN "e", "E" : PRINT "Vowel - E"
  60   WHEN "i", "I" : PRINT "Vowel - I"
  70   WHEN "o", "O" : PRINT "Vowel - O"
  80   WHEN "u", "U" : PRINT "Vowel - U"
  90   OTHERWISE PRINT "Consonant - "; UCASE C
 100   ENDCASE
 110 NEXT C
 120 END
> run
Consonant - H
Vowel - E
Consonant - L
Consonant - L
Vowel - O
Consonant - ,
Consonant -
Consonant - W
Vowel - O
Consonant - R
Consonant - L
Consonant - D
> _
```

A WHEN can use one of the relational operators <, <=, >, >=, or <>:

```
> list
   10 FOR FLOW = -5 TO 25 STEP 5
   20   PRINT "Flow is "; FLOW; " L/s: ";
   30   CASE FLOW
   40   WHEN < 0
   50     PRINT "Alert!  Flowing in wrong direction!"
   60   WHEN 0
   70     PRINT "Flow stopped"
   80   WHEN <= 5
   90     PRINT "Warning: flow is slow; blockage?"
  100   WHEN > 20
  110     PRINT "Alert!  Flow is dangerously high!"
  120   OTHERWISE
  130     PRINT "Flow is within operating limits"
  140   ENDCASE
  150 NEXT FLOW
  160 END
> run
Flow is -5 L/s: Alert!  Flowing in wrong direction!
Flow is 0 L/s: Flow stopped
Flow is 5 L/s: Warning: flow is slow; blockage?
Flow is 10 L/s: Flow is within operating limits
```

```
Flow is 15 L/s: Flow is within operating limits
Flow is 20 L/s: Flow is within operating limits
Flow is 25 L/s: Alert!  Flow is dangerously high!
> _
```

CASE statements can be nested. Each nested CASE statement must have a matching ENDCASE must be completely contained within a single WHEN or OTHERWISE block.

Notes

You can write END CASE as two separate words and CoreBASIC will change the two words to ENDCASE automatically.

CATALOG

Synopsis

CATALOG

CATALOG folder

Display folder content.

Description

CATALOG displays the contents of the current or specified folder.

Example

```
> catalog

Directory of: /c/*.*

12/05/12   11:26         <DIR>   sys
01/01/12   00:00           144   work.bas
01/01/12   00:00         1,913   crc.bas
01/01/12   00:00         2,499   union.bas
01/01/12   00:00           194   analog.bas
01/01/12   00:00        13,291   trek.bas
01/01/12   00:00         1,986   calib.bas
01/01/12   00:00           227   blinky.bas
01/01/12   00:00           157   !run.bas
01/01/12   00:00         2,580   flag.bas
01/01/12   00:00           538   air.bas
01/01/12   00:00            88   ansi.bas
01/01/12   00:00           102   test.bas
01/01/12   00:00           910   cam.bas
01/01/12   00:00         <DIR>   www

> chdir "sys"
> catalog

Directory of: /c/sys/*.*

12/05/12   11:26         <DIR>   .
12/05/12   11:26         <DIR>   ..
01/01/12   00:00           896   history.log
01/01/12   00:00       491,520   core.fw
01/01/12   00:00           113   !network.bas
01/01/12   00:00            84   !boot.bas

> _
```

Note

Pressing F6 at the CoreBASIC command line will execute CATALOG.

CD

Synopsis

CD *folder*

Change working folder.

Description

CD changes the working folder to *folder*. The difference between CD and CHDIR is that CD interprets the text *folder* as a folder name without requiring quotation marks—this is excellent when traversing the file system interactively using CD to move and DIR to list contents.

```
> cd /c
> dir

Directory of: /c/*.*

12/05/12   11:26         <DIR>   sys
01/01/12   00:00           144   work.bas
01/01/12   00:00         1,913   crc.bas
01/01/12   00:00         2,499   union.bas
01/01/12   00:00           194   analog.bas
01/01/12   00:00        13,291   trek.bas
01/01/12   00:00         1,986   calib.bas
01/01/12   00:00           227   blinky.bas
01/01/12   00:00           157   !run.bas
01/01/12   00:00         2,580   flag.bas
01/01/12   00:00           538   air.bas
01/01/12   00:00            88   ansi.bas
01/01/12   00:00           102   test.bas
01/01/12   00:00           910   cam.bas
01/01/12   00:00         <DIR>   www

> cd sys
> dir *.fw

Directory of: /c/sys/*.fw

01/01/12   00:00       491,520   core.fw

> cd ..
> print $cwd
/c
> _
```

See also

CHDIR, $CWD, CATALOG

CEIL

Synopsis

`CEIL` *arg*

Compute ceiling.

Description

`CEIL` computes computes the smallest integer value not less than *arg*:

```
> print ceil -3.4 | 3.4
-3
4
> print ceil -6 | 6
-6
6
```

CHAIN

Synopsis

CHAIN *name*

Load and run program from storage device.

Description

CHAIN loads into memory the program contained in the file *name* and immediately starts it running. The current program name is set to *name* just as NAME would have set the current program name:

```
> chain "/c/welcome.bas"
Welcome to CoreBASIC on the SolderCore!
For more information, visit http://www.soldercore.com/
> list
   10 ' Welcome program for CoreBASIC.
   20 '
   30 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
   40 PRINT "For more information, visit http://www.soldercore.com/"
   50 '
   60 END
> _
```

You can use CHAIN in a program to swap between applications.

See also

LOAD, RUN

CHDIR

Synopsis

`CHDIR` *path*

Change working folder.

Description

`CHDIR` changes the working folder to *path*.

Example

```
> print $cwd
/c
> chdir "www"
> print $cwd
/c/www
> chdir "/c/sys/backups"
> print $cwd
/c/sys/backups
> chdir ".."
> print $cwd
/c/sys
> _
```

See also

CD, **$CWD**

CHECK

Synopsis

CHECK

Check program syntax.

Description

You can use CHECK to see whether your program contains any syntax errors before running it. In fact, RUN performs a CHECK before starting to execute your program.

Note

Pressing F7 at the CoreBASIC command line will execute CHECK.

CHR

Synopsis

CHR *arg*

Convert ASCII code to character.

Description

CHR returns a string containing the single character with ASCII code *arg*:

```
> print chr 65
A
> _
```

The ASCII code *arg* is silently converted modulo 256:

```
> print asc chr -1
255
> _
```

If *arg* is a nonempty string, CHR returns the first character of that string:

```
> print chr "ABC"
A
> _
```

If *arg* is an empty string, CHR stops with a dimension error:

```
> print chr ""
?dimension error
> _
```

If *arg* is an array, CHR threads recursively over the elements of the array:

```
> print chr [67, 111, 114, 101, 66, 65, 83, 73, 67]
["C", "o", "r", "e", "B", "A", "S", "I", "C"]
> _
```

See also

ASC

CINT

Synopsis

`CINT` *arg*

Convert to integer using banker's rounding.

Description

`CINT` converts *arg* to an integer using banker's rounding. Banker's rounding uses a special rule to round when the fractional part of *arg* is exactly one half.

For the most part, `CINT` converts numbers just as `FIX` does, by discarding the fractional part of *arg*:

```
> print cint 48.4  | fix 48.4
48
48
> print cint 48.9 | fix 48.9
48
48
> print cint -48.4 | fix -48.4
-48
-48
> _
```

However, when *arg* has a fractional part that is exactly one half (0.5), `CINT` rounds to the *nearest even*:

```
> print cint 48.5 | fix 48.5
48
48
> print cint 49.5 | fix 49.5
50
49
> print cint -48.5 | fix -48.5
-48
-48
> print cint -49.5 | fix -49.5
-50
-49
```

If *arg* is a string, it is converted to a number before applying `CINT` to the result:

```
> print cint "1.5"
2
> _
```

If *arg* is not recognized as a valid number, `CINT` returns a NaN:

```
> print cint "garbage"
nan
> _
```

If *arg* is an array, `CINT` threads recursively over the elements of the array:

```
> print cint [47.5, 48.5, 49.5, 50.5]
[48, 48, 50, 50]
> _
```

See also

FIX, INT

CIRCLE

Synopsis

`CIRCLE` *x*, *y*, *radius*
`FILL CIRCLE` *x*, *y*, *radius*

Draw or fill a circle.

Description

`CIRCLE` draws a circle centered on *x*, *y* with radius *r* pixels using the current graphics color set by `COLOR`:

This figure is generated by the following program which uses `CIRCLE`:

```
***../examples/random-circles.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "random-circles"` or `|random-circles`.

If `CIRCLE` is preceded by `FILL`, the whole circle is filled with the current graphics color.

This figure is generated by the following program which uses `FILL CIRCLE`:

```
***../examples/random-spots.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "random-spots"` or `|random-spots`.

CIS

Synopsis

`CIS` *arg*

Compute circular sine and cosine.

Description

`CIS` computes the sine and cosine of *arg* radians. The result is a complex number where the real part is the sine of *arg* and the imaginary part is the cosine of :

```
> print cis rad 30
0.866025+0.5j
> _
```

The result of `CIS` is a point that lies on the unit circle of the complex plane and as such as magnitude one:

```
> print abs cis rad 30
1
> _
```

You can use `CIS` to convert a complex number $r(\cos\theta + i \sin\theta)$ in polar form to Cartesian form used in CoreBASIC. For instance, taking r=5 and *theta*=3π/4:

```
> print 5 * cis(5*pi/6)
-4.33013+2.5j
> _
```

If *arg* is an array, `CIS` threads recursively over the elements of the array:

```
> print cis gen(0 to %pi in 4)
[1, 0.5+0.866025j, -0.5+0.866025j, -1-8.74224e-08j]
> _
```

See also

SIN, COS

CLG

Synopsis

`CLG`

Clear graphics display.

Description

The current graphics display is cleared to the display's default background color and the current drawing colour is set to the display's default foreground color.

Note that the default foreground and background colors are set for the particular device that is selected.

For true color LCD or VGA displays, the background color is usually black and the foreground color is usually white.

For monochrome LCD displays, the foreground color is usually black and the background color is whatever color the display uses). Some LCD displays use reverse polarity, such as orange on black, for effect. Whatever the actual colors on display, color 0 always denotes background and color 1 always denotes foreground.

Note that clearing the graphics display does not reset the origin to (0, 0). In order to reset the graphics origin, use `ORIGIN 0, 0`.

CLOSE

Synopsis

CLOSE *arg*

Close a channel.

Description

CLOSE closes the open channel *arg*. All pending data sent, but not yet written, is flushed to the device. For channels associated with disk files, the channel is closed, all buffers are written to the medium, and the file size is updated in the directory. For channels associated with sockets, all data are flushed to the network and the socket is closed gracefully.

```
> list
   10 F = OPEN("/c/log.txt", WRITE)
   20 IF F < 0 THEN PRINT "Open error: "; REPORT(F) : STOP
   30 FOR I = 1 TO 100
   40   PRINT #F, "Count = "; I
   50 NEXT I
   60 CLOSE F
   70 END
> _
```

See also

FLUSH

CLS

Synopsis

```
CLS
```

Clear screen.

Description

The standard output device's screen is cleared by writing the ANSI clear-screen escape sequence.

CMPLX

Synopsis

CMPLX(*real*, *imag*)

Construct complex number.

Description

CMPLX constructs the complex number *real* + *i* × *imag*:

```
> print cmplx(1, 2)
1+2j
> _
```

If *real* and *imag* are vectors, CMPLX creates an array of complex numbers with each real value from *real* paired with its corresponding imaginary value from *imag*:

```
> print cmplx([1, 2], [3, 4])
[1+3j, 2+4j]
> _
```

For this, the length of the *real* and *imag* vectors must be equal:

```
> print cmplx([1, 2], [3, 4, 5])
?dimension error
> _
```

Note

You can use the constant %I to create pure imaginary numbers and then combine these with a real to generate a complex:

```
> print 4 - 7 * %i
4-7j
> print (3 - 4 * %i) * (-2 + 4 * %i)
10+20j
> print %i * %i
-1
> _
```

Using %I in this way is marginally more expensive than using CMPLX to construct a complex number.

See also

%I, IM, RE

CNJ

Synopsis

CNJ *arg*

Computes conjugate.

Description

CNJ computes the conjugate of *arg*:

```
> print cnj 3
3
> print cnj cmplx(3, 5)
3-5j
> print cnj quat(1, -2, 3, -4)
1+2i-3j+4k
> _
```

If *arg* is an array, CNJ threads recursively over the elements of the array:

```
> print cnj [3, cmplx(3, 5), quat(1, -2, 3, -4)]
[3, 3-5j, 1+2i-3j+4k]
> _
```

CON

Synopsis

CON(*dimension*, ...)

Create unit matrix.

Description

CON uses the dimension list in its argument to create a unit matrix. A unit matrix is a matrix where each of its elements is one.

```
> print con(3, 3)
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]
> mat print con(3, 3)
1       1       1
1       1       1
1       1       1
> mat print con(1, 3)
1       1       1
> mat print con(3, 1)
1
1
1
> print con(3)
[1, 1, 1]
> print con(2, 2, 2)
[[[1, 1], [1, 1]], [[1, 1], [1, 1]]]
>
```

See also

IDN, ZER

COL

Synopsis

COL(*matrix*, *n*)

Extract matrix column.

Description

COL extracts column *n* of the matrix *matrix*.

```
> a = [[1, 2], [3, 4]]
> mat print a
1       2
3       4
> print col(a, 0) | col(a, 1)
[1, 3]
[2, 4]
> _
```

See also

ROW

COLOR

Synopsis

COLOR *color*

Set drawing color.

Description

COLOR sets the drawing color to *color*. If the graphics device surface is a true color device, *color* must be a 24-bit RGB true color value. If the graphics device uses a palette, *color* is the palette index to use.

If no graphics device is selected, an error is thrown.

See also

%COLOR

COLOR$

Synopsis

COLOR$ *arg*

Return HTML color specification.

Description

COLOR$ returns the HTML specification for the 24-bit color *arg* as a number sign followed by six uppercase hexadecimal numerals.

```
> print color$ %red
#FF0000
> _
```

If *arg* is an array, COLOR$ threads recursively over the elements of the array:

```
> print color$ [%red, %green, %blue]
["#FF0000", "#00FF00", "#0000FF"]
> _
```

See also

COLOR

CORE

Synopsis

C0RE

Core driver.

Description

C0RE evaluates to the CPU or "core" driver. The CPU driver provides access to the hardware resources of the host CPU.

For full information on the SolderCore CPU driver, see **SolderCore CPU**. For full information on the Freedom Board CPU driver, see **Freedom Board CPU**.

See also

SolderCore CPU, **Freedom Board CPU**

COS

Synopsis

COS *arg*

Compute circular cosine.

Description

COS computes the circular cosine of *arg* in radian measure. For real, complex, and quaternion types, COS returns a number of the same type as its operand:

```
> print cos 10
-0.839072
> print cos cmplx(2, 10)
-4583.12-10014.3j
> print cos quat(2, 3, 4, 5)
-244.987-227.111i-302.814j-378.518k
> _
```

If *arg* is an array, COS threads recursively over the elements of the array:

```
> print cos [2, 3, 4]
[-0.416147, -0.989992, -0.653644]
> _
```

See also

ACS, SIN, TAN

COSH

Synopsis

COSH *arg*

Compute hyperbolic cosine.

Description

Computes the hyperbolic cosine of *arg*. For real, complex, and quaternion types, COSH returns a number of the same type as its operand:

```
> print cosh 1
1.54308
> print cosh cmplx(1, 2)
3.0519+2.03272j
> print cosh quat(2, 3, 4, 5)
2.65366+1.09076i+1.45434j+1.81793k
> _
```

If *arg* is an array, COSH threads recursively over the elements of the array:

```
> print cosh [2, 3, 4]
[3.7622, 10.0677, 27.3082]
> _
```

See also

ACSH, SINH, TANH

CREDITS

Synopsis

CREDITS

Show credits.

Description

CREDITS shows those steely, dedicated individuals responsible for bringing CoreBASIC software and SolderCore hardware to you:

```
> credits

The SolderCore Project is brought to you by...

Paul Curtis at Rowley Associates
  Conceived, designed, wrote, documented, and eventually debugged CoreBASIC.
  www.rowley.co.uk

Iain Derrington at K&I Design
  Designed, prototyped, reworked, tweaked, and finally built SolderCore.
  kandi-electronics.co.uk

Be part of the SolderCore Community...

  Launch www.soldercore.com to start!
  Follow @SolderCore on Twitter.
  Chat on Google Groups at groups.google.com/d/forum/soldercore

> _
```

See also

The SolderCore Website

CROSS

Synopsis

`CROSS(x, y)`

Cross product.

Description

CROSS computes the cross product of *x* and *y*. *x* and *y* must be 3-element vectors of real numbers.

```
> print cross([1, 2, 3], [4, 5, 6])
[3, -6, 3]
> _
```

The cross product of a vector with itself is the zero vector:

```
> print cross([1, 2, 3], [1, 2, 3])
[0, 0, 0]
> _
```

See also

DOT

CRUNCH

Synopsis

CRUNCH
CRUNCH MAX

Compress program.

Description

CRUNCH removes all comments and redundant tokens from your program. If you find yourself running out of memory at runtime, for instance when creating large arrays, you can CRUNCH your program to recover some extra space for those arrays. You will, of course, lose all your comments!

All references to lines that are removed by CRUNCH are correctly re-targeted so execution is not affected:

```
> list
   10 REM Written by some idiot circa 2013.
   20 REM
   30 X = 5
   40 Y = 3
   50 Z = %I : REM Set up
   60 PRINT "A piece of nonsense" ' ...but not any old nonsense...
   70 REM
   80 IF X = 1 THEN GOTO 40 ELSE GOTO 100 : REM Some more...
   90 REM
  100 REM
  110 GOTO 90
> crunch
Program was:    236 bytes
Program is now: 104 bytes
Crunching saved 132 bytes and reduced program size by 55.9%
> list
   30 X = 5
   40 Y = 3
   50 Z = %I
   60 PRINT "A piece of nonsense"
   80 IF X = 1 THEN GOTO 40 ELSE GOTO 110
  110 GOTO 110
> _
```

CRUNCH leaves the structure and line layout of your program intact, removing redundant code only. Using CUNCH MAX, on the other hand, asks that your program be crunched even further—if possible—by using multi-statement lines:

```
> list
   10 REM Written by some idiot circa 2013.
   20 REM
   30 X = 5
   40 Y = 3
   50 Z = %I : REM Set up
   60 PRINT "A piece of nonsense" ' ...but not any old nonsense...
```

```
   70 REM
   80 IF X = 1 THEN GOTO 40 ELSE GOTO 100 : REM Some more...
   90 REM
  100 REM
  110 GOTO 90
> crunch
Program was:    236 bytes
Program is now: 104 bytes
Crunching saved 132 bytes and reduced program size by 55.9%
> list
   30 X = 5
   40 Y = 3
   50 Z = %I
   60 PRINT "A piece of nonsense"
   80 IF X = 1 THEN GOTO 40 ELSE GOTO 110
  110 GOTO 110
> crunch max
Program was:    104 bytes
Program is now: 84 bytes
Crunching saved 20 bytes and reduced program size by 19.2%
> list
   30 X = 5
   40 Y = 3 : Z = %I : PRINT "A piece of nonsense" : IF X = 1 THEN GOTO 40
  110 GOTO 110
> _
```

You may well find that a program compressed using CRUNCH MAX is incomprehensible and cannot be edited because the program lines are too long. You should only use CRUNCH MAX before running your program to achieve top speed and maximum space for data.

See also

MEMORY

CVF

Synopsis

CVF *arg*

Convert binary data to float.

Description

CVF converts *arg* to a floating point value by interpreting the bytes in the argument as a 32-bit IEEE-754 single precision floating value in PC byte order (little endian). *arg* is converted to a string, using the MERGE operator, before conversion takes place.

After the implicit MERGE, *arg* must be a string containing exactly four characters or CoreBASIC throws a dimension error.

```
> print cvf [0x00, 0x00, 0x80, 0x3f]              ' 1.0 is 3F'80'00'00
1
> _
```

You can use REVERSE to swap from network byte order (big endian) to PC byte order (little endian):

```
> print cvf reverse [0x3f, 0x80, 0x00, 0x00]      ' 1.0 is 3F'80'00'00
1
> _
```

See also

MKF, REVERSE

CVI

Synopsis

`CVI` *arg*

Convert binary data to integer.

Description

`CVI` converts *arg* to an integer by interpreting the bytes in the argument as a two's complement integer in PC byte order (little endian). *arg* is converted to a string, using the `MERGE` operator, before conversion takes place.

```
> a = [0x27, 0x59, 0x41, 0x31]
> print cvi a, hex cvi a
826366247       31415927
> _
```

To convert bytes in network byte order (big endian) to an integer, you can use REVERSE:

```
> a = [0x27, 0x59, 0x41, 0x31]
> print cvi reverse a, hex cvi reverse a
660160817       27594131
> _
```

Because the input is treated as two's complement on conversion, `CVI` will return negative values if the most significant bit is set:

```
> print cvi [0x00, 0xff]
-256
> _
```

You can use this capability to easily sign-extend an 8-bit value:

```
> print cvi chr 0x7f
127
> print cvi chr 0xfe
-2
```

See also

CVU, MERGE(), MKI

CVU

Synopsis

CVU *arg*

Convert binary data to to unsigned.

Description

CVU converts *arg* to an integer by interpreting the bytes in the argument as an unsigned integer in PC byte order (little endian). *arg* is converted to a string, using the MERGE operator, before conversion takes place.

```
> a = [0x27, 0x59, 0x41, 0x31]
> print cvu a, hex cvu a
826366247       31415927
> _
```

To convert bytes in network byte order (big endian) to an integer, you can use REVERSE:

```
> a = [0x27, 0x59, 0x41, 0x31]
> print cvu reverse a, hex cvu reverse a
660160817       27594131
> _
```

Because the input is treated unsigned on conversion, CVU will not return negative values even if the most significant bit is set:

```
> print cvu [0x00, 0xff]
65280
> _
```

Note

As the internal representation of integers in CoreBASIC is 32-bit signed, a 32-bit input with its most significant bit set will convert to a signed value:

```
> print cvu [0x98, 0xba, 0xdc, 0xef]
-270746984
> _
```

The internal representation, however, carries all significant bits:

```
> print hex cvu [0x98, 0xba, 0xdc, 0xef]
EFDCBA98
> _
```

See also

CVI, MERGE(), MKI

DATA

Synopsis

DATA *expression*, *expression*…

Provide data.

Description

DATA lists the data for READ statements.

See also

READ

DATE$

Synopsis

DATE$
DATE$(*arg*)

Return textual date.

Description

DATE$ returns the date set for the core in the format YYYY/MM/DD. *arg* is the number of seconds since 1 January 1970, the standard way of representing time in CoreBASIC. If the argument *arg* is negative, DATE$ returns ????/??/??.

DATE$ without an argument returns the time string for the current *core time* and is equivalent to DATE $(CORE.TIME).

```
> list
   10 PRINT "The date according to SolderCore is "; DATE$; "."
   20 END
> run
The date according to SolderCore is 2012/18/05.
> _
```

See also

TIME$

DATE%

Synopsis

DATE%

DATE%(*arg*)

Return day within month.

Description

DATE%(*arg*) returns the current day within the month for the time *arg*. *arg* is the number of seconds since 1 January 1970, the standard way of representing time in CoreBASIC. The result is a number from one to 31.

DATE% without an argument returns the current day within the month for the *core time* and is equivalent to `DATE%(CORE.TIME)`.

```
> list
   10 PRINT "Today is day "; SPOKEN$ DATE%; " of the month."
   20 END
> run
Today is day thirty of the month.
> _
```

See also

DAY%, MONTH%, YEAR%

DAY%

Synopsis

DAY%
DAY%(*arg*)

Return day within week.

Description

DAY%(*arg*) returns the current day within the week for the time *arg*. *arg* is the number of seconds since 1 January 1970, the standard way of representing time in CoreBASIC. The result is a number from one to seven, with Sunday encoded as 1, Monday as 2, and so on.

DAY% without an argument returns the current day within the week for the *core time* and is equivalent to DAY %(CORE.TIME).

```
> list
   10 DAYNAMES = ["---", "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]
   20 PRINT "Today is ";
   30 PRINT DAYNAMES(DAY%)
   40 END
> run
Today is Fri
> _
```

See also

DATE%, MONTH%, YEAR%

DEBUG

Synopsis

DEBUG

Start full screen debugger.

Description

DEBUG enters the full-screen editor, prepares to execute the program, and stops at the first line to be executed.

See also

RUN

DEFPROC ... ENDPROC

Synopsis

DEFPROC *name* [(*expr*, *expr*...)] [USING *var*, *var*...]
statements
ENDPROC

Define a procedure.

Description

DEFPROC defines a procedure which contains a sequence of statements that you can execute with CALL.

In its simplest form, DEFPROC can replace a repetitive sequence of statements that you use often:

```
> list
   10 CALL WELCOME
   20 PRINT | "Once again, please..." ||
   30 CALL WELCOME
   40 END
   50 '
   60 DEFPROC WELCOME
   70   PRINT "Welcome to SolderCore and CoreBASIC,"
   80   PRINT "we hope you enjoy the experience!"
   90 ENDPROC
> run
Welcome to SolderCore and CoreBASIC,
we hope you enjoy the experience!

Once again, please...

Welcome to SolderCore and CoreBASIC,
we hope you enjoy the experience!
> _
```

DEG

Synopsis

DEG *arg*

Convert radian measure to degree measure.

Description

DEG converts *arg* radians to degrees by multiplying *arg* by 180?π:

```
> print deg(2*pi)
360
> _
```

If *arg* is complex or a quaternion, each part is multiplied by 180?π:

```
> print deg cmplx(-%pi, %pi)
-180+180j
> print deg quat(-%pi, %pi, -1, 1)
-180+180i-57.29578j+57.29578k
> _
```

If *arg* is an array, DEG threads recursively over the elements of the array:

```
> print deg [-%pi, %pi]
[-180, 180]
> _
```

See also

RAD

DELETE

Synopsis

DELETE [*start*] [, [*end*]]

Delete lines from program.

Description

DELETE will delete all line numbers falling in the range *start* to *end* inclusive.

If the first line number is omitted, DELETE will use the first line number in the program:

```
> list
   10 ' A poem by Lewis Carroll.
   20 '
   30 PRINT "'Twas brillig, and the slithy toves"
   40 PRINT "  Did gyre and gimble in the wabe:"
   50 PRINT "All mimsy were the borogoves,"
   60 PRINT "  And the mome raths outgrabe."
   70 END
> delete , 55
> list
   60 PRINT "  And the mome raths outgrabe."
   70 END
> _
```

If the second line number is omitted, DELETE will delete to the end of the program:

```
> list
   10 ' A poem by Lewis Carroll.
   20 '
   30 PRINT "'Twas brillig, and the slithy toves"
   40 PRINT "  Did gyre and gimble in the wabe:"
   50 PRINT "All mimsy were the borogoves,"
   60 PRINT "  And the mome raths outgrabe."
   70 END
> delete 35,
> list
   10 ' A poem by Lewis Carroll.
   20 '
   30 PRINT "'Twas brillig, and the slithy toves"
> _
```

With a single line number, DELETE will delete only that line:

```
> list
   10 ' A poem by Lewis Carroll.
   20 '
   30 PRINT "'Twas brillig, and the slithy toves"
> delete 20
> list
   10 ' A poem by Lewis Carroll.
   30 PRINT "'Twas brillig, and the slithy toves"
> _
```

With two lines numbers, DELETE will delete between those lines:

```
> list
   10 ' A poem by Lewis Carroll.
   20 '
   30 PRINT "'Twas brillig, and the slithy toves"
   40 PRINT "  Did gyre and gimble in the wabe:"
   50 PRINT "All mimsy were the borogoves,"
   60 PRINT "  And the mome raths outgrabe."
   70 END
> delete 21, 60
> list
   10 ' A poem by Lewis Carroll.
   20 '
   70 END
> _
```

Typing a line number with no following text is equivalent to deleting just that line:

```
> list
   10 ' A poem by Lewis Carroll.
   20 '
   70 END
> 20
> list
   10 ' A poem by Lewis Carroll.
   70 END
> _
```

DELETE$

Synopsis

DELETE$(*str*, *pos*, *len*)

Delete part of string.

Description

DELETE$ deletes *len* characters from *str* starting at position *pos*:

```
> print delete$("CoreBASIC", 1, 3)
CBASIC
> _
```

If *pos* is beyond the end of *str*, the string is unchanged:

```
> print delete$("CoreBASIC", 9, 100)
CoreBASIC
> _
```

See also

INSERT$

DET

Synopsis

DET *arg*

Matrix determinant.

Description

DET computes the determinant of the two-dimensional matrix *arg*. CoreBASIC will halt with an dimension error if *arg* is not a square two-dimensional matrix.

```
> m = [[1, 2], [3, 4]]    ' Set up matrix
> mat print m             ' Print matrix in matrix form
1       2
3       4
> print det m             ' Compute determinant of 2x2 matrix
-2
> m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
> mat print m
1       2       3
4       5       6
7       8       9
> print det m
0
> _
```

DET uses LU decomposition to compute the determinant.

See also

INV

DFT

Synopsis

DFT *arg*

Compute discrete Fourier transform.

Description

DFT computes the discrete Fourier transform of *arg*:

```
> m = [1, 2, 3, 4]
> print m | dft m
[1, 2, 3, 4]
[4, 6, -2, -2]
> _
```

CoreBASIC will halt with a dimension error if the length of *arg* is not a power of two:

```
> m = [1, 2, 3]
> print dft m
?dimension error
> _
```

In the array, the real parts of samples are listed first, followed by the imaginary parts. If you have an array of complex values, you can use IM and RE to rearrange the array to the form DFT requires:

```
> c = [cmplx(1, 3), cmplx(2, 4)]
> print c
[1+3j, 2+4j]
> print dft c
?type mismatch
> print dft(re c & im c)
[4, 6, -2, -2]
> _
```

Note

The implementation of the DFT function uses floating point operations and is not intended for real-time digital signal processing of large data sets.

See also

IDFT

DIM

Synopsis

`DIM` var(*n*)

Dimension variable.

Description

The variable *var* is created as an array of *n* integers with initial value zero. `DIM A(N)` is equivalent to the statement `LET A = ZER(N)`.

DIR

Synopsis

`DIR`

`DIR` *folder*

Display folder content.

Description

`DIR` will list the contents of a folder as `CATALOG` does. The difference between `DIR` and `CATALOG` is that `DIR` interprets the text *folder* as a folder name or wildcard without requiring quotation marks—this is excellent when traversing the file system interactively using `CD` to move and `DIR` to list contents.

```
> dir

Directory of: /c/*.*

12/05/12   11:26         <DIR>   sys
01/01/12   00:00           144   work.bas
01/01/12   00:00         1,913   crc.bas
01/01/12   00:00         2,499   union.bas
01/01/12   00:00           194   analog.bas
01/01/12   00:00        13,291   trek.bas
01/01/12   00:00         1,986   calib.bas
01/01/12   00:00           227   blinky.bas
01/01/12   00:00           157   !run.bas
01/01/12   00:00         2,580   flag.bas
01/01/12   00:00           538   air.bas
01/01/12   00:00            88   ansi.bas
01/01/12   00:00           102   test.bas
01/01/12   00:00           910   cam.bas
01/01/12   00:00         <DIR>   www

> dir sys/*.fw

Directory of: /c/sys/*.fw

01/01/12   00:00       491,520   core.fw

> _
```

See also

CATALOG, CD

DNS

Synopsis

DNS *arg*

Look up a DNS address.

Description

DNS looks up the hostname *arg* using the current set of DNS servers. If *arg* is a dotted-decimal, the IP address is resolved without using a DNS lookup. The result is an array of four integers containing the resolved IP address:

```
> print dns "www.rowley.co.uk"
[212, 159, 9, 91]
> print dns "www.soldercore.com"
[50, 116, 85, 1]
> print dns "192.168.1.1"
[192, 168, 1, 1]
> _
```

If the host name is in error, no DNS servers are available, no DNS server responds within two seconds, or a DNS server returns an error, CoreBASIC returns the null IP address `[0, 0, 0, 0]`.

```
> print dns "www.bad.server"
[0, 0, 0, 0]
> _
```

You can check for a null IP address using SUM:

```
> list
   10 ADDR = DNS "www.bad.server"
   20 IF SUM(ADDR) = 0 THEN PRINT "Can't resolve address"
   30 END
> run
Can't resolve address
> _
```

DOT

Synopsis

`DOT(x, y)`

Dot product.

Description

`DOT` computes the dot product of *x* and *y*. *x* and *y* must be equal-length vectors of real numbers.

```
> print dot([1, 2, 3], [4, 5, 6])
32
> _
```

See also

CROSS

DRAW

Synopsis

DRAW *arg*

Display text on graphics screen.

Description

DRAW draws the string *arg* on the graphics display at the current pen position. You can select the font to use with FONT, and you can move the pen using MOVE.

Within the DRAW command, you can change the color, scale, and rotation using COLOR, SCALE, and ANGLE. This change stays in effect for the duration of the DRAW command or until changed. When the DRAW command completes, local drawing changes are discarded, reverting to the global drawing context.

Example

```
10 COLOR %WHITE
20 MOVE 10, 10
30 DRAW COLOR %RED; "Error: "
40 DRAW "Software crashed!"
50 END
```

This example set the global drawing color to white then proceeds to draw `Error:` in red. Once the DRAW command completes, that local change is canceled and `Software crashed!` is drawn in the global drawing color, white.

DUMP

Synopsis

DUMP

Display all variables.

Description

All variables, and their values, are shown in list form to standard output:

```
> list
   10 U = [1, 7, 2]
   20 B = "CoreBASIC"
   30 F = 1 / 3
> run
> dump
U = [1, 7, 2]
B = "CoreBASIC"
F = 0.333333
> _
```

EDIT

Synopsis

`EDIT`

`EDIT` *line*

`EDIT` *filename*

Edit a single line or entire program.

Description

`EDIT` with a line number will recall that program line to be edited in calculator mode:

```
> list
   10 PRINT "Hello"
   20 END
> edit 10
10 PRINT "Hello"
```

Here, the underline indicates the cursor position immediately after executing `EDIT 10`.

Visual editor

`EDIT` without a line number will invoke the CoreBASIC visual editor to edit the program in memory.

Load and edit

`EDIT` with a filename will load the file *filename* (as if using `LOAD`) and enter the visual editor to edit it.

Note

Pressing F4 at the CoreBASIC command line will execute `EDIT`. Not all terminal emulators will pass the F4 keystroke to the client.

See also

Visual editor keystrokes

EJECT

Synopsis

`EJECT`
`EJECT` *name*

Eject media.

Description

`EJECT` flushes all cached, unwritten sectors to the specified volume and unmounts it so that it can no longer be accessed by CoreBASIC. If *name* is omitted, CoreBASIC ejects the `/c` volume which is the microSD card on the SolderCore.

See also

MOUNT

ELSE

Synopsis

ELSE

ELSE *statements*

Alternative execution path for `IF`.

Description

See **IF ... THEN**.

END

Synopsis

`END`

Terminate program execution.

Description

CoreBASIC stops executing the program and returns to the command prompt. Note that, in contrast to `STOP`, this returns control without displaying any message.

Notes

CoreBASIC will automatically make the following changes when `END` is used in conjunction with another keyword:

- `END IF` is converted to `ENDIF`.
- `END CASE` is converted to `ENDCASE`.
- `END WHILE` is converted to `WEND`.

ENDCASE

Synopsis

ENDCASE

End a CASE statement.

Description

ENDCASE indicates the end of an CASE statement.

Notes

You can write END CASE as two separate words and CoreBASIC will change the two words to ENDCASE automatically.

See also

See **CASE ... ENDCASE**.

ENDIF

Synopsis

`ENDIF`

End an `IF` statement.

Description

`ENDIF` indicates the end of an `IF` statement.

Notes

You can write `END IF` as two separate words and CoreBASIC will change the two words to `ENDIF` automatically.

See also

IF ... THEN

ENDPROC

Synopsis

ENDPROC

End a procedure.

Description

See **DEFPROC ... ENDPROC**.

EOF

Synopsis

EOF *unary*

Inquire end of file.

Description

EOF inquires whether the file *unary* is positioned at the end. EOF will return true if the stream *unary* is positioned at the end and false otherwise. If *unary* is not an open file, or is a socket, EOF returns true.

EQV

Synopsis

x EQV *y*

Logical equivalence.

Description

EQV computes the logical equivalence of *x* and *y*. The result is true if *x* and *y* are both true or *x* and *y* are both false:

```
> list
   10 FOR X = FALSE TO TRUE
   20   FOR Y = FALSE TO TRUE
   30     PRINT TRUTH X; " EQV "; TRUTH Y; " = "; TRUTH(X EQV Y)
   40   NEXT Y
   50 NEXT X
   60 END
> run
False EQV False = True
False EQV True = False
True EQV False = False
True EQV True = True
> _
```

Both *x* and *y* are converted to integers before applying EQV. After conversion to integers, any nonzero value is considered true.

```
> print 1.5 eqv 0.2 | 1.1 eqv 1.2
0
1
> _
```

Arrays are processed element-by-element and a new array is created containing the elementwise equivalence of each pair:

```
> print [0, 1, 0, 1] eqv [0, 0, 1, 1]
[1, 0, 0, 1]
> _
```

ERROR

Synopsis

ERROR

Return last error code.

Description

ERROR returns the last error code caught by TRY. When the program is first run, ERROR is set to zero, indicating no previous error. If the statement following TRY executed without error, zero is assigned to ERROR; if the statement following TRY raised an error, the corresponding error code is assigned to ERROR.

```
> list
   10 ' Generate a programmed "type mismatch" error
   20 '
   30 PRINT "Current error is: "; REPORT ERROR
   40 TRY PRINT "Comparing 3 to 4: "; TRUTH(3 < 4)
   50 PRINT "Current error is: "; REPORT ERROR
   60 TRY PRINT "Comparing 3 to ""Four"": "; TRUTH(3 < "Four")
   70 PRINT "Current error is: "; REPORT ERROR
   80 END
> run
Current error is: OK
Comparing 3 to 4: True
Current error is: OK
Comparing 3 to "Four": Current error is: type mismatch
> _
```

This example shows that some items are printed by PRINT before the error is raised, and then PRINT is abandoned and no further items are output by PRINT.

See also

TRY, REPORT

EXAMPLE

Synopsis

EXAMPLE CATALOG
EXAMPLE

List examples available from network.

See **EXAMPLE CATALOG**.

Synopsis

EXAMPLE LOAD *name*
EXAMPLE *name*

Load example from network.

See **EXAMPLE LOAD**.

EXAMPLE CATALOG

Synopsis

EXAMPLE CATALOG

EXAMPLE

List examples available from network.

Description

EXAMPLE CATALOG contacts the SolderCore server at `www.soldercore.com` and lists the current set of examples that are stored on the server. EXAMPLE, with nothing following, is the same as EXAMPLE CATALOG.

```
> example catalog
Connecting to www.soldercore.com (192.232.216.121)...
Requesting /examples/ from network...

Index of /examples

  * Parent Directory
  * 3d-cube-1.bas
  * 3d-function-plot.bas
  * bouncing-lines.bas
  * charlcd.bas
  * colors-shield-message.bas
  * compass-demo.bas
  ?
  * welcome.bas

Apache Server at www.soldercore.com Port 80
> _
```

You can load an example from the list into CoreBASIC using EXAMPLE LOAD and omitting the "`.bas`" extension:

```
> example load "welcome"
Connecting to www.soldercore.com (192.232.216.121)...
Loading welcome.bas from network...
Program loaded and ready. Type RUN to execute.
> _
```

See also

EXAMPLE LOAD

EXAMPLE LOAD

Synopsis

EXAMPLE LOAD *name*

EXAMPLE *name*

Load example from network.

Description

EXAMPLE LOAD loads the example with the name *name* from the SolderCore website, `www.soldercore.com`. When EXAMPLE is followed by an expression, the LOAD is optional:

```
> example "welcome"
Connecting to www.soldercore.com (192.232.216.121)...
Loading welcome.bas from network...
Program loaded and ready. Type RUN to execute.
> list
   10 ' Welcome program for CoreBASIC.
   20 '
   30 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
   40 PRINT "For more information, visit http://www.soldercore.com/"
   50 '
   60 END
> run
Welcome to CoreBASIC on the SolderCore!
For more information, visit http://www.soldercore.com/
> _
```

You can abbreviate EXAMPLE LOAD to a single bar to quickly load a program:

```
> |welcome
Connecting to www.soldercore.com (192.232.216.121)...
Loading welcome.bas from network...
Program loaded and ready. Type RUN to execute.
> _
```

Note

You can list the set of available online examples using EXAMPLE CATALOG.

See also

EXAMPLE CATALOG

EXIT

Synopsis

```
EXIT FOR
```

Exit innermost FOR loop.

See **EXIT FOR**.

Synopsis

```
EXIT REPEAT
```

Exit innermost REPEAT loop.

See **EXIT REPEAT**.

Synopsis

```
EXIT WHILE
```

Exit innermost WHILE loop.

See **EXIT WHILE**.

EXIT FOR

Synopsis

EXIT FOR

Exit the innermost FOR loop.

Description

EXIT FOR exists the innermost FOR loop, transferring control to the statement following NEXT.

Note

Do not exit a FOR loop by assigning to the control variable and do not use GOTO to skip over the NEXT statement.

EXIT REPEAT

Synopsis

EXIT REPEAT

Exit the innermost REPEAT loop.

Description

EXIT REPEAT exists the innermost REPEAT loop, transferring control to the statement following UNIIL.

Note

Do not exit a REPEAT loop by using GOTO to skip over the UNTIL statement.

EXIT WHILE

Synopsis

`EXIT WHILE`

Exit the innermost `WHILE` loop.

Description

`EXIT WHILE` exists the innermost `WHILE` loop, transferring control to the statement following `WEND`.

Note

Do not exit a `WHILE` loop by using `GOTO` to skip over the `WEND` statement.

EXP

Synopsis

EXP *arg*

Exponential.

Description

EXP computes the exponential of *arg*.

For real, complex, and quaternion types, EXP returns a number of the same type as its operand:

```
> print exp 2
7.3891
> print exp cmplx(-0.5, 1)
0.32771+0.51038j
> print exp quat(2, 3, 4, 5)
5.2119+2.2222i+2.963j+3.7037k
> _
```

If *arg* is an array, EXP threads recursively over the elements of the array:

```
> print exp [2, 3, 4]
[7.3891, 20.0855, 54.5982]
> _
```

See also

LOG

EXPAND

Synopsis

EXPAND *arg*

Expand string to array.

Description

EXPAND converts its string argument into an array of integers corresponding to the ASCII code of each character in the string.

```
> print expand "SolderCore"
[83, 111, 108, 100, 101, 114, 67, 111, 114, 101]
> print expand ["Solder", "Core"]
[83, 111, 108, 100, 101, 114, 67, 111, 114, 101]
> _
```

Using CHR with EXPAND expands the string to individual characters:

```
> print chr expand "SolderCore"
["S", "o", "l", "d", "e", "r", "C", "o", "r", "e"]
> _
```

EXT

Synopsis

EXT *unary* = *expr*

Set file extent.

Description

Assignment to EXT sets the length of the file *unary* to *expr*. You cannot extend a file beyond its current length.

See also

EXT()

EXT()

Synopsis

EXT *arg*

Get file extent.

Description

EXT returns the extent, or length, of the open file *arg*. If *arg* does not refer to an open file, the result of EXT will be negative and indicate an error.

```
> list
   10 F = OPEN("/c/log.txt", READ)
   20 IF F < 0 THEN PRINT "Error opening /c/log.txt: "; REPORT F : END
   30 PRINT "The file /c/log.txt is "; EXT F; " bytes in size."
   40 CLOSE F
   50 END
> run
The file /c/log.txt is 7 bytes in size.
> dir "/c/log.txt"
             7   log.txt
> _
```

See also

EXT

FALSE

Synopsis

FALSE

Boolean false.

Description

FALSE is a synonym for zero as Boolean values are represented as integers in CoreBASIC.

```
> print false | truth false
0
False
> _
```

See also

TRUE

FILL

Synopsis

`FILL CIRCLE` *x*`,` *y*`,` *radius*
`FILL LINE` *x0*`,` *y0* `TO` *x1*`,` *y1* `TO` … `FILL RECTANGLE` *x0*`,` *y0* `TO` *x1*`,` *y1*

Also…

`FILL` *x0*`,` *y0* `TO` *x1*`,` *y1*

Draw a filled object.

Description

`FILL` instructs CoreBASIC to fill the area drawn by the following graphic command rather than simply drawing an outline. You can use `FILL` before the `CIRCLE`, `RECTANGLE`, and `LINE` commands.

`RECTANGLE` is optional with the `FILL` command. If you omit a command after `FILL`, `RECTANGLE` is assumed.

Note that `FILL : RECTANGLE`… is not valid: the `FILL` must immediately precede a graphics command.

See also

CIRCLE, **LINE**, **RECTANGLE**

FIRMWARE

Synopsis

`FIRMWARE CATALOG`

List firmware versions available from network.

See **FIRMWARE CATALOG**.

Synopsis

`FIRMWARE CHECK`

Check integrity of upgrade firmware.

See **FIRMWARE CHECK**.

Synopsis

`FIRMWARE GET`
`FIRMWARE GET` *version*

Get firmware from network.

See **FIRMWARE GET**.

Synopsis

`FIRMWARE KILL`

Erase upgrade firmware.

See **FIRMWARE KILL**.

Synopsis

`FIRMWARE SAVE`

Replace installed firmware.

See **FIRMWARE SAVE**.

Synopsis

`FIRMWARE RUN`
`FIRMWARE [RUN]` *version*

Download and replace installed firmware.

See **FIRMWARE RUN**.

FIRMWARE CATALOG

Synopsis

`FIRMWARE CATALOG`

List firmware versions available from network.

Description

`FIRMWARE CATALOG` contacts the SolderCore server at `www.soldercore.com` and lists the releases of CoreBASIC firmware compatible with your SolderCore.

```
> firmware catalog

Installed firmware is 0.9.5.

Connecting to www.soldercore.com (192.232.216.121)...
Requesting /firmware/soldercore-v1/ from network...

Index of /firmware/soldercore-v1

  * Parent Directory
  * core.fw
  * core-1.0.0.fw
  ?

Apache Server at www.soldercore.com Port 80
> _
```

See also

FIRMWARE GET, **FIRMWARE RUN**

FIRMWARE CHECK

Synopsis

`FIRMWARE CHECK`

Check integrity of upgrade firmware.

Description

`FIRMWARE CHECK` performs an integrity check of the firmware contained in the upgrade image file `/c/sys/core.fw` on the microSD card.

```
> firmware check
Verifying integrity of upgrade firmware...
Verified 491,520 bytes of 491,520 (100%)...with good CRC.
Upgrade firmware is verified to work on soldercore-v1.
Installed firmware is 1.0; upgrade firmware is 1.0.
Installed firmware identical to upgrade firmware.
> _
```

See also

FIRMWARE SAVE, FIRMWARE RUN

FIRMWARE GET

Synopsis

`FIRMWARE GET`

`FIRMWARE GET` *version*

Get firmware from network.

Description

`FIRMWARE GET` contacts the SolderCore server at `www.soldercore.com`, retrieves the most recent release of the CoreBASIC firmware, and stores the upgrade firmware in the file `/c/sys/core.fw` on the microSD card. If you follow `FILENAME RUN` with a filename, that specific firmware version is downloaded from the SolderCore website.

Once the upgrade firmware is successfully downloaded from the server, `FIRMWARE GET` proceeds with an integrity check of the upgrade firmware to ensure that the firmware is intended for this SolderCore model.

```
> firmware get
Connecting to www.soldercore.com (192.232.216.121)...
Requesting /firmware/soldercore-v1/core.fw...
Downloaded 491,520 bytes of 491,520 (100%)...
Verifying integrity of upgrade firmware...
Verified 491,520 bytes of 491,520 (100%)...with good CRC.
Upgrade firmware is verified to work on soldercore-v1.
Installed firmware is 0.9; upgrade firmware is 1.0.
Installed firmware differs from upgrade firmware.
> _
```

You can update the SolderCore to the upgrade firmware with `FIRMWARE SAVE`.

See also

FIRMWARE CATALOG, **FIRMWARE RUN**

FIRMWARE KILL

Synopsis

`FIRMWARE KILL`

Erase upgrade firmware.

Description

`FIRMWARE KILL` removes the file `/c/sys/core.fw` from the microSD card.

FIRMWARE RUN

Synopsis

`FIRMWARE RUN`
`FIRMWARE` [`RUN`] *version*

Download and replace installed firmware.

Description

`FIRMWARE RUN` contacts the SolderCore server at `www.soldercore.com`, retrieves the most recent release of the CoreBASIC firmware, and stores the upgrade firmware in the file `/c/sys/core.fw` on the microSD card. If you follow `FIRMWARE RUN` with a filename, that specific firmware version is downloaded from the SolderCore website.

Once the upgrade firmware is successfully downloaded from the server, `FIRMWARE RUN` proceeds with an integrity check of the upgrade firmware to ensure that the firmware is intended for this SolderCore model. If the upgrade firmware passes all checks, the SolderCore resets and the upgrade firmware is installed by the bootloader.

Note

`FIRMWARE RUN` combines `FIRMWARE GET` followed by `FIRMWARE SAVE` into a single operation.

See also

FIRMWARE GET, **FIRMWARE SAVE**, **FIRMWARE CATALOG**

FIRMWARE SAVE

Synopsis

`FIRMWARE SAVE`

Replace installed firmware.

Description

`FIRMWARE SAVE` performs an integrity check of the firmware contained in the upgrade image file `/c/sys/core.fw` on the microSD card (as `FIRMWARE CHECK`); if the upgrade firmware passes the integrity check and is suitable for the SolderCore model, the SolderCore is reset and the upgrade firmware is installed by the bootloader.

```
> firmware save
Verifying integrity of upgrade firmware...
Verified 491,520 bytes of 491,520 (100%)...with good CRC.
Upgrade firmware is verified to work on soldercore-v1.
Installed firmware is 0.9; upgrade firmware is 1.0.
Installed firmware differs from upgrade firmware.

FIRMWARE UPGRADE STARTED

This network connection will be closed, the
SolderCore will reset, and the upgrade firmware
will be programmed into the SolderCore's flash.

If you have problems with firmware replacement,
please visit the support forums.

Closing connection now.

Connection to host lost.

C:\Users\Paul>
```

If the installed firmware and upgrade firmware are identical, `FIRMWARE SAVE` *will not* start the bootloader to flash the firmware:

```
> firmware save
Verifying integrity of upgrade firmware...
Verified 491,520 bytes of 491,520 (100%)...with good CRC.
Upgrade firmware is verified to work on soldercore-v1.
Installed firmware is 1.0; upgrade firmware is 1.0.
Installed firmware identical to upgrade firmware.
> _
```

See also

FIRMWARE GET, FIRMWARE CHECK

FIX

Synopsis

`FIX` *arg*

Floor.

Description

`FIX` computes the largest integer value not greater than *arg*:

```
> print fix 6 | fix -6
6
-6
> print fix 6.4 | fix -6.4
6
-7
```

If *arg* is a complex number or quaternion, `FIX` is applied to the real, imaginary, and vector parts independently:

```
> print fix cmplx(6.4, -6.4)
6-7j
> print fix quat(1.1, -2.2, 3.3, -4.4)
1-3i+3j-5k
```

If *arg* is an array, `FIX` threads recursively over the elements of the array:

```
> print fix [1.1, -2.2, 3.3, -4.4]
[1, -3, 3, -5]
> _
```

See also

CINT, **INT**

FLT

Synopsis

FLT *arg*

Convert to floating.

Description

FLT converts *arg* to a floating value. In many cases you do not need to use FLT as CoreBASIC automatically converts integers to floating point when required. However, FLT also specifies that an input must be in floating point format with INPUT.

```
> print flt 1.5
1.5
> print flt 5
5
> _
```

See also

INPUT

FLUSH

Synopsis

FLUSH *arg*

Flush data.

Description

FLUSH ensures that all buffered data on the open channel *arg* is pushed to the network or written to disk. This is important for sockets where data written to a socket is held for transmission until either a TCP segment is filled or FLUSH is executed to push a partially-filled TCP segment.

See also

CLOSE

FOR ... NEXT

Synopsis

FOR *variable* = *first* TO *last* [STEP *step*]
statements
NEXT *variable*

FOR *variable* = *first* TO *last* IN *count*
statements
NEXT *variable*

Iterate a fixed number of times.

Description

FOR repeatedly executes the statements between the FOR and NEXT statements.

```
> list
   10 FOR I = 1 TO 10
   20   PRINT I; " ";
   30 NEXT I
   40 PRINT
   50 END
> run
1 2 3 4 5 6 7 8 9 10
> _
```

You can use STEP and the *step* parameter to count up in steps other than one:

```
> list
   10 FOR I = 1 TO 10 STEP 2
   20   PRINT I; " ";
   30 NEXT I
   40 PRINT
   50 END
> run
1 3 5 7 9
> _
```

The statements within the loop will not execute if the starting value is past the end value:

```
> list
   10 FOR I = 10 TO 0
   20   PRINT I
   30 NEXT I
   40 END
> run
> _
```

The *step* parameter can be negative to count down:

```
> list
```

```
   10 FOR I = 10 TO 1 STEP -2
   20   PRINT I; " ";
   30 NEXT I
   40 PRINT
   50 END
> run
10 8 6 4 2
> _
```

Rather than specifying a step, you can ask for the values to be evenly spaced between *start* and *end* by using `IN`.

```
> list
   10 FOR I = 1 TO 10 IN 9
   20   PRINT I; " ";
   30 NEXT I
   40 PRINT
   50 END
> run
1 2.125 3.25 4.375 5.5 6.625 7.75 8.875 10
> _
```

Note that a decreasing sequence still uses a positive value for `IN`:

```
> list
   10 FOR I = 2 TO 1 IN 4
   20   PRINT I; " ";
   30 NEXT I
   40 PRINT
   50 END
> run
2 1.66667 1.33333 1
> _
```

FOR EACH ... NEXT

Synopsis

FOR EACH *variable* IN *array*
statements
NEXT *variable*

Iterate over array.

Description

FOR EACH executes *statements* for each element taken from the array *array* and assigned to *variable*:

```
> list
   10 FOR EACH V IN ["Core", "BASIC", EXP(1), %I, %PI, 1, 0]
   20   PRINT "Processing element: "; V
   30 NEXT V
   40 END
> run
Processing element: Core
Processing element: BASIC
Processing element: 2.71828
Processing element: 0-1j
Processing element: 3.14159
Processing element: 1
Processing element: 0
> _
```

To iterate over the characters of a string, use EXPAND and CHR together:

```
> list
   10 FOR EACH CHAR IN CHR EXPAND "CoreBASIC"
   20   PRINT "Give me a """; CHAR; """!"
   30 NEXT V
   40 END
> run
Give me a "C"!
Give me a "o"!
Give me a "r"!
Give me a "e"!
Give me a "B"!
Give me a "A"!
Give me a "S"!
Give me a "I"!
Give me a "C"!
> _
```

See also

CHR, EXPAND

FONT

Synopsis

FONT *arg*

Select font.

Description

FONT selects the font named *arg* for subsequent DISPLAY commands. If the font does not exist, CoreBASIC returns to the command prompt with a "font not found" error.

FONT CATALOG lists the fonts provided with each release of CoreBASIC.

See also

FONT CATALOG

FONT CATALOG

Synopsis

FONT CATALOG

List resident fonts.

Description

FONT CATALOG lists the font names of all the resident fonts.

```
> font catalog

Bitmap fonts

4x6                     5x7                     6x9
7x13                    10x20                   PET

Vector fonts

ROMAN-SIMPLEX

> _
```

See also

FONT

GEN

Synopsis

GEN(*start* TO *end*)
GEN(*start* TO *end* STEP *step*)
GEN(*start* TO *end* IN *count*)

Generate arithmetic progression as an array.

Description

GEN creates an array containing values in an arithmetic progression. The first value in the array will be *start*, continuing in steps of *step*, ending at *end*; if STEP is not present, it is assumed to be one.

```
> print gen(1 to 10)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
> print gen(1 to 10 step 2)
[1, 3, 5, 7, 9]
> _
```

The *step* parameter can be negative to count down:

```
> print gen(10 to 1 step -1)
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
> print gen(10 to 1 step -2)
[10, 8, 6, 4, 2]
> _
```

And *step* can be fractional:

```
> print gen(1 to 2 step 0.25)
[1, 1.25, 1.5, 1.75, 2]
> _
```

Rather than specifying a step, you can ask for the values to be evenly spaced between *start* and *end* by using IN. The value *in* is converted to an integer and the array contains that many values, all equally spaced:

```
> print gen(1 to 10 in 9)
[1, 2.125, 3.25, 4.375, 5.5, 6.625, 7.75, 8.875, 10]
> print gen(2 to 1 in 4)
[2, 1.66667, 1.33333, 1]
> _
```

GET

Synopsis

GET

Read standard input.

Description

GET waits for the next character on the standard input stream, typically the keyboard. The value returned is the ASCII code of the character read.

CoreBASIC encodes standard keystrokes into ASCII codes with values above 128. Please refer to **CoreBASIC Keyboard codes** for further details on keyboard mappings.

See also

GET$

Note

GET is equivalent to `BGET %IN`.

GET$

Synopsis

GET$

Read standard input.

Description

GET waits for the next character on the standard input stream, typically the keyboard. The value returned is a string containing the single character read.

CoreBASIC encodes standard keystrokes into ASCII codes with values above 128. Please refer to **CoreBASIC Keyboard codes** for further details on keyboard mappings.

See also

GET

Note

GET$ is equivalent to `BGET$ %IN`.

GOTO

Synopsis

GOTO *line*

Transfer control to line.

Description

GOTO transfers control to line number *line*.

GFX

Synopsis

GFX.*property*
GFX *arg*

Inquire graphics capabilities.

Description

GFX returns the capability of the selected graphics device. You can inquire a subset of the capabilities using property syntax and the remainder by using a capability index.

Following GFX with a named property inquires the following capabilities:

Property	Description
WIDTH	Visible display width, in pixels.
HEIGHT	Visible display height, in pixels.
DEPTH	Color depth, in bits.
X	Graphics pen *x* co-ordinate.
Y	Graphics pen *y* co-ordinate.
FONT	Selected font name.

Following GFX with an expression *arg* inquires the capability of the graphics device by index. The value of *arg* selects the particular capability of interest according to the following:

Capability	Description
0	Logical display width, in pixels
1	Logical display height, in pixels
2	Visible display width, in pixels
3	Visible display height, in pixels
4	Color depth, in bits
6	Default background color as RGB888
7	Default foreground color as RGB888

Some LCD displays have a wider logical width than visible width, or a taller logical height than visible height. This is because the controller chips are able to control an LCD with more pixels than the current panel supports. You will be able to draw outside of the logical display region, however you will not see that on the screen because the LCD panel is not capable of showing it.

The color depth in bits tells you whether you're dealing with a true color display or a paletted display. For true color displays that have 24-bit color (using an RGB888 arrangement), the color depth will be 24. Some displays

are use and RGB666 arrangement with a color depth of 18. Other displays use an RGB565 arrangement with a color depth of 16.

If a display reports a color depth of 8 or below, it is paletted with a finite set of colors chosen from a wider palette.

If a display reports a color depth of 1 then it is monochrome and supports only two usually fixed, colors.

GOSUB

Synopsis

GOSUB *line*

Call subroutine.

Description

GOSUB transfers control to the subroutine starting at line number *line*. Execution returns to the statement immediately after the GOSUB when the called subroutine executes a RETURN.

See also

RETURN

GREEN%

Synopsis

GREEN% *arg*

Extract green component of a color.

Description

GREEN% extracts the green component of the 24-bit RGB color value *arg* and returns it as a number between 0 (fully desaturated) and 1 (fully saturated).

```
> x = rgb(0.7, 0.2, 0.1)
> print green% x
0.2
> _
```

See also

BLUE%, **RED%**, **RGB**

HELP

Synopsis

HELP
HELP *keyword…*
HELP "*driver-name*"

Display help.

Description

HELP on its own displays the help on HELP. If you follow HELP with a keyword, HELP will look up the HTML help text on the SolderCore website for that keyword and display the associated help page using colored text on the terminal:

```
> help green%
Connecting to www.soldercore.com (192.232.216.121)...
Requesting /manual/corebasic_green_percent.htm from network...

GREEN%

Synopsis

GREEN% arg

Extract green component of a color.

Description

GREEN% extracts the green component of the 24-bit RGB color value arg and
returns it as a number between 0 and 1 (fully saturated).

    > x = rgb(0.7, 0.2, 0.1)
    > print green% x
    0.2
    >

See also

BLUE%, RED%, RGB

> _
```

You can use multiple keywords for compound statements:

```
> help mat print
Connecting to www.soldercore.com (192.232.216.121)...
Requesting /manual/corebasic_mat_print.htm from network...

MAT PRINT

Synopsis
```

```
MAT PRINT expr

Print matrix.
?
```

You can use parentheses to select help on a function when there are statements and functions that use the same keyword:

```
> help run()
Connecting to www.soldercore.com (192.232.216.121)...
Requesting /manual/corebasic_run_function.htm from network...

RUN()

Synopsis

RUN
RUN(index)
?
```

HELP followed by a string displays information relating to the driver named by the string:

```
> help "bmp085"
Connecting to www.soldercore.com (192.232.216.121)...
Requesting /manual/corebasic_bosch_sensortec_bmp085.htm from network...

Bosch Sensortec BMP085 Driver

Installation

INSTALL "BOSCH-SENSORTEC-BMP085"
INSTALL "BMP085"
INSTALL "CORE-PRESSURE"

Options

None.
?
```

Keys

At the `--More--` prompt, you can use the following keys:

- *Space*: Show the next page.
- *Return*: Show the next line.
- .: Continue without paging.
- *Q*: Quit.

See also

NEWS, WEB, INSTALL

HEX

Synopsis

HEX *arg*

Convert to hexadecimal string.

Description

HEX converts *arg* to its equivalent 32-bit unsigned integer and returns a string containing the hexadecimal representation of that integer. The string returned will always have an even number of characters with leading zeros appended if required:

```
> print hex 32767 | hex 1023 | hex 0 | hex -1
7FFF
03FF
00
FFFFFFFF
> _
```

HIGH

Synopsis

`HIGH` *arg*

Compute upper bound.

Description

`HIGH` returns the upper bound of *arg*. If *arg* is a string, `HIGH` returns the number of characters in the string less one:

```
> print high "abc" | high ""
2
-1
> _
```

If *arg* is an array, `HIGH` returns high bound of the array, that is one less than the number of elements in the array:

```
> print high [11, 22, 33, 44] | high []
3
-1
> _
```

The rationale for `HIGH` is that indexes for *arg* run from 0 through one less than `LEN` *arg*, so it is more natural to write a loop over a string or an array using `HIGH` as the upper bound:

```
> list
   10 X = 10 * RND CON(5)
   20 FOR I = 0 TO HIGH X
   30   PRINT I, X(I)
   40 NEXT I
   50 END
> run
0       2.22801
1       9.19247
2       2.65284
3       7.44131
4       8.20336
> _
```

See also

LEN

HISTORY

Synopsis

`HISTORY` [`LIST`]

List command line history.

See **HISTORY LIST**.

Synopsis

`HISTORY KILL`

Clear command line history and erase history file.

See **HISTORY KILL**.

Synopsis

`HISTORY OFF`

Suspend command line recording.

See **HISTORY OFF**.

Synopsis

`HISTORY ON`

Resume command line recording.

See **HISTORY ON**.

Synopsis

`HISTORY` [`PICK`] *n*

Recall command line.

See **HISTORY PICK**.

HISTORY LIST

Synopsis

`HISTORY`

`HISTORY LIST`

List command line history.

Description

`HISTORY LIST` displays all recorded command line history. This history is saved in the file `/c/sys/history.log` on the SolderCore and persists over resets and power cycles.

```
> history list
> history on
> example "welcome"
Connecting to www.soldercore.com (192.232.216.121)...
Loading welcome.bas from network...
Program loaded and ready. Type RUN to execute.
> run
Welcome to CoreBASIC on the SolderCore!
For more information, visit http://www.soldercore.com/
> history list
    1  example "welcome"
    2  run
    3  history list
> _
```

HISTORY KILL

Synopsis

`HISTORY KILL`

Clear command line history and erase history file.

Description

`HISTORY KILL` deletes the history file `/c/sys/history.log`; however, if history recording is turned on, the next command line entered will automatically create the history file and continue to record command lines.

```
> history on
> history list
    1  example "welcome"
    2  run
    3  history list
> history kill
> history list
    1  history list
> history off
> history kill
> history list
> _
```

See also

HISTORY OFF

HISTORY OFF

Synopsis

`HISTORY OFF`

Suspend command line recording.

Description

`HISTORY OFF` suspends recording command lines to the history file.

See also

HISTORY ON

HISTORY ON

Synopsis

`HISTORY ON`

Resume command line recording.

Description

`HISTORY ON` resumes recording command lines to the history file.

See also

HISTORY OFF

HISTORY PICK

Synopsis

`HISTORY` [`PICK`] *n*

Recall command line.

Description

`HISTORY PICK` recalls the command entry *n* from the history file. If entry *n* does not exist, CoreBASIC throws an argument error.

```
> history list
    1  catalog
    2  example "welcome"
    3  run
    4  history list
> history pick 2
> example welcome_
```

You can omit `PICK` and use the single-character substitution `!` for `HISTORY`:

```
> history list
    1  catalog
    2  example "welcome"
    3  run
    4  history list
> !2
> example welcome_
```

Note

At the command prompt, the Up and Down cursor keys scroll through the saved history.

HOUR%

Synopsis

HOUR%
HOUR%(*arg*)

Return hour within day.

Description

HOUR%(*arg*) returns the current hour within the day for the time *arg*. *arg* is the number of seconds since 1 January 1970, the standard way of representing time in CoreBASIC. The result is a number from zero to 23 with zero representing midnight, 1 representing 1am, and so on.

HOUR% without an argument returns the current hour within the day for the *core time* and is equivalent to HOUR%(CORE.TIME).

```
> list
   10 PRINT "The hour is "; HOUR%; "."
   20 END
> run
The hour is 16.
> _
```

I2C

Synopsis

I2C *addr* [WRITE *data*, *data*...] [READ *n* TO *var*]
I2C USING *bus*; *addr* [WRITE *data*, *data*...] [READ *n* TO *var*] I2C # *bus*; *addr* [WRITE *data*, *data*...] [READ *n* TO *var*]

Issue transaction on I2C bus.

Description

I2C initiates a transaction on the I2C bus and waits for it to complete. If no USING clause is present, CoreBASIC selects the primary I2C bus routed to the Arduino headers. You can override the default I2C bus selection with a USING clause that specifies the bus to use for the transaction. The standard channel marker, #, is acceptable as a substitute for USING.

The expression *addr* is the 8-bit device address to read or write. The I2C statement takes care of setting or clearing the least significant bit of the address to correctly read and write to the bus.

data is an expression to write to the I2C bus. This is typically an array of integers or strings. *data* is automatically merged, as if by MERGE, before being sent to the device.

Example

```
I2C 0x10 WRITE 0x55, 0x20, "DATA"
I2C 0x10 READ 5 TO X
I2C 0x10 WRITE 0x10 READ 1 TO X
I2C USING CORE.I2C(1); 0x10 WRITE 0x55, 0x20
```

IDFT

Synopsis

`IDFT` *arg*

Compute inverse discrete Fourier transform.

Description

`IDFT` computes the inverse discrete Fourier transform of *arg*:

```
> m = [1, 2, 3, 4]
> print m | idft m
[1, 2, 3, 4]
[2, 3, -1, -1]
> _
```

CoreBASIC will halt with a dimension error if the length of *arg* is not a power of two:

```
> m = [1, 2, 3]
> print idft m
?dimension error
> _
```

In the output array, the real parts of samples are listed first, followed by the imaginary parts. Here is a snippet of code to convert these into a complex array:

```
> list
   10 ' Frequency domain input.
   20 R = [1, 2, 3, 4]
   30 '
   40 ' Dump data as a vector.
   50 PRINT "Frequency domain sample: "; R
   60 PRINT "Time domain (reals):     "; IDFT R
   70 '
   80 ' Convert to complex vector
   90 C = IDFT R
  100 C = CMPLX(LEFT(C, LEN C / 2), RIGHT(C, LEN C / 2))
  110 '
  120 ' Dump as complex.
  130 PRINT "Time domain (complex):   "; C
  140 END
> run
Frequency domain sample: [1, 2, 3, 4]
Time domain (reals):     [2, 3, -1, -1]
Time domain (complex):   [2-1j, 3-1j]
> _
```

Note

The implementation of the `IDFT` function uses floating point operations and is not intended for real-time digital signal processing of large data sets.

See also

DFT

IDN

Synopsis

`IDN(`*dimension*`, ...)`

Create identity matrix.

Description

`IDN` uses the dimension list in its argument to create an identity matrix. An identity matrix is a square matrix where the leading diagonal elements are one, and all other elements are zero.

```
> print idn(3, 3)
[[1, 0, 0], [0, 1, 0], [0, 1, 0]]
> mat print idn(3, 3)
1       0       0
0       1       0
0       0       1
> mat print idn(1, 3)
?dimension error                ' matrix must be square
> mat print idn(3, 3, 3)
?dimension error                ' require two dimensions
> _
```

See also

CON, **ZER**

IF ... THEN

Synopsis

IF *condition* THEN *true-statements* [ELSE *false-statements*]

IF *condition* THEN
true-statements
[ELSE IF *condition* THEN
true-statements]
[ELSE
false-statements]
ENDIF

Conditionally execute.

Description

IF will execute *true-statements* or *false-statements* depending upon the logical value of *condition*. Any nonzero value is considered true, and a zero value is false.

If *condition* is nonzero then IF executes *true-statements* following THEN, otherwise IF executes the *false-statements* following ELSE.

There are two forms of *IF* statement: a single-line IF statement and a multi-line "block IF" statement.

Single-line IF

A single-line IF statement has both the THEN part and the optional ELSE part on a single line which provides a compact way to code a decision if the THEN and ELSE part is short:

```
> list
   10 FOR I = 1 TO 5
   20   PRINT I; " ";
   30   IF I MOD 2 = 0 THEN PRINT "is even" ELSE PRINT "is odd"
   40 NEXT I
   50 END
> run
1 is odd
2 is even
3 is odd
4 is even
5 is odd
> _
```

You can omit the ELSE part of the IF and if *condition* is false, the THEN part is ignored and execution continues at the following statement.

```
> list
```

```
   10 REPEAT
   20   INPUT "Enter an even number: "; N
   30   IF N MOD 2 <> 0 THEN PRINT N; " is not an even number.  Try again."
   40 UNTIL N MOD 2 = 0
   50 PRINT "Yes, "; N; " is even."
   60 END
> run
Enter an even number: 5
5 is not an even number.  Try again.
Enter an even number: 16
Yes, 16 is even.
```

Muti-line IF

You can use the multi-line IF statement if your conditional code is too long to easily fit a single-line IF.

The multi-line IF is no different from the single-line IF other than you must indicate the end of the IF statement using ENDIF.

```
> list
   10 FOR I = 1 TO 5
   20   IF I MOD 2 = 0 THEN
   30     PRINT I; " is even";
   40   ELSE
   50     PRINT I; " is odd";
   60   ENDIF
   70   IF I = (CINT SQR I) ^ 2 THEN
   80     PRINT " and square";
   90   ENDIF
  100   PRINT
  110 NEXT I
  120 END
> run
1 is odd and square
2 is even
3 is odd
4 is even and square
5 is odd
> _
```

With a multi-line IF, you can place as many statements as you want between after the THEN or ELSE, and you can also nest IF statements inside the THEN and ELSE parts.

You can use ELSE IF for cascaded tests:

```
> list
   10 FOR I = 1 TO 5
   20   IF I = 1 THEN
   30     PRINT I; " is odd and square"
   40   ELSE IF I MOD 2 = 0 AND I = (CINT SQR I) ^ 2 THEN
   50     PRINT I; " is even and square"
   60   ELSE IF I MOD 2 = 0 THEN
   70     PRINT I; " is even"
   80   ELSE
   90     PRINT I; " is odd"
  100   ENDIF
  120 NEXT I
  130 END
```

```
> run
1 is odd and square
2 is even
3 is odd
4 is even and square
5 is odd
> _
```

See also

IFF

IFF

Synopsis

`IFF(`*expr*`,` *true-part*`,` *false-part*`)`

Conditional evaluation.

Description

`IFF` will select whether *true-part* or *false-part* is evaluated according to the control expression *expr*. If *expr* is true, *true-part* will be evaluated and returned as the result of `IFF` without *false-part* being evaluated. Conversely, if *expr* is false, *false-part* will be evaluated and and returned as the result of `IFF` without *true-part* being evaluated.

```
> list
   10 INPUT "Enter a number: "; N
   20 PRINT "The number "; N; " is ";
   30 PRINT IFF(N MOD 2, "odd", "even"); "."
   40 END
> run
Enter a number: 10
The number 10 is even.
> run
Enter a number: 11
The number 11 is odd.
> _
```

Note that the true part and the false part can evaluate to different types as necessary. For instance, the following `IFF` returns a string or a number depending upon the control expression:

```
> list
   10 INPUT "Enter a number: "; N
   20 PRINT "The reciprocal of "; N; " is ";
   30 PRINT IFF(N = 0, "not defined", 1 / N); "."
   40 END
> run
Enter a number: 10
The reciprocal of 10 is 0.1.
> run
Enter a number: 0
The reciprocal of 0 is not defined.
> _
```

See also

IF ... THEN

IM

Synopsis

`IM` *arg*

Extract imaginary part.

Description

`IM` extracts the imaginary part of a complex number:

```
> print im sqr -1
1
> _
```

If *arg* is a quaternion, `IM` returns a the vector part of the quaternion:

```
> print im quat(1, 2, 3, 4)
[2, 3, 4]
> _
```

If *arg* is an array, `IM` threads recursively over the elements of the array:

```
> print im cis gen (-1 to +1 in 4)
[-0.841471, -0.327195, 0.327195, 0.841471]
> _
```

See also

RE

IMP

Synopsis

x `IMP` *y*

Logical implication.

Description

`IMP` computes the logical implication of *x* and *y*. The result is false if and only if *x* is true and *y* is false:

```
> list
   10 FOR X = FALSE TO TRUE
   20   FOR Y = FALSE TO TRUE
   30     PRINT TRUTH X; " IMP "; TRUTH Y; " = "; TRUTH(X IMP Y)
   40   NEXT Y
   50 NEXT X
   60 END
> run
False IMP False = True
False IMP True = True
True IMP False = False
True IMP True = True
> _
```

Both *x* and *y* are converted to integers before applying `IMP`. After conversion to integers, any nonzero value is considered true.

```
> print 1.5 imp 0.2 | 1.1 imp 1.2
0
1
> _
```

Arrays are processed element-by-element and a new array is created containing the elementwise implication of each pair:

```
> print [0, 1, 0, 1] imp [0, 0, 1, 1]
[1, 0, 1, 1]
> _
```

IN

Synopsis

`FOR` *variable* = *first* `TO` *last* `IN` *count*

Iterate a fixed number of times.

See **FOR ... NEXT**.

Synopsis

`GEN(` *start* `TO` *end* `IN` *count* `)`

Generate arithmetic progression as an array.

See **GEN**.

INF

Synopsis

`INF`

`INF(`*arg*`)`

Generate or inquire infinity.

Description

`INF` with an argument *arg* inquires whether *arg* is an infinity:

```
> print 1/0 | 0/0 | 1/2
inf
nan
0.5
> print inf(1/0) | inf(0/0) | inf(1/2)
1
0
0
> _
```

`INF` on its own generates an infinity:

```
> print inf | inf(inf)
inf
1
> _
```

For complex and quaternion arguments, `INF` is true if any of the real, imaginary, or vector parts of *arg* are an infinity:

```
> print inf(cmplx(1, inf)) | inf(cmplx(1, 2))
1
0
> print inf(quat(1, 2, inf, 4)) | inf(quat(1, 2, 3, 4))
1
0
> _
```

If *arg* is an array, `INF` threads recursively over the elements of the array:

```
> print inf([-1/0, 0/0, +1/0])
[1, 0, 1]
> _
```

INK

Synopsis

`INK` *n*

Construct string to change foreground color.

Description

`INK` constructs a string that changes the foreground color of the text display to the color *n*. The `INK` function is most commonly used with the `PRINT` command to change the foreground color of text sent to standard output.

The value *n* must line in the range 0 to 15 inclusive.

Colors

Ink	Color
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White

Note

More descriptive strings to select a particular color are offered using the **$ keyword**.

See also

PAPER, **$constant**

INNER

Synopsis

`INNER(`*mulop*`, `*u*`, `*v*`, `*addop*`)`

Compute inner product.

Description

`INNER` computes the inner product of the vectors *u* and *v* using the multiplicative operator *mulop* and the additive operator *addop*. The inner product is a generalization of the dot product.

The elements of *u* and *v* are combined elementwise using the operator *addop*, and then the resulting vector is reduced using the operator *addop*.

For instance, we can compute the dot product of two vectors using `INNER`:

```
> u = [3, 2, -5]
> v = [4, -4, -7]
> print inner(*, u, v, +)
39
> _
```

This computes 3×4 + 2×(-4) + (-5)×(-7) = 39.

In this particular case, we can verify the result using `SUM` and element-by-element multiplication:

```
> print sum(u * v)
39
> _
```

We can use `INNER` to see whether two vectors have identical elements:

```
> u = [3, 2, -5]
> v = [3, 2, -5]
> print truth inner(=, u, v, and)
True
> v = [3, 2, +5]
> print truth inner(=, u, v, and)
False
> _
```

When using `INNER`, the *u* and *v* vectors must be the same length:

```
> u = [3, 2, -5, 1]
> v = [4, -4, -7]
> print inner(*, u, v, +)
?dimension error
> _
```

See also

REDUCE

INPUT

Synopsis

`INPUT` ["*string*`;`] *var* [AS *type*]…

Read from console.

Description

`INPUT` reads data from the standard input stream into program variables.

If no prompt is provided, CoreBASIC prompts with the name of the variable being read followed by a question mark:

```
> list
   10 INPUT X, Y
   20 PRINT "X is "; X; " and Y is "; Y; "."
   30 END
> run
X? 10
Y? 20
X is 10 and Y is 20.
> _
```

If you provide a prompt, CoreBASIC uses that prompt for each variable to be read in the current `INPUT` statement:

```
> list
   10 INPUT "Enter X and Y: "; X, Y
   20 PRINT "X is "; X; " and Y is "; Y; "."
   30 END
> run
Enter X and Y: 10
Enter X and Y: 20
X is 10 and Y is 20.
> _
```

You can change the prompt within an input statement which provides a better user experience:

```
> list
   10 INPUT "Enter X: "; X; "Enter Y: "; Y
   20 PRINT "You entered: "; X; " and "; Y; "."
   30 END
> run
Enter X: 10
Enter Y: 20
You entered: 10 and 20.
> _
```

You can remove the prompt `INPUT` prints by using an empty string:

```
> list
```

```
   10 PRINT "Blind prompt.  Press enter, OK? ";
   20 INPUT ""; X
   30 END
> run
Blind prompt.  Press enter, OK?
> _
```

```
10 INPUT "Enter a whole number: "; X AS INT
20 PRINT "You entered: "; X
30 END
```

```
> list
   10 INPUT "Enter a number: "; X AS FLT
   20 PRINT "You entered: "; X
   30 PRINT "The integer part of "; X; " is "; FIX X
   40 PRINT "The fractional part of "; X; " is "; ABS(X - FIX X)
   50 END
> run
Enter a number: foobar
?bad input for X: float expected. Try again.
Enter a number: 127.25
You entered: 127.25
The integer part of 127.25 is 127
The fractional part of 127.25 is 0.25
> _
```

```
> list
   10 INPUT "Enter any string: "; X AS STR
   20 PRINT "You entered: "; X
   30 PRINT X; " as upper case is "; UCASE X
   40 PRINT X; " as lower case is "; LCASE X
   50 END
> run
Enter any string: SolderCore CoreBASIC
You entered: SolderCore CoreBASIC
SolderCore CoreBASIC as upper case is SOLDERCORE COREBASIC
SolderCore CoreBASIC as lower case is soldercore corebasic
> _
```

INPUT$

Synopsis

`INPUT$(`*stream*`,` *count*`)`
`INPUT$(`*stream*`,` *count*`,` *timeout*`)`

Read characters from stream.

Description

`INPUT$` reads *count* characters from the stream *stream* with an optional timeout of *timeout* sections. If a timeout is not provided, it is assumed to be infinite.

`INPUT$` will return a string with as many characters as it can read, up to and including *count* characters. Fewer than *count* characters may be returned if the stream closes (because it is a socket stream) or the timeout expires with fewer than *count* characters read from the stream, or if an end of line is encountered (ASCII code 10, or `%LF`).

Note

It is not possible to interrupt `INPUT$` using **Ctrl+C**, so using a sensible timeout is highly advised. Characters read from the standard input stream, `%IN`, are not echoed and are not otherwise processed.

Example

```
***../examples/input-timeout-demo.bas not found ***
```

See also

READ$

INSERT$

Synopsis

`INSERT$(`*sub*`,` *str*`,` *pos*`)`

Insert into string.

Description

`INSERT$` inserts *sub* into *str* at position *pos*:

```
> print insert$("ore", "CBASIC", 1)
CoreBASIC
> _
```

If *pos* is beyond the end of the *str*, *sub* is appended to *str*:

```
> print insert$("BASIC", "Core", 55)
CoreBASIC
> _
```

If *pos* is negative, *sub* is prepended before *str*:

```
> print insert$("Core", "BASIC", -1)
CoreBASIC
> _
```

See also

DELETE$

INSTALL

Synopsis

`INSTALL` *driver* [AS *var*]
`INSTALL FIX` *driver* [AS *var*]

Install driver.

Description

`INSTALL` installs the device driver *driver*, which must be a string. The list of drivers supported by SolderCore are described in the drivers section.

If you omit the `FIX` keyword, the driver is installed as a temporary driver. A temporary driver remains installed until the program is changed.

If you specify the `FIX` keyword, the driver is installed as a permanent driver and remains installed until the SolderCore is reset.

See also

INSTALL CATALOG

INSTALL CATALOG

Synopsis

`INSTALL CATALOG`

List built-in drivers.

Description

`INSTALL CATALOG` lists the drivers that are built into CoreBASIC and that can be installed using `INSTALL`:

```
> install catalog

LED

SPARKFUN-BUTTONPAD                Prototype
SPARKFUN-RINGCODER-BREAKOUT       Prototype
JIMMIE-RODGERS-LOL-SHIELD         Shield
THINGM-BLINKM-BREAKOUT            Breakout
GRAVITECH-7SEG-SHIELD             Shield
ITEAD-STUDIO-COLORS-SHIELD        Shield

?

Prototyped Module

SPARKFUN-BUTTONPAD                Prototype
SPARKFUN-RINGCODER-BREAKOUT       Prototype
OLIMEX-MOD-NOKIA6610              Prototype

> _
```

See also

INSTALL

INSTALL LIST

Synopsis

`INSTALL LIST`
`INSTALL`

List install drivers.

Description

`INSTALL LIST` displays the list of installed drivers, whether they are installed as transitory drivers or fixed drivers, and the amount of high memory that they consume.

Example

```
> install

Driver                              Type        Himem Used
------------------------------------------------------------
SOLDERCORE-CPU                      Fixed                0
EXTENDED-USER-MEMORY                Fixed            1,024

> _
```

See also

INSTALL, INSTALL CATALOG

INSTR

Synopsis

`INSTR(`*sub*`,` *in*`)`
`INSTR(`*sub*`,` *in*`,` *start*`)`

Find substring in string.

Description

`INSTR` returns the index of the start of substring *sub* within the string *in*, with the search starting at index *start*. If *start* is not given it is assumed to be zero:

```
> print instr("o", "Hello, world")
4
> print instr("o", "Hello, world", 5)
8
> print instr("ip", "Mississippi")
7
> _
```

`INSTR` returns −1 if *sub* cannot be found in *in*:

```
> print instr("L", "Hello, world")
-1
> print instr("o", "Hello, world", 9)
-1
> print instr("ipi", "Mississippi")
-1
> _
```

See also

MATCH

INT

Synopsis

`INT` *arg*

Integer part.

Description

`INT` removes any fractional part from *arg* and returns only the integer part:

```
> print int 3.4 | int -3.4
3
-3
> print int 1.5e20
1.5e20
> _
```

If *arg* is a complex number or quaternion, `INT` is applied to the real, imaginary, and vector parts independently:

```
> print int cmplx(6.4, -6.4)
6-6j
> print int quat(1.1, -2.2, 3.3, -4.4)
1-2i+3j-4k
```

If *arg* is an array, `INT` threads recursively over the elements of the array:

```
> print fix [1.1, -2.2, 3.3, -4.4]
[1, -2, 3, -4]
> _
```

See also

CINT, **FIX**

INV

Synopsis

`INV` *arg*

Compute inverse.

Description

`INV` computes the multiplicative inverse of its argument. For real, complex, and quaternion types, `INV` *arg* is simply the reciprocal of *arg* (one divided by *arg*):

```
> print inv 3
0.333333
> print inv cmplx(2, 4)
0.1-0.2j
> print inv quat(1, 2, 3, 4)
0.0333333-0.066667i-0.1j-0.133333k
> _
```

If *arg* is an array, `INV` threads recursively over the elements of the array:

```
> print inv [2, 3, 4]
[0.5, 0.333333, 0.25]
> _
```

Matrix mode

In matrix mode, `INV` evaluates the inverse of the two-dimensional matrix *arg*. CoreBASIC will halt with an dimension error if *arg* is not a square two-dimensional matrix.

```
> m = [[1, 2], [3, 4]]    ' Set up matrix
> mat print m             ' Print matrix in matrix form
1       2
3       4
> _
> i = inv m               ' Compute inverse in scalar mode
> mat print i
1       0.5
0.33333 0.25
> _
> i = mat(inv m)          ' Compute inverse in matrix mode
> mat print i             ' Print inverse in matrix form
-2      1
1.5     -0.5
> _
> mat print m * i         ' Verify identity
1       0
0       1
> _
```

If *arg* is a singular matrix with zero determinant, the returned matrix will be filled with NaNs:

```
> m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
> mat print m
1       2       3
4       5       6
7       8       9
> print det m
0
> mat print inv m
nan     nan     nan
nan     nan     nan
nan     nan     nan
> _
```

`INV` uses Gauss-Jordan elimination with full pivoting to ensure numeric stability when computing the inverse.

See also

DET

IP$

Synopsis

IP$ *arg*

Convert to dotted-decimal IP address.

Description

IP$ converts *arg* to a string in dotted-decimal IPv4 format. *arg* can be either an integer or a 4-element array containing the IPv4 address.

```
> list
   10 PRINT "Network name: "; NET.NAME
   20 PRINT "IP address:   "; IP$(NET.IPADDR)
   20 PRINT "Net mask:     "; IP$(NET.MASK)
   30 PRINT "Gateway:      "; IP$(NET.GATEWAY)
   40 PRINT "DNS server:   "; IP$(NET.DNS(0)); " (primary)"
   50 END
> run
Network name: core-000001
IP address:   10.0.0.133
Net mask:     255.255.255.0
Gateway:      10.0.0.3
DNS server:   10.0.0.8 (primary)
> _
```

See also

NET, DNS

JOIN

Synopsis

`JOIN`(*arg*)
`JOIN`(*arg*, *separator*)
`JOIN`(*arg*, *separator*, *flag*)

Join strings.

Description

`JOIN` will concatenate all the strings in the array *arg* into a single string, placing the string *separator* between each of them:

```
> print join(["fee", "fi", "fo", "fum"], "-")
fee-fi-fo-fum
> _
```

No separators are added for a single string or empty array:

```
> print join(["fee"], "-") | join([], "-")
fee

> _
```

If you do not specify a *separator* parameter, the strings are concatenated without any separator:

```
> print join(["fee", "fi", "fo", "fum"])
feefifofum
> _
```

Note

If you have an array with empty strings, `JOIN` may not deliver exactly what you desire:

```
> print join(["A sentence", "", "Another sentence"], ". ") + "."
A sentence. . Another sentence.
> _
```

In this case you can use the third parameter, *flag*, to tell `JOIN` to ignore empty strings. If *flag* is false (the default), empty strings will be treated as normal; if *flag* is true, empty strings will be ignored when joining:

```
> print join(["A sentence", "", "Another sentence"], ". ", %true) + "."
A sentence. Another sentence.
> _
```

Setting *flag* to true when joining is more efficient than removing empty strings from the array with `PICK` before `JOIN`:

```
> s = ["A sentence", "", "Another sentence"]
> print join(pick(s, s <> ""), ". ") + "."
A sentence. Another sentence.
> _
```

See also

SPLIT

URI$

Synopsis

URI$ *arg*

Convert string to percent-encoded URI.

Description

URI$ returns a string containing the percent-encoded representation of *arg* according to RFC 3986 section 2.2. Percent-encoding replaces certain reserved characters with an escape sequence started with a percent and followed by two hexadecimal numerals.

```
> print uri$ "Hello"
Hello
> print uri$ "http://www.soldercore.com/manual/index.htm"
http%3A%2F%2Fwww.soldercore.com%2Fmanual%2Findex.htm
> _
```

Note

The following characters are reserved and percent-encoded by URI$:

```
: / ? # [ ] @ ! $
& ' ( ) * + , ; =
```

See also

RFC3986 - Uniform Resource Identifier (URI): Generic Syntax

JUSTIFY$

Synopsis

`JUSTIFY$(`*str*`,` *len*`)`

Justify string.

Description

`JUSTIFY$` will left justify or right justify *str* according to the *len* parameter. If *len* is zero or positive, *str* is right justified to *len* characters by padding *str* to the left with spaces. If *len* is negative, *str* is left justified to –*len* characters by padding *str* to the right with spaces.

```
> print "|"; justify$("CoreBASIC", 20); "|"
|           CoreBASIC|
> print "|"; justify$("CoreBASIC", -20); "|"
|CoreBASIC           |
> _
```

If the field width that *str* must be justified in is smaller than the length of *str*, *str* is trimmed to the left or right (according to the sign of *len*) such that the returned string's length is exactly the requested field width:

```
> print "|"; justify$("CoreBASIC", 6); "|"
|eBASIC|
> print "|"; justify$("CoreBASIC", -6); "|"
|CoreBA|
> _
```

See also

LEFT, RIGHT

KILL

Synopsis

`KILL` *path*

Erase program from storage device.

Description

`KILL` erases the file with file name *path*. If the file does not exist, or cannot be erased because it is read only, or if any other error occurs, CoreBASIC stops with an error.

Example

```
> kill "/c/!run.bas"
```

See also

RENAME

LCASE

Synopsis

LCASE *arg*

Convert string to lower case.

Description

LCASE converts the string *arg* to lower case:

```
> print lcase("www.CoreBASIC.com")
www.corebasic.com
> _
```

If *arg* is an array, LCASE threads recursively over the elements of the array:

```
> mat print lcase ["Core", "Basic"]
core
basic
> _
```

See also

UCASE

LEFT

Synopsis

LEFT(*arg*, *n*)

Slice left part of string or array.

Description

LEFT extracts the first *n* characters of the string *arg*:

```
> print left("CoreBASIC", 4)
Core
> _
```

If *n* is negative, it specifies that the number of characters is computed from the end of the string. For instance, you can drop the last four characters from a string:

```
> print left("CoreBASIC", -4)
CoreB
> _
```

LEFT also works on arrays in the same manner as strings:

```
> print left([5, 4, 3, 2, 1], 3)
[5, 4, 3]
> print left([5, 4, 3, 2, 1], -3)
[5, 4]
> _
```

LEN

Synopsis

LEN *arg*

Compute length.

Description

LEN computes the length of its argument. If *arg* is a string, LEN returns the number of characters in the string.

```
> print len "abc" | len ""
3
0
> _
```

If *arg* is an array, LEN returns the number of elements in the array:

```
> print len [11, 22, 33, 44] | len []
4
0
> _
```

See also

HIGH

LET

Synopsis

LET *variable* = *expression*

Assign variable.

Description

Sets *variable* to *expression*. LET is optional and, if not specified, is assumed.

LINE

Synopsis

`LINE` *x*, *y* [`TO` *x*, *y*]...
`LINE TO` *x*, *y* [`TO` *x*, *y*]...

Draw lines.

Description

`LINE` draws lines in the current color between the specified points. `LINE TO` draws a line from the current position to the first point.

See also

PLOT

LIST

Synopsis

`LIST` [*line1*] [`,` [*line2*]]

List a range of lines.

Description

`LIST` will list the program between the lines *line1* and *line2* inclusive.

`LIST` on its own will list the whole program:

```
> example "welcome"
Connecting to www.soldercore.com (192.232.216.121)...
Loading welcome.bas from network...
Program loaded and ready. Type RUN to execute.
> list
   10 ' Welcome program for CoreBASIC.
   20 '
   30 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
   40 PRINT "For more information, visit http://www.soldercore.com/"
   50 '
   60 END
> _
```

If the first line number is omitted, `LIST` lists from the start of the program:

```
> list ,30
   10 ' Welcome program for CoreBASIC.
   20 '
   30 PRINT "Welcome to CoreBASIC on the SolderCore!"
> _
```

If the second line number is omitted, `LIST` lists to the end of the program:

```
> list 40,
   40 PRINT "For more information, visit http://www.soldercore.com/"
   50 '
   60 END
> _
```

With a single line number, `LIST` will list only that line:

```
> list 30
   30 PRINT "Welcome to CoreBASIC on the SolderCore!"
> _
```

With two lines numbers, `LIST` will list everything between those lines:

```
> list 25, 45
   30 PRINT "Welcome to CoreBASIC on the SolderCore!"
```

```
   40 PRINT "For more information, visit http://www.soldercore.com/"
> _
```

Note

Pressing F12 at the CoreBASIC command line will execute LIST.

See also

LIST USING

LIST USING

Synopsis

`LIST USING` *format* [`,` [*line1*] [`,` [*line2*]]]

List a range of lines with selected formatting.

Description

`LIST USING` will list the program, as `LIST` does, between the lines *line1* and *line2* inclusive. Please refer to `LIST` for a description of the line number range.

The *format* argument is a number that defines the style of the listing. This is a combination of the listing mode (plain, highlighted, HTML) or-ed with some formatting options (line number display, control structure indentation).

The mode is one of:

- 0 — Syntax-highlighted listing using ANSI escapes for colors.
- 1 — Plain listing, no syntax highlighting.
- 2 — HTML-formatted, syntax-highlighted listing for inclusion in a web page.

The options, when set, are:

- 8 — Inhibit initial line number in listing.
- 16 — Do not indent control structures.
- 32 — Enable page mode.

The standard mode that `LIST` uses is mode 32: syntax-highlighted with initial line numbers and indented control structures, in paged mode.

A 1980's style plain listing with line numbers and no indentation is mode 17, which is the inclusive-or of the mode, 1, with the no-indent option, 16.

To prepare an HTML-styled listing for the web, without line numbers, you would use mode 10, which is the inclusive-or of the mode, 2, with the no-line-number option, 8.

Of course, other combinations are possible. For instance, SAVE uses listing mode 1 when writing the program to a file as plain text.

See also

LIST

LOCK

Synopsis

LOCK *arg*

Make file read only.

Description

LOCK makes the file with name *arg* read only by setting the read-only attribute. When a file is locked and read only, it cannot be overwritten or removed. In order to remove or overwrite a read-only file, you must first make it writable using UNLOCK.

```
> example "welcome"
Connecting to www.soldercore.com (192.232.216.121)...
Loading welcome.bas from network...
Program loaded and ready. Type RUN to execute.
> save $work
> lock $work
> save $work
?read-only file
> kill $work
?read-only file
> unlock $work
> save $work
> _
```

See also

UNLOCK

LOG

Synopsis

LOG *arg*

Compute natural logarithm.

Description

LOG computes the base-*e* (natural) logarithm of *arg*. For real, complex, and quaternion types, LOG tries to return a number of the same type as its operand:

```
> print log 10
2.30259
> print log cmplx(2, 10)
2.3222+1.3734j
> print log quat(2, 3, 4, 5)
1.99449+0.549487i+0.73265j+0.915812k
> _
```

An exception to this rule exists for negative real values. As a negative real value has no corresponding logarithm that is real, LOG will return a complex number:

```
> print log 10
2.30259
> print log -10
2.30259+3.14159j
> _
```

If *arg* is an array, LOG threads recursively over the elements of the array:

```
> print log [2, 3, 4]
[0.693147, 1.09861, 1.38629]
> _
```

Note

The implementation of LOG follows the ISO standard for BASIC and the way that Microsoft have historically implemented LOG. Significantly, BBC BASIC uses LN for natural logarithms and LOG for common logarithms, so beware of this difference if you are converting programs written for BBC BASIC.

See also

EXP, LOG10, LOG2, SQR

LOG10

Synopsis

`LOG10` *arg*

Compute common logarithm.

Description

`LOG10` computes the base-10 (common) logarithm of *arg*. The definition of `LOG10(x)` is simply `LOG(x)/LOG(10)`.

See also

LOG, **LOG2**

LOG2

Synopsis

LOG2 *arg*

Compute binary logarithm.

Description

LOG2 computes the base-2 (binary) logarithm of *arg*. The definition of `LOG2(x)` is simply `LOG(x)/LOG(2)`.

See also

LOG, **LOG10**

LOAD

Synopsis

LOAD

LOAD *name*

Load program from storage device.

Description

LOAD loads into memory the program contained in the file *name*. The current program name is set to *name* just as NAME would have set the current program name.

With LOAD on its own, the program is loaded from disk using the current program name.

```
> list
> load "/c/welcome.bas"
> list
   10 ' Welcome program for CoreBASIC.
   20 '
   30 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
   40 PRINT "For more information, visit http://www.soldercore.com/"
   50 '
   60 END
> _
```

See also

NAME, SAVE

LTRIM

Synopsis

`LTRIM` *arg*

Remove leading whitespace.

Description

`LTRIM` removes all leading whitespace from the string *arg*. If *arg* is an array of strings, a new array is created with each string having leading whitespace removed:

```
> print "|"; ltrim "   CoreBASIC   "; "|";
|CoreBASIC   |
> _
```

If *arg* is an array, `LTRIM` threads recursively over the elements of the array:

```
> mat print "|" + ltrim ["  Core  ", "  BASIC  "] + "|"
|Core  |
|BASIC  |
> _
```

See also

RTRIM, TRIM

MAIL

Synopsis

`MAIL` *to*, *from*, *subject*, *body*

Send e-mail.

Description

`MAIL` sends a mail message to the e-mail address *to* with the from field *from*. The subject of the e-mail is in *subject* and the body of the e-mail is *body*.

In order to send an e-mail, the SMTP server to use as the agent needs to be set up by assigning the SMTPSERVER property of the NET object.

```
> list
   10 SUBJECT = "Hello from SolderCore!"
   20 FROM = NET.NAME + "@local"  ' fill in your own e-mail address
   30 TOO = "soldercore@googlegroups.com"
   40 BODY = "Hello from " + NET.NAME + "." + $NL
   50 BODY = BODY + "My device address is " + IP$(NET.IPADDR) + "." + $NL
   60 MAIL TOO, FROM, SUBJECT, BODY
   70 PRINT "e-mail away!"
   80 END
> run
e-mail away!
> _
```

E-mail address format

The e-mail addresses *from* and *to* can either be plain e-mail addresses such as:

- `soldercore@googlegroups.com`
- `plc@rowley.co.uk`

Or they can be augmented with a presentation name such as

- `SolderCore <soldercore@googlegroups.com>`
- `Paul Curtis <plc@rowley.co.uk>`

See also

NET

MAT

Synopsis

MAT *var* = *expr*
MAT LET *var* = *expr*

Assign matrix.

See **MAT LET**.

Synopsis

MAT PRINT *expr*

Print matrix.

See **MAT PRINT**.

Synopsis

MAT *arg*

Evaluate *arg* in matrix mode.

See **MAT()**.

MAT LET

Synopsis

MAT *var* = *expr*

MAT LET *var* = *expr*

Assign matrix.

Description

MAT LET evaluates *expr* in matrix mode and assigns the result to *var*.

```
> a = [[1, 2], [3, 4]]
> let b = a * a
> print b
[[1, 4], [9, 16]]          ' element-wise multiplcation
> mat b = a * a
> print b
[[7, 10], [15, 22]]        ' matrix multiplication
> _
```

MAT PRINT

Synopsis

MAT PRINT *expr*

Print matrix.

Description

MAT PRINT evaluates *expr* in matrix mode and prints it in matrix form.

```
> mat print [[1, 2, 3], [4, 5, 6]]
1       2       3
4       5       6
> _
```

If *expr* is a vector, the vector is printed in row form:

```
> mat print [1, 2, 3, 4]
1       2       3       4
> _
```

If you want the vector printed in column format, convert it to a matrix and use TRN to transpose it:

```
> a = [1, 2, 3, 4]
> mat print a
1       2       3       4
> mat print trn [a]
1
2
3
4
> _
```

MAT()

Synopsis

MAT *arg*

Evaluate *arg* in matrix mode.

Description

MAT evaluates *arg* in matrix mode. In matrix mode, some operators and functions work differently to the way they usually work. In scalar mode, all operations on are matrix are computed element-by-element. In matrix mode, operations are computed using their standard mathematical definitions.

```
> a = [[1, 2], [3, 4]]            ' 2x2 matrix
> mat print a
1       2
3       4
> p = a*a
> mat print p                     ' each element is squared
1       4
9       16
> p = mat(a*a)                    ' multiplication in matrix mode
> mat print p                     ' uses standard mathematical matrix multiply
7      10
15     22
> _
```

MATCH

Synopsis

`MATCH(`*pattern*`, `*string*`)`

Wildcard match.

Description

`MATCH` performs a wildcard match to see if *string* matches *pattern*.

The match is performed using the following rules:

- Matching is always case sensitive.
- The character * matches a sequence of zero or more characters.
- The character ? matches exactly one character.

You can use this to check to see whether a filename has a particular extension:

```
> print match("*.bas", "foo.bas")
1
> print match("*.bas", "foo.c")
0
> _
```

However, remember that this match is case sensitive:

```
> print match("*.bas", "FOO.BAS")
0
> _
```

If you want to match filenames without regard to case, use UCASE or LCASE to normalize letter case to begin with:

```
> print match("*.bas", lcase "FOO.BAS")
1
> _
```

See also

INSTR

MAX

Synopsis

x MAX *y*

Maximum.

Description

MAX returns the maximum of *x* and *y*.

For complex operands, the maximum is the complex number with the greatest complex modulus (magnitude). If both have the same magnitude, the maximum is the number with the greatest phase angle.

See also

MAX(), **MIN**

MAX()

Synopsis

MAX *array*

Maximum of array.

Description

MAX returns the maximum value held in the argument *array*:

```
> marks = int(100 * rnd con(10))]
> print marks | max marks
[51, 17, 30, 53, 94, 17, 70, 22, 49, 12]
94
> _
```

Note

MAX(x) is equivalent to REDUCE(MAX, x).

See also

MAX, MIN(), REDUCE

MEMORY

Synopsis

`MEMORY`

Display program memory use.

Description

`MEMORY` displays the current program's memory use and the maximum available program size that CoreBASIC can support.

For example, on a SolderCore:

```
> example "corempu-confidence-test"
Connecting to www.soldercore.com (192.232.216.121)...
Loading corempu-confidence-test.bas from network...
Program loaded and ready. Type RUN to execute.
> memory

Region            Total    Used   Free   Units    Load  Comment
-------------------------------------------------------------------------------
CoreBASIC RAM    67,584   7,604  59,980  bytes   11.3%  Total available RAM
Program text     65,016   5,044  59,972  bytes    7.8%  Compress with CRUNCH
High memory         512     512       0  bytes  100.0%  Drivers and long names
Variable names      512     133     379  bytes   26.0%  Stored in high memory
Scratchpad          256      22     234  cells    8.6%  Fixed runtime overhead
Object store      7,494       0   7,494  cells    0.0%  Arrays and strings

> _
```

To reduce memory usage, you can use `CRUNCH`, which reports how much memory was saved:

```
> crunch
Program was:    5,044 bytes
Program is now: 2,768 bytes
Crunching saved 2,276 bytes and reduced program size by 45.1%
> _
```

See also

CRUNCH

MERGE

Synopsis

MERGE *name*

Merge program from storage device.

Description

MERGE loads the file with file name *name* without deleting the existing program in memory, effecting a merge of two programs. If *name* has no file extension, "`.bas`" is added.

Lines from the loaded program replace existing lines in memory.

Example

Merges `other.bas` from the root folder of the SD card.

```
> merge "/c/other.bas"
> _
```

MERGE()

Synopsis

MERGE *arg*

Merge data into string.

Description

MERGE flattens arrays of integers and strings into a single string. Integers are treated as the ASCII code of a character, converted to a string, and then merged. Arrays are merged recursively.

```
> print merge "SolderCore"
SolderCore
> print merge ["Solder", "Core"]
SolderCore
> print merge ["Solder", 67, 111, 114, 101]
SolderCore
> print merge [["Solder"], [67, 111, "r"], 101]
SolderCore
> print merge 65
A
> _
```

MID

Synopsis

`MID(`*arg*`, `*pos*`)`
`MID(`*arg*`, `*pos*`, `*n*`)`

Slice a string or array.

Description

`MID` extracts at most *n* characters of the string *arg* starting at position *pos*, or first *n* elements the array `arg` starting at position *pos*. If *n* is omitted, all characters or array elements starting at index *n* to the end are returned:

```
> print mid("SolderCore", 1, 5)
older
> print mid("SolderCore", 6)
Core
> _
```

If *pos* is negative, it specifies that the start position is is computed from the end of the array:

```
> print mid("SolderCore", -4)
Core
> _
```

See also

LEFT, RIGHT

MIN

Synopsis

x `MIN` *y*

Minimum.

Description

`MIN` returns the minimum of *x* and *y*.

For complex operands, the minimum is the complex number with the smallest complex modulus (magnitude). If both have the same magnitude, the minimum is the number with the smallest phase angle.

See also

MIN(), **MAX**

MIN()

Synopsis

`MIN` *array*

Minimum value of array.

Description

`MIN` returns the minimum value in the argument *array*:

```
> marks = int(100 * rnd con(10))]
> print marks | min marks
[51, 17, 30, 53, 94, 17, 70, 22, 49, 12]
12
> _
```

Note

`MIN(x)` is equivalent to `REDUCE(MIN, x)`.

See also

MIN, MAX(), REDUCE

MINUTE%

Synopsis

`MINUTE%`

`MINUTE%(`*arg*`)`

Return minute within hour.

Description

`MINUTE%(`*arg*`)` returns the current minute within the hour for the time *arg*. *arg* is the number of seconds since 1 January 1970, the standard way of representing time in CoreBASIC. The result is a number from zero to 59.

`MINUTE%` without an argument returns the current minute within the hour for the *core time* and is equivalent to `MINUTE%(CORE.TIME)`.

```
> list
   10 PRINT "The minute is "; MINUTE%; "."
   20 END
> run
The minute is 23.
> _
```

MKDIR

Synopsis

`MKDIR` *path*

Create folder.

Description

`MKDIR` creates the folder specified by *path*. `MKDIR` will not create a folder that already exists or create a folder with the same name as a file that exists. If there is an error creating the folder *path*, CoreBASIC throws an error.

See also

MKDIR()

Example

```
> dir "/c/usr/paul/demos"
?file not found
> mkdir "/c/usr/paul/demos"
> dir "/c/usr/paul/demos"
>
```

See also

MKDIR()

MKDIR()

Synopsis

`MKDIR` *path*

Create folder.

Description

`MKDIR` creates the folder specified by *path* and returns a status code indicating success or failure. `MKDIR` will not create a folder that already exists or create a folder with the same name as a file that exists.

This form of `MKDIR` makes it slightly easier to create a folder than using the statement `MKDIR`. You can, however, still use a statement `MKDIR` and catch errors using `TRY`.

See also

MKDIR

MKF

Synopsis

MKF *arg*

Convert float to binary data.

Description

MKF forces *arg* to floating point and then encodes that value as a 4-character string in PC (little-endian) byte order:

```
> x = mkf(1)                ' 1.0 is 3F'80'00'00
> print hex expand x
["00", "00", "80", "3F"]
> _
```

You can use REVERSE to encode the floating point value in network (big-endian) byte order:

```
> x = mkf(1)                ' 1.0 is 3F'80'00'00
> print hex reverse expand x
["3F", "80", "00", "00"]
> _
```

See also

CVF, REVERSE

MKI

Synopsis

`MKI` *arg*

Convert integer to binary data.

Description

`MKI` forces *arg* to an integer and then encodes that integer as a 4-character string in PC (little-endian) byte order.

```
> x = mki(32767)
> print hex expand x
["FF", "7F", "00", "00"]
> _
```

You can use REVERSE to encode the integer value in network (big-endian) byte order:

```
> x = mki(32767)
> print hex reverse expand x
["00", "00", "7F", "FF"]
> _
```

See also

REVERSE

MOD

Synopsis

x MOD *y*

Remainder after division.

Description

MOD returns the remainder after division of *x* by *y*. Both *x* and *y* must be compatible; if they are incompatible, CoreBASIC throws an exception. For instance, CoreBASIC cannot divide strings by numbers directly:

```
> print "3" MOD 2
?type mismatch
> _
```

Numbers

Real numbers divided using standard mathematical rules:

```
> print 11 mod 3
2
> print 10 mod (2*pi)
3.71681
> _
```

It is not possible to use MOD with complex or quaternion operands.

Arrays

Arrays are divided element-by-element and a new array is created containing the remainder after division of each pair.

```
> print [7, 8, 9] mod [2, 3, 4]
[1, 2, 1]
> _
```

You can compute the remainder after a division of a scalar value by an array, or an array by a scalar:

```
> print [1, 2, 3] mod 2
[1, 0, 1]
> print 2 mod [1, 2, 3]
[0, 0, 2]
> _
```

See also

/

MODULES

Synopsis

`MODULES`

List built-in CoreBASIC modules.

Description

`MODULES` lists the modules that the current firmware has installed. Because CoreBASIC is modular, a specific build of CoreBASIC will well select a subset of the available modules to reduce both code space and data space in order to fit into the target platform.

`MODULES` is of most use to CoreBASIC developers when debugging reduced-functionality firmware to ensure that the correct modules are indeed linked in and available for use.

For a full-feature SolderCore, the following modules are built into the firmware:

```
> modules

Installed CoreBASIC modules:

  CORE-MODULE
  SOLDERCORE-MODULE
  STRINGS-MODULE
  FILES-MODULE
  TRANSCENDENTALS-MODULE
  EDITING-MODULE
  GRAPHICS-MODULE
  NETWORKING-MODULE
  DEVICE-MODULE

> _
```

MONTH%

Synopsis

`MONTH%`
`MONTH%(`*arg*`)`

Return month within year.

Description

`MONTH%(`*arg*`)` returns the current month within the year for the time *arg*. *arg* is the number of seconds since 1 January 1970, the standard way of representing time in CoreBASIC. The result is a number from one to 12 representing January through December.

`MONTH%` without an argument returns the current month within the month for the *core time* and is equivalent to `MONTH%(CORE.TIME)`.

```
> list
   10 NAMES = ["---", "Jan", "Feb", "Mar", "Apr", "May", "Jun"]
   10 NAMES = NAMES & ["Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]
   20 PRINT "This month is ";
   30 PRINT NAMES(MONTH%)
   40 END
> run
This month is Dec
> _
```

See also

DATE%, **DAY%**, **YEAR%**

MORE

Synopsis

MORE *name*

List contents of file.

Description

MORE opens the file *name* for reading and lists its contents to the console. The program in memory is not altered at all.

For instance, to list the contents of file `welcome.bas` on the root folder of the SD card:

```
> more "/c/welcome.bas"
   10 ' Welcome program for CoreBASIC.
   20 '
   30 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
   40 PRINT "For more information, visit http://www.soldercore.com/"
   50 '
   60 END
> _
```

MORSE$

Synopsis

MORSE$ *arg*

Encode string into Morse code.

Description

MORSE$ encodes its argument into ITU Morse code using dots, dashes, and spaces.

```
> print morse$ "sos"
... --- ...
> _
```

Numbers are converted to strings using STR and encoded:

```
> print morse$ 100.3
.---- ----- ----- .-.-.- ...--
> _
```

Characters that have no representation in Morse code are ignored:

```
> print "|"; morse$ "[]"; "|"
||
> _
```

See also

Recommendation ITU-R M.1667-1 - International Morse Code
Morse Code - Wikipedia, the free encyclopedia

MOUNT

Synopsis

`MOUNT`
`MOUNT` *name*

Mount media.

Description

`MOUNT` attempts to mount the specified volume so that it can be accessed by CoreBASIC. If *name* is omitted, CoreBASIC attempts to mount the `/c` volume which, on the SolderCore, is the microSD card.

See also

EJECT

MOVE

Synopsis

MOVE *x*, *y*

Move pen position.

Description

MOVE moves the graphics pen position to the co-ordinate (*x*, *y*) without drawing on the display.

$NAME

Synopsis

$NAME

Program work file name.

Description

$NAME returns the work file name of the current program as set by NAME.

```
> print $name
/c/work.bas
> name "/c/welcome.bas"
> print $name
/c/welcome.bas
> _
```

See also

NAME

NAME

Synopsis

NAME

NAME *path*

Set or display program name.

Description

NAME on its own displays the default program name that LOAD and SAVE will use if no filename is given to them.

NAME with *path* given will set the current program name to *path*. If *path* has no file extension, "`.bas`" is added. Note that *path* is not validated when the name is set, it is only validated when used by SAVE and LOAD.

When CoreBASIC starts, the current program name is set to `/c/work.bas`. You can override this default by including a NAME command in the CoreBASIC auto-exec file `/c/!run.bas`.

See also

LOAD, **SAVE**, **$NAME**

NAN

Synopsis

NAN

NAN(*arg*)

Generate or inquire NaN.

Description

NAN with an argument *arg* inquires whether *arg* is a NaN:

```
> print 1/0 | 0/0
inf
nan
> print nan(1/0) | nan(0/0)
0
1
> _
```

NAN on its own generates a Not-a-Number (NaN):

```
> print nan | nan(nan)
nan
1
> _
```

For complex and quaternion arguments, NAN is true if any of the real, imaginary, or vector parts of *arg* are a NaN:

```
> print nan(cmplx(1, nan)) | nan(cmplx(1, 2))
1
0
> print nan(quat(1, 2, nan, 4)) | nan(quat(1, 2, 3, 4))
1
0
> _
```

If *arg* is an array, NAN threads recursively over the elements of the array:

```
> print nan([-1/0, 0/0, +1/0])
[0, 1, 0]
> _
```

NET

Synopsis

NET

Network driver.

Description

NET evaluates to the SolderCore network driver. The network driver provides access to the network features of the SolderCore CPU.

For full information on the network driver, see **SolderCore Network**.

See also

SolderCore Network

NEW

Synopsis

NEW

Start new program.

Description

Your existing program is removed from memory without warning. Note that NEW only clears your program from memory and does not delete the program from disk. The working filename is reset to /c/work.bas.

```
> list
   10 PRINT "'Twas brillig, and the slithy toves"
   20 PRINT "  Did gyre and gimble in the wabe:"
   30 PRINT "All mimsy were the borogoves,"
   40 PRINT "  And the mome raths outgrabe."
   50 END
> run
'Twas brillig, and the slithy toves
  Did gyre and gimble in the wabe:
All mimsy were the borogoves,
  And the mome raths outgrabe.
> new
> list
> _
```

NEWS

Synopsis

NEWS

Show release history.

Description

You can use NEWS to display the release history and what has changed in each each CoreBASIC version. Releases are in reverse chronological order, with the most recent shown first.

```
> news
Connecting to www.soldercore.com (192.232.216.121)...
Requesting /manual/corebasic_change_history.htm from network...

Change history

While we have done all we can to test CoreBASIC, this is early in the
development cycle and, as such, please be warned that there will be the
inevitable bug. We urge you to report any bugs to us so they can be corrected
for all CoreBASIC users.

Release 1.3.0 changes

Major items:

  * Added HTTP-SERVER to serve HTTP requests from the SolderCore. SolderCore
    gets a web server! See HTTP Server.
  * Added FTP-SERVER which enables upload and download of files with an FTP
?
> _
```

Keys

At the `--More--` prompt, you can use the following keys:

- *Space*: Show the next page.
- *Return*: Show the next line.
- .: Continue without paging.
- *Q*: Quit.

See also

HELP, WEB

NEXT

Synopsis

NEXT *var*

End of FOR or FOR EACH loop.

Description

See **FOR ... NEXT** and **FOR EACH ... NEXT**.

NOT

Synopsis

NOT *arg*

Logical negation.

Description

NOT computes the logical negation of *arg*. The result is true *arg* is false, and false if *arg* is true:

```
> list
   10 FOR X = FALSE TO TRUE
   20   PRINT "NOT "; TRUTH X; " = "; TRUTH NOT X
   30 NEXT X
   40 END
> run
NOT False = True
NOT True = False
> _
```

arg is converted to an integer before applying NOT. After conversion to an integer, any nonzero value is considered true:

```
> print not 1.2 | not 0.2
0
1
> _
```

If *arg* is an array, logical negation threads recursively over the elements of the array:

```
> print not [0, 1]
[1, 0]
> _
```

Note

NOT does not perform bitwise negation, it performs logical negation, and there is a difference in the way that nonzero values are treated. To perform bitwise negation, use XOR with −1:

```
> x = [-2, -1, 0, 1, 2]
> print not x | -1 xor x
[0, 0, 1, 0, 0]
[-1, 0, -1, -2, -3]
> _
```

See also

XOR

NUMBER$

Synopsis

NUMBER$ *arg*

Convert number to spoken string.

Description

NUMBER$ converts its argument to a string of English words that you would speak.

```
> print number$ 10
ten
> print number$ 2012
two thousand and twelve
> _
```

The resulting string is not grammatically correct English in that some words are not hyphenated. This is done on purpose so that the string can be separated into words using SPLIT, or encoded again for voice output. For example, the number 1023 is correctly spelled out as "one thousand and twenty-three", but NUMBER$ drops the hyphen:

```
> print number$ 1023
one thousand and twenty three
> _
```

Negative values use "minus" as a prefix, not "negative":

```
> print number$ -1
minus one
> _
```

For large values, SPOKEN$ uses the American definition of billion rather than the English one:

```
> print number$ 1000000001
one billion and one
> _
```

Fractional values are spoken as decimals, not as fractions:

```
> print number$ 0.5
zero point five
> print number$ pi
three point one four one five nine
> _
```

For even exceptionally large or small values, NUMBER$ resorts to standard form:

```
> print number$ 3.14e15
three point one four times ten to the power of fifteen
```

```
> print number$ 0.00001
one times ten to the power of minus five
> _
```

If *arg* is an array, NUMBER$ threads recursively over the elements of the array:

```
> print number$ [5, 4, 3, 2, 1]
["five", "four", "three", "two", "one"]
> _
```

Infinities and not-a-numbers are converted appropriately:

```
> mat print number$ [1/0, -1/0, 0/0]
infinity
minus infinity
not a number
> _
```

OPEN

Synopsis

OPEN(*filepath*)
OPEN(*filepath*, READ)
OPEN(*filepath*, WRITE)
OPEN(*filepath*, READ WRITE)
OPEN(*filepath*, APPEND)

Open file for read, write, or update.

Description

OPEN attempts to open the file *filepath* according to the access mode. If no access mode is given, it defaults to READ.

If value returned is negative, the file could not be opened; if the value returned is positive, it is the channel allocated to the file.

The effect of opening a nonexistent file is different depending upon open mode:

- in READ mode it is an error.
- in WRITE mode, CoreBASIC will truncate the file, if the file can be written.
- in READ WRITE, a new file is created if it does not already exist; if the file already exists, it is opened for update and positioned at the start.
- in APPEND, a new file is created if it does not already exist; if the file already exists, it is opened for update and positioned at the end.

Example

```
> list
   10 ' A self-listing program
   20 F = OPEN($WORK, READ)
   30 IF F < 0 THEN PRINT "Open failed: "; REPORT F : STOP
   40 WHILE NOT EOF(F)
   50   INPUT #F, TEXT AS STR
   60   PRINT TEXT
   70 WEND
   80 END
> save $work
> run
   10 ' A self-listing program
   20 F = OPEN($WORK, READ)
   30 IF F < 0 THEN PRINT "Open failed: "; REPORT F : STOP
   40 WHILE NOT EOF(F)
   50   INPUT #F, TEXT AS STR
   60   PRINT TEXT
   70 WEND
   80 END
>
```

OR

Synopsis

x OR *y*

Logical disjunction.

Description

OR computes the bitwise disjunction of *x* and *y*. A bit is set in the result if the corresponding bit is set in either *x* or *y*:

```
> print hex(0x1234 or 0xfedc)
fefc
> _
```

If *x* and *y* are logical expressions, OR will compute the logical disjunction of *x* and *y* according to the standard truth table:

```
> list
   10 FOR X = 0 TO 1
   20   FOR Y = 0 TO 1
   30     PRINT X; " OR "; Y; " = "; X OR Y
   40   NEXT Y
   50 NEXT X
   60 END
> run
0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1
> _
```

Both *x* and *y* are converted to integers before applying OR. After conversion to integers, any nonzero value is considered true.

```
> print 1.5 or 0.2 | 0.1 or 0.2
1
0
> _
```

Arrays are processed element-by-element and a new array is created containing the elementwise disjunction of each pair:

```
> print [0, 1, 0, 1] or [0, 0, 1, 1]
[0, 1, 1, 1]
> _
```

OR ELSE

Synopsis

x OR ELSE *y*

Short-circuit logical disjunction.

Description

OR ELSE computes the short-circuit logical disjunction of *x* and *y*. The result OR ELSE is *x* if *x* is nonzero, otherwise it is *y*. If *x* is nonzero, *y* is not evaluated.

```
> list
   10 FOR X = FALSE TO TRUE
   20   FOR Y = FALSE TO TRUE
   30     PRINT TRUTH X; " OR ELSE "; TRUTH Y; " = "; TRUTH(X OR ELSE Y)
   40   NEXT Y
   50 NEXT X
   60 END
> run
False OR ELSE False = False
False OR ELSE True = True
True OR ELSE False = True
True OR ELSE True = True
> _
```

Note that *x* OR ELSE *y* is equivalent to `IFF(`*x*`,`*x*`,`*y*`)` with *x* evaluated only once.

```
> print 0 or else "Surprising"
Surprising
> print 2 or else "Surprising"
2
> _
```

Note

For bitwise operations, use OR.

See also

AND THEN, **IFF**, **OR**

ORIGIN

Synopsis

`ORIGIN` *x*, *y*
`ORIGIN CENTER`

Set graphics origin.

Description

`ORIGIN` sets the graphics origin to the display co-ordinate (*x*, *y*). The co-ordinate (*x*, *y*) is an absolute co-ordinate and is not affected by any origin already set.

All graphics commands executed after an origin is set will be relative to the new origin.

`ORIGIN CENTER` sets the graphic origin to the center of the display and is equivalent to

```
ORIGIN GFX.WIDTH/2, GFX.HEIGHT/2
```

OTHERWISE

Synopsis

```
OTHERWISE
```

Catch-all for CASE statement.

Description

See **CASE ... ENDCASE**.

PAPER

Synopsis

PAPER *n*

Construct string to change background color.

Description

PAPER constructs a string that changes the background color of the text display to the color *n*. The PAPER function is most commonly used with the `PRINT` command to change the background color of text sent to standard output.

The value *n* must line in the range 0 to 7 inclusive.

Colors

Paper	Color
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White

See also

INK

PAUSE

Synopsis

PAUSE *n*

Pause execution.

Description

Execution of the CoreBASIC program is paused for at least *n* seconds. During the pause, background tasks, such as networking, LCD refreshing, and other updates, continue to run. CoreBASIC will check for break-in requests every 500 ms so you will not cause CoreBASIC to become unresponsive by an inadvertent programming error.

Example

```
PAUSE 1.5   ' pause for 1.5 seconds
PAUSE 0.01  ' pause for 1/100th of a second
```

PI

Synopsis

`PI`

Approximation for π.

Description

`PI` evaluates to an approximation to π.

Example

```
> list
   10 INPUT "Radius? "; R
   20 PRINT "Circumference of a circle = "; 2 * PI * R
   30 PRINT "Area of a circle = "; PI * R^2
   40 PRINT "Volume of a sphere = "; 4/3 * PI * R^3
   50 END
> run
Radius? 15.2
Circumference of a circle = 94.248
Area of a circle = 706.86
Volume of a sphere = 14137.2
> _
```

PICK

Synopsis

`PICK(`*source*`, `*flags*`)`

Pick elements from array.

Description

`PICK` will select elements from *source* according to the Boolean values in the array *flags*. The array that is returned contains only those elements from *source* where the corresponding element in *flags* is nonzero:

```
> v = ["Co", "re", "BA", "SIC"]
> print pick(v, [0, 0, 1, 0])
["BA"]
> _
```

The lengths of the arrays *source* and *flags* must be identical:

```
> v = ["Co", "re", "BA", "SIC"]
> print pick(v, [0, 0, 1])
?dimension error
> _
```

Usually you would use `PICK` together with some relation to select the elements of the array. For instance, to select all elements of an array that exceed a threshold:

```
> marks = int(100 * rnd con(10))]
> print v
[86, 23, 23, 45, 56, 67, 61, 28, 34, 87]
> print marks > 50]
[1, 0, 0, 0, 1, 1, 1, 0, 0, 1]
> print pick(marks, marks > 50)
[86, 56, 67, 61, 87]
> _
```

See also

SELECT

PIN

Synopsis

`PIN` *x* `AS DIGITAL OUTPUT`
`PIN` *x* `AS DIGITAL INPUT`
`PIN` *x* `AS ANALOG OUTPUT`
`PIN` *x* `AS ANALOG INPUT`

Configure pin.

Description

This sets the mode of the pin *x* to be either a digital or analog input or output. If the pin cannot support the particular mode selected, a "bad pin configuration" error is thrown.

x can be an integer value that indicates a single pin or an array of integer values indicating a set of pins, for instance:

```
> pin 2 as digital input
> pin [1, 3, 5] as digital output
> _
```

Synopsis

`PIN` *x* = *value*

Drive pin.

Description

This sets the pin *x* to the value *value*.

If the pin *x* is a digital output, it is set high if *value* is greater than zero and set low if *value* is less than or equal to zero.

If the pin is an analog output, *value* must be between zero and one. If *value* is less than zero, is is set to zero and if greater than one it is set to one. The analog output is then set to *value*, with zero being the smallest analog output and one being the largest analog output.

If the analog pin is controlled by a DAC, the DAC will output the selected voltage. If the pin is not attached to a DAC but does support PWM, the pin is set to be a digital output in PWM mode and *value* sets the duty cycle of the pin. *value* is clamped between zero and one as it is with an analog output. A value of zero indicates a zero duty cycle, a value of 0.5 is a 50% duty cycle, and a value of 1 indicates a 100% duty cycle.

If *x* is an array of pins, all pins in the array are set to *value*:

```
> pin [1, 3, 5] as digital output
> pin [3, 5] = 0
```

See also

Arduino-style header pinout

PIN CATALOG

Synopsis

`PIN CATALOG`

List pins available and in use.

Description

`PIN CATALOG` lists all pins that are available for use and what, if anything, they are configured for.

SolderCore

On reset, the pin catalog of the SolderCore is:

```
> pin catalog

Pin#  Name      Function        Claim
----------------------------------------
   0  D0        Floating        None
   1  D1        Floating        None
   2  D2        Floating        None
   3  D3        Floating        None
   4  D4        Floating        None
   5  D5        Floating        None
   6  D6        Floating        None
   7  D7        Floating        None
   8  D8        Floating        None
   9  D9        Floating        None
  10  D10       Floating        None
  11  D11       Floating        None
  12  D12       Floating        None
  13  D13       Floating        None
  14  A0        Floating        None
  15  A1        Floating        None
  16  A2        Floating        None
  17  A3        Floating        None
  18  A4        Floating        None
  19  A5        Floating        None
  20  A4_ANA    Floating        None
  21  A5_ANA    Floating        None
  22  SCL1      Floating        None
  23  SDA1      Floating        None
  24  CSMEM1    Floating        None
  25  CSMEM2    Floating        None
  26  UL        Digital Output  Fixed
  27  IDL       Digital Output  Fixed
  28  CS        Digital Output  Fixed
> _
```

See **Arduino-style header pinout**.

Raspberry Pi

On reset, the pin catalog of the Raspberry Pi is:

```
> pin catalog

Pin#  Name      Function        Claim
----------------------------------------
   0  D0        Floating        None
   1  D1        Floating        None
   2  D2        Floating        None
   3  D3        Floating        None
   2  D2        Floating        None
   3  D3        Floating        None
   4  D4        Floating        None
   5  D5        Floating        None
   6  D6        Floating        None
   7  D7        Floating        None
   8  D8        Floating        None
   9  D9        Floating        None
  10  D10       Floating        None
  11  D11       Floating        None
  12  D12       Floating        None
  13  D13       Floating        None
  14  A0        Floating        None
  15  A1        Floating        None
  16  A2        Floating        None
  17  A3        Floating        None
  18  A4        Floating        None
  19  A5        Floating        None
> _
```

Freedom Board

On reset, the pin catalog of the Freedom Board is:

```
> pin catalog

Pin#  Name      Function        Claim
----------------------------------------
   0  D0        RS232 Rx        Locked
   1  D1        RS232 Tx        Locked
   2  D2        Floating        None
   3  D3        Floating        None
   4  D4        Floating        None
   5  D5        Floating        None
   6  D6        Floating        None
   7  D7        Floating        None
   8  D8        Floating        None
   9  D9        Floating        None
  10  D10       Floating        None
  11  D11       Floating        None
  12  D12       Floating        None
  13  D13       Floating        None
  14  A0        Floating        None
  15  A1        Floating        None
  16  A2        Floating        None
  17  A3        Floating        None
  18  A4        Floating        None
  19  A5        Floating        None
  20  D12       Floating        None
  21  D13       Floating        None
  22  J1_01     Floating        None
  23  J1_03     Floating        None
  24  J1_05     Floating        None
```

```
  25  J1_07     Floating        None
  26  J1_09     Floating        None
  27  J1_11     Floating        None
  28  J1_13     Floating        None
  29  J1_15     Floating        None
  30  J2_01     Floating        None
  31  J2_03     Floating        None
  32  J2_05     Floating        None
  33  J2_07     Floating        None
  34  J2_09     Floating        None
  35  J2_11     Floating        None
  36  J2_13     Floating        None
  37  J2_17     Floating        None
  38  J2_19     Floating        None
  39  J9_01     Floating        None
  40  J9_03     Floating        None
  41  J9_05     Floating        None
  42  J9_06     Floating        None
  43  J9_07     Floating        None
  44  J9_09     Floating        None
  45  J9_11     Floating        None
  46  J9_13     Floating        None
  47  J9_15     Floating        None
  48  J10_01    Floating        None
  49  J10_03    Floating        None
  50  J10_05    Floating        None
  51  J10_07    Floating        None
  52  J10_09    Floating        None
  53  J10_11    Floating        None

> _
```

PIN LIST

Synopsis

`PIN LIST`

List pins in use.

Description

`PIN LIST` lists the pins that are currently configured for use and what they are configured for.

The pin function listed is the mode the pin is configured for and is one of:

- *Floating*: Pin is not configured.
- *Digital Output*: Configured as a digital output.
- *Digital Input*: Configured as a digital input.
- *Analog Output*: Configured as a DAC or PWM output.
- *Analog Input*: Configured as an ADC input.
- *SCL*: Configured as an I2C SCL signal.
- *SDA*: Configured as an I2C SDA signal.
- *MOSI*: Configured as an SPI MOSI signal.
- *MISO*: Configured as an SPI MISO signal.
- *SCK*: Configured as an SPI SCK signal.
- *RS232 Tx*: Configured as an RS232 transmit signal.
- *RS232 Rx*: Configured as an RS232 receive signal.

The pin claim indicates how the pin is being used:

- *None*: The pin is not claimed and can be reconfigured for other functions as needed.
- *Shared*: The pin is claimed for shared functional use. For example, SPI and I2C clock and data signals can be shared by many devices. Attempting to claim this pin for other uses or for exclusive access will result in a pin configuration error.
- *Exclusive*: The pin is claimed for exclusive use by a device driver and cannot be shared and cannot be reconfigured. Many shield drivers will claim exclusive use of certain pins because they cannot be shared at a hardware level. The pin will be unlocked for reuse when the program is edited or run.
- *Fixed*: As **Exclusive**, but the pin will not be unlocked for reuse when the program is edited or run.
- *Locked*: The pin is claimed for exclusive use by CoreBASIC and has no user-level access. This prevents inadvertently configuring the target such that it is inaccessible.

SolderCore

On reset, the pin list of the SolderCore is:

```
> pin list
```

```
Pin#  Name      Function        Claim      Mode
-------------------------------------------------------------
  22  UL        Digital Output  Fixed      Standard, 2 mA
  23  IDL       Digital Output  Fixed      Standard, 2 mA
  24  CS        Digital Output  Fixed      Standard, 2 mA
  29  SCK       SPI SCK         Locked     Standard, 8 mA
  30  MISO      SPI MISO        Locked     Standard, 8 mA
  31  MOSI      SPI MOSI        Locked     Standard, 8 mA

> _
```

Raspberry Pi

On reset, the pin list of the Raspberry Pi is empty:

```
> pin catalog

Pin#  Name      Function        Claim      Mode
-------------------------------------------------------------

> _
```

Freedom Board

On reset, the pin list of the Freedom Board is:

```
> pin list

Pin#  Name      Function        Claim      Mode
-------------------------------------------------------------
   0  D0        RS232 Rx        Locked     Standard
   1  D1        RS232 Tx        Locked     Standard

> _
```

PIN()

Synopsis

`PIN` *arm*

Sample pin.

Description

`PIN` reads the current state of pin *arg*.

If the pin is a digital input, *PIN* returns 1 for a high signal and 0 for a low signal. If the pin is an analog input, `PIN` returns a value between zero and one when the analog voltage swings between the full scale voltages, with zero indicating the smallest voltage and one indicating the largest voltage on the pin.

If *arg* is an array, `PIN` is threaded over the array:

```
> pin [1, 3] as digital input
> pin [15, 16] as analog input
> print pin [1, 3, 15, 16]
[1, 0, 0.523949, 0.757576]
> _
```

See also

Arduino-style header pinout

PLOT

Synopsis

PLOT *position* [TO *position*]...

Draw points or lines.

If you specify a single position, a single pixel is plotted at that position on the graphics device using the selected color. If you specify more than one point, lines are drawn to connect each point.

Example

```
PLOT 10, 50
```

Plot a single point at the position 10, 50.

```
PLOT 10, 50 TO 15, 4
```

Draws a line between the points 10, 50 and 15, 4.

```
PLOT 10, 50 TO 15, 4 TO 1, 3
```

Draws a line between the points 10, 50 and 15, 4 and a second line from 15, 4 to 1, 3.

PRINT

Synopsis

`PRINT` *expr* [`,` | `;` | `|`] *expr*…
`PRINT` [*subclause*`,` …] *expr* [`,` | `;` | `|`] [*subclause*`,` …] *expr*…

Print data.

Subclause use

`USING` *expr*
#channel

Description

`PRINT` prints the values of each expression in the list. Numbers are printed in decimal:

```
> print 1/7
0.142857
> _
```

Strings print as themselves:

```
> print "CoreBASIC"
CoreBASIC
> _
```

Complex numbers and quaternions print their component parts:

```
> print cmplx(1, 2) | quat(2, -3, 4, -5)
1+2j
2-3i+4j-5k
> _
```

If the imaginary part of a complex number or any of the vector parts of a quaternion are zero, they are still printed:

```
> print cmplx(1, 0) | quat(2, 0, 0, -5)
1+0j
2-0i+0j-5k
> _
```

Arrays print in *source-like notation*, enclosed in square brackets:

```
> print [1, "CoreBASIC", cmplx(1, 0)]
[1, "CoreBASIC", 1+0j]
> _
```

Separators

Each expression in `PRINT` statement is separated from the next by either a comma, semicolon, or vertical bar. These *format effectors* alter the spacing between values printed.

Using a semicolon effector leaves no gap between one item and the next:

```
> print 1; 2; 3; "Core"; "BASIC"
123CoreBASIC
> _
```

A comma effector uses a horizontal tab character to tabulate each item into columns:

```
> print 1, 2, 3, "Core", "BASIC"
1       2       3       Core    BASIC
> _
```

A vertical bar effector starts a new line:

```
> print 1 | 2 | 3 | "Core" | "BASIC"
1
2
3
Core
BASIC
> _
```

You can use an effector at the end of a statement to modify the way that the PRINT statement ends. If you do not specify an effector, CoreBASIC assumes a vertical bar and prints a new line. If you use a semicolon, no newline is printed at the end of the statement. If you use a comma, a horizontal tab is printed:

```
> list
   10 PRINT "Core";
   20 PRINT "BASIC",
   30 PRINT "OK" ||
   40 END
> run
CoreBASIC       OK

> _
```

To print in matrix mode, you can use MAT PRINT.

See also

MAT PRINT, VDU

PTR

Synopsis

PTR *unary* = *expr*

Set position within file.

Description

Assignment to PTR sets the position within the open file *unary* to *expr*. You cannot position a file beyond its current length or before its start.

See also

PTR()

PTR()

Synopsis

PTR *arg*

Get position within file.

Description

PTR returns the current position of the open file *arg*. If *arg* does not refer to an open file, the result of PTR will be negative and indicate an error.

See also

PTR

QUAT

Synopsis

QUAT(*a*, *b*, *c*, *d*)
QUAT(*r*, *v*)

Construct quaternion.

Description

With four arguments, QUAT constructs the quaternion $a + b \times i + c \times j + d \times k$:

```
> print quat(1, -2, 3, -4)
1-2i+3j-4k
> _
```

With two arguments, QUAT constructs the quaternion with scalar part *r* and vector part *v*:

```
> print quat(1, [-2, 3, -4])
1-2i+3j-4k
> _
```

See also

IM, RE

RAD

Synopsis

RAD *arg*

Convert degree measure to radian measure.

Description

RAD converts *arg* degrees to radians by multiplying *arg* by π?180:

```
> print rad 180
3.14159
> _
```

If *arg* is complex or a quaternion, each part is multiplied by π?180:

```
> print rad cmplx(-180, 180)
-3.14159+3.14159j
> print rad quat(-180, 180, -1, 1)
-3.14159+3.14159i-0.0174533j+0.0174533k
> _
```

If *arg* is an array, RAD threads recursively over the elements of the array:

```
> print rad [-180, 180]
[-3.14159, 3.14159]
> _
```

See also

DEG

RANDOMIZE

Synopsis

RANDOMIZE [*arg*]

Seed random number generator.

Description

RANDOMIZE seeds the random number generator with *arg*. *arg* can be any whole or fractional number. Seeding the random number generator with the same value results in the same sequence of random numbers:

```
> randomize 10
> print rnd 100 | rnd 100
13.8641
86.102
> randomize 10
> print rnd 100 | rnd 100
13.8641
86.102
> _
```

If *arg* is omitted, CoreBASIC uses the internal timer, CORE.TICK, which changes rapidly and is hard to predict. This may not be the best source of randomness during the boot sequence, but is a good source of randomness after CoreBASIC has started and is waiting for user input.

Note

An implicit RANDOMIZE is executed each time a user program is run.

RAVEL

Synopsis

RAVEL *array*

Ravel an array to a vector

Description

RAVEL will ravel an array (either regular or ragged), of dimensionality up to and including 10, into a vector:

```
> v = [[1, 2], 3, [4, ["Five"]], [[[6]]]]
> print ravel v
[1, 2, 3, 4, "Five", 6]
> _
```

RE

Synopsis

RE *arg*

Extract real part.

Description

RE extracts the real part of a complex number:

```
> print re cmplx(4, pi)
4
> _
```

If *arg* is a quaternion, RE returns the real part of the quaternion:

```
> print re quat(1, 2, 3, 4)
1
> _
```

If *arg* is an array, RE threads recursively over the elements of the array:

```
> print re cis gen (-1 to +1 in 4)
[0.540302, 0.944957, 0.944957, 0.540302]
> _
```

See also

IM

READ

Synopsis

READ *var*, *var*…

Read data.

Description

READ reads the next item of data form a DATA statement and advances the data pointer. If there are insufficient data items to satisfy the READ, CoreBASIC stops with an *out of data* error.

Example

Try to read 10 data items, with insufficient data:

```
> list
  10 PRINT "Try to read 10 items of simple data..."
  20 FOR I = 1 TO 10
  30   READ X
  40   PRINT "Read "; X
  50 NEXT I
  60 END
 100 DATA "Here are", "some data items", %PI, %E
> run
Reading simple data...
Read Here are
Read some data items
Read 3.14159
Read 2.71828

?out of data in 30: READ X
> _
```

You can also read vectors:

```
> list
  10 PRINT "Read some vectors..."
  20 FOR I = 1 TO 3
  30   READ X
  40   PRINT X
  50 NEXT I
  60 END
 100 DATA [1, 0, 0], [0, 1, 0], [0, 0, 1]
> run
Read some vectors...
[1, 0, 0]
[0, 1, 0]
[0, 0, 1]
> _
```

It doesn't matter where the DATA statements are in the program, CoreBASIC will find them:

```
> list
```

```
  10 PRINT "Read some vectors..."
  15 DATA [0, 0, 1]
  20 FOR I = 1 TO 3
  30   READ X
  35   DATA [0, 1, 0]
  40   PRINT X
  50 NEXT I
  60 END
 100 DATA [0, 0, 1]
> run
Read some vectors...
[1, 0, 0]
[0, 1, 0]
[0, 0, 1]
> _
```

See also

RESTORE

READ$

Synopsis

READ$(*stream*, *count*)
READ$(*stream*, *count*, *timeout*)

Read characters from stream.

Description

READ$ reads *count* characters from the stream *stream* with an optional timeout of *timeout* sections. If a timeout is not provided, it is assumed to be infinite.

READ$ will return a string with as many characters as it can read, with no more than *count* characters. Fewer than *count* characters may be returned if the stream closes (because it is a socket stream) or the timeout expires with fewer than *count* characters read from the stream.

Note

It is not possible to interrupt READ$ using **Ctrl+C**, so using a sensible timeout is highly advised.

See also

INPUT$

REBOOT

Synopsis

REB00T

Reboot SolderCore into CoreBASIC.

Description

REB00T causes the SolderCore to reset and restart CoreBASIC as if you had pressed the reset switch on the SolderCore. You can use REB00T after changing the network parameters in the network startup file to have SolderCore restart with the new network configuration.

Note

External shields and devices that rely on the RESET signal *will not be reset by* REB00T as the RESET signal is not asserted by REB00T. If you require a hard reset of all devices, you can program a GPIO as an output pin, connect that output to the RESET signal, and then set the GPIO low to reset both the SolderCore and all devices attached to the RESET signal.

RECTANGLE

Synopsis

`RECTANGLE` x_0`,` y_0 `TO` x_1`,` y_1

`FILL RECTANGLE` x_0`,` y_0 `TO` x_1`,` y_1

Draw or fill a rectangle.

Description

`RECTANGLE` draws the rectangle with corners at x_0, y_0 and x_1, y_1 with the current graphics color set by `COLOR`.

This figure is generated by the following program which uses `RECTANGLE`:

```
***../examples/random-rectangles.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "random-rectangles"` or `|random-rectangles`.

If `RECTANGLE` is preceded by `FILL`, the whole rectangle is filled with the current graphics color.

This figure is generated by the following program which uses `FILL RECTANGLE`:

```
***../examples/random-slabs.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "random-slabs"` or `|random-slabs`.

RECYCLE

Synopsis

RECYCLE

Force garbage collection.

Description

CoreBASIC will run garbage collection whenever necessary to make space for new arrays, strings, or drivers. RECYCLE is present so that the CoreBASIC garbage collector can be tested when developing the interpreter.

RED%

Synopsis

RED% *arg*

Extract red component of a color.

Description

RED% extracts the red component of the 24-bit RGB color value *arg* and returns it as a number between 0 (fully desaturated) and 1 (fully saturated).

```
> x = rgb(0.7, 0.2, 0.1)
> print red% x
0.7
> _
```

See also

BLUE%, **GREEN%**, **RGB**

REDUCE

Synopsis

REDUCE(*op*, *v*)

Reduce vector.

Description

REDUCE reduces the vector *v* using the binary operator *op* by interposing *op* between all elements of *v* and calculating the result. For instance:

```
> print reduce(*, [1, 2, 3, 4])
24
> print reduce(+, [4, 5, 6])
15
> _
```

You can use this, for instance, to compute the maximum and minimum values of an array:

```
> v = [10, 15, 2, 7]
> print reduce(max, v) | reduce(min, v))
15
2
> _
```

Or, you could ask whether every value in an array is less than 10; or greater than zero:

```
> v = [10, 15, 2, 7]
> print reduce(and, v < 10) | reduce(and, v > 0)
0
1
> _
```

Note

For convenience, CoreBASIC defines SUM(x) as equivalent to REDUCE(+, x), MAX(x) equivalent to REDUCE(MAX, x), and MIN(x) equivalent to REDUCE(MIN, x),

See also

INNER, MAX(), MIN(), SUM

REM

Synopsis

REM *comment*

' *comment*

No-operation.

Description

All text following REM, to the end of the line, is skipped. You can use REM to add comments to your program.

RENAME

Synopsis

RENAME *from*, *to*

Rename file.

Description

RENAME changes the filename *from* to the filename *to*. *from* can be a full file specification, including volume. *from* must be a simple file name, without any path specification.

See also

KILL

RENUMBER

Synopsis

RENUMBER

RENUMBER *start* [, *increment*]

Renumber program.

Description

RENUMBER will renumber the lines within a program and ensure that any reference to a line in the program is correctly changed to the newly renumbered line.

```
> example "welcome"
Connecting to www.soldercore.com (192.232.216.121)...
Loading welcome.bas from network...
Program loaded and ready. Type RUN to execute.
> list
   10 ' Welcome program for CoreBASIC.
   20 '
   30 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
   40 PRINT "For more information, visit http://www.soldercore.com/"
   50 '
   60 END
> renumber 100, 2
> list
  100 ' Welcome program for CoreBASIC.
  102 '
  104 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
  106 PRINT "For more information, visit http://www.soldercore.com/"
  108 '
  110 END
> _
```

References to existing lines are automatically updated:

```
> 109 goto 104
> list
  100 ' Welcome program for CoreBASIC.
  102 '
  104 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
  106 PRINT "For more information, visit http://www.soldercore.com/"
  108 '
  109 GOTO 104
  110 END
> renumber 50
> list
   50 ' Welcome program for CoreBASIC.
   60 '
   70 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
   80 PRINT "For more information, visit http://www.soldercore.com/"
   90 '
  100 GOTO 70
  110 END
> _
```

If no additional parameters are given, the program is renumbered starting from line 10 in steps of 10:

```
> renumber
> list
   10 ' Welcome program for CoreBASIC.
   20 '
   30 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
   40 PRINT "For more information, visit http://www.soldercore.com/"
   50 '
   60 GOTO 30
   70 END
> _
```

Programs with references to nonexistent line numbers will not be renumbered:

```
> 60 goto 22
> list
   10 ' Welcome program for CoreBASIC.
   20 '
   30 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
   40 PRINT "For more information, visit http://www.soldercore.com/"
   50 '
   60 GOTO 22
   70 END
> renumber
?no such line in 60: GOTO 22
> _
```

See also

CRUNCH

REPEAT ... UNTIL

Synopsis

REPEAT
statements
UNTIL *condition*

Repetitively execute statements.

Description

REPEAT executes *statements* until *condition* is true. *statements* are executed at least once; if this is a problem, consider using the WHILE statement. See **WHILE ... WEND**.

Example

Wait for a button press on pin 2 where signal is low when the button is pressed and high when it is released.

```
10 PIN 2 AS DIGITAL INPUT
20 PRINT "Press button on pin 2."
30 REPEAT
40   STATE = PIN(2)
50 UNTIL STATE = 0
60 PRINT "Thanks."
70 END
```

REPEAT$

Synopsis

REPEAT$(*n*, *str*)

Replicate a string or character.

Description

REPEAT$ constructs a new string containing *n* copies of the string *str*:

```
> print repeat$(10, "=")
==========
> print repeat$(3, "123")
123123123
> _
```

Repeating a string zero times, with *n* equal to zero, is an empty string:

```
> print "|"; repeat$(0, "x"); "|"
||
> _
```

If *n* is negative, CoreBASIC stops with an argument error:

```
> print repeat$(-1, "x")
?argument error
> _
```

See also

STRING$, SPC

REPORT

Synopsis

REPORT *expr*

Report error.

Description

REPORT prints the error message that corresponds to the error *expr*. If *expr* indicates no error, REPORT prints "OK".

Note

The statement REPORT *x* is identical to `PRINT REPORT(x)`.

See also

REPORT()

REPORT()

Synopsis

REP0RT *arg*

Decode error code.

Description

REP0RT decodes the error code *arg* into a printable string. You can use REP0RT to display errors that are returned from CoreOS or CoreBASIC.

See also

REPORT

RESTORE

Synopsis

RESTORE

RESTORE *line*

RESTORE RUN

Set data pointer.

Description

RESTORE on its own sets the data pointer to the current line. This is useful when entering a procedure to reset the data pointer to a position that is independent of line numbering. Usually you would use RESTORE in a procedure with DATA statements inside, or just following, the procedure.

RESTORE RUN sets the data pointer to the start of the program.

RESTORE *line* sets the data pointer to line number *line* and is not recommended for structured programs.

After setting the data pointer, subsequent READ statements will read data from the first DATA statement following the data pointer.

Example

Use RESTORE RUN to read data twice.

```
> list
  10 FOR N = 1 TO 2
  20   PRINT "Pass "; N; " reading data..."
  30   FOR I = 1 TO 2
  40     READ X
  50     PRINT "Read "; X
  60   NEXT I
  70   RESTORE RUN
  80 NEXT N
  90 END
 100 DATA 3.1415926, 2.7182818
> run
Pass 1 reading data...
Read 3.14159
Read 2.71828
Pass 2 reading data...
Read 3.14159
Read 2.71828
> _
```

Example

Use a RESTORE in a procedure to read local data.

```
> list
  10 CALL DATA2
  20 CALL DATA1
```

```
  30 END
  40 DEFPROC DATA1
  50   RESTORE
  60   CALL DUMP_DATA
  70   DATA 1, 2, 3, -99
  80 ENDPROC
  90 DEFPROC DATA2
 100   RESTORE
 110   CALL DUMP_DATA
 120   DATA 3, 2, 1, -99
 130 ENDPROC
 140 DEFPROC DUMP_DATA
 150   READ X
 160   WHILE X >= 0
 170     PRINT X;
 180     READ X
 190   WEND
 200   PRINT
 210 ENDPROC
> run
> _
```

RETURN

Synopsis

RETURN

Return control to caller.

Description

RETURN returns control to the statement following the GOSUB that called the subroutine. If there is no corresponding GOSUB the program halts with a "return without GOSUB" error.

REVERSE

Synopsis

REVERSE *var*

Reverse a string or array variable.

Description

REVERSE will reverse the order of characters in a string or the elements in the array *var*.

If *arg* is an array, REVERSE will reverse the array in place:

```
> v = gen(1 to 10)
> print v
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
> reverse v
> print v
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
> _
```

If *arg* is a string, REVERSE will reverse the string in place:

```
> v = "CISABeroC"
> print v
CISABeroC
> reverse v
> print v
CoreBASIC
> _
```

Note

You can use reverse as a function to reverse a string during evaluation. See **REVERSE()**.

REVERSE()

Synopsis

REVERSE *arg*

Reverse a string or array.

Description

If *arg* is an array, REVERSE will create a new array with the elements of *arg* in reverse order:

```
> v = gen(1 to 10)
> print v | reverse(v)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
> _
```

If *arg* is a string, REVERSE will create a new string with the characters in *arg* in reverse order:

```
> print reverse "CISABeroC"
CoreBASIC
> _
```

Note

You can use REVERSE to reverse the encoding order of a floating point or integer value created by MKF and MKI:

```
> x = mkf(1)              ' 1.0 is 3F'80'00'00
> print hex expand x
["3F", "80", "00", "00"]
> print hex reverse expand x
["00", "00", "80", "3F"]
> _
```

RGB

Synopsis

RGB(*r*, *g*, *b*)

Construct 24-bit true color.

Description

The three parameters *r*, *g*, and *b*, define the saturation of the red, green, and blue channels. These values should lie in the range 0 to 1 inclusive. Values outside this range are clipped as appropriate, rather than causing an error.

You can use RGB to set the drawing color:

```
COLOR RGB(1, 0, 0)    ' fully saturated red
COLOR RGB(0, 0.5, 0)  ' 50% (dull) green
```

See also

BLUE%, **GREEN%**, **RED%**

RIGHT

Synopsis

`RIGHT(`*arg*`, `*n*`)`

Slice right part of string or array.

Description

`RIGHT` extracts the last *n* characters of the string *arg*:

```
> print right("CoreBASIC", 5)
BASIC
> _
```

If *n* is negative, it specifies that the number of characters is computed from the start of the string. For instance, you can drop the first four characters from a string:

```
> print right("CoreBASIC", -4)
Core
> _
```

`RIGHT` also works on arrays in the same manner as strings:

```
> print right([5, 4, 3, 2, 1], 3)
[3, 2, 1]
> print right([5, 4, 3, 2, 1], -3)
[2, 1]
> _
```

RMDIR

Synopsis

`RMDIR` *path*

Remove folder.

Description

`RMDIR` removes the folder specified by *path*. `RMDIR` will not remove a folder unless it is empty. If there is an error removing the folder *path*, CoreBASIC throws an error.

See also

RMDIR()

RMDIR()

Synopsis

RMDIR *path*

Remove folder.

Description

RMDIR removes the folder specified by *path* and returns a status code indicating success or failure. RMDIR will not remove a folder unless it is empty.

```
> list
   10 E = RMDIR("/c/folder")
   20 IF E < 0 THEN REPORT E ELSE PRINT "Removed OK"
   30 END
> _
```

See also

RMDIR

RND

Synopsis

RND *arg*

Computes pseudo-random number.

Description

RND generates a pseudo-random number between zero and *arg*. Each time RND is executed, it picks the next random number in the sequence:

```
> print rnd 3 | rnd 3
1.77252
2.29368
> _
```

The argument may be negative, in which case the random number returned will also be negative:

```
> print rnd -5
-4.37283
> _
```

You can use RND with RGB to generate a random color:

```
> c = rgb(rnd 1, rnd 1, rnd 1)
> print hex c
C6C1F3
> _
```

If *arg* is an array, RND threads recursively over the elements of the array:

```
> print rnd [10, 10, 10, 10, 10]
[5.34515, 9.47601, 1.71722, 7.02209, 2.2641]
> _
```

You can use this facility, for instance, to generate a vector containing 10 rolls of a 6-sided die:

```
> print 1 + int rnd(6*con(10))
[2, 5, 3, 3, 6, 1, 2, 6, 5, 3]
> _
```

See also

RANDOMIZE

ROT

Synopsis

ROT *arg*

Convert to rotation matrix form.

Description

ROT converts *arg* to rotation matrix form. If *arg* is real or complex, ROT reduces *arg* to a unit complex and converts it to a 2×2 matrix:

```
> mat print rot cmplx(1, 2)
0.447214        -0.894427
0.894427        0.447214
> _
```

The zero complex is reduced to the identity matrix:

```
> mat print rot cmplx(0, 0)
1       0
0       1
> _
```

Using ROT together with CIS provides a way to generate a 2×2 rotation matrix for an angle. For example, the matrix for counter-clockwise rotation through 30 degrees is:

```
> mat print rot cis rad 30
0.866025 -0.5
0.5      0.866025
> _
```

If *arg* is a quaternion, ROT reduces *arg* to a unit quaternion and converts it to a 3×3 rotation matrix:

```
> mat print rot quat(1, 2, 3, 4)
-0.666667       0.133333        0.733333
0.666667        -0.333333       0.666667
0.333333        0.933333        0.133333
> _
```

The zero quaternion is reduced to the identity matrix:

```
> mat print rot quat(0, 0, 0, 0)
1       0       0
0       1       0
0       0       1
> _
```

ROW

Synopsis

ROW(*matrix*, *n*)

Extract matrix row.

Description

ROW extracts row *n* of the matrix *matrix*.

Example

```
> a = [[1, 2], [3, 4]]
> mat print a
1       2
3       4
> print row(a, 0) |  row(a, 1)
[1, 2]
[3, 4]
> _
```

See also

COL

RTRIM

Synopsis

RTRIM *arg*

Remove trailing whitespace.

Description

RTRIM removes all trailing whitespace from the string *arg*. If *arg* is an array of strings, a new array is created with each string having trailing whitespace removed:

```
> print "|"; rtrim("   CoreBASIC   "); "|";
|   CoreBASIC|
> _
```

If *arg* is an array, RTRIM threads recursively over the elements of the array:

```
> mat print "|" + rtrim ["  Core  ", "  BASIC  "] + "|"
|  Core|
|  BASIC|
> _
```

See also

LTRIM, TRIM

RUN

Synopsis

RUN

RUN *expr*, *expr*…

Executes program.

Description

RUN starts to run your stored program. All variables are cleared, TIMER is set to zero, and execution starts from the first program line.

```
> example "welcome"
Connecting to www.soldercore.com (192.232.216.121)...
Loading welcome.bas from network...
Program loaded and ready. Type RUN to execute.
> list
   10 ' Welcome program for CoreBASIC.
   20 '
   30 PRINT "Welcome to CoreBASIC on the "; CORE.NAME; "!"
   40 PRINT "For more information, visit http://www.soldercore.com/"
   50 '
   60 END
> run
Welcome to CoreBASIC on the SolderCore!
For more information, visit http://www.soldercore.com/
> _
```

You can provide additional arguments to your program after RUN. The list of values ofter RUN is collected and stored for future use using RUN as a function.

Auto-save feature

To provide extra protection from hardware and software crashes, you can enable auto-save mode with SAVE AUTO ON. With auto-save turned on, CoreBASIC saves your program automatically, as if by SAVE, before the program is run.

If your program cannot be auto-saved, CoreBASIC stops and does not attempt to continue and run your program.

Note

Pressing F5 at the CoreBASIC command line will execute RUN.

See also

RUN(), corebasic_save_auto.

RUN()

Synopsis

RUN

RUN(*index*)

Return program arguments.

Description

RUN used as a function provides access to the program parameters after the RUN statement. The list of values ofter RUN is collected and stored for future use:

```
> list
   10 PRINT "You have "; LEN RUN; " arguments."
   20 PRINT "Your program arguments are: "; RUN
   30 END
> run
You have 0 arguments.
Your program arguments are: []
> run 1/3
You have 1 arguments.
Your program arguments are: [0.33333]
> run sin(%pi/3), "CoreBASIC", gen(1 to 5)
You have 3 arguments.
Your program arguments are: [0.86603, "CoreBASIC", [1, 2, 3, 4, 5]]
> _
```

You can iterate over the program arguments using using FOR EACH and RUN:

```
> list
   10 IF LEN RUN = 0 THEN PRINT "No program arguments" : END
   20 PRINT "Your first argument is: "; RUN(0)
   30 PRINT "The remainder of your program arguments are:"
   40 FOR EACH A IN RUN(1 TO)
   50   PRINT A
   60 NEXT A
   70 END
> run
No program arguments
> run "CoreBASIC", "Core", "BASIC"
Your first argument is: CoreBASIC
The remainder of your program arguments are:
Core
BASIC
> _
```

Note

Variables are cleared before evaluating program arguments, so any variable used in a program argument will default to zero:

```
> list
   10 PRINT RUN
   20 END
```

```
> run "CoreBASIC"
["CoreBASIC"]
> x = "CoreBASIC"
> run x
[0]
> _
```

SAMPLE

Synopsis

SAMPLE(*property*, *n*)
SAMPLE(*property*; *property*; ... *property*, *n*)

Repeatedly sample properties.

Description

Creates an new array with *n* samples of the *property* parameter.

You can use this to sample an analog input:

```
> v = sample(core.a17, 100) ' take 100 samples of analog input 3
> _
```

Or one of the digital pins:

```
> v = sample(core.d2, 33)  ' take 33 samples of digital input 2
> _
```

Or multiple digital signals in parallel using a bus:

```
> install "parallel-bus" using core.d2, core.d3, core.d4 as bus
> v = sample(bus.input, 10)
> _
```

You don't need to know the input to sample in advance. You might want to ask the user about it:

```
> list
   10 INPUT "What analog channel to sample? "; C
   20 INPUT "How many samples? "; N
   30 S = SAMPLE(CORE.A(C), N)
   40 PRINT "Samples are: "
   50 MAT PRINT S
   60 END
> _
```

You can sample more than one item by separating the properties using semicolons. For instance, to sample 100 pairs of inputs form analog inputs 3 and 4 in parallel:

```
> v = sample(core.a17; core.a18, 100)
> _
```

The output array has the readings interleaved: the first sample from input 3, then input 4, then input 3, and so on.

SAMPLE can be very flexible. For instance, to acquire 50 samples form analog channels 3 and 4, as above, and to timestamp each sample:

```
> v = sample(core.tick; core.a17; core.a18, 50)
```

```
> _
```

SAVE

Synopsis

SAVE

SAVE *name*

Save program to storage device.

Description

SAVE saves the program in memory to a file on disk. If a file name follows SAVE, the current program name is set to *name* just as NAME would have set the current program name, and the program is saved.

With SAVE on its own, the program in memory is saved to the current program name.

Example

Saves `hello.bas` to the root folder of the SD card:

```
> save "/c/hello.bas"
> _
```

See also

NAME, LOAD

SAVE AUTO

Synopsis

SAVE AUTO [ON]
SAVE AUTO OFF

Turn auto-save on or off.

Description

SAVE AUTO enables or disables auto-save mode. Auto-save is a feature intended to provide extra protection when developing programs that deal with general hardware prototyping that may sometimes cause the SolderCore or CoreBASIC to become unresponsive.

With auto-save enabled by SAVE AUTO ON, your program is automatically saved to disk, as if by SAVE, just before the program is run. Your program is saved if you type RUN from the CoreBASIC command line or if you start running your program inside the CoreBASIC editor by pressing F5.

If your program cannot be auto-saved, CoreBASIC stops and does not attempt to continue and run your program.

By default, auto-save is turned off; you can turn auto-save on by adding SAVE AUTO ON to your boot file.

See also

SAVE, RUN

SECOND%

Synopsis

SECOND%
SECOND%(*arg*)

Return second within minute.

Description

SECOND%(*arg*) returns the current second within the minute for the time *arg*. *arg* is the number of seconds since 1 January 1970, the standard way of representing time in CoreBASIC. The result is a number from zero to 59.

SECOND% without an argument returns the current second within the minute for the *core time* and is equivalent to MINUTE%(CORE.TIME).

```
> list
   10 PRINT "The second is "; SECOND%; "."
   20 END
> run
The second is 45.
> _
```

SELECT

Synopsis

SELECT(*array*, *indexes*)

Select elements from array.

Description

SELECT will select elements from *array* using the indexes from the array *indexes*.

```
> v = ["Computers", "are", "useless", "They", "can", "only", "give", "you", "answers."]
> print select(v, [0, 6, 7, 2, 8])
["Computers", "give", "you", "useless", "answers."]
> _
```

If any of the indexes are beyond the dimensions of the source array, CoreBASIC throws an subscript error:

```
> v = ["Computers", "are", "useless", "They", "can", "only", "give", "you", "answers."]
> print select(v, [0, 9])
?subscript error
> _
```

See also

PICK

SHA1$

Synopsis

SHA1$ *arg*

Compute SHA-1 hash.

Description

SHA1$ computes the SHA-1 hash of the string *arg* according to FIPS 180-1. This example exchange replicates the computation of hashes found in FIPS 180-1:

```
> print merge hex expand sha1$ "abc"
A9993E364706816ABA3E25717850C26C9CD0D89D
> print merge hex expand sha1$ "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"
84983E441C3BD26EBAAE4AA1F95129E5E54670F1
> _
```

Notes

The hash is computed over the whole string in a single operation; currently there is no support for initialization, incremental update, and finalization of the hash computation.

See also

FIPS 180-1 - Secure Hash Standard

SGN

Synopsis

SGN *arg*

Compute signum.

Description

SGN computes the signum (or sign) of *arg*. The definition of SGN(x) is x/ABS(x) for nonzero *arg* and 0 for zero x.

Real numbers

```
> print sgn -3, sgn 0, sgn 3
-1      0       1
> _
```

Complex numbers

The signum of a complex number *arg* is a *unit complex* that is the point on the unit circle of the complex plane that is nearest to *arg*:

```
> print sgn(cmplx(1, 3))
0.316228+0.948683j
> print sgn(cmplx(0.1, 0.1))
0.707107+0.707107j
> _
```

The inverse of a unit complex number *x* is simply the complex conjugate of *x*:

```
> x = cmplx(3, 4)           ' arbitrary complex number
> print x | 1/x | cnj x
3+4j
0.12-0.16j
3-4j
> x = sgn x                 ' convert to unit complex
> print x | 1/x | cnj x
0.6+0.8j
0.6-0.8j
0.6-0.8j
> _
```

Quaternions

The signum of a quaternion *arg* works just the same as for complex numbers:

```
> x = quat(1, 2, 3, 4)      ' arbitrary quaternion
> print x | 1/x | cnj x
```

```
12i+3j+4k
0.0333333-0.066667i-0.1j-0.133333k
1-2i-3j-4k
> x = sgn x                ' convert to unit quaternion
> print x | 1/x | cnj x
0.182574+0.365148i+0.547723j+0.730297k
0.182574-0.365148i-0.547723j-0.730297k
0.182574-0.365148i-0.547723j-0.730297k
> _
```

See also

ABS

SHUFFLE

Synopsis

SHUFFLE *arr*

Randomly shuffle an array.

Description

The elements within the array *arr* are shuffled in random order:

```
> v = gen(1 to 10)
> print v | shuffle(v)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[7, 5, 2, 6, 1, 8, 3, 10, 4, 9]
> _
```

SHUFFLE will not shuffle a string, but by using EXPAND and MERGE with SHUFFLE you can achieve the same:

```
> print merge shuffle expand "CoreBASIC"
eCrAICoSB
> _
```

SIN

Synopsis

`SIN` *arg*

Compute circular sine.

Description

`SIN` computes the circular sine of *arg* in radian measure.

For real, complex, and quaternion types, `SIN` returns a number of the same type as its operand:

```
> print sin 10
-0.544021
> print sin cmplx(2, 10)
10014.3-4583.12j
> print sin quat(2, 3, 4, 5)
535.31-103.939i-138.585j-173.232k
> _
```

If *arg* is an array, `SIN` threads recursively over the elements of the array:

```
> print sin [2, 3, 4]
[0.909297, 0.14112, -0.756802]
> _
```

See also

ASN, **COS**, **TAN**

Graphing the sine function

Here is a graph of the sine function over the reals from $-\pi$ to π:

The figure above is generated by the following program:

```
***../examples/sine-function.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "sine-function"` or `|sine-function`.

SINH

Synopsis

`SINH` *arg*

Compute hyperbolic sine.

Description

Computes the hyperbolic sine of *arg*. For real, complex, and quaternion types, `SINH` returns a number of the same type as its operand:

```
> print sinh 1
1.1752
> print sinh cmplx(1, 2)
-1.9596+3.16578j
> print sinh quat(2, 3, 4, 5)
2.5582+1.13146i+1.50861j+1.88577k
> _
```

If *arg* is an array, `SINH` threads recursively over the elements of the array:

```
> print sinh [2, 3, 4]
[3.62686, 10.0179, 27.2899]
> _
```

See also

ASNH, **COSH**, **TANH**

SOCKET

Synopsis

SOCKET(*addr*, *port*)

Opens a socket.

Description

SOCKET opens a socket to the IP address specified in *addr* on port *port*. *addr* must be a string which is resolved using DNS. As such, *addr* can be a standard host name or a dotted decimal.

Example

```
10 REM Open a socket to the SolderCore website
20 S = SOCKET("www.soldercore.com", 80)
30 IF S < 0 THEN REPORT S : STOP
```

SORT

Synopsis

SORT *arr*

Sort array into ascending order.

Description

The elements of the array are sorted into ascending order.

```
> v = shuffle gen(1 to 10)
> print v | sort v
[1, 4, 8, 5, 6, 3, 10, 2, 7, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
> _
```

To sort the array into descending order, simply use REVERSE:

```
> print v | reverse sort v
[1, 4, 8, 5, 6, 3, 10, 2, 7, 9]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
> _
```

You can sort strings:

```
> print sort ["Matthew", "Bethany", "Andrew", "Ben"])
["Andrew", "Ben", "Bethany", "Matthew"]
> _
```

All values in *arr* must be of the compatible with each other. If *arr* contains elements that cannot be compared, e.g. a combination of strings and integers, execution stops with a type mismatch error.

```
> print sort [67, "A"]
?type mismatch
> _
```

You can normalize numbers and strings in an array using STR before sorting:

```
> print sort str [67, "A"]
["67", "A"]
> _
```

See also

REVERSE

SPI

Synopsis

SPI *device* [WRITE *data*, *data*...] [READ *n* TO *var*]

Issue transaction on SPI bus.

Description

SPI initiates a transaction on the SPI bus and waits for it to complete. The expression *device* is a device that supports SPI transfers and is typically a variable created by installing the SPI-DEVICE driver.

data is an expression to write to the SPI bus. This is typically an array of integers or strings. *data* is automatically merged, as if by MERGE, before being sent to the device.

CoreBASIC takes care of device selection and protocol setup: you do not need to select the device yourself. When SPI executes, CoreBASIC will select the device before use, execute and write and read, and then deselect the device.

See also

Reading an MPL115A1 pressure sensor using SPI

SPLIT

Synopsis

SPLIT(*arg*, *separators*)

Split string.

Description

SPLIT breaks the string *arg* into an array of strings, splitting at the character `separator`:

```
> print split("http://www.soldercore.com", ":")
["http", "//www.soldercore.com"]
> print split("HTTP/1.1 200 OK", " ")
["HTTP/1.1", "200", "OK"]
> _
```

You can specify more than one character in *separators* and SPLIT will split *arg* at any character in *separators*:

```
> print split("CoreBASIC is great. I love it!  Truly awesome...", ".!")
["CoreBASIC is great", " I love it", "  Truly awesome", "", "", ""]
> _
```

You can remove the empty strings created by adjacent separators in *arg* using PICK:

```
> s = split("CoreBASIC is great. I love it!  Truly awesome...", ".!")
> print s
["CoreBASIC is great", " I love it", "  Truly awesome", "", "", ""]
> s = pick(s, s <> "")
> print s
["CoreBASIC is great", " I love it", "  Truly awesome"]
> _
```

And then you can clean it up using TRIM:

```
> print trim s
["CoreBASIC is great", "I love it", "Truly awesome"]
> _
```

See also

JOIN, PICK, TRIM

SPC

Synopsis

SPC *n*

Construct string containing spaces.

Description

SPC constructs a string containing *n* space characters:

```
> print "|"; spc(5); "|"
|     |
> _
```

See also

REPEAT$**, **STRING$

SPOKEN$

Synopsis

SPOKEN$ *arg*

Convert string to spoken words.

Description

SPOKEN$ converts *arg* to a string where each character in *arg* is converted to an American English word for that character:

```
> print spoken$ "$"
dollar
> print spoken$ "z"
zed
> print spoken$ "z$"
zed dollar
> _
```

Space characters are left unconverted:

```
> print spoken$ "z $"
zed dollar
> _
```

SQR

Synopsis

SQR *arg*

Compute square root.

Description

SQR computes the square root of *arg*. For real, complex, and quaternion types, SQR tries to return a number of the same type as its operand:

```
> print sqr 10
3.16228
> print sqr cmplx(2, 10)
2.46962+2.0246j
> print sqr quat(2, 3, 4, 5)
1.99449+0.549487i+0.73265j+0.915812k
> _
```

An exception to this rule exists for negative real values. As a negative real value has no corresponding square root that is real, SQR will return a complex number:

```
> print sqr 10
3.16228
> print sqr -10
0+3.16228j
> _
```

If *arg* is an array, SQR threads recursively over the elements of the array:

```
> print sqr [2, 3, 4]
[1.41421, 1.73205, 2]
> _
```

See also

EXP, **LOG**

STEP

Synopsis

FOR *variable* = *first* TO *last* STEP *step*

Iterate a fixed number of times.

See **FOR ... NEXT**.

Synopsis

GEN (*start* TO *end* STEP *step*)

Generate arithmetic progression as an array.

See **GEN**.

STOP

Synopsis

ST0P

Unconditionally stop execution.

Description

CoreBASIC stops executing the program and returns to the command prompt.

STR

Synopsis

STR *arg*

Convert to string.

Description

STR converts *arg* to a string as it would print using PRINT:

```
> print "|"; str cmplx(1/3, 1/7); "|"
|0.333333+0.142857j|
> _
```

Strings convert to themselves:

```
> print str "CoreBASIC"
CoreBASIC
> _
```

If *arg* is an array, STR threads recursively over the elements of the array:

```
> print str [-0.5, sqr(-1)]
["-0.5", "0+1j"]
> _
```

STRING$

Synopsis

`STRING$(`*n*`,` *str*`)`
`STRING$(`*n*`,` *code*`)`

Replicate a string.

Description

STRING$ constructs a new string containing *n* copies of the first character in the string *str*:

```
> print string$(10, "=")
==========
> print string$(3, "ABC")
AAA
> _
```

Repeating a string zero times, with *n* equal to zero, is an empty string:

```
> print "|"; string$(0, "x"); "|"
||
> _
```

If *n* is negative or the string is, CoreBASIC stops with an argument error:

```
> print string$(-1, "x")
?argument error
> _
```

If *str* is empty, CoreBASIC stops with a dimension error:

```
> print string$(10, "")
?dimension error
> _
```

If the second argument is an integer, it is interpreted as the ASCII code of the character to repeat:

```
> print string$(10, 88)
XXXXXXXXXX
> _
```

See also

REPEAT$, SPC

SUBST$

Synopsis

SUBST$(*str*, *old*, *new*)

Substitute string.

Description

SUBST$ replaces all occurrences of *old* in *str* with *new*.

```
> print subst$(subst$("Misisipi", "s", "ss"), "p", "pp")
Mississippi
> _
```

If *old* is empty, the original string is returned whatever *new* contains:

```
> print subst$("CoreBASIC", "", "Something")
CoreBASIC
> _
```

Using an empty *new* string has the effect of deleting all occurrences of *old* from *str*:

```
> print subst$("Transitivity", "it", "")
Transivy
> _
```

See also

INSERT$, DELETE$

SUM

Synopsis

SUM *arg*

Compute sum over data.

Description

SUM computes the sum of *arg*. If *arg* is an array, each of the elements is summed:

```
> print sum [1, 2, 3]
6
> print sum [1, cmplx(2, 4)]
3+4j
> print sum gen(1 to 100)
5050
> _
```

SUM will concatenate string arrays:

```
> print sum ["Core", "BASIC"]
CoreBASIC
> _
```

Note

SUM(x) is equivalent to REDUCE(+, x).

See also

REDUCE

SYSTEM

Synopsis

SYSTEM *command*

Execute CoreOS command.

Description

SYSTEM executes the CoreOS command *command*.

Example

```
> system "dir"
```

The SYSTEM statement may seem cumbersome when compared to issuing a CoreOS command using the * command, but the SYSTEM statement lets you create commands on the fly and send them to CoreOS, for instance to eject a disk:

```
10 INPUT "Eject which volume? "; VOL
20 SYSTEM "eject " + VOL
30 END
```

TAB

Synopsis

TAB(*x*, *y*)

Construct string to position the cursor.

Description

TAB constructs a string that positions the cursor at column *x* of row *y* on the display. Both row and column number from zero, so the top left of the display is (0, 0).

The TAB function is most commonly used with the PRINT command to position the cursor on the display.

Example

The following code uses TAB to position the cursor to write directly to the display and emulate the view of The Matrix in the film of the same name.

```
***../examples/matrix-rain.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "matrix-rain" or |matrix-rain.

TAN

Synopsis

TAN *arg*

Compute circular tangent.

Description

TAN computes the circular tangent of *arg* in radian measure. For real, complex, and quaternion types, TAN returns a number of the same type as its operand:

```
> print tan 10
0.648361
> print tan cmplx(1, 2)
0.0338128+1.01479j
> print tan quat(2, 3, 4, 5)
-1.12712e-06+0.424264i+0.565686j+0.707107k
> _
```

If *arg* is an array, TAN threads recursively over the elements of the array:

```
> print tan [2, 3, 4]
[-2.18504, -0.142547, 1.15782]
> _
```

See also

ATN, **SIN**, **COS**

TANH

Synopsis

TANH *arg*

Compute hyperbolic tangent.

Description

Computes the hyperbolic tangent of *arg*. For real, complex, and quaternion types, TANH returns a number of the same type as its operand:

```
> print tanh 1
0.761594
> print tanh cmplx(1, 2)
-1.01479+0.0338128j
> print tanh quat(2, 3, 4, 5)
2.65366+1.09076i+1.45434j+1.81793k
> _
```

If *arg* is an array, TANH threads recursively over the elements of the array:

```
> print tanh [2, 3, 4]
[0.964028, 0.995055, 0.999329]
> _
```

See also

ATNH, SINH, COSH

THEN

Synopsis

THEN

THEN *statements*

True execution path for `IF`.

Description

See **IF ... THEN**.

TIME$

Synopsis

`TIME$`

`TIME$(`*arg*`)`

Return textual time.

Description

`TIME$(`*arg*`)` returns a string in `HH:MM:SS` format for the time *arg*. *arg* is the number of seconds since 1 January 1970, the standard way of representing time in CoreBASIC. If the argument *arg* is negative, `TIME$` returns `??:??:??`.

`TIME$` without an argument returns the time string for the current *core time* and is equivalent to `TIME$(CORE.TIME)`.

```
> list
   10 PRINT "The time according to SolderCore is "; TIME$; "."
   20 END
> run
The time according to SolderCore is 21:20:22.
> _
```

See also

DATE$

TIMER

Synopsis

`TIMER`

`TIMER` = *expression*

Return or assign tick counter.

Description

`TIMER` returns the current tick counter which is the number of milliseconds elapsed since CoreBASIC started or the program started to execute.

`TIMER` is reset to zero by `RUN` or when a program is edited, but you can set `TIMER` to zero or any value you want in a program by assigning to it:

```
> list
   10 PRINT "Time how long it takes to compute 10,000 sines..."
   20 TIMER = 0
   30 FOR I = 1 TO 10000
   40   K = SIN 0
   50 NEXT I
   60 T = TIMER  ' read timer immediately
   70 PRINT "Took "; T/1000; " seconds"
   80 END
> run
Time how long it takes to compute 10,000 sines...
Took 0.18 seconds
> _
```

Notes

CoreBASIC is designed to use standard SI units whenever possible, so `PAUSE` uses seconds for timing. The way that `TIMER` works using milliseconds breaks this rule and may well be surprising, but this concession makes porting programs from other BASIC dialects a little easier as they use a millisecond `TIMER`.

The counter's resolution is to 10 milliseconds, not the millisecond, so `TIMER` counting freely will return 0, 10, 20, and so on.

TO

Synopsis

FOR *variable* = *first* TO *last*

Iterate a fixed number of times.

See **FOR ... NEXT**.

Synopsis

GEN (*start* TO *end*)

Generate arithmetic progression as an array.

See **GEN**.

Synopsis

RECTANGLE x_0, y_0 TO x_1, y_1
FILL RECTANGLE x_0, y_0 TO x_1, y_1

Draw or fill a rectangle.

See **RECTANGLE**.

TRIM

Synopsis

`TRIM` *arg*

Remove leading and trailing whitespace.

Description

`TRIM` removes all leading and trailing whitespace from the string *arg*.

```
> print "|"; trim "   CoreBASIC   "; "|";
|CoreBASIC|
> _
```

If *arg* is an array, `TRIM` threads recursively over the elements of the array:

```
> mat print "|" + trim ["  Core  ", "  BASIC  "] + "|"
|Core|
|BASIC|
> _
```

See also

LTRIM, RTRIM

TRN

Synopsis

TRN *arg*

Transpose matrix.

Description

TRN evaluates the transpose of the two-dimensional matrix *arg*.

```
> a = [[1, 2, 3], [4, 5, 6]]
> mat print a
1       2       3
4       5       6
> mat print trn a
1       4
2       5
3       6
> _
```

CoreBASIC will halt with an argument error if *arg* is a jagged two-dimensional matrix:

```
> a = [[1, 2, 3], [4, 5]]
> mat print trn a
?dimension error
> _
```

CoreBASIC will also transpose matrices with string elements:

```
> a = [["Core", "BASIC"]]
> mat print a
Core    BASIC
> mat print trn a
Core
BASIC
> _
```

Note

TRN only transposes the matrix *arg*, even if *arg* has complex elements:

```
> a = [[1+2 * %i, 2 - %i, 3 + %i], [4, 5 - 3*%i, 6 + %i]]
> mat print a
1+2j    2-1j    3+1j
4       5-3j    6+1j
> mat print trn a
1+2j    4
2-1j    5-3j
3+1j    6+1j
> _
```

If you need to compute the conjugate transpose of *arg*, use CNJ and TRN together:

```
> mat print cnj trn a
1-2j    4
2+1j    5+3j
3-1j    6-1j
> _
```

See also

CNJ

TRUE

Synopsis

TRUE

Boolean true.

Description

TRUE is a synonym for 1 as Boolean values are represented as integers in CoreBASIC.

```
> print true | truth true
1
True
> _
```

See also

FALSE

TRUTH

Synopsis

TRUTH *arg*

Convert logical to string.

Description

TRUTH converts *arg* to a string; any nonzero integer is considered true:

```
> print truth 0 | truth 1
False
True
> _
```

For real arguments, *arg* is converted to an integer by truncation. After truncation to an integer, any nonzero value is considered true:

```
> print truth -0.5 | truth -1 | truth 1.5
False
True
True
> _
```

If *arg* is an array, TRUTH threads recursively over the elements of the array:

```
> print truth [-0.5, -1, 1.5]
["False", "True", "True"]
> _
```

See also

%FALSE, **%TRUE**

TRY

Synopsis

TRY *statement*

Execute statement catching.

Description

TRY tries to execute *statement* and, if *statement* would have resulted in an error causing CoreBASIC to stop, the error is recorded, execution of *statement* is abandoned, and control transfers to the statement following TRY *statement*.

The last recorded error is accessible using ERROR. If *statement* executed without error, ERROR is set to zero. If *statement* raised an error, ERROR will be negative. You can use REPORT to report the error or decode the specific error raised by *statement*.

A nice example of using TRY to catch errors is when processing data coming from a socket.

```
> list
   10 S = SOCKET("www.google.com", 80)
   20 IF S < 0 THEN REPORT S : END
   30 PRINT #S, "GET /index.html HTTP/1.1"
   40 PRINT #S, "Accept: text/plain"
   50 PRINT #S, "Host: www.google.com"
   60 PRINT #S
   70 REPEAT
   80   TRY INPUT #S, HEADER AS STR
   90   IF NOT ERROR THEN PRINT HEADER
  100 UNTIL ERROR OR ELSE HEADER = ""
  110 IF ERROR THEN PRINT "Error processing headers: "; REPORT ERROR
  120 TRY CLOSE S
  130 END
> run
HTTP/1.1 302 Found
Location: http://www.google.co.uk/
Cache-Control: private
Content-Type: text/html; charset=UTF-8
?
Date: Fri, 08 Jun 2012 11:07:57 GMT
Server: gws
Content-Length: 221
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

> _
```

The REPEAT loop attempts to read the socket and find the end of the returned headers, indicated by a blank line. If an error happens during processing the data on the socket using INPUT, it is caught by TRY.

Because execution of INPUT is abandoned on an error, you cannot be sure what HEADER will contain after an error-generating INPUT statement: it may be empty, or it may be partially filled with data read, or it could be

unchanged. However, as ERROR is set to a nonzero value if INPUT is abandoned, the loop exits if there is an error on INPUT or INPUT completes correctly and a blank line is found.

See also

ERROR, REPORT

UCASE

Synopsis

UCASE *arg*

Convert to upper case.

Description

UCASE converts the string *arg* to upper case.

```
> print ucase("www.CoreBASIC.com")
WWW.COREBASIC.COM
> _
```

If *arg* is an array, UCASE threads recursively over the elements of the array:

```
> mat print ucase ["Core", "Basic"]
CORE
BASIC
> _
```

See also

LCASE

UNLOCK

Synopsis

UNLOCK *arg*

Make file writable.

Description

UNLOCK makes the file with name *arg* writable by removing the read only attribute. When a file is locked and read only, it cannot be overwritten or removed. In order to remove or overwrite a read-only file, you must first make it writable using UNLOCK.

```
> example "welcome"
Connecting to www.soldercore.com (192.232.216.121)...
Loading welcome.bas from network...
Program loaded and ready. Type RUN to execute.
> save $work
> lock $work
> save $work
?read-only file
> kill $work
?read-only file
> unlock $work
> save $work
> _
```

See also

LOCK

UNTIL

Synopsis

`UNTIL` *expression*

End a repeat loop.

Description

See **REPEAT ... UNTIL**.

UTF

Synopsis

UTF *arg*

Convert code point to UTF-8.

Description

UTF returns a string containing the UTF-8 sequence for the Unicode code point *arg*.

```
> print hex expand utf 65
["41"]
> print utf 65
A
> print hex expand utf 960
["CF", "80"]
> print utf 960
#
> _
```

VAL

Synopsis

VAL *arg*

Convert to number.

Description

VAL converts its argument to a number. If *arg* is already a real number, complex number, or quaternion, it is returned unchanged.

```
> print val 3.25
3.25
> _
```

If *arg* is a string, VAL converts the string to a number:

```
> print val "3.25"
3.25
> print val "1.66666666"
1.66667
> print val "-1e3"
-1000
> _
```

If *arg* is not recognized as a valid number, VAL returns a NaN:

```
> print val "garbage"
nan
> _
```

Both leading and trailing whitespace are ignored during conversion, but embedded whitespace is significant and will cause a conversion failure:

```
> print val "    3.25  "
3.25
> print val "3. 25"
nan
> _
```

Numbers that are too big to represent convert to an infinity:

```
> print val "1e40"
inf
> _
```

Numbers that are too small convert to zero:

```
> print val "1e-40"
0
> _
```

If *arg* is an array, VAL threads recursively over the elements of the array:

```
> print val ["3.1415", 2.781828, "CoreBASIC"]
[3.1415, 2.78183, nan]
> _
```

VDU

Synopsis

VDU *arg*, *arg*…
VDU *#channel*, *arg*, *arg*…

Display characters.

Description

VDU evaluates and prints each *arg* to the display. If *arg* is a string, it is printed in its entirety; numbers are treated as ASCII codes and converted to strings before output.

If VDU is followed by a channel specification, output is written to that channel rather than to the current output stream.

```
> vdu "Core", 13, 10
Core
> _
```

In contrast to PRINT, no separator characters are inserted between arguments on the output and no newline is added at the end:

```
> print 0x43, 0x6f, 0x72, 0x65
67      111     114     101
> vdu 0x43, 0x6f, 0x72, 0x65
Core>
```

If *arg* is an array, VDU threads recursively over the elements of the array:

```
> vdu ["Core", 66, 65, [83, 73], 67], $crlf
CoreBASIC
> _
```

Note

VDU x, y, z is functionally equivalent to PRINT MERGE [x, y, z];.

See also

PRINT, MERGE().

VERSION

Synopsis

`VERSION`

Show CoreBASIC firmware version.

Description

`VERSION` displays the installed CoreBASIC firmware version.

```
> version
CoreBASIC firmware 0.9.12
> _
```

The firmware version number is also provided by the `CORE.VERSION` property so you can read the installed firmware version from your program.

See also

SolderCore CPU

WAIT

Synopsis

`WAIT` *expr*

Wait for condition.

Description

`WAIT` will continuously evaluate *expr* until it is true. For example, to wait until the signals D2 and D3 are low:

```
WAIT CORE.D2 = 0 AND CORE.D3 = 0
```

Note

`WAIT` *expr* is equivalent to `REPEAT UNTIL` *expr*.

WATCHDOG

Synopsis

`WATCHDOG REBOOT`

Enable watchdog.

See **WATCHDOG REBOOT**.

Synopsis

`WATCHDOG RESTORE`

Service watchdog.

See **WATCHDOG RESTORE**.

Synopsis

`WATCHDOG THROW`

Debug hardware watchdog.

See **WATCHDOG THROW**.

Synopsis

`WATCHDOG TIMER` = *expr*

Set watchdog timeout period.

See **WATCHDOG TIMER**.

Synopsis

`WATCHDOG`

Remaining watchdog period.

See **WATCHDOG()**.

WATCHDOG()

Synopsis

```
WATCHDOG
```

Remaining watchdog period.

Description

WATCHDOG, as an expression, returns the number of seconds remaining before the watchdog times out.

See also

WATCHDOG TIMER

WATCHDOG REBOOT

Synopsis

`WATCHDOG REBOOT`

Enable hardware watchdog.

Description

`WATCHDOG REBOOT` enables the hardware watchdog in protection mode. Once the watchdog is enabled, the CoreBASIC program must service the watchdog within the timeout period set by `WATCHDOG TIMER` or the microcontroller resets. To service the watchdog, you must execute `WATCHDOG RESTORE`.

Note

After the watchdog is enabled in protection mode, it remains enabled even if the program returns to the CoreBASIC prompt. This behavior is intentional and caters for programming mistakes that cause the application to halt and return to the CoreBASIC prompt. In this case, rather than sitting idle waiting for user input, the microcontroller resets and starts running again.

See also

WATCHDOG THROW, **WATCHDOG TIMER**, **WATCHDOG RESTORE**

WATCHDOG RESTORE

Synopsis

`WATCHDOG RESTORE`

Service watchdog.

Description

`WATCHDOG RESTORE` services the watchdog and resets the watchdog to expire after the period set by `WATCHDOG TIMER`.

Note

If the watchdog is not enabled, executing `WATCHDOG RESTORE` does nothing.

See also

WATCHDOG TIMER, **WATCHDOG REBOOT**, **WATCHDOG THROW**

WATCHDOG TIMER

Synopsis

`WATCHDOG TIMER` = *expr*

Set watchdog timeout period.

Description

`WATCHDOG TIMER` sets the expiry period of the watchdog to *expr* seconds. When the CoreBASIC program starts, the expiry period is set to ten seconds. You can lengthen or shorten this period at any point in your application by reprogramming the timeout using `WATCHDOG TIMER`.

When the watchdog is enabled but is not serviced by `WATCHDOG RESTORE` within *expr* seconds, it expires and:

- the microcontroller will reset if the watchdog is configured for protection mode (set by `WATCHDOG REBOOT`).
- the program will halt with a watchdog timeout error if the watchdog is configured for debug mode (set by `WATCHDOG THROW`).

See also

WATCHDOG RESTORE, **WATCHDOG REBOOT**, **WATCHDOG THROW**

WATCHDOG THROW

Synopsis

`WATCHDOG THROW`

Debug hardware watchdog.

Description

`WATCHDOG THROW` enables the hardware watchdog in debug mode. Once the watchdog is enabled, the CoreBASIC program must service the watchdog within the timeout period set by `WATCHDOG TIMER` or the program exits with a watchdog timeout error. To service the watchdog, you must execute `WATCHDOG RESTORE`.

Note

Using `WATCHDOG THROW` rather than `WATCHDOG REBOOT` means that you can test your program to ensure that it meets its deadlines without continual microcontroller resets. Once you are satisfied that your application works well and meets its deadlines, you can deploy it in the field, using `WATCHDOG REBOOT` to guard against unexpected lock-ups.

See also

WATCHDOG REBOOT, **WATCHDOG RESTORE**, **WATCHDOG TIMER**

WEB

Synopsis

WEB *expr*

Display Web content.

Description

WEB requests an HTML page from the web and displays it on the console. The *web formatter* is very primitive, but it color-codes links, bold and italic text, and headers, and tries to format HTML pages as best it can with appropriate text wrapping.

The network part of the web formatter will not process anything other than "200" return codes; it will not redirect to new content with "page moved" codes.

```
> web "retro.hackaday.com"
Connecting to retro.hackaday.com (74.53.119.195)...
Requesting / from network...

[image omitted]

       __ __           __              ___
      / // /__ _____/ /__  ___ _  / _ \___ ___ __
     / _  / _ `/ __/  '_/ / _ `/ / // / _ `/ // /
    /_//_/\_,_/\__/_/\_\  \_,_/ /____/\_,_/\_, /
    retro edition                          /___/

Now optimized for embedded devices!

About
Successes
…
```

Keys

At the `--More--` prompt, you can use the following keys:

- *Space*: Show the next page.
- *Return*: Show the next line.
- .: Continue without paging.
- *Q*: Quit.

Notes

If the web formatter doesn't work on your favorite site, that is indeed a shame. Don't think we're going to fix it. The main purpose of the web formatter is to show CoreBASIC online help, using a restricted set of HTML, by way of HELP.

See also

HELP, NEWS

WEND

Synopsis

`WEND`

End a while loop.

Description

See **WHILE ... WEND**.

Notes

You can write `END WHILE` as two separate words and CoreBASIC will change the two words to `WEND` automatically.

WHEN

Synopsis

`WHEN` *expr*`,` *expr*`,` *expr*…
`WHEN` *relation expr*…

Match list for CASE statement.

Description

See **CASE ... ENDCASE**.

WHILE ... WEND

Synopsis

`WHILE` *condition*
statements
`WEND`

Repetitively execute statements.

Description

Execute *statements* while *condition* is true. If *condition* is initially false, *statements* are not executed.

XOR

Synopsis

x XOR *y*

Logical exclusive-or.

Description

XOR computes the logical exclusive or of *x* and *y*. The result is true if either *x* or *y* is true, but not both:

```
> list
   10 FOR X = FALSE TO TRUE
   20   FOR Y = FALSE TO TRUE
   30     PRINT TRUTH X; " XOR "; TRUTH Y; " = "; TRUTH(X XOR Y)
   40   NEXT Y
   50 NEXT X
   60 END
> run
False XOR False = False
False XOR True = True
True XOR False = True
True XOR True = False
> _
```

Both *x* and *y* are converted to integers before applying XOR. After conversion to integers, any nonzero value is considered true.

```
> print 1.5 xor 0.2 | 0.1 xor 0.2
1
0
> _
```

Arrays are processed element-by-element and a new array is created containing the elementwise exclusive-or of each pair:

```
> print [0, 1, 0, 1] xor [0, 0, 1, 1]
[0, 1, 1, 0]
> _
```

YEAR%

Synopsis

YEAR%

YEAR%(*arg*)

Return month within year.

Description

YEAR%(*arg*) returns the current year for the time *arg*. *arg* is the number of seconds since 1 January 1970, the standard way of representing time in CoreBASIC.

YEAR% without an argument returns the current year for the *core time* and is equivalent to YEAR %(CORE.TIME).

```
> list
   10 PRINT "Year "; SPOKEN$ YEAR%; "."
   20 END
> run
Year two thousand and twelve.
> _
```

See also

DATE%, DAY%, MONTH%

ZER

Synopsis

ZER(*dimension*, ...)

Create zero matrix.

Description

ZER uses the dimension list in its argument to create a zero matrix. A zero matrix is a matrix where each of its elements is zero.

```
> print zer(3, 3)
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
> mat print zer(3, 3)
0       0       0
0       0       0
0       0       0
> mat print zer(1, 3)
0       0       0
> mat print zer(3, 1)
0
0
0
> print zer(3)
[0, 0, 0]
> print zer(2, 2, 2)
[[[0, 0], [0, 0]], [[0, 0], [0, 0]]]
>
```

See also

IDN, ZER

Miscellaneous information

Here is some other miscellaneous information about CoreBASIC.

Command line keystrokes

The keystrokes recognized at the CoreBASIC command line are:

Completion	
Enter	Send line to CoreBASIC.
Moving	
Left, Right	Move insertion point in the direction of the arrow.
Home or Ctrl+A	Move insertion point to the start of the line.
End or Ctrl+E	Move insertion point to the end of the line.
Editing	
Backspace or Ctrl+H	Delete character before the insertion point.
Delete or Ctrl+G	Delete character after the insertion point .
Ctrl+K	Delete from the insertion point to the end of the line.
Ctrl+U	Delete entire line.
Macros	
F1	Execute `HELP`. (Ignored by some terminal emulators.)
F2	Execute `SAVE`. (Ignored by some terminal emulators.)
F3	Execute `LOAD`. (Ignored by some terminal emulators.)
F4	Execute `EDIT`. (Ignored by some terminal emulators.)
F5	Execute `RUN`.
F6	Execute `CATALOG`.
F7	Execute `CHECK`.
F10	Execute `DEBUG`.
F11	Execute `PRINT PAPER 4; INK 15; $CLS; $ON` which resets the terminal colors, clears the screen, and turns on the cursor.
F12	Execute `LIST`.

During program execution, **Ctrl+C** will request a break-in and stop the program running.

Visual editor keystrokes

The keystrokes recognized in the full screen editor, invoked by `EDIT`, are:

Control	
Ctrl+D	Exit the editor to the CoreBASIC command prompt.
F4	As Ctrl+D. (Ignored by some terminal emulators.)
Moving	
Up, Down, Left, Right	Move cursor in the direction of the arrow.
Home	Move cursor to the start of the line.
End	Move cursor to the end of the line.
PgDown, PgUp	Move cursor down one page or up one page.
Editing	
Return	Split line at cursor.
Backspace	Delete character to the left of the cursor.
Delete or Ctrl+G	Delete character to the right of the cursor.
Ctrl+K	Delete from the cursor to the end of the line.
Ctrl+U	Delete from the cursor to the beginning of the line.
Ctrl+X	Delete the current line.
Utility	
Ctrl+L	Center listing around current line.
F2	Save program. (Ignored by some terminal emulators.)
F5	Execute program.
F7	Check program for syntax errors.
F10	Execute one line of program.

CoreBASIC Keyboard codes

CoreBASIC encodes standard keystrokes into ASCII codes with values above 128:

Key	ASCII Value	Hexadecimal value	Notes
Left	136	0x88	
Right	137	0x89	
Up	138	0x8a	
Down	139	0x8b	
Home	140	0x8c	
End	141	0x8d	
PgUp	142	0x8e	
PgDn	143	0x8f	
F1	145	0x91	Not available on network virtual terminal
F2	146	0x92	Not available on network virtual terminal
F3	147	0x93	Not available on network virtual terminal
F4	148	0x94	Not available on network virtual terminal
F5	149	0x95	
F6	150	0x96	
F7	151	0x97	
F8	152	0x98	
F9	153	0x99	
F10	154	0x9a	
F11	155	0x9b	
F12	156	0x9c	

There are some common keys too:

Key	ASCII Value	Hexadecimal value	Notes
Backspace	8	0x08	May depend upon terminal configuration
Tab	9	0x09	
Return	10	0x0a	
Delete	127	0x7f	May depend upon terminal configuration

Note

When connecting to the SolderCore using Telnet, the keyboard keys F1 through F4 are not passed through the network connection because the standard VT100 terminal assigns F1 through F4 to local operations.

However, the PS/2 keyboard driver will return ASCII codes for F1 through F4.

CoreBASIC Driver Reference

This is the online documentation set for **CoreBASIC**. CoreBASIC is a programming language for embedded microcontrollers. It's easy to use, it's powerful, and most of all it's *interactive*.

Drivers by function

This section describes the shield and device drivers that you can install to extend the capability of CoreBASIC. Some drivers are very simple, such as reading from a sensors, while others are highly complex such as controlling LCD devices. CoreBASIC relieves you of the complexity of configuring your devices and provides tested, working, and robust drivers.

ADCs and DACs

ADCs convert analog voltages to digital value, and DACs convert digital values to analog voltages.

Devices

LTC2309
MCP342x
MCP4725

Accelerometers

Accelerometers measure gravity and linear acceleration. See Accelerometers.

Devices

ADIS16400
ADXL345
ADXL362
BMA150
BMA250
MMA8451Q
MMA8491Q
KXP84
KXTF9
MPU-6000 and MPU-6050
LIS302DL
LIS331DL
LIS331HH
LIS3DSH
LIS3LV02DL
SCA3000
SMB380

Products

CoreMPU
CoreTilt

Gyroscopes

Gyroscopes measure rotation around one or more axes. See Gyroscopes.

Devices

ADIS16400
IMU-3000
ITG-3200
MPU-6000 and MPU-6050
MPU-9150

Products

CoreGyro
CoreMPU
IMU-3000 Combo
MPU-6050EVB
MPU-9150EVB

Magnetometers

Magnetometers measure the earth's magnetic field. See Magnetometers.

Devices

ADIS16400
AK8975
BMM150
HMC5843
HMC5883L
HMC6343
HMC6352
MAG3110

Products

CoreMag
CoreMPU

Inertial measurement units

Inertial measurement units (IMUs) combine accelerometers, gyroscopes, and magnetometers so you know how you're moving. See **Inertial measurement units**.

Algorithms

AHRS

Devices

ADIS16400
MPU-6000
MPU-6050
MPU-9150

Products

ATAVRSBIN1
ATAVRSBIN2
CoreMPU
IMU-3000 Combo
MPU-6050EVB
MPU-9150EVB

Parallel Buses

Parallel buses are a common way to hook up digital integrated circuits. See **Parallel buses**.

Drivers

Parallel Bus

Devices

MCP23x08
MCP23016
MCP23017
PCF8575

Humidity Sensors

Humidity sensors measure the moisture content of the environment.

Devices

HIH6130
SHT1x
SHT2x
Si7005
DHT11, DHT21, DHT22, RHT03

Temperature Sensors

Temperature sensors measure temperature, but how they sense it varies between sensors. See **Temperature sensors**.

Devices

ADT7410
LM75
DHT11, DHT21, DHT22, RHT03
HIH6130
SHT1x
SHT2x
Si7005
TC77
TMP100
TMP102

Products

CoreTemp

Die temperature sensor

BMP085
MAG3110
MPU-6000 and **MPU-6050**

Pressure Sensors

Pressure sensors measure barometric pressure which can translate into altitude. See **Pressure sensors**.

Devices

BMP085
MPL115A1
MPL115A2
MPL3115A2
LPS331AP

Light Sensors

Light sensors measure light level; and some sense color. See **Light sensors**.

Devices

TSL2561
ISL29023

Graphic Displays

Graphic displays come in many forms: VGA output, LED arrays, and LCD displays.

VGA displays

SolderCore Arcade Shield

LED displays

Jimmie Rodgers LoL Shield
ITead Studio Colors Shield

LCD displays

Adafruit TFT Touch Shield
AirSensor 128GLCD
AirSensor 192GLCD
ITead Studio ITDB02-2.2 LCD Module
ITead Studio ITDB02-2.4D LCD Module
ITead Studio ITDB02-2.4E LCD Module
ITead Studio ITDB02-2.8 LCD Module
ITead Studio ITDB02-3.2S LCD Module
ITead Studio ITDB02-3.2WD LCD Module
ITead Studio ITDB02-4.3 LCD Module
ITead Studio ITDB02-5.0 LCD Module
NuElectronics 3310 LCD Shield
NuElectronics TFT LCD Shield
Seeed Studio TFT Touch Shield
SolderCore LCD Shield
SparkFun Color LCD Shield
Watterott electronic mSD Shield
Watterott electronic S65 Shield

OLED displays

Seeed Studio 96x16 OLED Brick
Seeed Studio 96x96 OLED Twig
Seeed Studio 128x64 OLED Twig

Accelerometers

Properties

All accelerometers implement a standard set of properties for reading acceleration in up to three axes. The properties are:

Acceleration		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		
`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.

Drivers

CoreBASIC is delivered with the following accelerometer drivers:

- Analog Devices ADIS16400 Driver
- Analog Devices ADXL345 Driver
- Analog Devices ADXL362 Driver
- Bosch Sensortec BMA150 Driver
- Bosch Sensortec BMA250 Driver
- Bosch Sensortec SMB380 Driver
- InvenSense MPU-6000 Driver
- InvenSense MPU-6050 Driver
- Kionix KXP84 Driver
- Kionix KXTF9 Driver
- SolderCore CoreMPU Driver
- STMicroelectronics LIS302DL Driver
- STMicroelectronics LIS331DLH Driver
- STMicroelectronics LIS331HH Driver
- STMicroelectronics LIS3DSH Driver
- STMicroelectronics LIS3LV02DL Driver
- VTI SCA3000 Driver

Example

When you receive a new board with a gyroscope on it, usually it is marked with the axis orientation. However, some boards aren't, and tracking down the documentation might be a bit difficult. So, to figure out what the axis orientation is, you can use this simple program. Do what it says in the comments and it will relieve a great deal of stress.

```
***../examples/coretilt-confidence-test.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE `"coretilt-confidence-test"` or `|coretilt-confidence-test`.

Gyroscopes

Properties

All gyroscopes implement a standard set of properties for reading rotation rate in up to three axes. The properties are:

Angular rate		
G	Analog Read	An array of three numbers containing the rotation rates measured around the *x*, *y*, and *z* axes, in degrees per second.
GX *or* X	Analog Read	Rotation rate around the *x* axis, in degrees per second.
GY *or* Y	Analog Read	Rotation rate around the *y* axis, in degrees per second.
GZ *or* Z	Analog Read	Rotation rate around the *z* axis, in degrees per second.
Configuration		
RANGE	Analog R/W	Selected full scale range of the gyroscope, in degrees per second.
BANDWIDTH	Analog R/W	Selected bandwidth of the of the gyroscope, in hertz.

When RANGE is written with *x*, the gyroscope's range is set to the nearest range that handles at most *x* dps. If *x* exceeds the maximum full scale range of the gyroscope, the gyroscope is set to the highest range.

For instance, assume a gyroscope offers ranges in degrees per second of 250, 500, 1000, and 2000. When RANGE is written as 350, the gyroscope driver will select the 500 dps range as this is the best range to use for that particular rate. When you read RANGE back, you will not read 350 because the gyroscope driver selected the 500 dps range, so you will instead read 500.

Now assume you write RANGE with 3000. The gyroscope can't deliver a full scale range of 3000 dps, so it selects the highest range possible, which is 2000 dps in this case. When you read RANGE now, you will read 2000 dps.

Drivers

CoreBASIC is delivered with the following gyroscope drivers:

Analog Devices ADIS16400 Driver
InvenSense IMU-3000 Driver
InvenSense ITG-3200 Driver
InvenSense MPU-6000 Driver

InvenSense MPU-6050 Driver

SolderCore CoreMPU Driver

Example

When you receive a new board with a gyroscope on it, usually it is marked with the axis orientation. However, some boards aren't, and tracking down the documentation might be a bit difficult. So, to figure out what the axis orientation is, you can use this simple program. Do what it says in the comments and it will relieve a great deal of stress.

```
***../examples/coregyro-confidence-test.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE `"coregyro-confidence-test"` or `|coregyro-confidence-test`.

Magnetometers

Properties

All magnetometers implement a standard set of properties for reading magnetic field in up to three axes. The properties are:

Magnetic field		
`M`	Analog Read	An array of three numbers containing the magnetic field measured in the *x*, *y*, and *z* axes, in microtesla.
`MX` *or* `X`	Analog Read	Magnetic field measured in the *x* direction, in microtesla.
`MY` *or* `Y`	Analog Read	Magnetic field measured in the *y* direction, in microtesla.
`MZ` *or* `Z`	Analog Read	Magnetic field measured in the *z* direction, in microtesla.
Configuration		
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the magnetometer, in hertz.

Drivers

CoreBASIC is delivered with the following magnetometer drivers:

Analog Devices ADIS16400 Driver
Asahi Kasei AK8975 Driver
Bosch Sensortec BMM150 Driver
Freescale MAG3110 Driver
Honeywell HMC5843 Driver
Honeywell HMC5883L Driver
Honeywell HMC6343 Driver
Honeywell HMC6352 Driver
SolderCore CoreMPU Driver

Inertial measurement units

Inertial measurement units (IMUs)
AHRS Driver
Analog Devices ADIS16400 Driver
Atmel ATAVRSBIN1 Driver
Atmel ATAVRSBIN2 Driver
InvenSense MPU-6000 Driver
InvenSense MPU-6050 Driver
SolderCore CoreMPU Driver
SparkFun IMU-3000 Combo

Parallel buses

Temperature sensors

Properties

All temperature sensors, whether they measure ambient temperature, die temperature, or some other temperature, implement a property to return the temperature in degrees Celsius:

Sensor		
TEMP	Analog Read	Temperature, in degrees Celsius.

Drivers

CoreBASIC is delivered with the following temperature sensor drivers:

Analog Devices ADT7410 Driver
Maxim MAX6675 Driver
Microchip TC77 Driver
National Semiconductor LM75 Driver
Texas Instruments TMP100 Driver
Texas Instruments TMP102 Driver

Die temperatuire

Some devices can sense their own temperature, commonly called the die temperature:

Bosch Sensortec BMP085 Driver
Freescale MAG3110 Driver
InvenSense MPU-6000 Driver
InvenSense MPU-6050 Driver
SolderCore CoreMPU Driver

Pressure sensors

Pressure sensors
Bosch Sensortec BMP085 Driver
STMicroelectronics LPS331AP
Freescale MPL115A1 Driver
Freescale MPL115A2 Driver
Freescale MPL3115A2 Driver

Light sensors

Light sensors
AMS TSL2561 Driver
Intersil ISL29023 Driver

Graphic displays

VGA displays

- SolderCore Arcade Shield
- SolderCore Graphics Shield

LED displays

- Jimmie Rodgers LoL Shield
- ITead Studio Colors Shield

LCD displays

- Adafruit TFT Touch Shield
- AirSensor 128GLCD
- AirSensor 192GLCD
- ITead Studio ITDB02-2.2 LCD Module
- ITead Studio ITDB02-2.4D LCD Module
- ITead Studio ITDB02-2.4E LCD Module
- ITead Studio ITDB02-2.8 LCD Module
- ITead Studio ITDB02-3.2S LCD Module
- ITead Studio ITDB02-3.2WD LCD Module
- ITead Studio ITDB02-4.3 LCD Module
- ITead Studio ITDB02-5.0 LCD Module
- NuElectronics 3310 LCD Shield
- NuElectronics TFT LCD Shield
- Seeed Studio TFT Touch Shield
- SolderCore LCD Shield
- SparkFun Color LCD Shield
- Watterott electronic mSD Shield
- Watterott electronic S65 Shield

OLED displays

- Seeed Studio 96x16 OLED Brick
- Seeed Studio 96x96 OLED Twig
- Seeed Studio 128x64 OLED Twig

Character displays

Character display drivers — LCD, LED, and e-Paper
Hitachi HD44780 Driver
Jee Labs LCD Plug
SparkFun e-Paper Breakout

Joysticks and joypads

Single sticks

All analog joystick implement a standard set of properties for reading joystick position. Joypads emulate a digital joystick with up to nine directional settings.

For hardware that has a single joystick, the following properties are implemented:

Position		
H	Analog Read	Sense horizontal joystick position; −1 is fully left, +1 is fully right, and 0 is centered.
V	Analog Read	Sense vertical joystick position; −1 is fully down, +1 is fully up, and 0 is centered.
POS	Analog Read	A complex number where the real component is the horizontal position of the joystick and the imaginary component is the vertical position of the joystick.
Selection		
PRESS	Digital Read	If the joystick can be pressed (not all can), this property reads `1` when the joystick is pressed and `0` when it is not.

Multiple sticks

For hardware that has multiple joysticks, or both joysticks and joypads (such as the SparkFun Joystick Shield with has both), each property can be indexed to select a particular joystick or joypad:

Position		
H(n)	Analog Read	Sense horizontal joystick position *n*; −1 is fully left, +1 is fully right, and 0 is centered.
V(n)	Analog Read	Sense vertical joystick position *n*; −1 is fully down, +1 is fully up, and 0 is centered.
POS(n)	Analog Read	A complex number where the real component is the horizontal position of joystick *n* and the imaginary component is the vertical position of joystick *n*.
Selection		

PRESS(n)	Digital Read	If joystick *n* can be pressed (not all can), this property reads `1` when the joystick is pressed and `0` when it is not. If the joystick cannot be pressed, PRESS always reads as `0`.

Drivers by vendor

This section is a complete reference to the drivers delivered in CoreBASIC.

CPUs	CoreBASIC	Adafruit	AirSensor
SolderCore CPU Freedom Board CPU Raspberry Pi CPU	AHRS Driver ANSI Graphics Driver Matrix Keyboard Driver NMEA Parser Parallel Bus Driver Software I2C Bus Driver Software SPI Bus Driver SolderCore Network System UART Driver Xterm Graphics Driver	TFT Touch Shield	128GLCD 192GLCD
AMS	Analog Devices	Asahi Kasei	Atmel
TSL2561	ADIS16400 ADT7410 ADXL345 ADXL362	AK8975	ATAVRSBIN1 ATAVRSBIN2
Bosch Sensortec	Freescale	Gravitech	Hitachi
BMA150 BMA250 BMM150 BMP085 SMB380	MAG3110 MMA8451Q MMA8491Q MPL115A1 MPL115A2 MPL3115A2	7-Segment Shield	HD44780

Honeywell HIH6130 HMC5843 HMC5883L HMC6343 HMC6352 HMC5883L + MPU-6050	Intersil ISL29023	InvenSense IMU-3000 ITG-3200 MPU-6000 MPU-6050 MPU-6050 + HMC5883L MPU-6050 + AK8975 MPU-9150	ITead Studio Colors Shield ITDB02-2.2 LCD Module ITDB02-2.4D LCD Module ITDB02-2.4E LCD Module ITDB02-2.8 LCD Module ITDB02-3.2S LCD Module ITDB02-3.2WD LCD Module ITDB02-4.3 LCD Module ITDB02-5.0 LCD Module
Jee Labs LCD Plug Pressure Plug RTC Plug	Jimmie Rodgers LoL Shield	Kionix KXP84 KXTF9	Linear Technology LTC6904
Liquidware Input Shield	MaxDetect DHT11, DHT21, DHT22, RHT03	Maxim DS1340 MAX6675	Microchip MCP23008 MCP23016 MCP23017 MCP342x MCP4725 TC77
Modkit MotoProto Shield	National Semiconductor LM75	Nintendo Classic Controller Nunchuk Controller	NuElectronics 3310 LCD Shield TFT LCD Shield
NXP PCF8575	Seeed Studio 96x16 OLED Brick 96x96 OLED Twig 128x64 OLED Twig TFT Touch Shield	Sensirion SHT1x SHT2x	Silicon Labs Si7005

SolderCore	SparkFun	STMicroelectronics	Texas Instruments
Arcade Shield	ArduMoto Shield	LIS302DL	TMP100
CoreLight Module	Color LCD Shield	LIS331DLH	TMP102
CoreMPU Module	El Escudo	LIS331HH	
CorePressure Module	e-Paper Breakout	LIS3DSH	
CoreSpin Module	IMU-3000 Combo	LIS3LV02DL	
CoreTemp Module	Joystick Shield	LPS331AP	
CoreTilt Module	MIDI Shield	LSM303DLH	
Graphics Shield	OLED Carrier		
LCD Shield	RingCoder Breakout		
Motor Shield	Spectrum Shield		
SenseCore	Touch Shield		
Servo Shield	VoiceBox Shield		
VTI	Watterott electronic		
SCA3000	mSD Shield		
	S65 Shield		

Adafruit TFT Touch Shield

Installation

```
INSTALL "ADAFRUIT-TFT-TOUCH-SHIELD"
INSTALL "TFT-TOUCH-SHIELD"
```

Options

None.

Description

Installing the TFT Touch Shield driver provides a 240×320 true color graphic display. You can use all the CoreBASIC graphics commands to drive the LCD display.

Resources

http://www.adafruit.com/products/376

Benchmarks

Here is the result of running the SolderCore Graphics Benchmarks application:

```
> run
Graphics display benchmark for ADAFRUIT-TFT-TOUCH-SHIELD

   Circles:    2397 ms
     Discs:   26380 ms
Rectangles:     710 ms
     Slabs:   24416 ms
     Lines:    6689 ms
  Polygons:    7656 ms
      Text:    2131 ms
> _
```

AHRS Driver

Installation

`INSTALL "AHRS" USING` *sensor*`,` *sensor*...

Options

None.

Description

Installs a driver that fuses accelerometers, gyroscopes, and optional magnetometers to provide a complete attitude and heading reference system (AHRS).

To create an AHRS successfully, the AHRS driver must be provided with a 3-axis accelerometer and 3-axis gyroscope instance in the `USING` list. To update heading correctly, a magnetometer is also required in the `USING` list.

To create a fully functioning AHRS from an ADXL345, an ITG-3200, and an HMC5883L, you would use:

```
INSTALL "ADXL345" AS TILT
INSTALL "ITG-3200" AS SPIN
INSTALL "HMC5883L" AS COMPASS
INSTALL "AHRS" USING TILT, SPIN, COMPASS AS AHRS
```

It doesn't matter which order you specify the sensors in, the AHRS driver will search the `USING` list to determine the type of sensor and capabilities.

Some sensors, such as the MPU-6050, provide an accelerometer and gyroscope in the same package; in this case, you only need provide the MPU-6050 instance to the AHRS driver, like this:

```
INSTALL "MPU-6050" AS MPU
INSTALL "AHRS" USING MPU AS AHRS
```

That's it! This is all you need to do to have a fully functioning IMU using the MPU-6050, except that heading (yaw) drift is not compensated. In order to compensate for gyroscope drift in yaw, a magnetometer needs to be provided. If you have, for instance, an HMC5843L magnetometer, you can create a full AHRS with long-term yaw drift correction using:

```
INSTALL "MPU-6050" AS MPU
INSTALL "HMC5843" AS COMPASS
INSTALL "AHRS" USING MPU, COMPASS AS AHRS
```

Some drivers, such as the CoreMPU and ADIS16400, provide an accelerometer, gyroscope, and magnetometer in the same package or board, and then it's even simpler to get the AHRS up and running:

```
INSTALL "CORE-MPU" AS MPU
INSTALL "AHRS" USING MPU AS AHRS
```

Properties

Algorithm results		
Q	Analog Read	A unit quaternion (an "attitude quaternion") representing the current orientation relative to the world frame.
PITCH	Analog Read	Pitch angle θ about the *y* axis, in degrees; derived from the quaternion Q.
ROLL	Analog Read	Roll angle φ about the *x* axis, in degrees; derived from the quaternion Q.
YAW	Analog Read	Yaw angle ψ about the *z* axis, in degrees; derived from the quaternion Q.

Example

```
***../examples/ahrs-orientation-demo.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "ahrs-orientation-demo"` or `|ahrs-orientation-demo`.

Notes

See **Accelerometers**, **Gyroscopes**, and **Magnetometers**.

AirSensor 128GLCD

Installation

```
INSTALL "AIRSENSOR-128GLCD"
```

Options

ADDR=*integer*

Set the 8-bit I2C address of the port expander. By default the driver uses the address `0x40`.

Description

Installing the AirSensor 128GLCD driver provides a 128×64 monochrome graphic display. You can use all the CoreBASIC graphics commands to drive the LCD display.

Properties

Backlight		
`LIGHT`	Digital Write	Backlight control. Writing `0` to `LIGHT` turns the backlight off, if the display has a controllable backlight, and writing `1` turns it on.
Orientation		
`ROTATION`	Digital Write	Sets the rotation angle of the display, in units of 90 degrees. Rotation 0 is native orientation, rotation 1 is 90 degree rotation, 2 is 180 degrees, and 3 is 270 degrees.

See also

AirSensor 192GLCD

AirSensor 192GLCD

Installation

```
INSTALL "AIRSENSOR-192GLCD"
```

Options

ADDR=*integer*

Set the 8-bit I2C address of the port expander. By default the driver uses the address `0x40`.

Description

Installing the AirSensor 192GLCD driver provides a 192×64 monochrome graphic display. You can use all the CoreBASIC graphics commands to drive the LCD display.

Properties

Backlight		
`LIGHT`	Digital Write	Backlight control. Writing `0` to `LIGHT` turns the backlight off, if the display has a controllable backlight, and writing `1` turns it on.
Orientation		
`ROTATION`	Digital Write	Sets the rotation angle of the display, in units of 90 degrees. Rotation 0 is native orientation, rotation 1 is 90 degree rotation, 2 is 180 degrees, and 3 is 270 degrees.

See also

AirSensor 128GLCD

AMS TSL2561 Driver

Installation

```
INSTALL "AMS-TSL2561"
INSTALL "TSL2561"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x72`. The TSL2561 can be configured to use addresses `0x52` and `0x92`.

Description

Installs a light sensor driver for the TSL2561.

Properties

Light sensor		
`LIGHT`	Analog Read	Reads the current ambient light level, in lux.
Configuration		
`GAIN`	Digital R/W	Reads and writes the internal GAIN bit. Setting `GAIN` to 16 selects 16× high gain mode and setting `GAIN` to 1 selects 1× mode.
General		
`VERSION`	String Read	A string containing the detected TSL2561 device and its silicon revision.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Analog Devices ADIS16400 Driver

Installation

```
INSTALL "ANALOG-DEVICES-ADIS16400" USING select
INSTALL "ADIS16400" USING select
```

Options

None.

Description

Installs a driver for the Analog Devices ADIS16400 iSensor. The ADIS16400 is a highly accurate Inertial Measurement Unit, or IMU. This driver provides an accelerometer, gyroscope, and magnetometer that can be fused together using the AHRS for an attitude and heading reference system. See **AHRS Driver** for more information on putting this device to work.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ	Analog Read	Acceleration measured along the *z* axis, in *g*.
Gyroscope		
G	Analog Read	An array of three numbers containing the rotation rates measured around the *x*, *y*, and *z* axes, in degrees per second.
GX	Analog Read	Rotation rate around the *x* axis, in degrees per second.
GY	Analog Read	Rotation rate around the *y* axis, in degrees per second.
GZ	Analog Read	Rotation rate around the *z* axis, in degrees per second.
Magnetometer		

`M`	Analog Read	An array of three numbers containing the magnetic field measured in the *x*, *y*, and *z* axes, in microtesla.
`MX`	Analog Read	Magnetic field measured in the *x* direction, in microtesla.
`MY`	Analog Read	Magnetic field measured in the *y* direction, in microtesla.
`MZ`	Analog Read	Magnetic field measured in the *z* direction, in microtesla.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See **Accelerometers**, **Gyroscopes**, and **Magnetometers**.

Analog Devices ADT7410 Driver

Installation

```
INSTALL "ANALOG-DEVICES-ADT7410"
INSTALL "ADT7410"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x94`.

Description

Installs a temperature sensor driver for the ADT7410.

Properties

Measurement		
TEMP	Analog Read	Temperature in degrees Celsius.
Configuration		
RESOLUTION	Analog R/W	The selected resolution of the temperature sensor. The ADT7410 supports resolutions of 0.0625 (default) and 0.0078125 degrees Celsius. Writing this property will configure the sensor to deliver measurements to one of those resolutions.
Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
WORD(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

Analog Devices ADXL345 Driver

Installation

```
INSTALL "ANALOG-DEVICES-ADXL345"
INSTALL "ADXL345"
INSTALL "CORE-TILT"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0xA6`; to configure the primary address, use `ADDR=0x3A`.

Description

Installs an accelerometer driver using the ADXL345 on the CoreTilt SExI module which mounts into a SenseCore and initializes the accelerometer to the 2*g* range:

http://www.soldercore.com/products/sensecore/coretilt/

Although this is intended to fit into an SenseCore, you can use it as a standard breakout module and mount it wherever you wish or use it with something other than a SolderCore.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
ROLL	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.

PITCH	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		
RANGE	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer GAIN and BIAS to the defaults for the selected range.
BANDWIDTH	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
BIAS	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
GAIN	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
WORD(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See **Accelerometers**.

References

The SolderCore CoreTilt SExI module:

http://www.soldercore.com/products/sensecore/coretilt/

SparkFun offer an ADXL345 breakout:

http://www.sparkfun.com/products/9836

Analog Devices ADXL362 Driver

Installation

```
INSTALL "ANALOG-DEVICES-ADXL362" USING select
INSTALL "ADXL362" USING select
```

Options

None.

Description

Installs an accelerometer driver using the ADXL362. The `USING` clause specifies the device select signal to select the ADXL362.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		
`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.

`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See **Accelerometers**.

References

SparkFun offer an ADXL362 breakout:

http://www.sparkfun.com/products/11446

ANSI Graphics Driver

Installation

```
INSTALL "ANSI-GRAPHICS"
```

Options

None.

Description

Installing the ANSI Graphics driver provides you with a graphics display that is emulated on the terminal. The 24-bit color space is mapped to the eight standard colors of an ANSI terminal. After installation, you can use all the CoreBASIC graphics commands to write to the display.

Because the graphics are emulated, this driver makes it possible to run every standard graphics demonstration without graphics hardware.

Note

If you use a graphics command without installing a graphics driver, CoreBASIC automatically installs and initializes an `ANSI-GRAPHICS` driver.

See also

Xterm Graphics Driver

Asahi Kasei AK8975 Driver

Installation

```
INSTALL "ASAHI-KASEI-AK8975"
INSTALL "AK8975"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x18`.

Description

Installs a magnetometer driver for the AK8975.

Properties

Magnetometer		
`M`	Analog Read	An array of three numbers containing the magnetic field measured in the *x*, *y*, and *z* axes, in microtesla.
`MX` *or* X	Analog Read	Magnetic field measured in the *x* direction, in microtesla.
`MY` *or* Y	Analog Read	Magnetic field measured in the *y* direction, in microtesla.
`MZ` *or* Z	Analog Read	Magnetic field measured in the *z* direction, in microtesla.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See Magnetometers.

Atmel ATAVRSBIN1 Driver

Installation

```
INSTALL "ATMEL-ATAVRSBIN1"
INSTALL "ATAVRSBIN1"
```

Options

None.

Description

Installs a combination gyroscope, accelerometer, and magnetometer driver for the ATAVRSBIN1 module. The ATAVRSBIN1 integrates a Bosch Sensortec BMA150 accelerometer, an InvenSense ITG-3200 gyroscope, and a Asahi Kasei AK8975 magnetometer to provide a 9DOF sensor platform.

When these sensors are fused by the AHRS driver, you have excellent 9DOF attitude and heading reference system.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ	Analog Read	Acceleration measured along the *z* axis, in *g*.
`RANGE(0)`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH(0)`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
`BIAS(0)`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.

`GAIN(0)`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Gyroscope		
`G`	Analog Read	An array of three numbers containing the rotation rates measured around the *x*, *y*, and *z* axes, in degrees per second.
`GX`	Analog Read	Rotation rate around the *x* axis, in degrees per second.
`GY`	Analog Read	Rotation rate around the *y* axis, in degrees per second.
`GZ`	Analog Read	Rotation rate around the *z* axis, in degrees per second.
`RANGE(1)`	Analog R/W	Selected full scale range of the gyroscope, in degrees per second.
`BANDWIDTH(1)`	Analog R/W	Selected bandwidth of the of the gyroscope, in hertz.
`BIAS(1)`	Analog R/W	An array of three numbers containing the gyroscope bias, in degrees per second, for the *x*, *y*, and *z* axes.
`GAIN(1)`	Analog R/W	An array of three numbers containing the gain for one LSB, in degrees per second, for the *x*, *y*, and *z* axes.
Magnetometer		
`M`	Analog Read	An array of three numbers containing the magnetic field measured in the *x*, *y*, and *z* axes, in microtesla.
`MX`	Analog Read	Magnetic field measured in the *x* direction, in microtesla.
`MY`	Analog Read	Magnetic field measured in the *y* direction, in microtesla.
`MZ`	Analog Read	Magnetic field measured in the *z* direction, in microtesla.
`BANDWIDTH(2)`	Analog R/W	Selected bandwidth of the of the magnetometer, in hertz.
Algorithms		

`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
Additional		
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the sensor assembly as a whole, in hertz. This is the smaller of `BANDWIDTH(0)` and `BANDWIDTH(1)`.

See also

See **Accelerometers**, **Gyroscopes**, **Magnetometers**, and **AHRS Driver**.

Atmel ATAVRSBIN2 Driver

Installation

```
INSTALL "ATMEL-ATAVRSBIN2"
INSTALL "ATAVRSBIN2"
```

Options

None.

Description

Installs a combination gyroscope, accelerometer, and magnetometer driver for the ATAVRSBIN2 module. The ATAVRSBIN2 integrates a Kionix KXTF9 accelerometer, an InvenSense IMU-3000 gyroscope, and a Honeywell HMC5883L magnetometer to provide a 9DOF sensor platform.

When these sensors are fused by the AHRS driver, you have excellent 9DOF attitude and heading reference system.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ	Analog Read	Acceleration measured along the *z* axis, in *g*.
`RANGE(0)`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH(0)`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
`BIAS(0)`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.

GAIN(0)	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Gyroscope		
G	Analog Read	An array of three numbers containing the rotation rates measured around the *x*, *y*, and *z* axes, in degrees per second.
GX	Analog Read	Rotation rate around the *x* axis, in degrees per second.
GY	Analog Read	Rotation rate around the *y* axis, in degrees per second.
GZ	Analog Read	Rotation rate around the *z* axis, in degrees per second.
RANGE(1)	Analog R/W	Selected full scale range of the gyroscope, in degrees per second.
BANDWIDTH(1)	Analog R/W	Selected bandwidth of the of the gyroscope, in hertz.
BIAS(1)	Analog R/W	An array of three numbers containing the gyroscope bias, in degrees per second, for the *x*, *y*, and *z* axes.
GAIN(1)	Analog R/W	An array of three numbers containing the gain for one LSB, in degrees per second, for the *x*, *y*, and *z* axes.
Magnetometer		
M	Analog Read	An array of three numbers containing the magnetic field measured in the *x*, *y*, and *z* axes, in microtesla.
MX	Analog Read	Magnetic field measured in the *x* direction, in microtesla.
MY	Analog Read	Magnetic field measured in the *y* direction, in microtesla.
MZ	Analog Read	Magnetic field measured in the *z* direction, in microtesla.
BANDWIDTH(2)	Analog R/W	Selected bandwidth of the of the magnetometer, in hertz.
Algorithms		

`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
Additional		
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the sensor assembly as a whole, in hertz. This is the smaller of `BANDWIDTH(0)` and `BANDWIDTH(1)`.

See also

See **Accelerometers**, **Gyroscopes**, **Magnetometers**, and **AHRS Driver**.

Bosch Sensortec BMA150 Driver

Installation

`INSTALL "BOSCH-SENSORTEC-BMA150"` [`USING` *i2c-bus*]
`INSTALL "BMA150"` [`USING` *i2c-bus*]

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x80`.

Description

Installs an accelerometer driver for the BMA150 and initializes the accelerometer to the 2*g* range. If a `USING` clause is present, the sensor is initialized on the I2C bus specified by *i2c-bus*; if no `USING` clause is present, the sensor is initialized on the primary I2C bus.

Properties

General		
`VERSION`	String Read	A string containing the silicon revision of the BMA150.
Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.

Configuration		
`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See **Accelerometers**.

Bosch Sensortec BMA250 Driver

Installation

`INSTALL "BOSCH-SENSORTEC-BMA150"` [`USING` *i2c-bus*]
`INSTALL "BMA250"` [`USING` *i2c-bus*]

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x30`.

Description

Installs an accelerometer driver for the BMA250 and initializes the accelerometer to the 2*g* range. If a `USING` clause is present, the sensor is initialized on the I2C bus specified by *i2c-bus*; if no `USING` clause is present, the sensor is initialized on the primary I2C bus.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		

`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See **Accelerometers**.

Bosch Sensortec BMM150 Driver

Installation

```
INSTALL "BOSCH-SENSORTEC-BMM150"
INSTALL "BMM150"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x22`.

Description

Installs a magnetometer driver for the BMM150.

Properties

Magnetometer		
`M`	Analog Read	An array of three numbers containing the magnetic field measured in the *x*, *y*, and *z* axes, in microtesla.
`MX` *or* X	Analog Read	Magnetic field measured in the *x* direction, in microtesla.
`MY` *or* Y	Analog Read	Magnetic field measured in the *y* direction, in microtesla.
`MZ` *or* Z	Analog Read	Magnetic field measured in the *z* direction, in microtesla.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See **Magnetometers**.

Bosch Sensortec BMP085 Driver

Installation

```
INSTALL "BOSCH-SENSORTEC-BMP085"
INSTALL "BMP085"
INSTALL "CORE-PRESSURE"
```

Options

None.

Description

Installs a pressure sensor driver using the BMP085 on the CorePressure SExI module which mounts into a SenseCore:

http://www.soldercore.com/products/sensecore/corepressure/

Although this is intended to fit into an SenseCore, you can use it as a standard breakout module and mount it wherever you wish or use it with something other than a SolderCore.

Properties

Sensors		
PRESSURE	Analog Read	Pressure measured in pascals.
TEMP	Analog Read	Die temperature measured in degrees Celsius.
ALL	Analog Read	An array of two numbers containing the pressure in pascals and the temperature in degrees Celsius.
Control		
HEIGHT	Analog Read	Current height above sea level, in meters. For HEIGHT to work correctly, you must assign the barometric pressure at sea level to ORIGIN.
RESOLUTION	Digital R/W	Sensor pressure resolution in bits. The BMP085 supports 16 to 19 bits of resolution with higher resolutions taking longer to convert as the sensor uses oversampling. The default is RESOLUTION=17.

`ORIGIN`	Analog R/W	The pressure at sea level. You need to set this property to the current barometric pressure at sea level if you want the `HEIGHT` property to work correctly.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

References

The SolderCore CorePressure half-width SExI module:

http://www.soldercore.com/products/sensecore/corepressure/

SparkFun and Jee Labs offer BMP085 breakout boards:

http://www.sparkfun.com/products/9694
http://jeelabs.com/products/pressure-plug

Example

```
***../examples/corepressure-demo.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "corepressure-demo"` or `|corepressure-demo`.

Bosch Sensortec SMB380 Driver

Installation

`INSTALL "BOSCH-SENSORTEC-SMB380"` [`USING` *i2c-bus*]
`INSTALL "SMB380"` [`USING` *i2c-bus*]

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x70`.

Description

Installs an accelerometer driver for the SMB380 and initializes the accelerometer to the 2*g* range. If a `USING` clause is present, the sensor is initialized on the I2C bus specified by *i2c-bus*; if no `USING` clause is present, the sensor is initialized on the primary I2C bus.

Properties

General		
`VERSION`	String Read	A string containing the silicon revision of the SMB380.
Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.

Configuration		
RANGE	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer GAIN and BIAS to the defaults for the selected range.
BANDWIDTH	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
BIAS	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
GAIN	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
WORD(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See **Accelerometers**.

References

Olimex offer an SMB380 breakout board:

http://www.olimex.com/dev/mod-smb380.html

Extended User Memory

Installation

```
INSTALL "EXTENDED-USER-MEMORY".
```

Options

`SIZE=`*integer*

Description

Installs a driver that provides additional memory for variable names. When CoreBASIC starts, it allocates 512 bytes to hold *long* variable names, that is, variable names longer than two characters. CoreBASIC manages this area as you edit your program, add, rename, and remove variables, recovering free space in the background.

However, you may find that 512 bytes simply is not enough for longer programs with long variable names. In this case, you can extend the memory that CoreBASIC sets aside for long variable names, by installing the `EXTENDED-USER-MEMORY` driver.

The parameter `SIZE` specifies the number of additional bytes to set aside for long variable names. The memory for CoreBASIC programs and runtime data are reduced by the same amount.

It's best to install this driver in the `/c/sys/!boot.bas` file, so it becomes part of the boot process.

Notes

You should install the extended user memory driver before any other driver that requires "high memory" for operation (and most do). The extended memory driver ensures that it can extend the variable name space and, if it cannot because other drivers have been installed beforehand, it will throw a "using error" exception.

Example

```
   ______                ____  ___   _____ ___________
  / ____/___  _________  / __ )/   | / ___//  _/ ____/
 / /   / __ \/ ___/ _ \/ __  / /| | \__ \ / // /
/ /___/ /_/ / /  /  __/ /_/ / ___ |___/ // // /___
\____/\____/_/   \___/_____/_/  |_/____/___/\____/

Welcome to CoreBASIC on the SolderCore.
Copyright (c) 2004-2014 Rowley Associates Limited.

65,016 bytes free.  Core at 80 MHz.  Firmware 1.3.0.

> memory

Region            Total    Used   Free   Units    Load  Comment
-------------------------------------------------------------------------------
CoreBASIC RAM    67,584   2,566  65,018  bytes    3.8%  Total available RAM
Program text     65,016       6  65,010  bytes    0.0%  Compress with CRUNCH
High memory         512     512       0  bytes  100.0%  Drivers and long names
Variable names      512       0     512  bytes    0.0%  Stored in high memory
Scratchpad          256       0     256  cells    0.0%  Fixed runtime overhead
```

```
Object store      8,124       0   8,124  cells    0.0%  Arrays and strings

> install "extended-user-memory size=1024"
> memory

Region            Total    Used   Free   Units    Load  Comment
-------------------------------------------------------------------------------
CoreBASIC RAM    67,584   3,590  63,994  bytes    5.3%  Total available RAM
Program text     63,992       6  63,986  bytes    0.0%  Compress with CRUNCH
High memory       1,536   1,536       0  bytes  100.0%  Drivers and long names
Variable names    1,536       0   1,536  bytes    0.0%  Stored in high memory
Scratchpad          256       0     256  cells    0.0%  Fixed runtime overhead
Object store      7,996       0   7,996  cells    0.0%  Arrays and strings

> install list

Driver                              Type          Himem Used
-------------------------------------------------------------
SOLDERCORE-CPU                      Fixed             0 bytes
EXTENDED-USER-MEMORY                Fixed         1,024 bytes
> _
```

Freedom Board Accelerometer

Installation

```
INSTALL "FREEDOM-ACCELEROMETER"
INSTALL "ACCELEROMETER"
```

Options

None.

Description

Installs an accelerometer driver for the on-board MMA8451Q accelerometer of the Freedom Board. The accelerometer is connected to the internal I2C bus of the Freedom Board, not the I2C bus that is routed to the Arduino-style headers on the PCB.

This is a special driver for the Freedom Board and is not available on the SolderCore. However, a generic MMA8451Q driver is provided by the SolderCore for attaching such a sensor to the standard I2C bus.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		

`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

See also

See **Freescale MMA8451Q Driver** and **Accelerometers**.

Freedom Board CPU

Installation

The Freedom Board CPU driver is automatically installed as a fixed driver when CoreBASIC starts.

Description

The Freedom Board CPU provides the base hardware drivers for digital and analog I/O, SPI, and I2C. You access the CPU driver using the `CORE` keyword.

Properties

Information		
`NAME`	String Input	Presentation name for the model that runs CoreBASIC. For the standard Freedom Board, this is "`Freedom Board`".
`MODEL`	String Input	Platform that CoreBASIC is running on. For the Freedom Board, this is "`FREEDOM-V1`".
`VERSION`	String Input	CoreBASIC version number.
Digital and analog I/O		
`D0` through `D19`	Digital R/W	See expanded pin description below.
`A0` through `A19`	Digital R/W	See expanded pin description below.
`D(n)`	Digital R/W	Equivalent to `D0` through `D19`, indexed by *n*.
`A(n)`	Digital R/W	Equivalent to `A0` through `A19`, indexed by *n*.
LEDs		
`R`	Digital Write	Illuminates the red channel of the tricolor LED when written to a nonzero value, and extinguishes it when written to zero.
`G`	Digital Write	Illuminates the green channel of the tricolor LED when written to a nonzero value, and extinguishes it when written to zero.
`B`	Digital Write	Illuminates the blue channel of the tricolor LED when written to a nonzero value, and extinguishes it when written to zero.

LED	Digital Write	Writes a 24-bit true color value to the tricolor LED.
Timing		
FREQUENCY	Digital Write	Core tick frequency. For the Freedom Board, this reads as 24,000,000 indicating 24 MHz.
TICK	Digital Write	Core tick. The tick increments at the core tick frequency, FREQUENCY.

See also

CORE

Freescale MAG3110 Driver

Installation

```
INSTALL "FREESCALE-MAG3110"
INSTALL "MAG3110"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x1C`.

Description

Installs a magnetometer driver for the MAG3110.

Properties

Magnetometer		
`M`	Analog Read	An array of three numbers containing the magnetic field measured in the *x*, *y*, and *z* axes, in microtesla.
`MX` *or* `X`	Analog Read	Magnetic field measured in the *x* direction, in microtesla.
`MY` *or* `Y`	Analog Read	Magnetic field measured in the *y* direction, in microtesla.
`MZ` *or* `Z`	Analog Read	Magnetic field measured in the *z* direction, in microtesla.
Additional		
`TEMP`	Analog Read	Die temperature, in degrees Celsius. (See note below.)
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

The die temperature is not returned correctly by the MAG3110. We raised a service request with Freescale regarding this and their response is as follows:

"Yes, sorry, the die temperature is not being trimmed for MAG3110. This should have been in the errata. What you are seeing is correct. You can get around this by trimming the output in software since the sensitivity of the sensor is OK."

See **Magnetometers**.

References

SparkFun and Olimex both offer MAG3110 breakout boards:

http://www.sparkfun.com/products/10619
http://www.olimex.com/dev/mod-mag.html

Freescale MMA8451Q Driver

Installation

```
INSTALL "FREESCALE-MMA8451Q"
INSTALL "MMA8451Q"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x3A`. To select the alternate address, use `ADDR=0x38`.

Description

Installs an accelerometer driver for the MMA8451Q.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		

`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

See also

See **Freedom Board Accelerometer**.

Notes

See **Accelerometers**.

Freescale MMA8491Q Driver

Installation

```
INSTALL "FREESCALE-MMA8491Q"
INSTALL "MMA8491Q"
```

Parameters

`USING` *bus*`,` *enable*
`USING` *enable*

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0xAA`.

Description

Installs an accelerometer driver for the MMA8491Q. The MMA8491Q requires an additional signal, *enable*, connected to the ENABLE pin of the accelerometer.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.

Configuration		
`RANGE`	Analog R	Fixed at ±8*g*.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

See also

See **Accelerometers**.

Freescale MPL115A1 Driver

Installation

```
INSTALL "FREESCALE-MPL115A1" USING select
INSTALL "MPL115A1" USING select
```

Options

None.

Description

Installs a pressure sensor driver for the Freescale MPL115A1 SPI pressure sensor. The USING clause specifies the device select signal to select the MPL115A1.

Properties

Sensors		
PRESSURE	Analog Read	Pressure measured in pascals.
TEMP	Analog Read	Returns NaN because the Freescale do not document how to turn the raw ADC values to a temperature estimation. The temperature sensor does, however, compensate the pressure measurement.
Control		
HEIGHT	Analog Read	Current height above sea level, in meters. For HEIGHT to work correctly, you must assign the barometric pressure at sea level to ORIGIN.
ORIGIN	Analog R/W	The pressure at sea level. You need to set this property to the current barometric pressure at sea level if you want the HEIGHT property to work correctly.
Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
WORD(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

Example

```
***../examples/mpl115a1-demo.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE `"mpl115a1-demo"` or `|mpl115a1-demo`.

Freescale MPL115A2 Driver

Installation

```
INSTALL "FREESCALE-MPL115A2"
INSTALL "MPL115A2"
```

Options

ADDR=*integer*

Set the I2C 8-bit address of the MPL115A2. By default the driver uses the address `0xC0`.

Description

Installs a pressure sensor driver for the Freescale MPL115A2 I2C pressure sensor.

Properties

Sensors		
`PRESSURE`	Analog Read	Pressure measured in pascals.
`TEMP`	Analog Read	Returns NaN because the Freescale do not document how to turn the raw ADC values to a temperature estimation. The temperature sensor does, however, compensate the pressure measurement.
Control		
`HEIGHT`	Analog Read	Current height above sea level, in meters. For `HEIGHT` to work correctly, you must assign the barometric pressure at sea level to `ORIGIN`.
`ORIGIN`	Analog R/W	The pressure at sea level. You need to set this property to the current barometric pressure at sea level if you want the `HEIGHT` property to work correctly.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Freescale MPL3115A2 Driver

Installation

```
INSTALL "FREESCALE-MPL3115A2"
INSTALL "MPL3115A2"
```

Options

ADDR=*integer*

Set the I2C 8-bit address of the MPL3115A2. By default the driver uses the address `0xC0`.

Description

Installs a pressure sensor driver for the Freescale MPL3115A2 I2C pressure sensor.

Properties

Sensors		
`PRESSURE`	Analog Read	Pressure measured in pascals.
`TEMP`	Analog Read	Temperature measured in degrees Celsius.
Control		
`HEIGHT`	Analog Read	Current height above sea level, in meters. For `HEIGHT` to work correctly, you must assign the barometric pressure at sea level to `ORIGIN`.
`ORIGIN`	Analog R/W	The pressure at sea level. You need to set this property to the current barometric pressure at sea level if you want the `HEIGHT` property to work correctly.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

FTP Server

Installation

```
INSTALL "FTP-SERVER".
```

Options

None.

Description

Installs the FTP server to transfer files to and from the SolderCore. The root of the FTP server file system is `/c`. The FTP server is always installed as a fixed driver and continues to run in parallel with CoreBASIC.

The web server takes approximately 4 kilobytes of RAM from high memory for operation.

Notes

If you are going to use the FTP server, it's best to install it in `/c/sys/!boot.bas` as part of the boot process.

Gravitech 7-Segment Shield

Installation

```
INSTALL "GRAVITECH-7SEG-SHIELD"
INSTALL "7SEG-SHIELD"
INSTALL "7SEG"
```

Options

ADDR=*integer*

Set the I2C 8-bit address of the SAA1064. By default the driver uses the address `0x92`.

Description

The 7-Segment shield combines a four digit 7-segment driver, an LM75 temperature sensor, an 24LC128 (16K×8) EEPROM, and RGB LED.

Properties

Tricolor LED		
`LED`	Analog Write	When written, sets the red, green, and blue color of the RGB LED.
`COLOR`	Analog Write	As `LED`.
`R`	Analog Write	Sets the brightness of the red component of the RGB LED.
`G`	Analog Write	Sets the brightness of the green component of the RGB LED.
`B`	Analog Write	Sets the brightness of the blue component of the RGB LED.
7-Segment LEDs — Direct Addressing		
`COL(n)`	Digital Write	Writes a single ASCII character to digit *n*, where digit 0 is the leftmost and digit 3 the rightmost. If the ASCII character has no corresponding encoding, the character is ignored.
`X(n)`	Digital Write	As `COL(n)`.
`A(n)`	Digital Write	Directly drives segment A of digit **n**.
`B(n)`	Digital Write	Directly drives segment B of digit **n**.
`C(n)`	Digital Write	Directly drives segment C of digit **n**.

`D(n)`	Digital Write	Directly drives segment D of digit **n**.
`E(n)`	Digital Write	Directly drives segment E of digit **n**.
`F(n)`	Digital Write	Directly drives segment F of digit **n**.
`G(n)`	Digital Write	Directly drives segment G of digit **n**.
`POINT(n)`	Digital Write	Directly drives the decimal point segment of digit **n**.
7-Segment LEDs — Character I/O		
`OUTPUT`	Digital Write	Sends ASCII characters to the 7-segment display. Characters are shifted in from the right. If the ASCII character has no corresponding encoding, the character is ignored.
`PRINT`	Digital Write	As `OUTPUT`.

Example

```
 10 INSTALL "GRAVITECH-7SEG-SHIELD" AS DIS
 20 DIS.OUTPUT = "HELO"
 30 PAUSE 1
 40 FOR I = 1 TO 4
 50   DIS.OUTPUT = " "
 60   PAUSE 0.2
 70 NEXT I
 80 FOR I = 0 TO DIS.WIDTH - 1
 90   DIS.A(I) = 1
100   DIS.D(DIS.WIDTH - 1 - I) = 1
110   PAUSE 0.2
120 NEXT I
130 FOR I = 0 TO DIS.WIDTH - 1
140   DIS.A(I) = 0
150   DIS.D(DIS.WIDTH - 1 - I) = 0
160   PAUSE 0.2
170 NEXT I
180 FOR I = 0 TO DIS.WIDTH - 1
190   DIS.B(I) = 1 : DIS.C(I) = 1 : PAUSE 0.2
210   DIS.E(I) = 1 : DIS.F(I) = 1 : PAUSE 0.2
230 NEXT I
240 FOR I = 1 TO 4
250   DIS.OUTPUT = "    " : PAUSE 0.1
270   DIS.OUTPUT = "DONE" : PAUSE 0.2
290 NEXT I
300 END
```

Hitachi HD44780 Driver

Installation

`INSTALL "HITACHI-HD44780" USING` *databus*`,` *rs*`,` *e*`[,` *rw*`]`

Options

None.

Description

Installs a generic driver for an HD44780 LCD.

Both 4-bit and 8-bit parallel buses are supported by this driver. If you are attaching using 4-bit mode, *databus* must be a 4-bit-wide bus; if attaching using 8-bit mode, *databus* must be an 8-bit-wide bus.

The required parameters *rs* and *e* specify the properties that drive the RS and E signals on the LCD.

The *rw* parameter is optional. If *rw* is specified, the databus is bidirectional and will be used to determine when the LCD is ready to accept further data. If the *rw* parameter is omitted, the databus is output only and the driver uses open-loop delays after each command to ensure that the LCD is ready.

Properties

Dimensions		
`WIDTH`	Digital R/W	The width of the display, in character positions.
`HEIGHT`	Digital R/W	The height of the display, in character lines.
Cursor		
X or `COL`	Digital R/W	The *x* co-ordinate of the cursor.
Y or `ROW`	Digital R/W	The *y* co-ordinate of the cursor.
`CURSOR`	Digital Write	Cursor control. Writing `0` to `CURSOR` turns the cursor off, and writing `1` turns it on.
`LINE`	String R/W	As Y and `ROW`.
`POS`	Digital R/W	An array containing the *x* and *y* co-ordinates of the cursor.
Control		
`LIGHT`	Digital Write	Backlight control. Writing `0` to `LIGHT` turns the backlight off, if the display has a controllable backlight, and writing `1` turns it on.

Character I/O		
`LINE(n)`	String R/W	As `Y(n)` and `ROW(n)`.
`Y(n)` or `ROW(n)`	String R/W	When read, returns the string being displayed on line *n* of the display. When written, overwrites the whole of line *n* of the display, filling the line with extra spaces if necessary.
`CENTER(n)`	String Write	Centers the string written on display line *n*. If the string is too long for the display, only the central part is displayed. If the string is narrower than the display width, it is padded left and right with spaces to the display width such that it lies central within the display.
`RIGHT(n)`	String Write	Right-justifies the string written on display line *n*. If the string is too long for the display, it is truncated on the left such that the rightmost part of the string covers the whole line. If the string is narrower than the display width, it is padded left with spaces to the display width such that it lies adjusted right on the display.

Example

```
***../examples/digital-spirit-level.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "digital-spirit-level"` or `|digital-spirit-level`.

Honeywell HIH6130 Driver

Installation

```
INSTALL "HONEYWELL-HIH1630"
INSTALL "HIH1630"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x4E`.

Description

Installs a humidity and temperature sensor driver for the HIH1630 sensor.

Properties

Measurement		
`TEMP`	Analog Read	Supply-compensated temperature in degrees Celsius.
`HUMIDITY`	Analog Read	True relative humidity, in percent.
`DEWPOINT`	Analog Read	Dew point in degrees Celsius, computed from the supply-compensated temperature `TEMP` and true relative humidity `HUMIDITY`.
`DATA`	Analog Read	An array of four values: the temperature, in degrees Celsius, the true relative humidity in percent, and the dew point in degrees Celsius.
`DATA(n)`	Analog Read	Direct access to temperature and humidity readings. `DATA(0)` is the supply-compensated temperature in degrees Celsius and is equivalent to the `TEMP` property; `DATA(1)` is the true relative humidity in percent and is equivalent to the `HUMIDITY` property; `DATA(2)` is the dew point in degrees Celsius and is equivalent to the `DEWPOINT` property.

Honeywell HMC5843 Driver

Installation

```
INSTALL "HONEYWELL-HMC5843"
INSTALL "HMC5843"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x3C`.

Description

Installs a magnetometer driver for the HMC5843.

Properties

Magnetometer		
`M`	Analog Read	An array of three numbers containing the magnetic field measured in the *x*, *y*, and *z* axes, in microtesla.
`MX` *or* `X`	Analog Read	Magnetic field measured in the *x* direction, in microtesla.
`MY` *or* `Y`	Analog Read	Magnetic field measured in the *y* direction, in microtesla.
`MZ` *or* `Z`	Analog Read	Magnetic field measured in the *z* direction, in microtesla.
Configuration		
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the magnetometer, in hertz.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See Magnetometers.

Honeywell HMC5883L Driver

Installation

```
INSTALL "HONEYWELL-HMC5883L"
INSTALL "HMC5883L"
INSTALL "CORE-MAG"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x3C`.

Description

Installs a magnetometer driver using the HMC5883L on the CoreMag SExI module which mounts into a SenseCore:

http://www.soldercore.com/products/sensecore/coremag/

Although this is intended to fit into an SenseCore, you can use it as a standard breakout module and mount it wherever you wish or use it with something other than a SolderCore.

If you require a complete attitude and heading reference system, the SolderCore CoreMPU provides an HMC5883L and MPU-6050 on a SExI module:

http://www.soldercore.com/products/sensecore/corempu/

Properties

Magnetometer		
`M`	Analog Read	An array of three numbers containing the magnetic field measured in the *x*, *y*, and *z* axes, in microtesla.
`MX` *or* X	Analog Read	Magnetic field measured in the *x* direction, in microtesla.
`MY` *or* Y	Analog Read	Magnetic field measured in the *y* direction, in microtesla.
`MZ` *or* Z	Analog Read	Magnetic field measured in the *z* direction, in microtesla.
Configuration		
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the magnetometer, in hertz.

Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
WORD(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See **Magnetometers**.

References

The SolderCore CoreMag half-width SExI module:

http://www.soldercore.com/products/sensecore/coremag/

SparkFun and LoveElectronics offer HMC5883L breakout boards:

http://www.sparkfun.com/products/10530
http://www.loveelectronics.co.uk/products/140/

Honeywell HMC6343 Driver

Installation

```
INSTALL "HONEYWELL-HMC6343"
INSTALL "HMC6343"
```

Options

None.

Description

Installs a compass driver for the HMC6343.

Properties

Compass		
HEADING	Analog Write	Heading in degrees.
PITCH	Analog Write	Pitch angle in degrees.
ROLL	Analog Write	Roll angle in degrees.
General		
ALL	Analog Write	Array of three numbers with the elements eading, pitch angle, and roll angle in egrees.

Honeywell HMC6352 Driver

Installation

```
INSTALL "HONEYWELL-HMC6352"
INSTALL "HMC6352"
```

Options

None.

Description

Installs a compass driver for the HMC6352.

Properties

Compass		
HEADING	Analog Read	Heading in degrees.

HTTP Server

Installation

```
INSTALL "HTTP-SERVER".
```

Options

None.

Description

Installs the HTTP server to serve web pages from `/c/www`. The HTTP server is always installed as a fixed driver and continues to run in parallel with CoreBASIC.

The web server takes approximately 2 kilobytes of RAM from high memory for operation.

Notes

If you are going to use the HTTP server, it's best to install it in `/c/sys/!boot.bas` as part of the boot process.

Intersil ISL29023 Driver

Installation

```
INSTALL "INTERSIL-ISL29023"
INSTALL "ISL29023"
INSTALL "CORE-LIGHT"
```

Options

None.

Description

Installs a light sensor driver for the ISL29023 on the CoreLight SExI module which mounts into a SenseCore:

http://www.soldercore.com/products/sensecore/corelight/

Although this is intended to fit into an SenseCore, you can use it as a standard breakout module and mount it wherever you wish or use it with something other than a SolderCore.

Properties

Light sensor		
`LIGHT`	Analog Write	Reads the current ambient light level, in lux.
Configuration		
`RANGE`	Digital R/W	Sets the ISL29023 sensor range, from 1 to 4.
`RESOLUTION`	Digital R/W	Sets the ISL29023 sensor resolution.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

References

The SolderCore CoreLight SExI module:

http://www.soldercore.com/products/sensecore/corelight/

Example

```
***../examples/corelight-demo.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "corelight-demo"` or `|corelight-demo`.

InvenSense IMU-3000 Driver

Installation

```
INSTALL "INVENSENSE-IMU-3000"
INSTALL "IMU-3000"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0xD0`.

Description

Installs a gyroscope driver for the IMU-3000.

Properties

Gyroscope		
`G`	Analog Read	An array of three numbers containing the rotation rates measured around the *x*, *y*, and *z* axes, in degrees per second.
`GX` *or* `X`	Analog Read	Rotation rate around the *x* axis, in degrees per second.
`GY` *or* `Y`	Analog Read	Rotation rate around the *y* axis, in degrees per second.
`GZ` *or* `Z`	Analog Read	Rotation rate around the *z* axis, in degrees per second.
Configuration		
`RANGE`	Analog R/W	Selected full scale range of the gyroscope, in degrees per second.
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the gyroscope, in hertz.
`BIAS`	Analog R/W	An array of three numbers containing the gyroscope bias, in degrees per second, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in degrees per second, for the *x*, *y*, and *z* axes.

Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
HALF(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

See also

SparkFun IMU-3000 Combo

Notes

See **Gyroscopes**.

InvenSense ITG-3200 Driver

Installation

```
INSTALL "INVENSENSE-ITG-3200"
INSTALL "ITG-3200"
INSTALL "CORE-SPIN"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0xD2`.

Description

Installs a gyroscope driver for the ITG-3200 on the CoreSpin SExI module which which mounts into a SenseCore:

http://www.soldercore.com/products/sensecore/coregyro/

Although this is intended to fit into an SenseCore, you can use it as a standard breakout module and mount it wherever you wish or use it with something other than a SolderCore.

Properties

Gyroscope		
G	Analog Read	An array of three numbers containing the rotation rates measured around the *x*, *y*, and *z* axes, in degrees per second.
GX *or* X	Analog Read	Rotation rate around the *x* axis, in degrees per second.
GY *or* Y	Analog Read	Rotation rate around the *y* axis, in degrees per second.
GZ *or* Z	Analog Read	Rotation rate around the *z* axis, in degrees per second.
Configuration		
RANGE	Analog R/W	Selected full scale range of the gyroscope, in degrees per second.
BANDWIDTH	Analog R/W	Selected bandwidth of the of the gyroscope, in hertz.
BIAS	Analog R/W	An array of three numbers containing the gyroscope bias, in degrees per second, for the *x*, *y*, and *z* axes.

GAIN	Analog R/W	An array of three numbers containing the gain for one LSB, in degrees per second, for the *x*, *y*, and *z* axes.
Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
HALF(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See **Gyroscopes**.

References

The SolderCore CoreSpin SExI module:

http://www.soldercore.com/products/sensecore/coregyro/

SparkFun offer an ITG-3200 breakout:

http://www.sparkfun.com/products/9801

InvenSense MPU-6000 Driver

Installation

```
INSTALL "INVENSENSE-MPU-6000" USING select
INSTALL "MPU-6000" USING select
```

Options

None.

Description

Installs a combination gyroscope and accelerometer driver for the InvenSense MPU-6000.

Because the MPU-6000 is an SPI device, the USING clause must specify a single-bit GPIO that controls the chip select connected to the MPU-6000 device. When the MPU-6000 is paired with the CoreBASIC AHRS driver, it makes an excellent 6DOF IMU. When the MPU-6000 and a magnetometer are fused by the the AHRS driver, you have an even better 9DOF attitude and heading reference system.

The MPU-6000 driver was tested on an InvenSense MPU-6000 Evaluation Board connected to a CoreProto rivet plugged into a SenseCore.

If you need an easy to use plug-and-go motion sensing and AHRS solution, take a look at the SolderCore **CoreMPU**.

Properties

General		
VERSION	String Read	A string containing the silicon revision of the MPU-6000.
Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ	Analog Read	Acceleration measured along the *z* axis, in *g*.

`RANGE(0)`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH(0)`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
`BIAS(0)`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN(0)`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Gyroscope		
`G`	Analog Read	An array of three numbers containing the rotation rates measured around the *x*, *y*, and *z* axes, in degrees per second.
`GX`	Analog Read	Rotation rate around the *x* axis, in degrees per second.
`GY`	Analog Read	Rotation rate around the *y* axis, in degrees per second.
`GZ`	Analog Read	Rotation rate around the *z* axis, in degrees per second.
`RANGE(1)`	Analog R/W	Selected full scale range of the gyroscope, in degrees per second.
`BANDWIDTH(1)`	Analog R/W	Selected bandwidth of the of the gyroscope, in hertz.
`BIAS(1)`	Analog R/W	An array of three numbers containing the gyroscope bias, in degrees per second, for the *x*, *y*, and *z* axes.
`GAIN(1)`	Analog R/W	An array of three numbers containing the gain for one LSB, in degrees per second, for the *x*, *y*, and *z* axes.
Algorithms		

ROLL	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
PITCH	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
Additional		
BANDWIDTH	Analog R/W	Selected bandwidth of the of the sensor assembly as a whole, in hertz. This is the smaller of `BANDWIDTH(0)` and `BANDWIDTH(1)`.
TEMP	Analog Write	Die temperature, in degrees Celsius.
Transport		
SPEED	Digital R/W	The speed of the bus, in hertz, when addressing the device. The default is 1 MHz, which is the maximum speed supported by the MPU-6000.
Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
HALF(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

References

The InvenSense MPU-6000 Evaluation Board manual:

http://www.invensense.com/mems/gyro/mpu6050.html

The SolderCore CoreMPU SExI module:

http://www.soldercore.com/products/sensecore/corempu/

Notes

See Accelerometers, Gyroscopes, SolderCore CoreMPU Driver, and AHRS Driver.

InvenSense MPU-6050 Driver

Installation

```
INSTALL "INVENSENSE-MPU-6050"
INSTALL "MPU-6050"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0xD0`.

Description

Installs a combination gyroscope and accelerometer driver for the InvenSense MPU-6050.

When the MPU-6050 is paired with the CoreBASIC AHRS driver, it makes an excellent 6DOF IMU. When the MPU-6050 and a magnetometer are fused by the the AHRS driver, you have an even better 9DOF attitude and heading reference system.

Note that the `CORE-MPU` driver configures the SolderCore CoreMPU in its entirety to simplify generating a 9DOF sensor-fusion AHRS.

Please see **SolderCore CoreMPU Driver** for more information on the CoreMPU driver and see **AHRS Driver** for more information on putting this device to work.

Properties

General		
VERSION	String Read	A string containing the silicon revision of the MPU-6050.
Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ	Analog Read	Acceleration measured along the *z* axis, in *g*.

`RANGE(0)`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH(0)`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
`BIAS(0)`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN(0)`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Gyroscope		
`G`	Analog Read	An array of three numbers containing the rotation rates measured around the *x*, *y*, and *z* axes, in degrees per second.
`GX`	Analog Read	Rotation rate around the *x* axis, in degrees per second.
`GY`	Analog Read	Rotation rate around the *y* axis, in degrees per second.
`GZ`	Analog Read	Rotation rate around the *z* axis, in degrees per second.
`RANGE(1)`	Analog R/W	Selected full scale range of the gyroscope, in degrees per second.
`BANDWIDTH(1)`	Analog R/W	Selected bandwidth of the of the gyroscope, in hertz.
`BIAS(1)`	Analog R/W	An array of three numbers containing the gyroscope bias, in degrees per second, for the *x*, *y*, and *z* axes.
`GAIN(1)`	Analog R/W	An array of three numbers containing the gain for one LSB, in degrees per second, for the *x*, *y*, and *z* axes.
Algorithms		

`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
Additional		
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the sensor assembly as a whole, in hertz. This is the smaller of `BANDWIDTH(0)` and `BANDWIDTH(1)`.
`TEMP`	Analog Write	Die temperature, in degrees Celsius.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`HALF(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

References

InvenSense MPU-6050:

http://www.invensense.com/mems/gyro/mpu6050.html

The SolderCore CoreMPU SExI module:

http://www.soldercore.com/products/sensecore/corempu/

Notes

See **Accelerometers**, **Gyroscopes**, **SolderCore CoreMPU Driver**, and **AHRS Driver**.

InvenSense MPU-6050EVB Driver

Installation

```
INSTALL "INVENSENSE-MPU-6050EVB"
INSTALL "MPU-6050EVB"
```

Options

ADDR=*integer*

Set the I2C 8-bit address of the InvenSense MPU. By default the driver uses the address `0xD0`.

Description

Installs a combination gyroscope, accelerometer, and magnetometer driver for the InvenSense MPU-6050EVB evaluation board. The MPU-6050EVB integrates an InvenSense MPU-6050 accelerometer and gyroscope and an Asahi Kasei AK8975 magnetometer to provide a 9DOF sensor platform.

This driver will also work with an MPU-9150EVB where all sensors are integrated into a single package which acts the same as the MPU-6050EVB.

The MPU driver automatically detects the type of MPU attached (MPU-6050, MPU-6150, or MPU-9150) and configures itself for that device. The `VERSION` property returns the type of device detected and its silicon revision.

When the MPU and AK8975 are fused by the AHRS driver, you have excellent 9DOF attitude and heading reference system.

Properties

General		
`VERSION`	String Read	A string containing the silicon revision of the MPU.
Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ	Analog Read	Acceleration measured along the *z* axis, in *g*.

`RANGE(0)`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH(0)`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
`BIAS(0)`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN(0)`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Gyroscope		
`G`	Analog Read	An array of three numbers containing the rotation rates measured around the *x*, *y*, and *z* axes, in degrees per second.
`GX`	Analog Read	Rotation rate around the *x* axis, in degrees per second.
`GY`	Analog Read	Rotation rate around the *y* axis, in degrees per second.
`GZ`	Analog Read	Rotation rate around the *z* axis, in degrees per second.
`RANGE(1)`	Analog R/W	Selected full scale range of the gyroscope, in degrees per second.
`BANDWIDTH(1)`	Analog R/W	Selected bandwidth of the of the gyroscope, in hertz.
`BIAS(1)`	Analog R/W	An array of three numbers containing the gyroscope bias, in degrees per second, for the *x*, *y*, and *z* axes.
`GAIN(1)`	Analog R/W	An array of three numbers containing the gain for one LSB, in degrees per second, for the *x*, *y*, and *z* axes.
Magnetometer		
`M`	Analog Read	An array of three numbers containing the magnetic field measured in the *x*, *y*, and *z* axes, in microtesla.

MX	Analog Read	Magnetic field measured in the *x* direction, in microtesla.
MY	Analog Read	Magnetic field measured in the *y* direction, in microtesla.
MZ	Analog Read	Magnetic field measured in the *z* direction, in microtesla.
BANDWIDTH(2)	Analog R/W	Selected bandwidth of the of the magnetometer, in hertz.
Algorithms		
ROLL	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
PITCH	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
Additional		
BANDWIDTH	Analog R/W	Selected bandwidth of the of the sensor assembly as a whole, in hertz. This is the smaller of BANDWIDTH(0) and BANDWIDTH(1).
TEMP	Analog Write	MPU-6050 die temperature, in degrees Celsius.

References

InvenSense MPU-6050 EVB:

http://www.invensense.com/mems/gyro/documents/AN-MPU-6000EVB.pdf

InvenSense MPU-9150 EVB:

http://www.invensense.com/mems/gyro/documents/AN-MPU-9150EVB-00.pdf

Notes

See **Accelerometers**, **Gyroscopes**, **Magnetometers**, and **AHRS Driver**.

InvenSense MPU-9150 Driver

Installation

```
INSTALL "INVENSENSE-MPU-9150"
```

Options

ADDR=*integer*

Set the I2C 8-bit address of the InvenSense MPU. By default the driver uses the address `0xD0`.

Description

Installs a combination gyroscope, accelerometer, and magnetometer driver for the InvenSense MPU-9150. The MPU-9150 integrates an accelerometer and gyroscope and an Asahi Kasei AK8975 magnetometer to provide a 9DOF sensor platform.

When the MPU and AK8975 are fused by the AHRS driver, you have excellent 9DOF attitude and heading reference system.

Properties

General		
`VERSION`	String Read	A string containing the silicon revision of the MPU.
Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ	Analog Read	Acceleration measured along the *z* axis, in *g*.
`RANGE(0)`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH(0)`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.

BIAS(0)	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
GAIN(0)	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Gyroscope		
G	Analog Read	An array of three numbers containing the rotation rates measured around the *x*, *y*, and *z* axes, in degrees per second.
GX	Analog Read	Rotation rate around the *x* axis, in degrees per second.
GY	Analog Read	Rotation rate around the *y* axis, in degrees per second.
GZ	Analog Read	Rotation rate around the *z* axis, in degrees per second.
RANGE(1)	Analog R/W	Selected full scale range of the gyroscope, in degrees per second.
BANDWIDTH(1)	Analog R/W	Selected bandwidth of the of the gyroscope, in hertz.
BIAS(1)	Analog R/W	An array of three numbers containing the gyroscope bias, in degrees per second, for the *x*, *y*, and *z* axes.
GAIN(1)	Analog R/W	An array of three numbers containing the gain for one LSB, in degrees per second, for the *x*, *y*, and *z* axes.
Magnetometer		
M	Analog Read	An array of three numbers containing the magnetic field measured in the *x*, *y*, and *z* axes, in microtesla.
MX	Analog Read	Magnetic field measured in the *x* direction, in microtesla.
MY	Analog Read	Magnetic field measured in the *y* direction, in microtesla.
MZ	Analog Read	Magnetic field measured in the *z* direction, in microtesla.
BANDWIDTH(2)	Analog R/W	Selected bandwidth of the of the magnetometer, in hertz.

Algorithms		
ROLL	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
PITCH	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
Additional		
BANDWIDTH	Analog R/W	Selected bandwidth of the of the sensor assembly as a whole, in hertz. This is the smaller of `BANDWIDTH(0)` and `BANDWIDTH(1)`.
TEMP	Analog Write	MPU-6050 die temperature, in degrees Celsius.

References

InvenSense MPU-9150:

http://invensense.com/mems/gyro/mpu9150.html

Notes

See **Accelerometers**, **Gyroscopes**, **Magnetometers**, and **AHRS Driver**.

ITead Studio Colors Shield

Installation

```
INSTALL "ITEAD-STUDIO-COLORS-SHIELD"
INSTALL "COLORS-SHIELD"
```

Options

None.

Description

Installing the Colors Shield Shield driver provides an 8×8 color graphic display using the high-power LEDs on the shield. You can use all the CoreBASIC graphics commands to drive the LED display.

Properties

Control		
FREQUENCY	Digital R/W	Line scanning frequency. To reduce flicker, at the expense of higher CPU overhead, you can increase the scanning frequency. To reduce CPU overhead, at the expense of increased flicker, decrease the scanning frequency. By default, the frequency is set to 600 Hz; as the array has eight lines, the default frame scanning frequency is 75Hz. Values assigned to FREQUENCY are clamped to lie in the range 100 to 1,000.

Example

This example scrolls a multi-color message across the LED display.

```
***../examples/colors-shield-message.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE `"colors-shield-message"` or `|colors-shield-message`.

Resources

http://iteadstudio.com/store/index.php?main_page=product_info&products_id=312

ITead Studio ITDB02-2.2 LCD Module

Installation

```
INSTALL "ITEAD-STUDIO-ITDB02-2.2"
INSTALL "ITDB02-2.2"
```

Options

None.

Description

Installing the ITDB02 shield with the 2.2" LCD Module provides a 176×220 color graphic display with touch screen input.

ITDB02 v2 shield

Using the ITDB02-2.2 LCD module and the ITDB02 v2 shield requires no configuration.

ITDB02 v1 shield

Using the ITDB02-2.2 LCD module and the ITDB02 v1 shield requires proper configuration to ensure success. The LCD functions in 8-bit mode, so all shorting blocks on the ITDB02 must be moved to the T&SD side.

The switch on the ITB02 shield selects between routing the SD card or the the touch controller signals to the SolderCore. This driver requires the switch in the `Touch` position and support the touch panel only.

Notes

The LCD and touch controller require a lot of pins on the SolderCore to drive it. If you really wish to have pins left over and outstanding performance, the **SolderCore LCD Shield** uses only a few pins, has a 2.8" LCD with touchscreen, and the firmware for the 400 MHz XMOS XS1-L1 device on board is fully open source.

Resources

http://iteadstudio.com/store/index.php?main_page=product_info&products_id=149 — shield
http://iteadstudio.com/store/index.php?main_page=product_info&products_id=529 — 2.2" LCD module

A reference for all the displays offered by ITead Studio is found here:

ITead Studio LCDs

Benchmarks

Here is the result of running the SolderCore Graphics Benchmarks application:

```
> run
Graphics display benchmark for ITEAD-STUDIO-ITDB02-2.2

   Circles:    3267 ms
```

```
     Discs:   27950 ms
Rectangles:     781 ms
     Slabs:   22730 ms
     Lines:    9648 ms
  Polygons:    7858 ms
      Text:    3907 ms
> _
```

ITead Studio ITDB02-2.4D LCD Module

Installation

```
INSTALL "ITEAD-STUDIO-ITDB02-2.4D"
INSTALL "ITDB02-2.4D"
```

Options

None.

Description

Installing the ITDB02 shield with the 2.4" LCD Module provides a 320×240 color graphic display with touch screen input.

There are two versions of the ITDB02-2.4D LCD with different revisions of the ILI9325 controller. This driver expects to use the 2.4D LCD panel with the ILI9325D controller; there is an earlier 2.4 panel using the ILI9325C controller, but unfortunately that is no longer manufactured and we have no means to test any code on it.

ITDB02 v2 shield

Using the ITDB02-2.4D LCD module and the ITDB02 v2 shield requires no configuration.

ITDB02 v1 shield

Using the 2.4" LCD module requires proper configuration of the ITDB02 v1 shield to ensure success. The LCD functions in 8-bit mode, so all shorting blocks on the ITDB02 must be moved to the T&SD side.

The switch on the ITB02 shield selects between routing the SD card or the the touch controller signals to the SolderCore. This driver requires the switch in the Touch position and support the touch panel only.

Notes

The LCD and touch controller require a lot of pins on the SolderCore to drive it. If you really wish to have pins left over and outstanding performance, the **SolderCore LCD Shield** uses only a few pins, has a 2.8" LCD with touchscreen, and the firmware for the 400 MHz XMOS XS1-L1 device on board is fully open source.

Resources

http://iteadstudio.com/store/index.php?main_page=product_info&products_id=149 — shield
http://iteadstudio.com/store/index.php?main_page=product_info&products_id=55 — 2.4" LCD module

A reference for all the displays offered by ITead Studio is found here:

ITead Studio LCDs

ITead Studio ITDB02-2.4E LCD Module

Installation

```
INSTALL "ITEAD-STUDIO-ITDB02-2.4E"
INSTALL "ITDB02-2.4E"
```

Options

None.

Description

Installing the ITDB02 shield with the ITDB02-2.4E LCD Module provides a 320×240 color graphic display with touch screen input.

You can purchase a combined ITDB02-2.4E shield which integrated the ITDB02 v2 shield electronics and the ITDB02-2.4E LCD module into a single combined. This offers no-hassle setup but doesn't enable you to swap LCD displays like the ITDB02 shield does.

ITDB02 v2 shield

Using the ITDB02-2.4E LCD module and the ITDB02 v2 shield requires no configuration.

ITDB02 v1 shield

Using the ITDB02-2.4E LCD module requires proper configuration of the ITDB02 v1 shield to ensure success. The LCD functions in 8-bit mode, so all shorting blocks on the ITDB02 must be moved to the T&SD side.

The switch on the ITB02 shield selects between routing the SD card or the the touch controller signals to the SolderCore. This driver requires the switch in the `Touch` position and support the touch panel only.

Notes

The LCD and touch controller require a lot of pins on the SolderCore to drive it. If you really wish to have pins left over and outstanding performance, the **SolderCore LCD Shield** uses only a few pins, has a 2.8" LCD with touchscreen, and the firmware for the 400 MHz XMOS XS1-L1 device on board is fully open source.

Benchmarks

Here is the result of running the SolderCore Graphics Benchmarks application:

```
> run
Graphics display benchmark for ITEAD-STUDIO-ITDB02-2.4E
```

```
   Circles:    2494 ms
     Discs:   26460 ms
Rectangles:     701 ms
     Slabs:   24267 ms
     Lines:    6961 ms
  Polygons:    7626 ms
      Text:    2224 ms
> _
```

Resources

http://iteadstudio.com/store/index.php?main_page=product_info&products_id=149 — ITDB02 shield
http://iteadstudio.com/store/index.php?main_page=product_info&products_id=55 — 2.4" LCD module
http://iteadstudio.com/store/index.php?main_page=product_info&products_id=473 — 2.4" LCD module and shield combined

A reference for all the displays offered by ITead Studio is found here:

ITead Studio LCDs

ITead Studio ITDB02-2.8 LCD Module

Installation

```
INSTALL "ITEAD-STUDIO-ITDB02-2.8"
INSTALL "ITDB02-2.8"
```

Options

None.

Description

Installing the ITDB02 shield with the 2.8" LCD Module provides a 320×240 color graphic display with touch screen input.

Setup

Using the 2.8" LCD module requires proper configuration of the ITDB02 shield to ensure success. The LCD functions in 8-bit mode, so all shorting blocks on the ITDB02 must be moved to the T&SD side.

The switch on the ITB02 shield selects between routing the SD card or the the touch controller signals to the SolderCore. This driver requires the switch in the `Touch` position and support the touch panel only.

The LCD and touch controller require a lot of pins on the SolderCore to drive it. If you really wish to have pins left over and outstanding performance, the **SolderCore LCD Shield** uses only a few pins, has a 2.8" LCD with touchscreen, and the firmware for the 400 MHz XMOS XS1-L1 device on board is fully open source.

Resources

http://iteadstudio.com/store/index.php?main_page=product_info&products_id=149 — shield
http://iteadstudio.com/store/index.php?main_page=product_info&products_id=530 — 2.8" LCD module

A reference for all the displays offered by ITead Studio is found here:

ITead Studio LCDs

ITead Studio ITDB02-3.2S LCD Module

Installation

```
INSTALL "ITEAD-STUDIO-ITDB02-3.2S"
INSTALL "ITDB02-3.2S"
```

Options

None.

Description

Installing the ITDB02 shield with the ITDB02-3.2S LCD Module provides a 320×240 color graphic display *but no touch screen input*.

Note that ITead Studio sold an ITDB02-3.2 module in the past—note the lack of a trailing "S". This driver is *not compatible* with the ITDB02-3.2 module.

Setup

Using the ITDB02-3.2S LCD module requires proper configuration of the ITDB02 shield to ensure success. The LCD only functions in 16-bit mode, so all shorting blocks on the ITDB02 must be moved to the `LCD_16bit` side.

Unfortunately, driving the LCD in 16-bit mode utilizes all the pins on the SolderCore, leaving nothing free. A disappointing consequence is that you cannot use the SD card interface, nor can you use the touch screen controller. (As an aside, the ITDB02 Mega Shield does support concurrent SD card and touch capability along with the LCD, but that is a different format shield.)

If you really wish to have pins left over and outstanding performance, the **SolderCore LCD Shield** uses only a few pins, has a 2.8" LCD with touchscreen, and the firmware for the 400 MHz XMOS XS1-L1 device on board is fully open source.

Resources

http://iteadstudio.com/store/index.php?main_page=product_info&products_id=149 — shield
http://iteadstudio.com/store/index.php?main_page=product_info&products_id=54 — 3.2" LCD module

A reference for all the displays offered by ITead Studio is found here:

ITead Studio LCDs

ITead Studio ITDB02-3.2WD LCD Module

Installation

```
INSTALL "ITEAD-STUDIO-ITDB02-3.2WD"
INSTALL "ITDB02-3.2WD"
```

Options

None.

Description

Installing the ITDB02 shield with the ITDB02-3.2WD LCD Module provides a 240×400 color graphic display *but no touch screen input*.

Note that ITead Studio sold an ITDB02-3.2WC module in the past. This driver is *not compatible* with the ITDB02-3.2WC LCD module.

Setup

Using the ITDB02-3.2WD LCD module requires proper configuration of the ITDB02 shield to ensure success. The LCD only functions in 16-bit mode, so all shorting blocks on the ITDB02 must be moved to the `LCD_16bit` side.

Unfortunately, driving the LCD in 16-bit mode utilizes all the pins on the SolderCore, leaving nothing free. A disappointing consequence is that you cannot use the SD card interface, nor can you use the touch screen controller. (As an aside, the ITDB02 Mega Shield does support concurrent SD card and touch capability along with the LCD, but that is a different format shield.)

If you really wish to have pins left over and outstanding performance, the **SolderCore LCD Shield** uses only a few pins, has a 2.8" LCD with touchscreen, and the firmware for the 400 MHz XMOS XS1-L1 device on board is fully open source.

Resources

This is the ITDB02 shield; currently only the v2 shield is offered for sale and is not compatible with the 3.2WD and SolderCore.

http://iteadstudio.com/store/index.php?main_page=product_info&products_id=149

This is the display module, ITead Studio part number DIS012:

http://iteadstudio.com/store/index.php?main_page=product_info&products_id=263

A reference for all the displays offered by ITead Studio is found here:

ITead Studio LCDs

ITead Studio ITDB02-4.3 LCD Module

Installation

```
INSTALL "ITEAD-STUDIO-ITDB02-4.3"
INSTALL "ITDB02-4.3"
```

Options

None.

Description

Installing the ITDB02 shield with the ITDB02-4.3 LCD Module provides a 480×272 color graphic display *but no touch screen input*.

Setup

Using the ITDB02-4.3 is unreliable with the ITDB02 v1.2 shield and is incompatible with the ITDB02 v2 shield, so you need to connect the ITDB02-4.3 module to the SolderCore headers directly.

Unfortunately, driving the LCD in 16-bit mode utilizes all the pins on the SolderCore, leaving nothing free. A disappointing consequence is that you cannot use the SD card interface, nor can you use the touch screen controller. (As an aside, the ITDB02 Mega Shield does support concurrent SD card and touch capability along with the LCD, but that is a different format shield.)

If you really wish to have pins left over and outstanding performance, the **SolderCore LCD Shield** uses only a few pins, has a 2.8" LCD with touchscreen, and the firmware for the 400 MHz XMOS XS1-L1 device on board is fully open source.

Resources

This is the display module:

http://www.itead-europe.com/index.php/display/tft-lcm/itdb02-4-3.html

A reference for all the displays offered by ITead Studio and for hand-wiring this display is found here:

ITead Studio LCDs

ITead Studio ITDB02-5.0 LCD Module

Installation

```
INSTALL "ITEAD-STUDIO-ITDB02-5.0"
INSTALL "ITDB02-5.0"
```

Options

None.

Description

Installing the ITDB02 shield with the ITDB02-5.0 LCD Module provides a 780×480 color graphic display *but no touch screen input*.

Setup

Using the ITDB02-5.0 is unreliable with the ITDB02 v1.2 shield and is incompatible with the ITDB02 v2 shield, so you need to connect the ITDB02-5.0 module to the SolderCore headers directly.

Unfortunately, driving the LCD in 16-bit mode utilizes all the pins on the SolderCore, leaving nothing free. A disappointing consequence is that you cannot use the SD card interface, nor can you use the touch screen controller. (As an aside, the ITDB02 Mega Shield does support concurrent SD card and touch capability along with the LCD, but that is a different format shield.)

If you really wish to have pins left over and outstanding performance, the **SolderCore LCD Shield** uses only a few pins, has a 2.8" LCD with touchscreen, and the firmware for the 400 MHz XMOS XS1-L1 device on board is fully open source.

Resources

This is the display module:

http://www.itead-europe.com/index.php/display/tft-lcm/itdb02-5-0.html

A reference for all the displays offered by ITead Studio and for hand-wiring this display is found here:

ITead Studio LCDs

Jee Labs LCD Plug

Installation

```
INSTALL "JEE-LABS-LCD-PLUG"
INSTALL "LCD-PLUG"
```

Options

ADDR=*integer*

Set the I2C 8-bit address of the MCP23028 bus expander. By default the driver uses the address `0x48`.

Description

Installing the LCD Plug driver provides a character-based LCD display with up to four lines and up to 40 characters per line.

Note that LCD displays that are 4×40 characters are composed of two separate HD44780s which share a common bus but have two select (or enable) signals to allow selection of each. It is not possible for the Jee Plug to drive these displays because it does not support two chip selects on the 16-pin LCD header.

Properties

Dimensions		
`WIDTH`	Digital R/W	The width of the display, in character positions.
`HEIGHT`	Digital R/W	The height of the display, in character lines.
Cursor		
X or `COL`	Digital R/W	The *x* co-ordinate of the cursor.
Y or `ROW`	Digital R/W	The *y* co-ordinate of the cursor.
`CURSOR`	Digital Write	Cursor control. Writing `0` to `CURSOR` turns the cursor off, and writing `1` turns it on.
`LINE`	String R/W	As Y and `ROW`.
`POS`	Digital R/W	An array containing the *x* and *y* co-ordinates of the cursor.
Control		
`LIGHT`	Digital Write	Backlight control. Writing `0` to `LIGHT` turns the backlight off, if the display has a controllable backlight, and writing `1` turns it on.

Character I/O		
`LINE(n)`	String R/W	As `Y(n)` and `ROW(n)`.
`Y(n)` or `ROW(n)`	String R/W	When read, returns the string being displayed on line *n* of the display. When written, overwrites the whole of line *n* of the display, filling the line with extra spaces if necessary.
`CENTER(n)`	String Write	Centers the string written on display line *n*. If the string is too long for the display, only the central part is displayed. If the string is narrower than the display width, it is padded left and right with spaces to the display width such that it lies central within the display.
`RIGHT(n)`	String Write	Right-justifies the string written on display line *n*. If the string is too long for the display, it is truncated on the left such that the rightmost part of the string covers the whole line. If the string is narrower than the display width, it is padded left with spaces to the display width such that it lies adjusted right on the display.

Resources

http://jeelabs.com/products/lcd-plug

Jimmie Rodgers LoL Shield

Installation

```
INSTALL "JIMMIE-RODGERS-LOL-SHIELD"
INSTALL "LOL-SHIELD"
```

Options

None.

Description

Installing the LoL Shield Shield driver provides a 14×9 monochrome graphic display using the LEDs on the shield. You can use all the CoreBASIC graphics commands to drive the LED display. In addition to this, this driver offers an additional set of properties to directly address individual rows and columns of the display, and the display as a whole, to ease animation and special effects.

Properties

Control		
POLARITY	Digital R/W	Display polarity. By default, the display is such that the LEDs light when writing in color *1* with POLARITY set to 0. Setting POLARTY to 1 inverts the display, keeping all displayed content unchanged. Using this property, you can easily flash the display by simply altering the display polarity or write in "inverse video".
FREQUENCY	Digital R/W	Display scanning frequency. To reduce flicker, at the expense of higher CPU overhead, you can increase the scanning frequency. To reduce CPU overhead, at the expense of increased flicker, decrease the scanning frequency. By default, the frequency is set at 600 Hz; values assigned to FREQUENCY are clamped to lie in the range 100 to 1,000.
Direct Output		

FRAME	Digital R/W	Display frame dump. When read, FRAME property returns an array of 9 numbers corresponding to the pixel state of each row of LEDs. The elements of the array are binary coded with bit 0 corresponding to the leftmost LED, and bit 13 corresponding to the rightmost LED. When written, the whole display frame is updated with the contents of the array assigned.
ROW(n)	Digital R/W	Binary-coded row data. When read, returns the row data for row *n*, with bit 0 corresponding to the leftmost LED, and bit 13 corresponding to the rightmost LED. When written, updates row *n* with the assigned binary-coded row data.
COL(n)	Digital R/W	Binary-coded column data. When read, returns the column data for column *n*, with bit 0 corresponding to the topmost LED, and bit 8 corresponding to the bottommost LED. When written, updates column *n* with the assigned binary-coded column data.

Scrolling message example

This application scrolls a simple message across the display, then flashes a message using the POLARITY property.

```
***../examples/lol-message.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "lol-message" or |lol-message.

Random dissolve example

This runs a few random dissolves on the display:

```
***../examples/lol-dissolve.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "lol-dissolve" or |lol-dissolve.

Static noise example

This shows how to assign the whole frame at once to produce a display of random "static".

```
***../examples/lol-static.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "lol-static"` or `|lol-static`.

Resources

This is the original from Jimmie Rodgers:

http://jimmieprodgers.com/kits/lolshield/

The original shield offered by Jimmie Rodgers needs to be hand-soldered, and that takes a long time. However, there are other shields on the market that come pre-assembled.

Compatible hardware is available, pre-assembled and tested, from Olimex:

http://www.olimex.com/dev/shield-lol.html

Of all the shields offered on the market now, the one with the best uniform brightness across all LEDs is the Olimex LoL Shield with SMD LEDs, the `SHIELD-LOL-SMT`. The LEDs on this shield are not as bright as other shields, but it produces a lovely uniform display.

Beware: the Olimex LoL Shield with 3mm green LEDs is so bright it could permanently burn an image into your retina!

Kionix KXP84 Driver

Installation

```
INSTALL "KIONIX-KXP84"
INSTALL "KXP84"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x30`; to configure the alternate address, use `ADDR=0x32`.

Description

Installs an accelerometer driver for the KXP84. The KXP84 has a fixed 2*g* range and a fixed bandwidth defined by a set of capacitors connected to the accelerometer.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		

RANGE	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer GAIN and BIAS to the defaults for the selected range.
BANDWIDTH	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz. This reads as 0 as the bandwidth is not configurable and not documented in the datasheet.
Calibration		
BIAS	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
GAIN	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
WORD(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

Kionix KXTF9 Driver

Installation

```
INSTALL "KIONIX-KXTF9"
INSTALL "KXTF9"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x1E`.

Description

Installs an accelerometer driver for the KXTF9. The KXP84 supports 2*g*, 4*g*, and 8*g* ranges with a fixed bandwidth.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		
`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.

BANDWIDTH	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz. This reads as 0 as the bandwidth is not configurable.
Calibration		
BIAS	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
GAIN	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
WORD(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

Linear Technology LTC2309 Driver

Installation

```
INSTALL "LINEAR-TECHNOLOGY-LTC2309"
INSTALL "LTC2309"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x10`.

Description

Installs an 8-channel ADC driver for the Linear Techonology LTC2309 analog to digital converter. Each channel is configured as a unipolar input.

Properties

Configuration		
RESOLUTION	Digital R	The sample resolution, in bits. This is fixed at 12.
Measurement		
A0 … A7	Analog Read	Measures the voltage on one of the analog channels 0 through 7. A*n* is equivalent to A(*n*).
A(n)	Analog Read	Measure the voltage on analog channel *n*. The value returned will range between 0 and +1 for full scale.
A	Analog Read	Measures the voltage on all eight channels sequentially and returns the measurements as an array of eight numbers.

Linear Technology LTC6904 Driver

Installation

```
INSTALL "LINEAR-TECHNOLOGY-LTC6904"
INSTALL "LTC6904"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x2E`; to configure the alternate address, use `ADDR=0x2C`.

Description

Installs an digital oscillator driver for the LTC6904. The LTC6904 can be programmed to oscillate between 1 kHz and 68 MHz.

Properties

Output		
`FREQUENCY`	Analog R/W	Selected oscillator frequency in hertz. Writing to this property will select the frequency nearest to the written value. When reading this property, the value read will be the oscillator frequency selected by the driver as the nearest frequency to the requested frequency.
`MODE`	Digital R/W	The selected oscillator mode. Mode 0 drives both CLK and nCLK with nCLK = –CLK; mode 1 drives nCLK only; mode 2 drives CLK only; and mode 3 powers down the oscillator.

Liquidware Input Shield

Installation

```
INSTALL "LIQUIDWARE-INPUT-SHIELD"
INSTALL "INPUT-SHIELD"
```

Options

`MODE=A`

Configures the driver for an InputShield with the mode switch in the A position.

`MODE=B`

Configures the driver for an InputShield with the mode switch in the B position.

Description

Installs a driver for the analog joystick and digital buttons on the InputShield. What makes the InputShield slightly different from many other joystick shield is the vibration motor for *haptic feedback*, also known as a *rumble function*.

When delivered, the InputShield's joystick outputs are configured to vary between 0V and +5V as the joystick's potentiometers are tied to the 5V rail. In order to use this shield with SolderCore, you need a simple hardware modification:

- Snip the 5V pin from the underside of the shield.
- Strap the 5V pin to the 3V3 pin next to it on the topside of the shield using a solder bridge or a small length of wire.

With that done, the InputShield works perfectly with SolderCore.

Properties

Buttons		
A	Digital Read	Sense A button. This property reads `1` when the A button is pressed and `0` when not.
B	Digital Read	Sense B button. This property reads `1` when the B button is pressed and `0` when not.
Analog Joystick		
PRESS	Digital Read	This property reads `1` when the joystick is pressed down and continues to read `1` until the joystick is released. It reads `0` when it is not pressed.

SELECT	Digital Read	As PRESS.
H	Analog Read	Sense horizontal joystick position; −1 is fully left, +1 is fully right, and 0 is centered.
V	Analog Read	Sense vertical joystick position; −1 is fully down, +1 is fully up, and 0 is centered.
POS	Analog Read	A complex number where the real component is the horizontal position of the joystick and the imaginary component is the vertical position of the joystick.
Vibration Motor		
MOTOR	Digital Write	Controls the vibration motor. When written to 0, the motor is turned off. When written to any nonzero value, the motor is turned on.

Resources

http://www.liquidware.com/shop/show/INPT/InputShield

Matrix Keyboard Driver

Installation

`INSTALL "MATRIX-KEYBOARD" USING` *rowbus*`,` *colbus*

Options

None.

Description

Installs a generic matrix keyboard driver using the two buses *rowbus* and *colbus* to scan the keyboard. The width of the buses *rowbus* and *colbus* must match the electrical "rectangular" layout of the matrix.

The matrix keyboard is scanned by driving a single row bus signal low on *rowbus* and then sensing the column outputs using *colbus*, for each row in the matrix. Each row on *rowbus* corresponds to a single row in the matrix, and each column on *colbus* corresponds to a single column in the matrix. Of course, the physical layout of they keys need to be taken into consideration, so the above simply reflects the *logical* layout of the keyboard matrix.

The position of a key on row *r*, column *c* is denoted (*r*, *c*). Both row and column numbers start from zero.

Scan codes for the matrix start from `1`. The scan code corresponding to the position (*r*, *c*), with *n* columns per row, is:

$r \times n + c + 1$

Hence, position (0, 0) corresponds to scan code 1. Scan code 0 is reserved to indicate a "no key pressed" condition.

Properties

Layout		
`ROW`	Digital Read	Number of rows in the keyboard matrix, and is equivalent to the `WIDTH` property of *rowbus*.
`COL`	Digital Read	Number of columns in the keyboard matrix, and is equivalent to the `WIDTH` property of *colbus*.
Scanning		
KEY	Digital Read	The scan code of the first key pressed in the matrix, scanning in row-wise and column-wise order from row 0, column 0. If no key is detected as pressed, `0` is output.

`ROW(r)`	Digital Read	Select row *r* and return the column sense. You can use this to see which keys are held down on an individual row. Bit *c* of the output is set to one if the key at position (*r*, *c*) in the matrix is held down.

Example

The following installs a matrix keyboard driver for the ITead Studio Ibridge Lite, which features a 3×3 matrix keypad. The row selects are connected to D2 through D4 and the column senses are connected to D5 through D7.

```
INSTALL "PARALLEL-BUS" USING CORE.D2, CORE.D3, CORE.D4 AS ROWBUS
INSTALL "PARALLEL-BUS" USING CORE.D5, CORE.D6, CORE.D7 AS COLBUS
INSTALL "MATRIX-KEYBOARD" USING ROWBUS, COLBUS AS KEYPAD
PRINT "Ready to scan keyboard.  Press some keys.  Key 9 to exit."
REPEAT
  SCAN = KEYPAD.INPUT
  IF SCAN THEN PRINT "Scan = "; SCAN
UNTIL SCAN = 9
END
```

MaxDetect DHT and RHT Driver

Installation (DHT21, RHT03)

```
INSTALL "MAXDETECT-DHT21" USING data
INSTALL "DHT21" USING data
```

Installation (DHT11)

```
INSTALL "MAXDETECT-DHT11" USING data
INSTALL "DHT11" USING data
```

Options

None.

Description

Installs a humidity and temperature sensor driver for MaxDetect DHT11 or DHT21 devices.

As the one-wire protocol for these devices is timing sensitive, it is not possible to install the DHT21 using a bus expander because the bus expander cannot drive and sense signals fast enough to meet the timing requirements of the MaxDetect one-wire protocol.

Properties

Measurement		
TEMP	Analog Read	Temperature in degrees Celsius.
HUMIDITY	Analog Read	Relative humidity, in percent.
DEWPOINT	Analog Read	Dew point in degrees Celsius, computed from the temperature TEMP and relative humidity HUMIDITY.
DATA	Analog Read	An array of three values: the temperature, in degrees Celsius, the relative humidity in percent, and the dewpoint in degrees Celsius.

Maxim DS1340 driver

Installation

```
INSTALL "MAXIM-DS1340"
INSTALL "DS1340"
INSTALL "RTC-PLUG"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0xD0`.

Description

Installs a real time clock driver using the DS1340. The DS1340 is a battery-backed real time clock that continues marking time while the SolderCore is turned off. You can set the system time from the RTC time, or the RTC time from the system, network, or GPS time if you wish to synchronize clocks.

Note

The DS1340 does not store the year, it only stores the year within the century. Because of this, the DS1340 driver assumes all dates are in the 21st century and that YEAR will be assigned a value between 2000 and 2099.

This driver takes care of the necessary BCD to binary and binary to BCD conversions that are necessary when reading and writing the RTC registers using the high-level properties. You can read the raw BCD values by using the low-level PEEK and POKE properties.

Properties

Unix time		
`TIME`	Digital R/W	Current time as seconds elapsed since 1 January 1970, the standard way of representing time in CoreBASIC, read atomically from the RTC. You can set the system time used by CoreBASIC by assigning this to `CORE.TIME`, or you can set the RTC time by assigning the time from another source (system, network, GPS) to the RTC. When assigning to `TIME`, the RTC day register will be updated to the correct day of the week derived from the assigned date.
Time		

SECOND	Digital R/W	Current second, 0 to 59.
MINUTE	Digital R/W	Current minute, 0 to 59.
HOUR	Digital R/W	Current hour, 0 to 23.
Date		
DAY	Digital Read	Current day within week, 1 to 7, with 1 as Sunday. This is computed from the date contained in the RTC registers and is not read from the RTC day register. If you wish to read the RTC day register directly, you can use the access property PEEK(3).
DATE	Digital R/W	Current date (within month), 1 to 31.
MONTH	Digital R/W	Current month, 1 to 12.
YEAR	Digital R/W	Current year, 2000 to 2099.
Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.

Maxim MAX6675 Driver

Installation

```
INSTALL "MAXIM-MAX6675" USING select
INSTALL "MAX6675" USING select
```

Options

None.

Description

Installs a temperature sensor driver for the Maxim MAX6675. The MAX6675 is an SPI K-type thermocouple sensor. The USING clause specifies the device select signal to select the MAX6675.

Properties

Sensors		
TEMP	Analog Read	Returns the temperature in degrees Celsius. If the thermocouple is detected as open, the value returned is NaN.

Microchip MCP23008 Driver

Installation

```
INSTALL "MICROCHIP-MCP23008"
INSTALL "MCP23008"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x48`.

Description

Installs an 8-bit parallel bus driver for the Microchip MCP23008.

Properties

Configuration		
`WIDTH`	Digital Read	The width of the bus, in bits. Reads as 8.
`MASK`	Digital Read	An integer with each bus bit set to one. The mask for a bus of width *n* is 2^n-1. Reads as `0xFF`.
Bus Output		
`D0` … `D7`	Digital R/W	Reads or writes individual bits on the bus. When written, the bus direction for that bit (only) is set to output, and the state of the bit is written. When read, the bus direction for that bit (only) is set to input and the state of the pin is read. Note that writing to `D0` is the same as to writing to `OUTPUT(0)` and reading from `D0` is the same as reading from `INPUT(0)`, and so on.
`OUTPUT`	Digital R/W	Reads or writes the output latches. When written, sets the entire bus to output and writes data to the latches which drive the pins. When read, returns the data set in the latches, *not the pins*, and *does not set the bus to input*. If you need to read the state of the pins, use `INPUT`.

`OUTPUT(n)`	Digital R/W	Reads or writes an individual output latch. When written, sets the bus direction for pin *n* to output and writes latch bit *n* which then drives the state of the pin. When read, returns the data set for latch *n*, *not the pin*, and *does not set bus pin n to input*. If you need to read the state of the an individual pins, use `INPUT(n)`.
`INPUT`	Digital Read	Sets the entire bus to input and reads the current state of the pins.
`INPUT(n)`	Digital Read	Sets the bus direction for pin *n* to input and reads the state of the pin.
`DATA`	Digital R/W	Reads or writes the entire bus. When written, it acts as the `OUTPUT` property and when read it acts as the `INPUT` property.
`DATA(n)`	Digital R/W	Reads or writes an individual pin. When written, it acts as the `OUTPUT(n)` property and when read it acts as the `INPUT(n)` property.
`DIRECTION`	Digital Read	Reads the entire bus direction. Port pins set to input mode read as `1`, port pins set to output mode read as `0`.
`DIRECTION(n)`	Digital Read	Reads the bus direction for pin *n*. Port pins set to input mode read as `1`, port pins set to output mode read as `0`.

Example

This example shows hot to use the MCP23008 bus expander which controls an HD44780 to provide a 16×2 character-based LCD display.

```
' The LCD plug is controlled by a MCP23008 bus expander.
INSTALL "MICROCHIP-MCP23008" AS EXPANDER

' The LCD operates in 4-bit mode with the four data bus bits connected
' to bits 0 through 3 of the port expander.  This creates a 4-bit bus
' using four bits on the 8-bit port expander.
INSTALL "PARALLEL-BUS" USING EXPANDER.D0, EXPANDER.D1, EXPANDER.D2, EXPANDER.D3 AS LCDBUS

' Install the HD44780 driver.  We use the 4-bit bus constructed above,
' the R/S signal (connected to bit 4 of the port expander), and the E signal
' (connected to bit 6 of the port expander).  The R/W signal is not used,
```

```
' so we don't specify it when installing this driver.
INSTALL "HITACHI-HD44780" USING LCDBUS, EXPANDER.D4, EXPANDER.D6 AS LCD

' We now have the LCD initialized and ready to work.
LCD.ROW = 1 : LCD.COLUMN = 1
LCD.OUTPUT = "Hello, world!"

' We can turn the cursor on and off.
LCD.ROW = 2 : LCD.COLUMN = 1
LCD.CURSOR = 1
```

This is, however, equivalent to:

```
INSTALL "JEE-LABS-LCD-PLUG" AS LCD
```

Microchip MCP23016 Driver

Installation

```
INSTALL "MICROCHIP-MCP23016"
INSTALL "MCP23016"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x48`.

Description

Installs an 8-bit parallel bus driver for the Microchip MCP23016.

Properties

Configuration		
`WIDTH`	Digital Read	The width of the bus, in bits. Reads as `16`.
`MASK`	Digital Read	An integer with each bus bit set to one. The mask for a bus of width *n* is 2^n-1. Reads as `0xFFFF`.
Bus Output		
`D0` … `D15`	Digital R/W	Reads or writes individual bits on the bus. When written, the bus direction for that bit (only) is set to output, and the state of the bit is written. When read, the bus direction for that bit (only) is set to input and the state of the pin is read. Note that writing to `D0` is the same as to writing to `OUTPUT(0)` and reading from `D0` is the same as reading from `INPUT(0)`, and so on.
`OUTPUT`	Digital R/W	Reads or writes the output latches. When written, sets the entire bus to output and writes data to the latches which drive the pins. When read, returns the data set in the latches, *not the pins*, and *does not set the bus to input*. If you need to read the state of the pins, use `INPUT`.

`OUTPUT(n)`	Digital R/W	Reads or writes an individual output latch. When written, sets the bus direction for pin *n* to output and writes latch bit *n* which then drives the state of the pin. When read, returns the data set for latch *n*, *not the pin*, and *does not set bus pin n to input*. If you need to read the state of the an individual pins, use `INPUT(n)`.
`INPUT`	Digital Read	Sets the entire bus to input and reads the current state of the pins.
`INPUT(n)`	Digital Read	Sets the bus direction for pin *n* to input and reads the state of the pin.
`DATA`	Digital R/W	Reads or writes the entire bus. When written, it acts as the `OUTPUT` property and when read it acts as the `INPUT` property.
`DATA(n)`	Digital R/W	Reads or writes an individual pin. When written, it acts as the `OUTPUT(n)` property and when read it acts as the `INPUT(n)` property.
`DIRECTION`	Digital Read	Reads the entire bus direction. Port pins set to input mode read as `1`, port pins set to output mode read as `0`.
`DIRECTION(n)`	Digital Read	Reads the bus direction for pin *n*. Port pins set to input mode read as `1`, port pins set to output mode read as `0`.
Split bus		
`BUS(n)`	Digital R/W	Write eight bits to the sub-bus *n*. If *n* is zero, addresses bits `D0` through `D7`, and if *n* is one, addresses bits `D8` through `D15`.

Microchip MCP23017 Driver

Installation

```
INSTALL "MICROCHIP-MCP23017"
INSTALL "MCP23017"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x48`.

Description

Installs an 16-bit parallel bus driver for the Microchip MCP23017. The low eight bits of the bus are mapped to port A of the MCP23017, and the high eight bits are mapped to port B.

Properties

Configuration		
`WIDTH`	Digital Read	The width of the bus, in bits. Reads as `16`.
`MASK`	Digital Read	An integer with each bus bit set to one. The mask for a bus of width *n* is 2^n-1. Reads as `0xFFFF`.
Bus Output		
`D0` … `D15`	Digital R/W	Reads or writes individual bits on the bus. When written, the bus direction for that bit (only) is set to output, and the state of the bit is written. When read, the bus direction for that bit (only) is set to input and the state of the pin is read. Note that writing to `D0` is the same as to writing to `OUTPUT(0)` and reading from `D0` is the same as reading from `INPUT(0)`, and so on.
`OUTPUT`	Digital R/W	Reads or writes the output latches. When written, sets the entire bus to output and writes data to the latches which drive the pins. When read, returns the data set in the latches, *not the pins*, and *does not set the bus to input*. If you need to read the state of the pins, use `INPUT`.

`OUTPUT(n)`	Digital R/W	Reads or writes an individual output latch. When written, sets the bus direction for pin *n* to output and writes latch bit *n* which then drives the state of the pin. When read, returns the data set for latch *n*, *not the pin*, and *does not set bus pin n to input*. If you need to read the state of the an individual pins, use `INPUT(n)`.
`INPUT`	Digital Read	Sets the entire bus to input and reads the current state of the pins.
`INPUT(n)`	Digital Read	Sets the bus direction for pin *n* to input and reads the state of the pin.
`DATA`	Digital R/W	Reads or writes the entire bus. When written, it acts as the `OUTPUT` property and when read it acts as the `INPUT` property.
`DATA(n)`	Digital R/W	Reads or writes an individual pin. When written, it acts as the `OUTPUT(n)` property and when read it acts as the `INPUT(n)` property.
`DIRECTION`	Digital Read	Reads the entire bus direction. Port pins set to input mode read as `1`, port pins set to output mode read as `0`.
`DIRECTION(n)`	Digital Read	Reads the bus direction for pin *n*. Port pins set to input mode read as `1`, port pins set to output mode read as `0`.
Split bus		
`BUS(n)`	Digital R/W	Write eight bits to the sub-bus *n*. If *n* is zero, addresses bits `D0` through `D7`, and if *n* is one, addresses bits `D8` through `D15`.

Microchip MCP342x Driver

Installation

```
INSTALL "MICROCHIP-MCP342X"
INSTALL "MCP342X"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0xD0`.

Description

Installs an 4-channel ADC driver for the Microchip MCP342x family of analog to digital converters. This driver will work with the MCP3424, MCP3423, and MCP3422.

Properties

Configuration		
`AMPLIFIER`	Digital R/W	Reads or writes the the gain of the programmable gain amplifier. The value assigned is the gain to select: writing 1, 2, 4, or 8 selects 1×, 2×, 4×, or 8× gain for subsequent measurements. 1× gain is selected by default. The programmable gain amplifier enables you to measure very small voltages with high resolution.
`RESOLUTION`	Digital R/W	Reads or writes the sample resolution. The value assigned is the sample resolution in bits. Writing 12, 14, 16, or 18 selects that many bits for subsequent measurements. Note that higher resolutions take longer when measuring: 12 bit resolution requires approximately 4 microseconds for a conversion and 18 bit resolution requires 267 microseconds.
Measurement		
`A0` … `A3`	Analog Read	Measures the voltage on one of the analog channels 0 through 3. A*n* is equivalent to A(*n*).

A(n)	Analog Read	Measure the voltage on analog channel *n* using the selected gain and resolution. The value returned will range between −1 and +1 for full scale which represents a full scale swing of −2.048 to +2.048 volts across channel *n* inputs.
A	Analog Read	Measures the voltage on all four channels sequentially using the selected gain and resolution and returns the measurements as an array of four numbers. Note that using this property will attempt to read all four channels even for the MCP3423 and MCP3422 which have only two channels: the third and fourth values in the array are undefined in this case.

Microchip MCP4725 Driver

Installation

```
INSTALL "MICROCHIP-MCP4725"
INSTALL "MCP4725"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0xC0`.

Description

Installs an single-channel DAC driver for the Microchip MCP4725 digital to analog converter.

Properties

Measurement		
`A0`	Analog Write	Writes the DAC. 0 sets the DAC to 0 volts and 1 sets the DAC to VREF volts, with values
`A(n)`	Analog Write	Implements only `A(0)` write and is identical to the `A0` property. Other values of *n* will throw a subscript error.

Microchip TC77 Driver

Installation

```
INSTALL "MICROCHIP-TC77" USING select
INSTALL "TC77" USING select
INSTALL "CORE-TEMP" USING select
```

Options

None.

Description

Installs an temperature sensor driver using the TC77 on the CoreTemp SExI module which mounts into a SenseCore:

http://www.soldercore.com/products/sensecore/coretemp/

Because the TC77 is an SPI device, the `USING` clause must specify a single-bit GPIO that controls the chip select connected to the TC77 device. Although this is intended to fit into an SenseCore, you can use it as a standard breakout module and mount it wherever you wish or use it with something other than a SolderCore.

Properties

Temperature Sensor		
TEMP	Analog Write	Reads the current temperature, in degrees Celsius.

Example

```
***../examples/coretemp-demo.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "coretemp-demo"` or `|coretemp-demo`.

Modkit MotoProto Shield

Installation

```
INSTALL "MODKIT-MOTOPROTO-SHIELD"
INSTALL "MOTOPROTO-SHIELD"
INSTALL "MOTOPROTO"
```

Options

None.

Description

The Modkit MotoProto Shield can control two DC motors and has an interface for a standard HD44780-based display.

Properties

Motor Drive		
`MOTOR(n)`	Analog Write	Sets the speed and direction of channel *n*. The value assigned to the property should lie between −1 and 1; values outside this range are clamped. The sign of the value written determines whether the motor runs forward or in reverse.
`LEFT` or `A`	Analog Write	Convenience property. Refers directly to channel 0, so `LEFT` is identical to `MOTOR(0)`.
`RIGHT` or `B`	Analog Write	Convenience property. Refers directly to channel 1, so `RIGHT` is identical to `MOTOR(0)`.
Button		
`PRESS`	Digital Read	Sense button SW2 connected to D12. This property reads `1` when the button is pressed and `0` when not.

LCD Properties

Dimensions		
`WIDTH`	Digital R/W	The width of the display, in character positions.
`HEIGHT`	Digital R/W	The height of the display, in character lines.

Cursor		
X or COL	Digital R/W	The *x* co-ordinate of the cursor.
Y or ROW	Digital R/W	The *y* co-ordinate of the cursor.
CURSOR	Digital Write	Cursor control. Writing 0 to CURSOR turns the cursor off, and writing 1 turns it on.
LINE	String R/W	As Y and ROW.
POS	Digital R/W	An array containing the *x* and *y* co-ordinates of the cursor.
Control		
LIGHT	Digital Write	Backlight control. Writing 0 to LIGHT turns the backlight off, if the display has a controllable backlight, and writing 1 turns it on.
Character I/O		
LINE(n)	String R/W	As Y(n) and ROW(n).
Y(n) or ROW(n)	String R/W	When read, returns the string being displayed on line *n* of the display. When written, overwrites the whole of line *n* of the display, filling the line with extra spaces if necessary.
CENTER(n)	String Write	Centers the string written on display line *n*. If the string is too long for the display, only the central part is displayed. If the string is narrower than the display width, it is padded left and right with spaces to the display width such that it lies central within the display.
RIGHT(n)	String Write	Right-justifies the string written on display line *n*. If the string is too long for the display, it is truncated on the left such that the rightmost part of the string covers the whole line. If the string is narrower than the display width, it is padded left with spaces to the display width such that it lies adjusted right on the display.

Resources

http://www.sparkfun.com/products/10018

National Semiconductor LM75 Driver

Installation

```
INSTALL "NATIONAL-SEMICONDUCTOR-LM75"
INSTALL "NATSEMI-LM75"
INSTALL "LM75"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x90`.

Description

Installs a temperature sensor driver for the LM75.

Properties

Measurement		
TEMP	Analog Read	Temperature in degrees Celsius.
Configuration		
RESOLUTION	Analog R/W	The selected resolution of the temperature sensor. The LM75 has a fixed resolution of 0.5 degrees Celsius, writing this property has no effect on the selected resolution of the sensor.
Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
WORD(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

Nintendo Classic Controller

Installation

```
INSTALL "NINTENDO-CLASSIC-CONTROLLER"
INSTALL "CLASSIC-CONTROLLER"
INSTALL "CLASSIC"
```

Options

None.

Description

Installs a driver for the Nintendo Classic Controller. The SolderCore WiiChuk Adapter fits into a SenseCore SExI socket and allows you to attach either a Nintendo Classic Controller or a Nintendo Nunchuk to the SolderCore.

The Classic Controller communicates over I2C using the fixed address `0xA4`. Because all controllers on an I2C bus must have distinct addresses, it means that you can only have a single Nintendo Classic Controller or Nintendo Nunchuk attached to an I2C bus.

In order for the Classic Controller to work, *it is imperative* that you ensure I2C pull-ups are fitted to the I2C bus. By default, the SenseCore provides I2C pull-ups, but if you are using some other adapter, such as the Wiichuk, *you must ensure those pull-ups are in place*. If you don't have pull-ups, communication with the Classic Controller or Nunchuk will be unreliable or will hang.

Properties

Digital buttons		
A	Digital Read	Sense A button. This property reads `1` when the A button is pressed and `0` when not.
B	Digital Read	Sense B button. This property reads `1` when the B button is pressed and `0` when not.
X	Digital Read	Sense X button. This property reads `1` when the X button is pressed and `0` when not.
Y	Digital Read	Sense Y button. This property reads `1` when the Y button is pressed and `0` when not.
L	Digital Read	Sense L button. This property reads `1` when the L button is pressed and `0` when not.

R	Digital Read	Sense R button. This property reads 1 when the R button is pressed and 0 when not.
ZL	Digital Read	Sense ZL button. This property reads 1 when the ZL button is *fully* pressed and 0 when not.
ZR	Digital Read	Sense ZR button. This property reads 1 when the ZR button is *fully* pressed and 0 when not.
Control buttons		
HOME	Digital Read	Sense Home button. This property reads 1 when the Home button is pressed and 0 when not.
SELECT	Digital Read	Sense Select button. This property reads 1 when the Select button is pressed and 0 when not.
START	Digital Read	Sense Start button. This property reads 1 when the Start button is pressed and 0 when not.
Digital pad		
UP	Digital Read	Sense joypad Up button. This property reads 1 when the Up button is pressed and 0 when not.
DOWN	Digital Read	Sense joypad Down button. This property reads 1 when the Down button is pressed and 0 when not.
LEFT	Digital Read	Sense joypad Left button. This property reads 1 when the Left button is pressed and 0 when not.
RIGHT	Digital Read	Sense joypad Right button. This property reads 1 when the Left button is pressed and 0 when not.
Analog joysticks		
H	Analog Read	Sense horizontal positions of all joysticks; returns an array of four positions for the two analog and two digital joysticks. −1 is fully left, +1 is fully right, and 0 is centered.
H(n)	Analog Read	Sense horizontal position of joystick *n*. −1 is fully left, +1 is fully right, and 0 is centered.

V	Analog Read	Sense vertical positions of all joysticks; returns an array of four positions for the two analog and two digital joysticks. −1 is fully down, +1 is fully up, and 0 is centered.
V(n)	Analog Read	Sense horizontal position of joystick *n*. −1 is fully down, +1 is fully up, and 0 is centered.
POS	Analog Read	Sense position of all joysticks. Returns an array of four elements where each element is a complex number. The complex number's real component is the horizontal position of the joystick and the imaginary component is the vertical position of the joystick.
POS(n)	Analog Read	Sense position of joystick *n*. Returns a complex number. The complex number's real component is the horizontal position of the joystick and the imaginary component is the vertical position of the joystick.
Controller memory		
PEEK(n)	Digital Read	Reads the 8-bit contents of memory location *n* on the extension controller.
POKE(n)	Digital Write	Write the 8-bit contents of memory location *n* on the extension controller.
BYTE(n)	Digital R/W	Reads or writes the 8-bit contents of memory location *n* on the extension controller.

Resources

This work couldn't have been possible without the pioneering effort of the WiiBrew community.

http://wiibrew.org/wiki/Wiimote/Extension_Controllers

The following are the formats used by original Nintendo controllers:

http://wiibrew.org/wiki/Wiimote/Extension_Controllers/Nunchuck
http://wiibrew.org/wiki/Wiimote/Extension_Controllers/Classic_Controller
http://wiibrew.org/wiki/Wiimote/Extension_Controllers/Wii_Motion_Plus

Nintendo Nunchuk Controller

Installation

```
INSTALL "NINTENDO-NUNCHUK-CONTROLLER"
INSTALL "NUNCHUK-CONTROLLER"
INSTALL "NUNCHUK"
```

Options

None.

Description

Installs a driver for the Nintendo Nunchuk Controller. The SolderCore WiiChuk Adapter fits into a SenseCore SExI socket and allows you to attach a Nintendo Classic Controller, a Nintendo Nunchuk, or a WiiMotion Plus to the SolderCore.

The Nunchuk Controller communicates over I2C using the fixed address `0xA4`. Because all controllers on an I2C bus must have distinct addresses, it means that you can only have a single Nintendo Classic Controller or Nintendo Nunchuk attached to an I2C bus.

In order for the Nunchuk Controller to work, *it is imperative* that you ensure I2C pull-ups are fitted to the I2C bus. By default, the SenseCore provides I2C pull-ups, but if you are using some other adapter, such as the Wiichuk, *you must ensure those pull-ups are in place*. If you don't have pull-ups, communication with the Classic Controller or Nunchuk will be unreliable or will hang.

Properties

Buttons		
Z	Digital Read	Sense Z button. This property reads `1` when the Z button is pressed and `0` when not.
C	Digital Read	Sense C button. This property reads `1` when the B button is pressed and `0` when not.
Analog Joystick		
H	Analog Read	Sense horizontal joystick position; −1 is fully left, +1 is fully right, and 0 is centered.
V	Analog Read	Sense vertical joystick position; −1 is fully down, +1 is fully up, and 0 is centered.

POS	Analog Read	A complex number where the real component is the horizontal position of the joystick and the imaginary component is the vertical position of the joystick.
PRESS	Digital Read	Returns the state of the Z button in order to emulate a press.
SELECT	Digital Read	Returns the state of the Z button in order to emulate a selection.
Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
BIAS	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
GAIN	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Algorithms		
ROLL	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
PITCH	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Controller memory		
PEEK(n)	Digital Read	Reads the 8-bit contents of memory location *n* on the extension controller.
POKE(n)	Digital Write	Write the 8-bit contents of memory location *n* on the extension controller.

BYTE (n)	Digital R/W	Reads or writes the 8-bit contents of memory location *n* on the extension controller.

NuElectronics 3310 LCD Shield

Installation

```
INSTALL "NUELECTRONICS-3310-LCD-SHIELD"
INSTALL "3310-SHIELD"
```

Options

None.

Description

Installing the 3310 LCD Shield driver provides a 84×38 monochrome graphic display. You can use all the CoreBASIC graphics commands to drive the LCD display.

Resources

http://www.nuelectronics.com/estore/index.php?main_page=product_info&products_id=12

NMEA Parser

Installation

```
INSTALL "NMEA-PARSER"
INSTALL "GPS-PARSER"
INSTALL "GPS"
```

Options

None.

Description

Installs a driver that parses GPS NMEA 0183 sentences to extract the fix and operating data.

The CoreBASIC NMEA 0183 parser will decode the following sentence:

- *GPGGA, Global Positioning System Fix Data*: This contains the time only, along with latitude and longitude.
- *GPRMC, Recommended Minimum Specific GNSS Data*: This contains the time and date along with latitude and longitude.
- *GPGSA, GNSS DOP and Active Satellites*: This contains the satellites used in the navigation solution reported by the GPGGA sentence.

Properties

Engine update		
SENTENCE	String Write	Send sentence to NMEA 0183 parser. The string is parsed by looking for a start-of-sentence character '$', the NMEA 0183 sentence type and data, and the requires checksum. Any data before the sentence start character is ignored. If the checksum presented in the sentence does not match the checksum computed by the parser, the sentence is discarded as invalid. If the sentence checksum is valid, the sentence is parsed as described above.
Position (derived from GPGGA or GPRMC)		

LONGITUDE	Analog Read	Longitude, in degrees. If the longitude is not yet available, as no appropriate sentence has been passed to the parser containing a valid fix, LONGITUDE is read as a NaN.
LATITUDE	Analog Read	Latitude , in degrees. If the longitude is not yet available, as no appropriate sentence has been passed to the parser containing a valid fix, LATITUDE is read as a NaN.
Combined time and date (derived from GPRMC)		
TIME	Digital Read	Current time as seconds elapsed since 1 January 1970, the standard way of representing time in CoreBASIC, computed from the time and date in the GPRMC sentence. You can set the system time used by CoreBASIC by assigning this to CORE.TIME.
Time (derived from GPGGA or GPRMC)		
SECOND	Digital Read	Current second, 0 to 59.
MINUTE	Digital Read	Current minute, 0 to 59.
HOUR	Digital Read	Current hour, 0 to 23.
Date (derived from GPRMC)		
DATE	Digital Read	Current date (within month), 1 to 31.
MONTH	Digital Read	Current month, 1 to 12.
YEAR	Digital Read	Current year, 2000 to 2099.
Solution data (derived from GPGSA)		
SATELLITES	Digital Read	An array of satellite PRN numbers that were used in the last fix.

Example

This example receives NMEA 0183 sentences from a GPS receiver attached to the UART, sends them to the NMEA 0183 parser, and prints the state of the parser after each sentence received.

```
***../examples/nmea-parser-demo.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "nmea-parser-demo" or |nmea-parser-demo.

NuElectronics TFT LCD Shield

Installation

```
INSTALL "NUELECTRONICS-TFT-LCD-SHIELD"
INSTALL "TFT-SHIELD"
```

Options

None.

Description

Installing the TFT LCD Shield driver provides a 240×320 true color graphic display. This driver works on both the 2.4-inch and 2.8-inch LCD displays that NuElectronics offer. You can use all the CoreBASIC graphics commands to drive the LCD display.

Resources

http://www.nuelectronics.com/estore/index.php?main_page=product_info&products_id=30
http://www.nuelectronics.com/estore/index.php?main_page=product_info&products_id=31

NXP PCF8575 Driver

Installation

```
INSTALL "NXP-PCF8575"
INSTALL "PCF8575"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x40`.

Description

Installs a 16-bit parallel bus driver for the NXP PCF8575 (and compatibles).

Properties

Configuration		
`WIDTH`	Digital Write	The width of the bus, in bits. Reads as 8.
`MASK`	Digital Read	An integer with each bus bit set to one. The mask for a bus of width *n* is 2^n-1. Reads as `0xFF`.
Digital I/O		
`D0` … `D7`	Digital R/W	Reads or writes individual bits on the bus. When written, the bus direction for that bit (only) is set to output, and the state of the bit is written. When read, the bus direction for that bit (only) is set to input and the state of the pin is read. Note that writing to `D0` is the same as to writing to `OUTPUT(0)` and reading from `D0` is the same as reading from `INPUT(0)`, and so on.
`OUTPUT`	Digital R/W	Reads or writes the output latches. When written, sets the entire bus to output and writes data to the latches which drive the pins. When read, returns the data set in the latches, *not the pins*, and *does not set the bus to input*. If you need to read the state of the pins, use `INPUT`.

`OUTPUT(n)`	Digital R/W	Reads or writes an individual output latch. When written, sets the bus direction for pin *n* to output and writes latch bit *n* which then drives the state of the pin. When read, returns the data set for latch *n*, *not the pin*, and *does not set bus pin n to input*. If you need to read the state of the an individual pins, use `INPUT(n)`.
`INPUT`	Digital Read	Sets the entire bus to input and reads the current state of the pins.
`INPUT(n)`	Digital Read	Sets the bus direction for pin *n* to input and reads the state of the pin.
`DATA`	Digital R/W	Reads or writes the entire bus. When written, it acts as the `OUTPUT` property and when read it acts as the `INPUT` property.
`DATA(n)`	Digital R/W	Reads or writes an individual pin. When written, it acts as the `OUTPUT(n)` property and when read it acts as the `INPUT(n)` property.
`DIRECTION`	Digital Read	Reads the entire bus direction. Port pins set to input mode read as `1`, port pins set to output mode read as `0`.
`DIRECTION(n)`	Digital Read	Reads the bus direction for pin *n*. Port pins set to input mode read as `1`, port pins set to output mode read as `0`.

Parallel Bus Driver

Installation

`INSTALL "PARALLEL-BUS" USING` *digital-io*`,` *digital-io*...

Options

None.

Description

Installs a generic parallel bus driver using discrete GPIOs. The first `USING` argument corresponds to the least significant bit of the bus, typically called `D0`, the second argument to the next bit on the bus, and so on.

Properties

Configuration		
`WIDTH`	Digital Read	The width of the bus, in bits.
`MASK`	Digital Read	An integer with each bus bit set to one. The mask for a bus of width *n* is 2^n-1. Reads as 0xFF.
Digital I/O		
`D0` … `D15`	Digital R/W	Reads or writes individual bits on the bus. When written, the bus direction for that bit (only) is set to output, and the state of the bit is written. When read, the bus direction for that bit (only) is set to input and the state of the pin is read. Note that writing to `D0` is the same as to writing to `OUTPUT(0)` and reading from `D0` is the same as reading from `INPUT(0)`, and so on.
`OUTPUT`	Digital Write	Sets the bus to output and writes data to the digital I/Os comprising the bus.
`INPUT`	Digital Read	Sets the bus to input and reads the digital I/Os comprising the bus.
`DATA`	Digital R/W	Reads or writes the bus. When written, it acts as the `OUTPUT` property and when read it acts as the `INPUT` property.

`OUTPUT(n)`	Digital Write	Sets the bus direction for pin *n* to output and writes the state of that pin.
`INPUT(n)`	Digital Read	Sets the bus direction for pin *n* to input and reads the state of that pin.
`DATA(n)`	Digital R/W	Reads or writes an individual pin. When written, it acts as the `OUTPUT(n)` property and when read it acts as the `INPUT(n)` property.

Example

Construct a 3-bit bus on the SolderCore's digital pins D4, D5, and D9, then use the bus to bring all signals high, then low.

```
INSTALL "PARALLEL-BUS" USING CORE.D4, CORE.D5, CORE.D9 AS BUS
BUS.OUTPUT = 7  ' three bits all high
BUS.OUTPUT = 0  ' three bits all low
END
```

Create a 4-bit parallel bus to show a binary count on the SolderCore Breakout Shield LEDs.

```
' Parallel bus driver example.
' Written by Paul Curtis of Rowley Associates.
' You're free to do what you like with this program.

' Make S a synonym for CORE.
LET S = CORE

' Install a 4-bit parallel bus driver on D2 through D5.
INSTALL "PARALLEL-BUS" USING S.D2, S.D3, S.D4, S.D5 AS BUS

' Run through the binary coding of 0 through 15 on the bus.
' If you use a Breakout Shield, you can see the binary count
' on the green and red LEDs.
FOR I = 0 TO BUS.MASK
  BUS.OUTPUT = I
  PAUSE 0.1
NEXT I
END
```

Raspberry Pi CPU

Installation

The Raspberry Pi CPU driver is automatically installed as a fixed driver when CoreBASIC starts.

Description

The Raspberry Pi Board CPU provides the base hardware drivers for digital and analog I/O, SPI, and I2C when used with a **Libelium Arduino Connection Bridge**.

Note that not all shields are compatible with the Connection Bridge. Shields that expect 5V signaling must not be used! Please refer to Libelium's information on the product.

You access the CPU driver using the `CORE` keyword.

Properties

Information		
`NAME`	String Input	Presentation name for the model that runs CoreBASIC. For the standard Raspberry Pi, this is "`Raspberry Pi`".
`MODEL`	String Input	Platform that CoreBASIC is running on. For the Raspberry Pi, this is "`MODEL-B`".
`VERSION`	String Input	CoreBASIC version number.
Digital and analog I/O		
`D0` through `D19`	Digital R/W	See expanded pin description below.
`A0` through `A19`	Digital R/W	See expanded pin description below.
`D(n)`	Digital R/W	Equivalent to `D0` through `D19`, indexed by *n*.
`A(n)`	Digital R/W	Equivalent to `A0` through `A19`, indexed by *n*.
Timing		
`FREQUENCY`	Digital Write	Core tick frequency. For the Freedom Board, this reads as 24,000,000 indicating 24 MHz.
`TICK`	Digital Write	Core tick. The tick increments at the core tick frequency, `FREQUENCY`.

Resources

Purchasing the adapter:

http://www.cooking-hacks.com/index.php/shop/raspberry-pi/raspberry-pi-to-arduino-shield-connection-bridge.html

Information about the adapter:

http://www.cooking-hacks.com/index.php/documentation/tutorials/raspberry-pi-to-arduino-shields-connection-bridge

See also

CORE

Seeed Studio 96x16 OLED Brick

Installation

```
INSTALL "SEEED-STUDIO-OLED-96x16-BRICK"
INSTALL "OLED-96x16-BRICK"
```

Description

Installing the Seeed Studio 96x16 OLED provides a 96×16 monochrome graphic display. You can use all the CoreBASIC graphics commands to drive the LCD display.

Resources

http://www.seeedstudio.com/depot/electronic-brick-oled-96x16-display-with-free-cable-p-704.html

Seeed Studio 96x96 OLED Twig

Installation

```
INSTALL "SEEED-STUDIO-OLED-96x96-TWIG"
INSTALL "OLED-96x96-TWIG"
```

Description

Installing the Seeed Studio 96x96 OLED provides a 96×96 16-level grayscale graphic display. You can use all the CoreBASIC graphics commands to drive the LCD display.

Resources

http://www.seeedstudio.com/depot/twig-oled-96x96-p-824.htm

Seeed Studio 128x64 OLED Twig

Installation

```
INSTALL "SEEED-STUDIO-OLED-128x64-TWIG"
INSTALL "OLED-128x64-TWIG"
INSTALL "OLED-128x64-BRICK"
```

Description

Installing the Seeed Studio 128x64 OLED provides a 128×64 monochrome graphic display. You can use all the CoreBASIC graphics commands to drive the LCD display.

Seeed Studio mounted this display on both an Electronic Brick and a Grove Twig module. The Electronic Brick is no longer available, but if you're lucky enough to have one, the Twig driver and the Brick driver are identical.

Resources

http://www.seeedstudio.com/depot/twig-oled-display-12864-p-781.html

Seeed Studio TFT Touch Shield

Installation

```
INSTALL "SEEED-STUDIO-TFT-TOUCH-SHIELD"
```

Options

None.

Description

Installing the TFT Touch Shield driver provides a 240×320 true color graphic display. You can use all the CoreBASIC graphics commands to drive the LCD display.

http://www.seeedstudio.com/depot/28-tft-touch-shield-p-864.html

Notes

This shield only works at 5V and will require the SolderCore Proteus Extender to work correctly with SolderCore.

Benchmarks

Here is the result of running the SolderCore Graphics Benchmarks application:

```
> run
Graphics display benchmark for SEEED-STUDIO-TFT-TOUCH-SHIELD

   Circles:    2379 ms
     Discs:   26550 ms
Rectangles:     702 ms
     Slabs:   23675 ms
     Lines:    6638 ms
  Polygons:    7899 ms
      Text:    2122 ms
> _
```

See also

SolderCore Proteus Extender

Sensirion SHT1x Driver

Installation

```
INSTALL "SENSIRION-SHT1X" USING sck, data
INSTALL "SHT1X" USING sck, data
```

Options

None.

Description

Installs a humidity and temperature sensor driver for the SHT1x family of sensors.

As the SHT1x is not a true I2C sensor, this driver requires two digital I/Os specified which are used for the SCK and DATA signals to the sensor.

Properties

Measurement		
TEMP	Analog Read	Supply-compensated temperature in degrees Celsius. For accuracy, this requires that the SOURCE property is set correctly to reflect the sensor's supply voltage.
HUMIDITY	Analog Read	True relative humidity, in percent. The driver reads both temperature and humidity and then supply-compensates the temperature reading and temperature-compensates the linear humidity reading to deliver true relative humidity.
DEWPOINT	Analog Read	Dew point in degrees Celsius, computed from the supply-compensated temperature TEMP and true relative humidity HUMIDITY.
DATA	Analog Read	An array of four values: the temperature, in degrees Celsius, the linear relative humidity in percent, the true relative humidity in percent, and the dew point in degrees Celsius.

DATA(n)	Analog Read	Direct access to temperature and humidity readings. DATA(-2) is the raw humidity ADC result; DATA(-1) is the raw temperature ADC result; DATA(0) is the supply-compensated temperature in degrees Celsius and is equivalent to the TEMP property; DATA(1) is the linear relative humidity in percent; DATA(2) is the true relative humidity in percent and is equivalent to the HUMIDITY property; DATA(3) is the dew point in degrees Celsius and is equivalent to the DEWPOINT property.
Configuration		
SOURCE	Analog R/W	The supply voltage, Vdd. When converting temperature, the driver selects from a range of coefficients to correct the temperature reading. You should set SOURCE to the voltage supplied to the Vdd pin. By default, SOURCE is set to 3 indicating a 3 V supply.
RESOLUTION	Digital R/W	Sensor resolution selection. When this property is written to zero, 14-bit humidity and 12-bit temperature readings are taken. When this property is written to one, 12-bit humidity and 8-bit temperature readings are taken. By default, RESOLUTION is set to 0 to select high resolution mode.
STATUS	Digital R/W	Reads and writes the entire SHT1x status register. Note that changing the resolution bit in the status register directly, without using the RESOLUTION property, will lead to incorrect measurement.
STATUS(n)	Digital R/W	Reads and writes bit *n* of the SHT1x status register. For instance, you can turn on the on-chip heater by writing one to STATUS(2) and turn it off by writing zero to STATUS(2).

Example

The following example shows how to read the temperature and humidity, and shows the measured temperature being affected by the on-chip heater.

```
***../examples/sensirion-sht11-demo.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "sensirion-sht11-demo"` or `|sensirion-sht11-demo`.

Sensirion SHT2x Driver

Installation

```
INSTALL "SENSIRION-SHT2X"
INSTALL "SHT2X"
```

Options

ADDR=*integer*

Set the I2C 8-bit address of the SHT2x. By default the driver uses the address `0x80`.

Description

Installs a humidity and temperature sensor driver for the SHT2x family of sensors.

Properties

Measurement		
`TEMP`	Analog Read	Supply-compensated temperature in degrees Celsius.
`HUMIDITY`	Analog Read	Relative humidity, in percent.
`DEWPOINT`	Analog Read	Dew point in degrees Celsius, computed from the temperature `TEMP` and relative humidity `HUMIDITY`.
`DATA`	Analog Read	An array of three values: the temperature, in degrees Celsius, the relative humidity in percent, and the dew point in degrees Celsius.
`DATA(n)`	Analog Read	Direct access to temperature and humidity readings. `DATA(0)` is the supply-compensated temperature in degrees Celsius and is equivalent to the `TEMP` property; `DATA(1)` is the true relative humidity in percent and is equivalent to the `HUMIDITY` property; `DATA(2)` is the dew point in degrees Celsius and is equivalent to the `DEWPOINT` property.
Configuration		

RESOLUTION	Digital R/W	Reads and writes the sensor resolution. The acceptable settings are 0x00 for 12-bit relative humidity and 14-bit temperature, 0x01 for 8-bit relative humidity and 12-bit temperature, 0x80 for 10-bit relative humidity and 13-bit temperature, 0x81 for 11-bit relative humidity and 11-bit temperature.
STATUS	Digital R/W	Reads and writes the entire SHT2x status register. Note that changing the resolution bit in the status register directly, without using the RESOLUTION property, will lead to incorrect measurement.
STATUS(n)	Digital R/W	Reads and writes bit *n* of the SHT2x status register. For instance, you can turn on the on-chip heater by writing one to STATUS(2) and turn it off by writing zero to STATUS(2).

Silicon Labs Si7005

Installation

```
INSTALL "SILICON-LABS-SI7005"
INSTALL "SI7005"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x80`.

Description

Installs a humidity and temperature sensor driver for the Si7005 sensor.

Properties

Measurement		
`TEMP`	Analog Read	Supply-compensated temperature in degrees Celsius.
`HUMIDITY`	Analog Read	True relative humidity, in percent.
`DEWPOINT`	Analog Read	Dew point in degrees Celsius, computed from the supply-compensated temperature `TEMP` and true relative humidity `HUMIDITY`.
`DATA`	Analog Read	An array of four values: the temperature, in degrees Celsius, the true relative humidity in percent, and the dew point in degrees Celsius.
`DATA(n)`	Analog Read	Direct access to temperature and humidity readings. `DATA(0)` is the supply-compensated temperature in degrees Celsius and is equivalent to the `TEMP` property; `DATA(1)` is the true relative humidity in percent and is equivalent to the `HUMIDITY` property; `DATA(2)` is the dew point in degrees Celsius and is equivalent to the `DEWPOINT` property.

Software I2C Bus Driver

Installation

`INSTALL "SOFTWARE-I2C-BUS" USING` *scl*, *sda*
`INSTALL "SOFTWARE-I2C" USING` *scl*, *sda*
`INSTALL "SOFT-I2C" USING` *scl*, *sda*

Options

None.

Description

Creates a software-driven I2C bus using the two digital pins *scl* and *sda*. This allows you great flexibility for isolating your I2C devices, or using pins other than A4 ad A5 as an I2C connection. However, note that using software-driven I2C imposes an additional overhead on the processor to handle each bit of communication over the I2C bus and, as such, will have lower performance than the primary and secondary I2C buses on the SolderCore.

Properties

Configuration		
SPEED	Digital R/W	Sets the speed of the bus in hertz.

Example

Construct an I2C bus on SolderCore's digital pins D2 (as SCL) and D3 (as SDA) and use that bus to control a BMP085 pressure sensor.

```
INSTALL "SOFTWARE-I2C-BUS" USING CORE.D2, CORE.D3 AS BUS
INSTALL "BOSCH-SENSORTEC-BMP085" USING BUS AS BMP085
PRINT "Current pressure: "; BMP085.PRESSURE
```

Software SPI Bus Driver

Installation

`INSTALL "SOFTWARE-SPI-BUS" USING` *sck*, *mosi*, *miso*
`INSTALL "SOFTWARE-SPI" USING` *sck*, *mosi*, *miso*
`INSTALL "SOFT-SPI" USING` *sck*, *mosi*, *miso*

Options

None.

Description

Creates a software-driven SPI bus using the three digital pins *sck*, `mosi`, and `msio`. This allows you great flexibility for isolating your SPI devices, or using pins other than D11, D12, and D13 as an SPI connection. However, note that using software-driven SPI imposes an additional overhead on the processor to handle each bit of communication over the SPI bus and, as such, will have lower performance than the primary SPI bus on the SolderCore.

Example

The ITead Studio IBridge with a Nokia 3310 LCD display has a nonstandard SPI arrangement on the shield which means that it is incompatible with the core SPI driver. However, with software-driven SPI, it's very simple to create an SPI bus that will drive the display:

```
***../examples/ibridge-soft-spi-demo.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "ibridge-soft-spi-demo"` or `|ibridge-soft-spi-demo`.

SolderCore Arcade Shield

Installation

```
INSTALL "SOLDERCORE-ARCADE-SHIELD"
INSTALL "ARCADE-SHIELD"
```

Options

None.

Description

Installing the Arcade Shield driver provides a 320×240 16-color graphic display.

Upon installation, the SolderCore inquires whether the Arcade Shield is fitted. If the shield is not fitted, or a SolderCore LCD shield is fitted instead of the Arcade Shield, the driver stops with an incorrect device error.

Note that you can use the SolderCore Graphics driver to choose between the Arcade Shield driver and LCD Shield driver at install time.

Properties

General		
`MODEL`	String Input	A string containing the model number of the Arcade Shield. There is currently only one model in the series and it is reported as `ARCADE-SHIELD-V1`.
`VERSION`	String Input	A string containing the firmware revision number of the Arcade Shield.
Color palette		
`COLOR(n)`	Digital Write	Assign color to palette entry. *n* is the palette entry, 0 through 15; the value assigned should be a standard 24-bit color value (constructed by RGB or other means).

See also

SolderCore Graphics Shield

Resources

http://www.soldercore.com/products/arcade-shield/

Benchmarks

Here is the result of running the SolderCore Graphics Benchmarks application:

```
> run
Graphics display benchmark for SOLDERCORE-ARCADE-SHIELD

   Circles:      10 ms
     Discs:      45 ms
Rectangles:      24 ms
     Slabs:      18 ms
     Lines:      48 ms
  Polygons:     274 ms
      Text:     572 ms
> _
```

SolderCore CoreMPU Driver

Installation

```
INSTALL "SOLDERCORE-CORE-MPU"
INSTALL "CORE-MPU"
```

Options

ADDR=*integer*

Set the I2C 8-bit address of the InvenSense MPU-6050. By default the driver uses the address `0xD0`.

Description

Installs a combination gyroscope, accelerometer, and magnetometer driver for the CoreMPU module. The CoreMPU integrates an InvenSense MPU-6050 accelerometer and gyroscope and a Honeywell HMC5883L magnetometer which provides a 9DOF sensor platform.

http://www.soldercore.com/products/sensecore/corempu/

When the MPU-6050 and HMC5883L are fused by the AHRS driver, you have excellent 9DOF attitude and heading reference system.

Although this is intended to fit into an SenseCore, you can use it as a standard breakout module and mount it wherever you wish or use it with something other than a SolderCore.

Properties

General		
VERSION	String Read	A string containing the device variant and silicon revision of the MPU-6050.
Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ	Analog Read	Acceleration measured along the *z* axis, in *g*.

`RANGE(0)`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH(0)`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
`BIAS(0)`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN(0)`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Gyroscope		
`G`	Analog Read	An array of three numbers containing the rotation rates measured around the *x*, *y*, and *z* axes, in degrees per second.
`GX`	Analog Read	Rotation rate around the *x* axis, in degrees per second.
`GY`	Analog Read	Rotation rate around the *y* axis, in degrees per second.
`GZ`	Analog Read	Rotation rate around the *z* axis, in degrees per second.
`RANGE(1)`	Analog R/W	Selected full scale range of the gyroscope, in degrees per second.
`BANDWIDTH(1)`	Analog R/W	Selected bandwidth of the of the gyroscope, in hertz.
`BIAS(1)`	Analog R/W	An array of three numbers containing the gyroscope bias, in degrees per second, for the *x*, *y*, and *z* axes.
`GAIN(1)`	Analog R/W	An array of three numbers containing the gain for one LSB, in degrees per second, for the *x*, *y*, and *z* axes.
Magnetometer		
`M`	Analog Read	An array of three numbers containing the magnetic field measured in the *x*, *y*, and *z* axes, in microtesla.

MX	Analog Read	Magnetic field measured in the *x* direction, in microtesla.
MY	Analog Read	Magnetic field measured in the *y* direction, in microtesla.
MZ	Analog Read	Magnetic field measured in the *z* direction, in microtesla.
BANDWIDTH(2)	Analog R/W	Selected bandwidth of the of the magnetometer, in hertz.
Algorithms		
ROLL	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
PITCH	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
Additional		
BANDWIDTH	Analog R/W	Selected bandwidth of the of the sensor assembly as a whole, in hertz. This is the smaller of BANDWIDTH(0) and BANDWIDTH(1).
TEMP	Analog Write	MPU-6050 die temperature, in degrees Celsius.

References

The SolderCore CoreMPU SExI module:

http://www.soldercore.com/products/sensecore/corempu/

Notes

See **Accelerometers**, **Gyroscopes**, **Magnetometers**, and **AHRS Driver**.

SolderCore CPU

Installation

The SolderCore CPU driver is automatically installed as a fixed driver when CoreBASIC starts.

Description

The SolderCore CPU provides the base hardware drivers for digital and analog I/O, SPI, I2C, networking, and mass storage. You access the CPU driver using the `CORE` keyword.

Properties

Information		
`NAME`	String Input	Presentation name for the model that runs CoreBASIC. For the standard SolderCore, this is "`SolderCore`". For the Windows SolderCore Emulator, it is "`SolderCore Emulator`".
`MODEL`	String Input	Platform that CoreBASIC is running on. For the SolderCore, this is "`SOLDERCORE-V1`". For the Windows SolderCore Emulator, it is "`SOLDERCORE-V1-EMULATOR`".
`SERIAL`	String Input	SolderCore serial number.
`VERSION`	String Input	CoreBASIC version number.
Digital and analog I/O		
`D0` through `D19`	Digital R/W	See expanded pin description below.
`A0` through `A19`	Digital R/W	See expanded pin description below.
`D(n)`	Digital R/W	Equivalent to `D0` through `D19`, indexed by *n*.
`A(n)`	Digital R/W	Equivalent to `A0` through `A19`, indexed by *n*.
LEDs		
`LED`	Digital R/W	User LED. Illuminates the USER LED when written to a nonzero value, and extinguishes it when written to zero.

RUN	Digital R/W	Run LED. Illuminates the RUN LED when written to a nonzero value, and extinguishes it when written to zero. The RUN LED is turned on when commencing a CoreBASIC program, and turned off on return to the command line. During program execution, however, the user's CoreBASIC program has complete control over the LED.
Y, G	Digital Write	Configure LAN yellow and green LED functions. Writing 0 or 1 turns the LED off and on and does not reflect any LAN function, allowing an additional two LEDs for application status output. Writing negative values configure the LEDs to indicate LAN status: −1 is "Link OK" (i.e. carrier present), −2 is "Tx or Rx activity", −6 is "100BASE-Tx mode" and illuminates when the link is running 100MHz, −7 is 10BASE-Tx mode" and illuminates when the link is running 10MHz, −8 is "Full Duplex" and illuminates when the link is in full duplex mode, and −9 is "Link OK and blink on Tx or Rx activity". The default is for the green LED to be connfigured as "Link OK" and the yellow LED as "Tx or Rx activity".
Operation		
TEMP	Analog Read	The die temperature in degrees Celsius. The LM3S9D92's specification states that the internal die temperature sensors accuracy is ±5 degrees Celsius, so this is simply an indication of how hot the die is running. This is not an indication of the ambient air temperature: if you require that, please use a separate temperature sensor.
Serial buses		

`I2C(n)`	Digital Read	Access SolderCore I2C bus *n*. Bus #0 is the primary I2C bus routed to pins A4 and A5 of the analog header. Bus #1 is the secondary I2C bus routed to the two-pin header between the reset button and the status LEDs on the SolderCore.
`SPI(n)`	Digital Read	Access SolderCore SPI bus *n*. Bus #0 is the primary SPI bus routed to pins D11, D12, and D13 of the digital header. Bus #1 is the secondary SPI bus routed to the two SOIC-8 memory sites on the reverse side of the SolderCore PCB and the microSD card slot.
Timing		
`FREQUENCY`	Digital Write	CPU operating frequency. For the Soldercore, this reads as 80,000,000 indicating 80 MHz.
`TICK`	Digital Write	CPU tick. The tick increments at the CPU operating frequency, `FREQUENCY`.
`TIME`	Digital R/W	Current time as seconds elapsed since 1 January 1970 the standard way of representing time in CoreBASIC. The core time can be set automatically from an NTP server or manually from some other time source such as a battery-backed RTC or a from a GPS receiver. Once set, the core time ticks independent of the original source.

See also

CORE

SolderCore Graphics Shield

Installation

```
INSTALL "SOLDERCORE-GRAPHICS-SHIELD"
INSTALL "GRAPHICS-SHIELD"
```

Options

None.

Description

The SolderCore Graphics Shield driver is a composite driver that will automatically select between a SolderCore LCD Shield and a SolderCore Arcade Shield at install time, depending upon which is connected to the SolderCore.

As the LCD Shield and Arcade Shield share substantially the same interface and protocol, the Graphics Shield driver inquires which shield is connected over SPI and then installs the appropriate driver for the device. The advantage of using the Graphics Shield driver is that demonstrations, and your own code, can run on the Arcade Shield or the LCD Shield with minimal changes.

You can determine the shield detected and installed at runtime by querying the `MODEL` property.

Properties

General		
`MODEL`	String Input	A string containing the model of the Arcade Shield or LCD Shield attached. There is currently only one model in each series and they are reported as either `ARCADE-SHIELD-V1` or `LCD-SHIELD-V1`.
`VERSION`	String Input	A string containing the firmware revision number of the Arcade Shield or LCD Shield.
Color palette		
`COLOR(n)`	Digital Write	Assign color to palette entry. *n* is the palette entry, 0 through 15; the value assigned should be a standard 24-bit color value (constructed by RGB or other means). This works as expected on an Arcade Shield and has no effect on an LCD Shield.

See Also

SolderCore Arcade Shield, SolderCore LCD Shield

Resources

http://www.soldercore.com/products/arcade-shield/
http://www.soldercore.com/products/lcd-shield/

SolderCore LCD Shield

Installation

```
INSTALL "SOLDERCORE-LCD-SHIELD"
INSTALL "LCD-SHIELD"
```

Options

None.

Description

Installing the LCD Shield driver provides a 320×240 64k-color graphic display.

Upon installation, the SolderCore inquires whether the LCD Shield is fitted. If the shield is not fitted, or a SolderCore Arcade shield is fitted instead of the LCD Shield, the driver stops with an incorrect device error.

Note that you can use the SolderCore Graphics driver to choose between the Arcade Shield driver and LCD Shield driver at install time.

Properties

General		
`MODEL`	String Input	A string containing the model number of the LCD Shield. There is currently only one model in the series and it is reported as `LCD-SHIELD-V1`.
`VERSION`	String Input	A string containing the firmware revision number of the LCD Shield.

See also

SolderCore Graphics Shield

Resources

http://www.soldercore.com/products/lcd-shield/

Benchmarks

Here is the result of running the SolderCore Graphics Benchmarks application:

```
> run
Graphics display benchmark for SOLDERCORE-LCD-SHIELD

   Circles:      64 ms
     Discs:     340 ms
Rectangles:     166 ms
     Slabs:     417 ms
```

```
     Lines:     223 ms
  Polygons:     286 ms
      Text:     572 ms
> _
```

SolderCore Network

Installation

The SolderCore network driver is automatically installed as a fixed driver when CoreBASIC starts.

Description

The SolderCore network driver provides access to the networking features of the SolderCore. You access the network driver using the NET keyword.

Properties

Information		
NAME	String I/O	SolderCore's NetBIOS network name.
MAC addressing		
MACADDR	Digital Write	An array of six octets specifying the MAC address of the integrated Ethernet network interface.
IP addressing		
IPADDR	Digital R/W	Assigned network IPv4 address. If configured for DCHP, reading this returns an array of four integers corresponding to the DHCP-assigned network address, or four zeros if the IP address is not yet configured. If DHCP is not configured, writing to IPADDR configures the IP address that the integrated Ethernet network interface uses.
MASK	Digital R/W	Network mask. IPv4 network mask of the integrated Ethernet network interface.
GATEWAY	Digital R/W	Gateway IPv4 address of the integrated Ethernet network interface; used when a nonlocal IP address needs routing.
SMTPSERVER	Digital R/W	Assigned SMTP server IPv4 address. CoreBASIC does not automatically configure the SMTP server address; you must assign this before using the MAIL statement.

TIMESERVER	Digital R/W	Assigned NTP server IPv4 address. CoreBASIC does not automatically configure the NTP server address; you must assign this to have CoreOS maintain network time.

See also

NET

SolderCore Motor Shield

Installation

```
INSTALL "SOLDERCORE-MOTOR-SHIELD"
INSTALL "MOTOR-SHIELD"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x22`.

Description

The SolderCore Motor Shield is a 6-channel brushed DC motor driver.

Properties

Motor Drive		
`MOTOR(n)`	Analog Write	Sets the speed and direction of channel *n*. The value assigned to the property should lie between −1 and 1; values outside this range are clamped. The sign of the value written determines whether the motor runs forward or in reverse.
`LEFT`	Analog Write	Convenience property. Refers directly to channel 0, so `LEFT` is identical to `MOTOR(0)`.
`RIGHT`	Analog Write	Convenience property. Refers directly to channel 1, so `RIGHT` is identical to `MOTOR(0)`.

SolderCore SenseCore Shield

Installation

```
INSTALL "SOLDERCORE-SENSECORE"
INSTALL "SENSECORE"
```

Options

None.

Description

Installing the SenseCore Shield driver provides a sites for up to four full-width SExI modules or eight half-width SExI modules.

Properties

High-Level selection		
`A … D`	Digital Write	Direct write to SExI SPI chip select signal for sites A through D.
Port I/O		
`D0 … D3`	Digital R/W	Read or write signals DIO0 through DIO3 which are the auxiliary digital I/O signals routed to SExI sites A through D.
`D(n)`	Digital R/W	Read or write signals DIO0 through DIO3 which are the auxiliary digital I/O signals routed to SExI sites A through D.
`TX(n)`	Digital R/W	Read or write signals TX0 through TX3 which are the TX digital I/O signals routed to SExI sites A through D.
`RX(n)`	Digital R/W	Read or write signals RX0 through RX3 which are the RX digital I/O signals routed to SExI sites A through D.
`SELECT(n)`	Digital Write	Indexed write to SExI SPI chip select bit for site sites A through D. `SELECT(0)` writes to the chip select for site A, `SELECT(1)` writes to the chip select for site B, and so on.

`OUTPUT`	Digital Write	Write all 16 bits of the port expander.
`DATA`	Digital Write	As `OUTPUT`.
`OUTPUT(n)`	Digital Write	Writes bit *n* of the port expander.
`INPUT(n)`	Digital Write	Reads bit *n* of the port expander.

Notes

Writing to the port expander directly may well select an SPI device onto the SPI bus; you'll need to consult the SenseCore schematic and be careful not to interfere with CoreBASIC's management of SExI sites. Also note that selecting a device on the SPI bus by writing directly to its chip select will not automatically deselect other devices on the SPI bus: if you intend to manage chip selection yourself, you must also manage deselection of selected devices.

Resources

http://www.soldercore.com/products/sensecore/

SolderCore Servo Shield

Installation

```
INSTALL "SOLDERCORE-SERVO-SHIELD"
INSTALL "SERVO-SHIELD"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x20`.

Description

The SolderCore Servo Shield is a 6-channel servo driver.

Properties

Motor Drive		
`MOTOR(n)`	Analog Write	Sets the position of channel *n*. The value assigned to the property should lie between −1 and 1; values outside this range are clamped.
`LEFT`	Analog Write	Convenience property. Refers directly to channel 0, so `LEFT` is identical to `MOTOR(0)`.
`RIGHT`	Analog Write	Convenience property. Refers directly to channel 1, so `RIGHT` is identical to `MOTOR(0)`.

SparkFun Ardumoto Shield

Installation

```
INSTALL "SPARKFUN-ARDUMOTO"
INSTALL "ARDUMOTO"
INSTALL "MONSTER-MOTO-SHIELD"
INSTALL "MONSTER-MOTO"
```

Options

None.

Description

The Ardumoto is a 2-channel brushed DC motor driver.

Properties

Motor Drive		
A	Analog Write	Sets speed and direction of drive A; equivalent to `MOTOR(0)`.
B	Analog Write	Sets speed and direction of drive B; equivalent to `MOTOR(1)`.
Alternative naming		
`LEFT`	Analog Write	Sets speed and direction of drive A; equivalent to `MOTOR(0)`.
`RIGHT`	Analog Write	Sets speed and direction of drive B; equivalent to `MOTOR(1)`.
Indexed Motor Drive		
`MOTOR(n)`	Analog Write	Sets the speed and direction of channel *n*. Channel 0 is drive A, and channel 1 is drive B.

Notes

We arbitrarily designate the A channel as the left-hand drive and B as the right-hand drive channel so that your programs read better if you use both left and right drives for a robot.

The value assigned to the drive channel should lie between −1 and +1; values outside this range are clamped. The sign of the value written determines whether the motor runs forward (when positive) or in reverse (when negative), or is stationary (when zero).

SparkFun Color LCD Shield

Installation

```
INSTALL "SPARKFUN-COLOR-LCD-SHIELD"
INSTALL "6610-SHIELD"
```

Options

`TYPE=GE8` *(default)*

Initializes the display to use an Epson-based GE8 display.

`TYPE=GE12`

Initializes the display to use an Philips-based GE12 display.

Description

Installing the SparkFun Color LCD Shield provides a 128×128 12-bit color graphic display. You can use all the CoreBASIC graphics commands to drive the LCD display.

Properties

Buttons		
`D1 … D3`	Digital Read	Senses individual switches S1, S2, and S3. The property reads `1` if the switch is pressed and `0` if it is released.
`ALL`	Digital Read	Senses the state of all switches S1, S2, and S3, and returns an array of three states. Index 0 corresponds to switch S1, index 1 to switch S2, and index 2 to switch S3. Each element is `1` if the corresponding switch is pressed and `0` if it is released.
`BUTTON`	Digital Read	As `ALL`.

Resources

http://www.sparkfun.com/products/9363

SparkFun El Escudo

Installation

```
INSTALL "SPARKFUN-EL-ESCUDO"
INSTALL "EL-ESCUDO"
INSTALL "ESCUDO"
```

Options

None.

Description

Installing the El Escudo driver sets the hardware to drive the electroluminescent wires (EL-wire) attached to D2 through D9 and the LED on D10. The SolderCore signal D2 drives segment A, D3 drives segment B, and so on.

Properties

Direct Single Wire Drive		
A … H	Digital Write	Drives EL segments A through H. Writing a zero turns the EL wire off, writing nonzero turns it on.
LED	Analog Write	Controls the intensity of the LED on the El Escudo using PWM.
Indexed Single Wire Drive		
SEGMENT(n)	Digital Write	Indexes 0 through 7 drive segments A through H; hence, SEGMENT(0) is the same as A, SEGMENT(1) is the same as B, and so on. This form of addressing makes it easier to write both simple and complex sequence-based animations.

Resources

http://www.sparkfun.com/products/9259

SparkFun e-Paper Breakout

Installation

```
INSTALL "SPARKFUN-EPAPER-BREAKOUT"
INSTALL "SPARKFUN-EPAPER"
INSTALL "EPAPER"
```

Options

None.

Description

Installs a driver for the 10×2 WINSTAR e-paper display used in the SparkFun e-Paper Breakout.

The driver takes care of all necessary housekeeping and timing for the e-paper display and automatically schedules updates independently from execution of your CoreBASIC application.

The CoreBASIC interface to the e-paper display is identical to character-based LCD displays using the HD44780. If you have an application using an HD44780 LCD display, moving it to the e-paper display is a simple matter of wiring the e-paper display to the SolderCore and changing the driver installed to handle the application's output.

The differences between a character LCD display and the e-paper display are:

- The e-paper display is much slower to change than an LCD display owning to the different fabrication technologies. You cannot update an e-paper display with quickly changing text and expect to be able to read it. The display can take half a second to a second to change, and longer to erase ghosting of old characters, inherent in e-paper technology.
- The LCD display has a flashing cursor which is controlled by the `CURSOR` property. The e-paper display does not feature a cursor, so the `CURSOR` property does nothing on an e-paper display.
- The LCD display has eight programmable characters. The e-paper display does not have this capability.
- An LCD display may optionally have its backlight controlled using the `LIGHT` property. Because an e-paper display does not require a backlight and maintains its contents after power is removed, the `LIGHT` property does nothing on an e-paper display.
- The e-paper display can be black on white or white on black, controlled by the `POLARITY` property. A character LCD display cannot invert its display and does not implement the `POLARITY` property.

Note

Because the e-paper display is updated by a separate task in the SolderCore, your application is not delayed when you output to the e-paper display: your output request completes almost instantaneously and the update of the display is taken care of, over time, by a independent update task.

Setup

The connections required to use this driver are:

- D6 — SLEEP
- D8 — LATCH
- D9 — EIO
- D11 — DI0
- D13 — XCK

Properties

Dimensions		
WIDTH	Digital Write	The width of the display, in character positions. Reads as 10.
HEIGHT	Digital Write	The height of the display, in character lines. Reads as 2.
Cursor		
X or COL	Digital R/W	The *x* co-ordinate of the cursor.
Y or ROW	Digital R/W	The *y* co-ordinate of the cursor.
LINE	Digital R/W	As Y and ROW.
POS	Digital R/W	An array containing the *x* and *y* co-ordinates of the cursor.
CURSOR	Digital Write	Cursor control. Writing to this property has no effect: the property is implemented only for compatibility with character-based LCD displays.
Control		
POLARITY	Digital R/W	The display polarity. By default, the display initialized to black on white with POLARITY set to 0. Setting POLARTY to 1 changes the display to white on black, keeping all displayed content unchanged.
LIGHT	Digital Write	Backlight control. Writing to this property has no effect: the property is implemented only for compatibility with character-based LCD display drivers.
Formatted I/O		

`Y(n)` or `ROW(n)`	String R/W	When read, returns the string being displayed on line *n* of the display. When written, overwrites the whole of line *n* of the display, filling display line *n* with extra spaces if necessary.
`LINE(n)`	String R/W	As `Y(n)` and `ROW(n)`.
`CENTER(n)`	String Write	Centers the string written on display line *n*. If the string is too long for the display, only the central part is displayed. If the string is narrower than the display width, it is padded left and right with spaces to the display width such that it lies central within the display.
`RIGHT(n)`	String Write	Right-justifies the string written on display line *n*. If the string is too long for the display, it is truncated on the left such that the rightmost part of the string covers the whole line. If the string is narrower than the display width, it is padded left with spaces to the display width such that it lies adjusted right on the display.
Direct Digit Output		
`DIGIT(n)`	Digital R/W	Reads or writes the 16-segment digit *n*. The coding of the segments to the bits in the value written is described below. Digit 0 is the first digit of the top line and digit 19 is the last digit of the second line.

Resources

http://www.sparkfun.com/products/10150 — display
http://www.sparkfun.com/products/10304 — breakout

SparkFun IMU-3000 Combo

Installation

```
INSTALL "SPARKFUN-IMU-3000-COMBO"
INSTALL "IMU-3000-COMBO"
```

Options

None.

Description

Installs drivers for the devices on the SparkFun IMU-3000 Combo. The IMU-3000 Combo has both an InvenSense IMU-3000 which is a 3-axis gyroscope and motion processor, and a Analog Devices ADXL345 3-axis accelerometer. Together, these provide a 6DOF IMU.

While the IMU-3000 has a capability to process motion data internally using its own Digital Motion Processor (DMP), the CoreBASIC driver does not use this function. There are great reasons for not using the IMU-3000's DMP capability: if we did, it would require that we devote a lot of internal RAM just to bootstrap the DMP, and RAM is always at a premium.

So, rather than use the internal DMP for sensor fusion, you can simply use the CoreBASIC AHRS driver to fuse the gyroscopes and accelerometers.

Features

This driver differs from the standard CoreBASIC `IMU-3000` driver as the IMU-3000 Combo board has an ADXL345 attached to the IMU-3000's auxiliary I2C bus. This driver programs up the IMU-3000 to deliver accelerometer readings to the internal registers of the IMU-3000 rather than querying the ADXL345 directly.

The `IMU-3000-COMBO` driver also compensates for the misaligned axes on the ADXL345 with respect to the IMU-3000 and presents accelerometer data in the same orientation as the gyroscope's body frame.

Integration with the AHRS driver

The IMU-3000 Combo is easily integrated into an 6DOF IMU using the AHRS driver:

```
INSTALL "IMU-3000-COMBO" AS IMU
INSTALL "AHRS" USING IMU AS AHRS
```

That's it! All you need to do now is query the AHRS driver to determine the orientation of the IMU-3000.

Properties

Accelerometer

A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ	Analog Read	Acceleration measured along the *z* axis, in *g*.
Gyroscope		
G	Analog Read	An array of three numbers containing the rotation rates measured around the *x*, *y*, and *z* axes, in degrees per second.
GX	Analog Read	Rotation rate around the *x* axis, in degrees per second.
GY	Analog Read	Rotation rate around the *y* axis, in degrees per second.
GZ	Analog Read	Rotation rate around the *z* axis, in degrees per second.
Algorithms		
ROLL	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.
PITCH	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings. Note that using the AHRS driver to fuse gyroscopes and accelerometers will provide better dynamic response and will compensate for linear acceleration.

See also

InvenSense IMU-3000 Driver

Notes

See **Accelerometers** and **Gyroscopes**.

SparkFun Joystick Shield

Installation

```
INSTALL "SPARKFUN-JOYSTICK-SHIELD"
```

Options

None.

Description

Installing the Joystick Shield provides a few digital switches and an analog joystick.

When delivered, the Joystick Shield's joystick outputs are configured to vary between 0V and +5V as the joystick's potentiometers are tied to the 5V rail. In order to use this shield with SolderCore, you need a simple hardware modification:

- Snip the 5V pin from the underside of the shield.
- Strap the 5V pin to the 3V3 pin next to it on the topside of the shield using a solder bridge or a small length of wire.

With that done, the Joystick Shield works perfectly with SolderCore.

Properties

Joypad		
`LEFT`	Digital Write	Sense left switch on the joystick. This property reads `1` When the joystick is pushed left and `0` when not.
`RIGHT`	Digital Write	As `LEFT` but for the right switch.
`UP`	Digital Write	As `LEFT` but for the up switch.
`DOWN`	Digital Write	As `LEFT` but for the down switch.
`PRESS`	Digital Write	This property reads `1` when the joystick is pressed down and continues to read `1` until the joystick is released. It reads `0` when it is not pressed.
`SELECT`	Digital Write	As `PRESS`.
Analog Joystick		
`H(n)`	Analog Read	Sense horizontal joystick position *n*; −1 is fully left, +1 is fully right, and 0 is centered.

V(n)	Analog Read	Sense vertical joystick position *n*; –1 is fully down, +1 is fully up, and 0 is centered.
POS(n)	Analog Read	A complex number where the real component is the horizontal position of joystick *n* and the imaginary component is the vertical position of joystick *n*.

Resources

http://www.sparkfun.com/products/9760

SparkFun MIDI Shield

Installation

```
INSTALL "SPARKFUN-MIDI-SHIELD"
INSTALL "MIDI-SHIELD"
```

Options

None.

Description

Installing the MIDI shield provides you with a means to send and receive MIDI messages over a standard MIDI cable. You can also read the state of the switches and rotary potentiometers directly.

Please make sure that the switch is set to the RUN position in order to route the MIDI connections to the Rx and Tx signals on the SolderCore.

Properties

LEDs		
RED	Digital Write	Drive the red LED. Writing 1 to RED will illuminate the red LED, and writing 0 will extinguish it.
GREEN	Digital Write	Drive the red LED. Writing 1 to GREEN will illuminate the red LED, and writing 0 will extinguish it.
Switches		
D2	Digital Read	Reads the state of switch D2. When switch D2 is pressed, D2 reads as zero and when released D2 reads nonzero.
D3	Digital Read	Reads the state of switch D2. When switch D2 is pressed, D3 reads as zero and when released D3 reads nonzero.
D4	Digital Read	Reads the state of switch D2. When switch D2 is pressed, D4 reads as zero and when released D4 reads nonzero.
Potentiometers		
A0	Analog Read	Reads the setting of potentiometer A0. This will deliver a result between 0 and 1, from fully off to fully on.

A1	Analog Read	Reads the setting of potentiometer A1. This will deliver a result between 0 and 1, from fully off to fully on.

Resources

http://www.sparkfun.com/products/9595

SparkFun OLED Carrier

Installation

```
INSTALL "SPARKFUN-OLED-CARRIER"
INSTALL "OLED-CARRIER"
```

Options

None.

Description

The OLED carrier mounts a 128×128 OLED graphic display.

Connections

The connections from the carrier to the SolderCore are:

RST to D0; D/C to D1; CS to D8.

SparkFun RingCoder Breakout

Installation

```
INSTALL "SPARKFUN-RINGCODER-BREAKOUT"
INSTALL "SPARKFUN-RINGCODER"
INSTALL "RINGCODER"
```

Options

The optional USING clause indicates how LATCH is connected.

Description

A driver for a single 16-segment RingCoder breakout.

The USING clause indicates how the RingCoder's LATCH signal is connected. If you do not specify a USING clause, the latch signal, LATCH, is set to use CORE.D8.

The RingCoder's CLR signal must be pulled up to Vcc, and EN must be pulled down to GND.

Properties

Configuration		
ORIGIN	Digital R/W	Read or write the 12-o'clock segment. By default, ORIGIN is set to zero.
LEDs		
LED	Digital R/W	Directly read or write all 16 bits of the shift register that controls the LEDs. The 12-o'clock position is not shifted by the ORIGIN property.
SEGMENT(n)	Digital R/W	Reads or writes the state of LED segment *n* (modulo 16). Writing a 1 turns the LED on, writing a 0 turns the LED off. The 12-o'clock segment is controlled by the ORIGIN property.

Example

```
INSTALL "RINGCODER" AS RINGCODER
CALL DISSOLVE(1)
CALL SWEEP(0) : CALL SWEEP(1)
CALL SWEEP(0) : CALL SWEEP(1)
CALL DISSOLVE(0)
```

```
END

DEFPROC SWEEP(STATE)
FOR I = 0 TO 15
  RINGCODER.SEGMENT(I) = STATE
  PAUSE 0.1
NEXT I
ENDPROC

DEFPROC DISSOLVE(STATE)
SEQ = SHUFFLE GEN(0 TO 15)
FOR EACH S IN SEQ
  RINGCODER.SEGMENT(S) = STATE
  PAUSE 0.1
NEXT S
ENDPROC
```

This can be written a little more succinctly:

```
INSTALL "RINGCODER" AS RINGCODER
CALL EFFECT(SHUFFLE GEN(0 TO 15), 1)
CALL EFFECT(GEN(0 TO 15), 0) : CALL EFFECT(GEN(0 TO 15), 1)
CALL EFFECT(GEN(0 TO 15), 0) : CALL EFFECT(GEN(0 TO 15), 1)
CALL EFFECT(SHUFFLE GEN(0 TO 15), 0)
END

DEFPROC EFFECT(SEQ, STATE)
FOR EACH S IN SEQ
  RINGCODER.SEGMENT(S) = STATE
  PAUSE 0.1
NEXT S
ENDPROC
```

SparkFun Spectrum Shield

Installation

```
INSTALL "SPARKFUN-SPECTRUM-SHIELD"
INSTALL "SPECTRUM-SHIELD"
```

Options

None.

Description

The SparkFun Spectrum Shield uses two graphic equalizer driver chips to analyze the spectrum of a stereo signal. You connect the audio to be analyzed into either of the two jacks, and the left and right channels are divided into frequency bands.

Properties

Direct Input		
`LEFT`	Analog Input	Equivalent to `CHANNEL(0)`.
`RIGHT`	Analog Input	Equivalent to `CHANNEL(1)`.
Indexed Input		
`CHANNEL(n)`	Analog Input	Returns a seven-element array for the spectrum of channel *n*. Channel 0 is the left channel and channel 1 is the right channel. Each element of the array lies between 0 and 1 inclusive.

Example

This example requires only the Spectrum Shield plugged into the SolderCore. It shows a bar graph, like you would see on a graphic equalizer, for the left channel.

```
***../examples/spectrum-shield-test.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "spectrum-shield-test"` or `|spectrum-shield-test`.

Resources

http://www.sparkfun.com/products/10306

SparkFun Touch Shield

Installation

```
INSTALL "SPARKFUN-TOUCH-SHIELD"
INSTALL "TOUCH-SHIELD"
```

Options

None.

Description

The SparkFun Touch Shield provides a 9-digit capacitive sense keypad.

Properties

`SINGLE` filters out a single capacitive sense touch and returns a string. If the string is empty, no keys are detected as pressed or multiple keys are pressed simultaneously (i.e. there is not a single unique key press).

`ALL` returns a string containing all the keys that are pressed, and is empty if no keys are pressed.

Note that both `SINGLE` and `ALL` return the instantaneous state of keys pressed—if you hold your finger on a key, that key is continually returned as pressed.

Example

This example requires only the Touch Shield plugged into the SolderCore. It shows how to wait for keys to be released before registering another key as pressed.

```
***../examples/touch-shield-pin.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "touch-shield-pin"` or `|touch-shield-pin`.

Resources

http://www.sparkfun.com/products/10508

SparkFun VoiceBox Shield

Installation

```
INSTALL "SPARKFUN-VOICEBOX-SHIELD"
INSTALL "VOICEBOX"
```

Options

None.

Description

The SparkFun VoiceBox Shield is integrated into CoreBASIC so you can easily add speech to your application. It's easy!

```
INSTALL "SPARKFUN-VOICEBOX-SHIELD" AS VOICE
VOICE.SAY = "Welcome to the SolderCore"
```

The VoiceBox shield routes the eight *event input* signals of the SpeakJet IC to digital pins 5 through 12. CoreBASIC configures the SpeakJet such that these pins are unused and do not speak phrases when the event signals change.

The SolderCore does not know how to "speak" each word that it sees. Rather, it uses a dictionary and looks up each word and how to speak it from the dictionary. It does this using the file `/c/sys/speakjet.txt` on the card in the SD slot. If that file doesn't exist and you use SAY, the SolderCore will substitute "error." In fact, if the word you're trying to say is not in the dictionary, the SolderCore will substitute "error" for that word too.

You can add additional words, or trim the vocabulary, by editing the `speakjet.txt` file.

Resources

The first version of the VoiceBox shield has an SKU (Stock Keeping Unit) of DEV-09624. You can find it here:

http://www.sparkfun.com/products/9624

The updated version of the VoiceBox shield has an SKU of DEV-09799. You can find it here:

http://www.sparkfun.com/products/9799

The only difference between the two is that the newer VoiceBox uses a surface-mounted SpeakJet IC whereas the older one uses a DIL package.

SparkFun then retired that version and replaced it with one that includes a 3.5mm jack:

http://www.sparkfun.com/products/10661

SPI Device Driver

Installation

`INSTALL "SPI-DEVICE" USING` *digital-io*

Options

None.

Description

Installs a SPI device driver for a device using the *digital-io* property as the device selector.

Each device on an SPI bus is selected using a *device select* or *chip select* signal. The *digital-io* property specifies which digital I/O the installed SPI driver will use to select the device.

Properties

Configuration		
`SPEED`	Digital Read	The speed of the bus, in hertz, when addressing the device. The bus speed must be between 1 kHz and 50 MHz. The default is 1 MHz.
`MODE`	Digital Read	The phase and polarity of the bus when addressing the device. Supported SPI modes are 0 through 3. The default is mode 0.
`WIDTH`	Digital Read	The width of one data item on the bus. Supported widths are 1 though 8. The default is 8 bits.
`IDLE`	Digital Read	The state of the MOSI when reading from the bus. The default is zero.

See also

SPI, Reading an MPL115A1 pressure sensor using SPI

STMicroelectronics LIS302DL Driver

Installation

```
INSTALL "STMICROELECTRONICS-LIS302DL"
INSTALL "LIS302DL"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x38`; to configure the alternate address, use `ADDR=0x3A`.

Description

Installs an accelerometer driver for the LIS302DL and initializes the accelerometer to the 2*g* range.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		

`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Specification

Parameter	Supported settings
Bandwidth (Hz)	100, 400
Range (*g*)	±2, ±8
Communication	up to 400 kHz (I2C) and 10 MHz (SPI)

Notes

See **Accelerometers**.

STMicroelectronics LIS331DLH Driver

Installation

```
INSTALL "STMICROELECTRONICS-LIS331DLH"
INSTALL "LIS331DLH"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x30`; to configure the alternate address, use `ADDR=0x32`.

Description

Installs an accelerometer driver for the LIS331DLH and initializes the accelerometer to the 2*g* range.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		

`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See **Accelerometers**.

STMicroelectronics LIS331HH Driver

Installation

```
INSTALL "STMICROELECTRONICS-LIS331HH"
INSTALL "LIS331HH"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x30`; to configure the alternate address, use `ADDR=0x32`.

Description

Installs an accelerometer driver for the LIS331HH and initializes the accelerometer to the 6*g* range.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		

`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Notes

See **Accelerometers**.

STMicroelectronics LIS3DSH Driver

Installation

```
INSTALL "STMICROELECTRONICS-LIS3DSH"
INSTALL "LIS3DSH"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x38`; to configure the alternate address, use `ADDR=0x3A`.

Description

Installs an accelerometer driver for the LIS3DSH and initializes the accelerometer to the 2*g* range.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		

`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Specification

Parameter	Supported settings
Bandwidth (Hz)	50, 200, 400, 800
Range (*g*)	±2, ±4, ±6, ±8, ±16
Communication	up to 400 kHz (I2C) and 10 MHz (SPI)

Notes

See **Accelerometers**.

STMicroelectronics LIS3LV02DL Driver

Installation

```
INSTALL "STMICROELECTRONICS-LIS3LV02DL"
INSTALL "LIS3LV02DL"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x3A`.

Description

Installs an accelerometer driver for the LIS3LV02DL and initializes the accelerometer to the 2*g* range.

Properties

Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		
`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.

`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

Specification

Parameter	Supported settings
Bandwidth (Hz)	40, 160, 640, 2560
Range (*g*)	±2, ±6
Communication	up to 400 kHz (I2C) and 8 MHz (SPI)

Notes

See **Accelerometers**.

STMicroelectronics LPS331AP

Installation

```
INSTALL "STMICROELECTRONICS-LPS331AP"
INSTALL "LPS331AP"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0xBA`. The LPS331Ap can be configured to use addresses `0xB8` using the SA0 signal.

Description

Installs a pressure sensor driver for the LPS331AP.

Properties

Sensors		
PRESSURE	Analog Read	Pressure measured in pascals.
TEMP	Analog Read	Die temperature measured in degrees Celsius.
ALL	Analog Read	An array of two numbers containing the pressure in pascals and the temperature in degrees Celsius.
Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
WORD(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

STMicroelectronics LSM303DLH Driver

Installation

```
INSTALL "STMICROELECTRONICS-LIS331DL" AS accel
INSTALL "HONEYWELL-HMC5883L" AS compass
```

Options

None.

Description

The STMicroelectronics LSM303DLH device is the combination of a STMicroelectronics LIS331DL accelerometer and a Honeywell HMC5883L magnetometer in a single integrated circuit. Each device has its own (standard) I2C address and shares the same I2C bus. Rather than provide a single driver for the LSM303DLH in CoreBASIC, it is much easier to consider the single package as two separate devices and install drivers for each device separately, as above. Hence, CoreBASIC fully supports the LSM303DLH but does not provide a specific driver for it.

System UART Driver

Installation

```
INSTALL "SYSTEM-UART"
INSTALL "UART"
```

Install UART driver.

Options

PORT=*integer*

Selects port to use. Port 0 is routed to pins D0/Rx and D1/Tx, and port 1 is routed to pins D8/Tx and D10/Rx. If no port is specified, the UART uses port 0.

RXBUF=*integer*

Set receive buffer size, in bytes. The receive buffer size defaults to 128 bytes if not specified.

TXBUF=*integer*

Set transmit buffer size, in bytes. The transmit buffer size defaults to 128 bytes if not specified.

Description

Installs a UART driver for the system UART.

For the SolderCore:

- UART port 0 is brought out to pins 0 and 1 of the digital connector. Pin 0 is receive (for data transmitted to the SolderCore) and pin 1 is transmit (for data transmitted by the SolderCore).
- UART port 1 is brought out to pins 8 and 10 of the digital connector. Pin 8 is transmit and pin 10 is receive.

All UART input and output is interrupt driven and buffered by CoreOS. If received characters are dropped because the receive buffer isn't emptied quickly enough by your program, you should increase the buffer size when installing the driver.

If the transmit buffer is full and you try to output to the UART, your program will wait until the transmit buffer empties sufficiently such that more data can be accepted. Hence, you will never drop characters on output, but your program may be delayed waiting for the buffer to empty when you output data with the transmit buffer full.

Properties

Line protocol		
SPEED	Digital R/W	Baud rate; from 9600 to 1Mbaud. Default is 9600 baud.

PARITY	Digital R/W	Parity for each character. 0 is no parity, 1 is odd parity, 2 is even parity. Default is no parity.
DATA	Digital R/W	Number of data bits for each character, range is 5 to 8 inclusive. Default is 8 data bits.
STOP	Digital R/W	Number of stop bits for each character, range is 1 to 2 inclusive. Default is 1 stop bit.
Writing		
PRINT	String Write	Convert written item to a string and write that to the UART.
WRITE, TX	String Write	Write merged data to UART.
Status		
READY	Digital Read	Number of items that are ready to be read from the UART input buffer.
LEFT	Digital Read	Number of items that remain in the UART output buffer, waiting to be sent to the UART. You can wait on this property to ensure that all buffered data has been sent to the UART for transmission (by using, for instance, `WAIT UART.LEFT = 0`).
Reading		
READ	Digital Read	Read a single item from the UART and return it as a number.
READ(n)	String Read	Read exactly n items from the UART and return them as a string.
RX	Digital Read	Read a single item from the UART and return it as a number. If the UART has no item immediately available, RX will read as –1.
RX(n)	String Read	Read at most n items from the UART and return them as a string. If the UART does not have *n* items immediately available, the read will consume all items that can be read. The length of the returned string indicates the number of items read from the buffer.

Example

The following example shows how you can print messages to the UART and poll the UART for input:

```
***../examples/uart-tick-tock.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "uart-tick-tock" or |uart-tick-tock.

Example

The following example will append all data received by the SolderCore UART to the file /c/log.txt and will echo the received data to the transmitter.

```
***../examples/uart-sniffer.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "uart-sniffer" or |uart-sniffer.

Example

The following example will test that you can transmit and receive correctly on both SolderCore UARTs and is a demonstration of how to use two UARTs.

```
***../examples/dual-uart-loopback-test.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "dual-uart-loopback-test" or |dual-uart-loopback-test.

When this is run you should see this output:

```
> run

Setting initial conditions.

Testing loopback at 9600 baud.

Transmit from UART 0 to UART 1
..............................................
Transmit from UART 1 to UART 0
..............................................

Testing loopback at 38400 baud.

Transmit from UART 0 to UART 1
..............................................
Transmit from UART 1 to UART 0
..............................................

Testing loopback at 31250 baud.

Transmit from UART 0 to UART 1
..............................................
Transmit from UART 1 to UART 0
..............................................

Testing loopback at 115200 baud.

Transmit from UART 0 to UART 1
```

```
.............................................
Transmit from UART 1 to UART 0
.............................................

Test passed.
> _
```

Texas Instruments TMP100 Driver

Installation

```
INSTALL "TEXAS-INSTRUMENTS-TMP100"
INSTALL "TMP100"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x90`.

Description

Installs a temperature sensor driver for the TMP100.

Properties

Measurement		
TEMP	Analog Read	Temperature in degrees Celsius.
Configuration		
RESOLUTION	Analog R/W	The selected resolution of the temperature sensor. The TMP100 0.5 (default), 0.25, 0.125, and 0.0625 degrees Celsius. Writing this property will configure the sensor to deliver measurements to one of those resolutions. Note that higher resolutions require longer conversion times.
Device		
PEEK(n)	Digital Read	Reads the 8-bit device register *n*.
POKE(n)	Digital Write	Writes the 8-bit device register *n*.
BYTE(n)	Digital R/W	Reads or writes the 8-bit device register *n*.
WORD(n)	Digital R/W	Reads or writes the 16-bit device register *n*.

Texas Instruments TMP102 Driver

Installation

```
INSTALL "TEXAS-INSTRUMENTS-TMP102"
INSTALL "TMP102"
```

Options

ADDR=*integer*

Set the I2C 8-bit address. By default the driver uses the address `0x90`.

Description

Installs a temperature sensor driver for the TMP102.

Properties

Measurement		
`TEMP`	Analog Read	Temperature in degrees Celsius.
Configuration		
`RESOLUTION`	Analog R/W	The fixed resolution of the temperature sensor, 0.0625 degrees Celsius. Writing this property has no effect as the TMP102 has fixed resolution.
`MODE`	Digital R/W	Select TMP102 Extended Mode. Writing a nonzero value to `MODE` enables Extended Mode where temperatures to to 150 degrees Celsius can be measured and reported. The default for `MODE` is zero to set the TMP102 to Normal Mode.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.
`WORD(n)`	Digital R/W	Reads or writes the 16-bit device register *n*.

VTI SCA3000 Driver

Installation

`INSTALL "VTI-SCA3000" USING` *select*
`INSTALL "SCA3000" USING` *select*

Options

`DEVICE=`*string*

Sets the specific device variant, e.g. `DEVICE=D01`.

Description

Installs an accelerometer driver for the SCA3000 in SPI mode. The SCA has a fixed range, according to the device variant, and a fixed bandwidth.

Properties

General		
`VERSION`	String Read	A string containing the silicon revision of the SCA3000.
Accelerometer		
A	Analog Read	An array of three numbers containing the accelerations measured along the *x*, *y*, and *z* axes, in *g*.
AX *or* X	Analog Read	Acceleration measured along the *x* axis, in *g*.
AY *or* Y	Analog Read	Acceleration measured along the *y* axis, in *g*.
AZ *or* Z	Analog Read	Acceleration measured along the *z* axis, in *g*.
Algorithms		
`ROLL`	Analog Read	Roll angle φ about the *x* axis, in degrees; also called bank angle. This is computed directly from the accelerometer readings.
`PITCH`	Analog Read	Pitch angle θ about the *y* axis, in degrees; also called elevation. This is computed directly from the accelerometer readings.
Configuration		

`RANGE`	Analog R/W	Selected full scale range of the accelerometer, in *g*. Note that changing the range will reset the accelerometer `GAIN` and `BIAS` to the defaults for the selected range.
`BANDWIDTH`	Analog R/W	Selected bandwidth of the of the accelerometer, in hertz. This reads as 0 as the bandwidth is not configurable.
Calibration		
`BIAS`	Analog R/W	An array of three numbers containing the accelerometer bias, in *g*, for the *x*, *y*, and *z* axes.
`GAIN`	Analog R/W	An array of three numbers containing the gain for one LSB, in *g*, for the *x*, *y*, and *z* axes.
Device		
`PEEK(n)`	Digital Read	Reads the 8-bit device register *n*.
`POKE(n)`	Digital Write	Writes the 8-bit device register *n*.
`BYTE(n)`	Digital R/W	Reads or writes the 8-bit device register *n*.

Notes

See **Accelerometers**.

Watterott electronic mSD Shield

Installation

```
INSTALL "WATTEROTT-MSD-SHIELD"
INSTALL "MSD-SHIELD"
```

Options

None.

Description

Installing the MSD LCD Shield driver provides a 240×320 true color graphic display.

Resources

http://www.watterott.com/en/Arduino-mSD-Shield

Benchmarks

Here is the result of running the SolderCore Graphics Benchmarks application:

```
> run
Graphics display benchmark for WATTEROTT-MSD-SHIELD

   Circles:    3120 ms
     Discs:   10515 ms
Rectangles:     281 ms
     Slabs:    7284 ms
     Lines:    8723 ms
  Polygons:    4048 ms
      Text:    2801 ms
> _
```

Watterott electronic S65 Shield

Installation

```
INSTALL "WATTEROTT-S65-SHIELD"
INSTALL "S65-SHIELD"
```

Options

None.

Description

Installing the S65 LCD Shield driver provides a 176×132 true color graphic display.

Resources

http://www.watterott.com/en/Arduino-S65-Shield

Xterm Graphics Driver

Installation

```
INSTALL "XTERM-GRAPHICS"
```

Options

None.

Description

Installing the Xterm Graphics driver provides you with a graphics display that is emulated using the 256-color mode of an Xterm terminal emulator. The 24-bit color space is mapped to the 6x6x6 color cube supported by Xterm which provides a 216-color graphics capability. After installation, you can use all the CoreBASIC graphics commands to write to the display.

Because the graphics are emulated, this driver makes it possible to run every standard graphics demonstration without graphics hardware.

Notes

Tera Term provides a 256-color Xterm emulation.

See also

ANSI Graphics Driver

SolderCore Reference

This section is a software and hardware reference for the SolderCore and the accessories in the SolderCore product portfolio.

Arduino-style header pinout

The SolderCore has a standard Arduino pinout:

Pin	Digital	Analog	PWM Channel	Other	LM3S Port
D0	I/O	Input #15	Output #0	UART #0 Rx	D0
D1	I/O	Input #3		UART #0 Tx	E4
D2	I/O	Input #2			E5
D3	I/O	Input #1	Output #4		E6
D4	I/O		Output #5		G1
D5	I/O		Output #0		G0
D6	I/O	Input #14	Output #1		D1
D7	I/O		Output #1		F1
D8	I/O			UART #1 Tx	C7
D9	I/O		Output #6		C4
D10	I/O	Input #13	Output #2	UART #1 Rx	D2
D11	I/O		Output #7	MOSI	A5
D12	I/O		Output #6	MISO	A4
D13	I/O		Output #4	SCK	A2
A0	I/O	Input #4			D7
A1	I/O	Input #5			D6
A2	I/O	Input #6			D5
A3	I/O	Input #7			D4
A4	I/O			SDA	B3
A5	I/O			SCL	B2

Voltages

All I/O pins operating in digital mode are 5V tolerant.

All analog input voltages must be between 0V and 3V; voltages outside this range may well damage the microprocessor on the SolderCore. Analog-capable pins operating in digital mode are 5V tolerant.

Boot sequence

When you turn on your SolderCore, or reset the SolderCore using the small button on the PCB, the SolderCore performs a *cold start sequence*, or *boot sequence* to start networking, mount the microSD card, and start CoreBASIC. The exact sequence is:

1. Initialize all hardware functions.
2. Mount the `/c` drive from the microSD card.

Default boot sequence

If the microSD card cannot be mounted, the SolderCore uses a default boot sequence as follows:

1. Start the networking service.
2. Request a DHCP address from an available DHCP server.
3. Request the current time from a time server on the network.
4. Start the WINS server to respond to the name `core-`*xxxxxx* where *xxxxxx* is the serial number of the SolderCore found on the underside of the PCB.

Once the SolderCore is booted in this mode, you can find the IPv4 address assigned by the DHCP server by looking at the records maintained by your particular DHCP server. However, if you have the NetBIOS or the Windows Internet Name Service configured, you can simply use the name `core-`*xxxxxx* to address the SolderCore over the network.

To find the network-assigned IPv4 address using `ping`:

```
C:\Users\Paul>ping core-0000be

Pinging core-0000be.rowley.co.uk [10.0.0.46] with 32 bytes of data:
Reply from 10.0.0.46: bytes=32 time<1ms TTL=64
Reply from 10.0.0.46: bytes=32 time<1ms TTL=64
Reply from 10.0.0.46: bytes=32 time<1ms TTL=64
Reply from 10.0.0.46: bytes=32 time<1ms TTL=64

Ping statistics for 10.0.0.46:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Paul>
```

User boot sequence

If the microSD is mounted without error, the SolderCore uses a more elaborate boot sequence as follows:

1. Start the networking service.
2. Loads and run `/c/sys/!network.bas`, if it exists, to configure the network.
3. If DHCP is configured, request a DHCP address from an available DHCP server.
4. Request the current time from a time server on the network.
5. Start the WINS server to respond to the network name set in `NET.NAME`.

6. Loads and run `/c/sys/!boot.bas`, if it exists, to allow further user configuration.
7. Loads and run `/c/!run.bas`, if it exists, to automatically start a user application.

Benchmarking CoreBASIC

In the 1980s, Kilobaud published a set of benchmarks for BASIC interpreters, and those benchmarks were adopted by Personal Computer World (PCW), a magazine printed in the United Kingdom. They became the de-facto method to measure the performance of a BASIC interpreter. It's fun to see just how far things have progressed, so here is a table of results for some vintage computers and the SolderCore:

Computer	BM1	BM2	BM3	BM4	BM5	BM6	BM7	BM8	Avg
BBC B	0.6	3.2	8.1	8.8	9.9	14.3	21.9	48	14.3
IBM PC	1.5	5.2	12.1	12.6	13.6	23.5	37.4	35	17.6
Acorn Atom	0.5	5.1	9.5	10.8	13.9	19.1	31.1	92	22.8
VIC-20	1.4	8.3	15.5	17.1	18.3	27.2	42.7	99	28.7
Apple II	1.3	8.5	16.0	17.8	19.1	28.6	44.8	107	30.4
Dragon 32	1.6	10.2	19.7	21.6	23.3	34.3	50.0	129	36.2
Oric Atmos	1.6	15.2	25.4	27.4	33.0	45.6	68.5	136	44.1
SVI-328	1.6	5.4	17.9	19.6	20.6	30.7	42.2	236	46.7
ZX81 (fast)	4.5	6.9	16.4	15.8	18.6	49.7	68.5	229	51.2
Microtan65	1.9	12.8	24.7	27.8	29.6	43.2	68.9	243	56.5
ZX Spectrum	4.8	8.7	21.1	20.4	24.0	55.3	80.7	253	58.5
Oric-1	1.8	17.1	29.0	31.4	38.0	51.8	77.8	230	59.6
Atari 600XL	2.2	7.2	19.1	22.8	25.8	37.6	58.3	412	73.1
TI-99/4A	2.9	8.8	22.8	24.5	26.1	61.6	84.4	382	76.6
SolderCore	0.0016	0.013	0.031	0.034	0.037	0.058	0.092	0.078	0.043

The SolderCore's time to run all eight benchmarks is just 0.388 seconds, which is quicker than the fastest benchmark time recorded above—for a single benchmark!

Why does SolderCore run so fast? Well, it's a combination of pure grunt from an 80MHz Cortex-M3 core (the Cortex-M3 is a 32-bit processor rather than an 8-bit or 16-bit processor) and CoreBASIC is a very slick piece of software.

Here are the benchmarks that we ran:

BM1

```
***../examples/kilobaud-bm1.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "kilobaud-bm1" or |kilobaud-bm1.

BM2

```
***../examples/kilobaud-bm2.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "kilobaud-bm2" or |kilobaud-bm2.

BM3

```
***../examples/kilobaud-bm3.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "kilobaud-bm3" or |kilobaud-bm3.

BM4

```
***../examples/kilobaud-bm4.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "kilobaud-bm4" or |kilobaud-bm4.

BM5

```
***../examples/kilobaud-bm5.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "kilobaud-bm5" or |kilobaud-bm5.

BM6

```
***../examples/kilobaud-bm6.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "kilobaud-bm6" or |kilobaud-bm6.

BM7

```
***../examples/kilobaud-bm7.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "kilobaud-bm7" or |kilobaud-bm7.

BM8

```
***../examples/kilobaud-bm8.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "kilobaud-bm8" or |kilobaud-bm8.

The SolderCore bootloader

Before booting CoreBASIC, the integrity of the CoreBASIC firmware is verified by the SolderCore *bootloader*. The bootloader tries to prevent you bricking your SolderCore and that the SolderCore can always be recovered.

The SolderCore will upgrade the CoreBASIC firmware from the microSD slot when commanded. If there is an error upgrading the firmware, the bootloader will halt and report the error using the LEDs.

How the bootloader flashes your program

When the SolderCore is reset, the bootloader initializes the SolderCore hardware and checks the integrity of the installed CoreBASIC firmware. If the CoreBASIC firmware is intact, the bootloader transfers control to CoreBASIC and CoreBASIC continues with its standard boot sequence.

If the CoreBASIC firmware is damaged, the bootloader initiates a flash of the CoreBASIC firmware from the upgrade image stored in the file `/c/sys/core.fw` on the microSD card.

To flash the CoreBASIC firmware, the bootloader takes the following steps:

- Initialize the microSD card.
- Mount the microSD card as drive `/c`.
- Open the upgrade firmware image file `/c/sys/core.fw` for reading.
- Verify the integrity of the firmware image by decrypting the firmware, checking the CRC, and checking that the firmware label indicates that the firmware is intended for this model of SolderCore.
- Erase the current CoreBASIC firmware and verify that the firmware area in flash is blank.
- Read, decrypt, and flash the firmware image from `/c/sys/core.fw`.
- Once the image is flashed, verify the flashed image and check the firmware CRC.
- Reset the SolderCore to start running the new CoreBASIC firmware.

What to expect during firmware upgrade

Although the SolderCore and the CoreBASIC `FIRMWARE` commands provide many checks before requesting the bootloader flash the CoreBASIC firmware, it is still possible to *manually* place corrupt firmware, or firmware intended for another model, into `/c/sys/core.fw`. The SolderCore bootloader will, however, reject invalid firmware by checking the firmware CRC and the firmware label before reprogramming the firmware which makes it impossible, in normal operation, to brick your SolderCore.

During flashing, the LED next to the microSD card slot will be lit, indicating access to the SD card and flash reprogramming. The bootloader checks the firmware upgrade at each stage. If something unexpected happens during the upgrade, the the bootloader indicates the firmware upgrade failure by extinguishing the microSD LED and blinking both red IDL and USR LEDs on the SolderCore PCB: you'll find these LEDs next to the reset button.

LED blinks	Description
2	No media in microSD slot.

3	Media in microSD slot is not recognized as a FAT file system.
4	The media in the microSD slot is a valid FAT file system, but the file upgrade image `/c/sys/core.fw` does not exist.
5	Read error on `/c/sys/core.fw` indicating a corrupted FAT file system, or the microSD card was ejected during flashing.
6	The firmware upgrade image contained in `/c/sys/core.fw` does not pass CRC verification; the firmware or file system is corrupt.
7	The firmware upgrade image is intact and passes CRC verification but the upgrade firmware is not intended for this model of SolderCore.
8	Firmware erase or blank check failed.
9	Firmware flash programming failed.
10	Newly installed firmware failed CRC check.

Stellaris port mapping

The SolderCore has a number of signals that have dedicated functions. If you are programming the SolderCore in C, this section is a quick summary of the pinning of the SolderCore and the LM3S9D92 port assignment. However, if you are doing serious development with SolderCore, please refer to the schematic for exact details.

Internal signals

The signals internal to the SolderCore are mapped as follows:

SolderCore use	Stellaris port assignment
User LED	C5
Run LED	E7
microSD LED	J4
microSD Chip Select	G7
Ethernet Yellow LED	F2
Ethernet Green LED	F3
MEM1 Chip Select	J3
MEM2 Chip Select	J5

SolderCore headers

The signals brought out to the SolderCore headers are mapped as follows:

SolderCore pin	Stellaris port assignment
D0	D0
D1	E4
D2	E5
D3	E6
D4	G1
D5	G0
D6	D1
D7	F1
D8	C7
D9	C4
D10	D2
D11	A5
D12	A4
D13	A2

A0	D7
A1	D6
A2	D5
A3	D4
A4	B3
A5	B2

Alternative pinning

The SolderCore can be configured, using solder jumpers, to route pins A4 and A5 to Stellaris analog inputs rather than use them for digital I/O, primarily I2C. CoreBASIC does not support this configuration, but if you move the solder jumpers, the alternate routings are:

SolderCore pin	**Stellaris port assignment**
A4	E3
A5	E2

A historical perspective...

The SolderCore compares extremely well with a 1977 popular computer:

Parameter	PET 2001	SolderCore	Comparison
Introduced	1977	2012	35 years pass…
RAM	4K	96K	24× bigger
ROM	12K	512K	43× bigger
Processor speed	1MHz	80MHz	80× faster
Storage capacity	170KB per disk	16GB per SD card	100,000× bigger
Write speed	1.8KB per second	6MB per second	3,500,000× faster!
Cost	$795 + $795 = $1590	$80	1/20th the cost
Number of ICs	81	1	1/81× the chips

Actually, if we account for inflation, things become even more surreal. $795 in 1977 is approximately $4,500 in 2012; so, the cost of a Commodore PET plus a dual disk drive is $9,000—meaning the $80 SolderCore is 1/112th the adjusted price of a Commodore PET!

XMOS Firmware Development

This section describes how to build the Arcade Shield and LCD Shield firmware factory image from source and how to download them to the Arcade Shield using an XMOS XTAG-2 Debug adapter. We also describe how to prepare a firmware upgrade release.

Preparing a factory image release	Describes how to develop XMOS firmware to run in RAM for testing and how to flash that into the SolderCore Arcade Shield or LCD Shield.
Generating a fimware upgrade release	Describes how to prepare an upgrade image of new XMOS firmware and how to load that into the SolderCore Arcade Shield or LCD Shield using CoreBASIC and a SolderCore.

Preparing a factory image release

You will need:

- *The the XMOS Desktop Tools*: I happen to be using version 11.2.2 on Windows. You can download these open tools from XMOS, **http://www.xmos.com/products/development-tools**.
- *The firmware sources*: Download and extract the sources to the Arcade Shield and LCD Shield firmware from the SolderCore website.
- *An XTAG-2 programmer*: You can purchase an XTAG-2 programmer from a number of places, including DigiKey: **http://www.digikey.com/catalog/en/partgroup/xtag-2-debug-adapter-xcard-xtag-2/20055**
- *A 20-pin to 10-pin format converter*: You can purchase this from Rowley Associates.

Compiling the sources for a factory image

1. Open an XMOS command prompt.
2. Change to the firmware folder
3. Compile the sources...

To compile the Arcade Shield sources:

```
xcc -O3 8x8.c Graphics.c Main.xc VGAThread.xc ArcadeShield.xn -lflash -o factory.xe
```

To compile the LCD Shield sources:

```
xcc -O3 Main.xc 8x8.xc LCDShield.xn -lflash -o factory.xe
```

You now have an executable file, `factory.xe`, that contains the firmware for the target.

Running the factory firmware in RAM

First, connect the XMOS XTAG-2 adapter to the PC and make sure the drivers install. Next, plug the XTAG-2 debug adapter into the format converter and connect the 10-conductor cable from the format converter into the LCD or Arcade Shield using the red wire to ensure proper polarity between the format converter and the shield.

To run the firmware in RAM:

```
xrun factory.xe
```

That's all there is to it! However, the firmware will evaporate if the shield is reset or power cycled. If you want to make your new firmware a permanent addition, you need to write it to the SPI flash.

Burning the factory firmware into SPI flash

To burn the firmware into flash:

```
xflash factory.xe
```

You should see some messages like this:

```
Warning: F03098 Factory image and boot loader cannot be write-protected on flash device on
 node "0".
Site 0 has finished.
```

If you see some messages about the flash not being able to be identified, unplug the XTAG-2 and power cycle the shield and try again—the XTAG-2 can be a little flaky sometimes.

Generating a fimware upgrade release

The factory firmware programmed into the Arcade Shield and LCD shield is capable of upgrading itself over SPI. If you wish to build and distribute a firmware upgrade for the Arcade Shield or LCD Shield, you don't actually need an XMOS XTAG-2 adapter, as long as you are 100% confident your application will never crash!

This example shows how to compile firmware for upgrade, produce an upgrade image, and then convert that image into a CoreBASIC program that will present the image over SPI for a field upgrade by the SolderCore.

You will need:

- *The the XMOS Desktop Tools*: I happen to be using version 11.2.2 on Windows. You can download these open tools from XMOS, **http://www.xmos.com/products/development-tools**.
- *The firmware sources*: Download and extract the sources to the Arcade Shield and LCD Shield firmware from the SolderCore website.
- *A SolderCore running CoreBASIC*: You can purchase this from Rowley Associates.

Step 1. Build the upgrade firmware

You compile the sources as you would when creating a factory image to generate an '.xe' file; we'll call this `upgrade.xe`.

Step 2. Build the upgrade image

You use xflash to prepare the upgrade image. Each upgrade image requires a unique number; the bootloader will choose the firmware with the highest upgrade image number. We typically use the date of distribution of the firmware image as the version number, in year-month-day format, and this date is displayed by the shield firmware on its splash screen.

```
xflash --upgrade 20120426 upgrade.xe -o upgrade.xi
```

We now have a binary upgrade image, `upgrade.xi`, which can be placed into the SPI flash.

Step 3. Build the conversion utility

The SolderCore can run programs that send data over an SPI bus. So, to get the image into the shield, we format the image into a CoreBASIC program which the SolderCore can run and send the upgrade image to the shield.

You can compile the program `mkdata.c` which processes the '.xi' file into CoreBASIC DATA statements. Microsoft offer Visual Studio Express which can do this for you. Here I'm compiling with Visual Studio 2008 Standard Edition:

```
E:\boards> cl mkdata.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.21022.08 for 80x86
```

```
Copyright (C) Microsoft Corporation.  All rights reserved.

mkdata.c
Microsoft (R) Incremental Linker Version 9.00.21022.08
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:mkdata.exe
mkdata.obj

E:\boards>
```

Step 4. Convert upgrade image to DATA statements

Now I process the firmware image:

```
E:\boards> cd ArcadeShield
E:\boards\ArcadeShield> ..\mkdata upgrade.xi >upgrade.bas
E:\boards\ArcadeShield>
```

Step 5. Merge firmware upgrade fragment

Now I have the raw data as a set of CoreBASIC DATA statements; all that's left to do now is prepend the following to those DATA statements:

```
  10 INSTALL "SOLDERCORE-ARCADE-SHIELD" AS U
  20 READ L
  30 PRINT "New firmware image requires "; L; " bytes."
  40 PRINT "Upgrading firmware."
  50 ' Send Firmware Upgrade command.
  60 U.WORD = 0xF00DBABE
  70 U.WORD = 0xDEADBEEF
  80 ' Send size of new firmware.
  90 U.WORD = L
 100 ' Send firmware image.
 110 WHILE L > 0
 120   READ X : U.WORD = X
 130   L = L - 4
 140 WEND
 150 PRINT
 160 PRINT "Firmware upgraded.  Please power cycle your device."
 170 END
```

Step 6. Upgarade the firmware!

Now you can load this into a SolderCore, execute it, and upgrade your firmware:

```
> load "/c/upgrade.bas"
> run
New firmware image requires 22784 bytes.
Upgrading firmware.

Firmware upgraded.  Please power cycle your device.
>
```

Done!

Example programs

Here are a few examples we cooked up with CoreBASIC.

Timing methods How to time how long things take using the timers CoreBASIC provides.	**Removing noise** A simple filter to eliminate noise from a signal.	**Deinterlacing samples** How to deinterlace samples gathered by the SAMPLE function.
Median filtering How to apply a median filter for algorithms such as touch screen samples.	**Monte Carlo simulation** Use random numbers to simulate and estimate.	**Parsing GPS sentences** Use CoreBASIC's advanced string and array handling to easily pick apart GPS sentences.
Calibrating touch screens Calibrate your touch screen and apply those calibrations to new samples.	**Four-parameter calibration of a compass for hard iron effects** Eliminate hard iron effects from your magnetometer readings.	**Reading an MPL115A1 pressure sensor using SPI** Read from a pressure sensor using CoreBASIC's built-in SPI support.
Downloading firmware using CoreBASIC An example that demonstrates network and file access to download new firmware by hand.	**ITead Studio LCDs** Select a working combination of components from ITead Studio for your SolderCore.	**Conway's Game Of Life** Run Conway's classic Game of Life on nothing but a SolderCore.

Hangman Play a version of Hangman with the computer choosing a randome word from a web server.	**Sign a Twitter request with an OAuth signature** Shows how to create an OAuth signature for a Twitter request.	

Timing methods

The SolderCore has three different ways of measuring time:

- `CORE.TIME`, the time of day clock, counts once per second.
- `TIMER`, the millisecond timer, counts 1000 times per second, but is only accurate to 1/100th of a second.
- `CORE.TICK`, the high-precision timer, counts `CORE.FREQUENCY` times per second. For SolderCore, the core frequency is 80 MHz and `CORE.TICK` increments 80 million times per second.

Which timer you use depends upon the type of time you're looking to measure, how long you need to measure over, and what accuracy you would like to achieve.

Let's say that we'd like to time how long it takes to run one million `FOR…NEXT` iterations; we will do this using the three timing methods available to us.

Time of day clock

The time of day clock `CORE.TIME` increments once per second. The format of `CORE.TIME` happens to be *Unix* or *POSIX* time, which is the number of elapsed seconds since midnight UTC, 1 January 1970 and is the standard way of representing time in CoreBASIC. See **Unix time - Wikipedia, the free encyclopedia**.

```
> list
   10 LIMIT = 10000000
   20 PRINT "About to time a count to "; LIMIT; " using time of day"
   30 T0 = CORE.TIME
   40 WAIT CORE.TIME <> T0  ' wait for one second tick to advance
   50 T0 = CORE.TIME
   60 FOR I = 1 TO LIMIT : NEXT I
   70 SECONDS = CORE.TIME - T0
   80 PRINT "That took "; SECONDS; " seconds..."
   90 PRINT "...or "; SECONDS / LIMIT * 1000000; " microseconds/iteration"
  100 END
> run
About to time a count to 10000000 using time of day
That took 16 seconds...
...or 1.6 microseconds/iteration
> _
```

Millisecond timer

The millisecond timer `TIMER` increments by 10 counts each centisecond. You can use `TIMER` to measure periods of up to $2^{31}-1$ milliseconds, which is 2,147,483 seconds, or just under 25 days, to one hundredth of a second:

```
> list
   10 LIMIT = 10000000
   20 PRINT "About to time a count to "; LIMIT; " using millisecond timer"
   30 TIMER = 0
   40 FOR I = 1 TO LIMIT : NEXT I
   50 SECONDS = TIMER / 1000
   60 PRINT "That took "; SECONDS; " seconds..."
```

```
   70 PRINT "...or "; SECONDS / LIMIT * 1000000; " microseconds/iteration"
   80 END
> run
About to time a count to 10000000 using millisecond timer
That took 15.8 seconds...
...or 1.58 microseconds/iteration
> _
```

High-precision timer

The high-precision timer `CORE.TICK` increments `CORE.FREQUENCY` times per second. The actual frequency delivered by `CORE.FREQUENCY` is dependent upon which microprocessor is running CoreBASIC: for SolderCore, `CORE.TICK` increments 80 million times per second, and for the Freescale Freedom board it increments 24 million times per second.

Considering SolderCore, you can use `CORE.TICK` to measure periods of up to 2^{31}–1 / 80,000,000 seconds, which is about 26.84 seconds, to an accuracy of 1/80,000,000th of a second:

```
> list
   10 LIMIT = 10000000
   20 PRINT "About to time a count to "; LIMIT; " using high precision timer"
   30 T0 = CORE.TICK
   40 FOR I = 1 TO LIMIT : NEXT I
   50 SECONDS = (CORE.TICK - T0) / CORE.FREQUENCY
   60 PRINT "That took "; SECONDS; " seconds..."
   70 PRINT "...or "; SECONDS / LIMIT * 1000000; " microseconds/iteration"
   80 END
> run
About to time a count to 10000000 using high precision timer
That took 16.051 seconds...
...so it is 1.6051 microseconds/iteration
> _
```

If you run this on a Freedom board:

```
> run
About to time a count to 10000000 using high precision timer
That took 39.4006 seconds...
...or 3.94006 microseconds/iteration
> _
```

Removing noise

If you have a noisy signal, you might like to remove that noise by using a simple filter. For instance, the SparkFun Spectrum Shield is prone to noise on both left and right channels which leads to unsightly graphic equalizer displays. We'd like to eliminate the noise, and we have a couple of simple ways to do this.

The first is to simply force values below a threshold to zero:

```
> v = rnd con(10)
> print v
[0.513855, 0.17572, 0.308624, 0.534515, 0.947601, 0.171722, 0.702209, 0.22641, 0.494751,
 0.124695]
> print v * (v >= 0.2)
[0.513855, 0, 0.308624, 0.534515, 0.947601, 0, 0.702209, 0.22641, 0.494751, 0]
> _
```

This code generates an array of Boolean values, either zero or one, where each value indicates whether the element exceeds the noise threshold; 0.2 in this case. Multiplying this array by the original array forces values less than the threshold to zero and leaves other values unchanged.

If we wish to remove those noise values completely, we can use `PICK`:

```
> v = rnd con(10)
> print v
[0.513855, 0.17572, 0.308624, 0.534515, 0.947601, 0.171722, 0.702209, 0.22641, 0.494751,
 0.124695]
> print pick(v, v >= 0.2)
[0.513855, 0, 0.308624, 0.534515, 0.947601, 0, 0.702209, 0.22641, 0.494751, 0]
> _
```

This uses the same Boolean array as before, and selects only those values which we are interested in, discarding the rest.

Of course, it's possible to use different selection criteria; for instance, to select only those values falling into a particular band or interval:

```
> v = rnd con(10)
> print v
[0.513855, 0.17572, 0.308624, 0.534515, 0.947601, 0.171722, 0.702209, 0.22641, 0.494751,
 0.124695]
> print pick(v, 0.2 <= v and v <= 0.6)
[0.513855, 0.308624, 0.534515, 0.22641, 0.494751]
> _
```

As they say, the possibilities are endless...

Deinterlacing samples

The SAMPLE function delivers its results in a single array:

```
> v = sample(core.a17; core.a18; core.a19, 3)
> print v
[0.083893, 0.389618, 0.277222, 0.36804, 0.983429, 0.53537, 0.765656, 0.646454, 0.76712]
> _
```

This might be inconvenient for some algorithms or digital filters, which require sets of samples from a single source. The samples from A3 are held in the array at indexes 0, 3, 6; the samples from A4 are held at 1, 4, 7; and the samples from A5 are at 2, 5, and 8. We can use GEN together with SELECT to deinterlace these:

```
> a3 = select(v, gen(0 to high v step 3))
> print a3
[0.083893, 0.36804, 0.765656]
> a4 = select(v, gen(1 to high v step 3))
> print a4
[0.389618, 0.983429, 0.646454]
> a5 = select(v, gen(2 to high v step 3))
> print a5
[0.277222, 0.53537, 0.76712]
> _
```

Median filtering

As a follow-on from the previous sections, we can now put this into practice by using a median filtering algorithm to filter the samples from a resistive touch panel.

We'll present only the sampling and filtering in this section; setting up the resistive touch panel depends very much on how the panel is connected, so we'll assume that it is already set up and all we need to do is sample the inputs.

Let's assume that we will be sampling the panel on analog input 2. We'll take `N` samples.

The median is found by sorting all the samples into ascending order and picking the middle one. If there is an even number of samples, there is no single middle value and the median is then the average of the two middle values.

We can code it like this:

```
T = SORT SAMPLE(CORE.A15, N)
N = N / 2
IF N \<\> INT N THEN M = T(N) ELSE M = (T(N-1) + T(N)) / 2
```

After this, `M` contains the median value of the samples.

Monte Carlo simulation

The value of π has been a lingering interest for me ever since I received a calculator, as a present from my aunt, with a single unfathomable symbol on one of its keys. The off-the-cuff remark by a mathematics teacher one afternoon, simply stating that you can approximate π using random numbers, set off a light bulb. After inquiring a bit deeper after the lesson, I went away and wrote a program to approximate π during the morning break.

The approximation of π by the Monte Carlo method is simple to understand:

- Draw a square, then inscribe a circle within it.
- Scatter rice over the square.
- Count the total number of grains scattered and those grains falling within the circle.
- Calculate the ratio of the number of grains falling within the circle to the total number of grains scattered. This is simply an estimate of the ratio of the two areas, which is π/4.

Rather than scattering rice over a circle, we can use a computer to virtually scatter a single grain of rice using two random numbers. The two numbers, both between 0 and 1, represent the *x* and *y* coordinate of the grain of rice.

We use Pythoagoras to determine whether the grain of rice lies within a unit circle; it lies in the unit circle if $x^2 + y^2 \leq 1$.

We continue to scatter rice, or generating random numbers, and so develop a more accurate estimation of π—as long as the random numbers are indeed uniformly random, and we don't start repeating.

Here's a single line of CoreBASIC to provide an estimate of *pi* using n grains of rice:

```
PRINT 4 * SUM((RND CON(N))^2 + (RND CON(N))^2 \<= 1)/N
```

We can scatter 10 grains and see how well we do:

```
> n = 10
> print 4 * sum((rnd con(n))^2 + (rnd con(n))^2 <= 1)/n
2.4
> print 4 * sum((rnd con(n))^2 + (rnd con(n))^2 <= 1)/n
3.2
> print 4 * sum((rnd con(n))^2 + (rnd con(n))^2 <= 1)/n
3.6
> _
```

Well, not too accurate. We can try 1,000:

```
> n = 1000
> print 4 * sum((rnd con(n))^2 + (rnd con(n))^2 <= 1)/n
3.10797
> print 4 * sum((rnd con(n))^2 + (rnd con(n))^2 <= 1)/n
3.18397
> print 4 * sum((rnd con(n))^2 + (rnd con(n))^2 <= 1)/n
3.11197
> _
```

The SolderCore will run this single-liner up to approximately 2,000 grains, but not much beyond, because of memory constraints. To venture beyond 2,000 grains, we can construct a simple program which scatters one grain at a time, and tallies everything at the end:

```
> list
***../examples/monte-carlo-pi.bas not found ***> run
How many iterations? 1000
After 1000 iterations an approximation to pi is 3.152
> run
How many iterations? 10000
After 10000 iterations an approximation to pi is 3.1416
> _
```

You can load this into CoreBASIC using EXAMPLE `"monte-carlo-pi"` or `|monte-carlo-pi`.

You might like to compare how fast array operations are compared to pure iteration. Try timing the program above over a million operations. Now, rewrite the program to scatter 1,000 grains at a time, and iterate that 1,000 times so that you scatter one million grains.

Which do you predict to be faster? And which is actually faster?

Parsing GPS sentences

Although CoreBASIC has a GPS driver that will validate and parse GPS sentences, it's interesting to show how to pick apart a GPS sentence and illustrate the simplicity and elegance of CoreBASIC.

Validating the checksum

A GPS sentence starts with a dollar sign and ends with a checksum after an asterisk. The checksum is computed over the sentence between the dollar and asterisk.

Here is a simple sentence:

```
S = "$GPRMC,192157.110,A,4208.3427,N,02445.0243,E,0.13,92.58,211111,,,A*5E"
```

To validate the checksum, we will start by discarding the dollar sign:

```
S = S(1 TO)
```

We could have written this in functional form using `MID(S, 1)`; it comes down to personal preference.

Now we divide the string into two at the asterisk:

```
V = SPLIT("*", S)
```

After execution of the SPLIT, `V(0)` contains the GPS sentence and `V(1)` contains the 8-bit checksum as two hexadecimal numerals.

The GPS checksum is a longitudinal redundancy check over the ASCII characters in the sentence. A longitudinal redundancy check computes the even parity over a parallel set of bits along the sentence; it just happens that the XOR Boolean operation does this for us.

We must iterate over each character in `V(0)` and exclusive-or a running checksum with the ASCII value of each character. The first thing that comes to mind is a loop like this:

```
CHECKSUM = 0
FOR I = 0 TO HIGH V(0)
  CHECKSUM = CHECKSUM XOR V(0)(I)
NEXT I
```

This would do the trick, but it's rather clumsy. Instead, we can use the EXPAND and REDUCE functions to compute the checksum for us. EXPAND *x*, when *x* is a string, creates a new array where each value in the array is the ASCII code of the corresponding character in *x*. Using REDUCE to apply the XOR operation over the expanded data gives us the checksum:

```
CHECKSUM = REDUCE(XOR, EXPAND V(0))
```

Finally, we confirm that our computed checksum matches the two-digit checksum following the sentence by converting the binary checksum to a hexadecimal string and comparing:

```
IF HEX CHECKSUM = V(1) THEN PRINT "GOOD" ELSE PRINT "BAD"
```

Tidying up, the code to validate the checksum is:

```
S = "$GPRMC,192157.110,A,4208.3427,N,02445.0243,E,0.13,92.58,211111,,,A*5E"
V = SPLIT("*", S(1 TO))
CHECKSUM = REDUCE(XOR, EXPAND V(0))
IF HEX CHECKSUM = V(1) THEN PRINT "GOOD" ELSE PRINT "BAD"
END
```

Separating the fields

Now that we have validated the checksum, we turn to separating the fields of the GPS sentence. Each field in a GPS sentence is separated from the next by a comma; we can do that using SPLIT again:

```
V = SPLIT(",", V(0))
```

Now V contains an array of strings which correspond to the fields in the GPS sentence. The first field is the sentence type, and the NMEA standard defines a number of standard sentence types. We'll only decode the sentence type GPRMC, the recommended minimum *essential* GPS data. We can use a CASE statement to switch to parsing routines for each sentence type:

```
CASE V(0)
  WHEN "GPRMC": CALL GPRMC(V)
  WHEN "GPGGA": CALL GPGGA(V)
  WHEN "GPGSA": CALL GPGSA(V)
  OTHERWISE PRINT "Unhandled GPS sentence type "; V(0)
ENDCASE
```

You can extend this with additional sentence types by following the format above and adding extra WHEN clauses.

Implementing GPRMC is now simply a matter of examining and decoding each of the fields in the array. Showing the time is easy enough:

```
DEFPROC GPRMC(X)
  PRINT "GPS time is "; X(1)
```

Latitude and longitude are spread over two fields: one gives the angle and the other says whether it's north or south, and east or west. We would rather combine the the two fields into one where north/south is indicated by sign, as is east/west:

```
  LAT = IFF(X(4) = "N", VAL X(3), -VAL X(3))
  LONG = IFF(X(6) = "E", VAL X(5), -VAL X(5))
  PRINT "Latitude:  "; LAT
  PRINT "Longitude: "; LONG
ENDPROC
```

Adding more processing is simply a matter of more string processing. We can stub-out the procedures to decode the other sentences and, in the best traditions of text books, leave it as an exercise for the reader:

```
DEFPROC GPGSA(X) : ENDPROC
DEFPROC GPGGA(X) : ENDPROC
```

Here is the full application:

```
S = "$GPRMC,190415,A,3456.246,N,07650.437,W,000.0,0,040505,0,W*7A"
V = SPLIT("*", S(1 TO))
CHECKSUM = REDUCE(XOR, EXPAND V(0))
IF HEX CHECKSUM = V(1) THEN PRINT "GOOD" ELSE PRINT "BAD"
V = SPLIT(",", V(0))
CASE V(0)
  WHEN "GPRMC": CALL GPRMC(V)
  WHEN "GPGGA": CALL GPGGA(V)
  WHEN "GPGSA": CALL GPGSA(V)
  OTHERWISE PRINT "Unhandled GPS sentence type "; V(0)
ENDCASE
END

DEFPROC GPRMC(X)
  PRINT "GPS time is "; X(1)
  LAT = IFF(X(4) = "N", VAL X(3), -VAL X(3))
  LONG = IFF(X(6) = "E", VAL X(5), -VAL X(5))
  PRINT "Latitude:  "; LAT
  PRINT "Longitude: "; LONG
ENDPROC

DEFPROC GPGSA(X) : ENDPROC
DEFPROC GPGGA(X) : ENDPROC
```

Calibrating touch screens

One of the critical tasks that a developer needs implement when using a touch panel is simply calibration: making sure that touches on the panel are accurately mapped to pixel positions on the LCD.

There are a number of ways to achieve this, but a nice algorithm is presented by Analog Devices in Application Note AN-1021:

http://www.analog.com/static/imported-files/application_notes/AN-1021.pdf

We've implemented this algorithm in CoreBASIC and it works out very nicely using CoreBASIC's matrix functions.

```
***../examples/touch-calibration-demo.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "touch-calibration-demo"` or `|touch-calibration-demo`.

```
> example "touch-calibration-demo"
Connecting to www.soldercore.com (192.232.216.121)...
Loading touch-calibration-demo.bas from network...
Program loaded and ready. Type RUN to execute.
> run
Solution vectors

KX =
1.01116
-0.00188613
-25.7803

KY =
0.0097182
1.00611
-126.257

0       Calibrated=     3938    3856    Error=  7       7
1       Calibrated=     2044    3854    Error=  -3      5
2       Calibrated=     162     3843    Error=  -2      -6
3       Calibrated=     3925    2044    Error=  -6      -3
4       Calibrated=     2047    2034    Error=  0       -13
5       Calibrated=     167     2053    Error=  3       6
6       Calibrated=     3932    245     Error=  1       -1
7       Calibrated=     2046    250     Error=  -1      4
8       Calibrated=     165     249     Error=  1       3
> _
```

Four-parameter calibration of a compass for hard iron effects

This example shows how to use CoreBASIC's matrix mathematics to correct magnetometer samples for hard iron effects.

The theory

There are a number of ways to achieve this, but a nice algorithm is presented by Freescale in Application Note AN4246:

http://cache.freescale.com/files/sensors/doc/app_note/AN4246.pdf

We've implemented this algorithm in CoreBASIC and it works out very nicely using CoreBASIC's matrix functions. This example replicates *Worked Example 1* in the application note, to make sure that we get the implementation right.

```
***../examples/compass-calibration-demo.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "compass-calibration-demo"` or `|compass-calibration-demo`.

```
> example "compass-calibration-demo"
Connecting to www.soldercore.com (192.232.216.121)...
Loading compass-calibration-demo.bas from network...
Program loaded and ready. Type RUN to execute.
> run
Known dependent variables vector Y:
 95189.4
 76457.9
 76310.3
110773.5
 85487.9
 96260.6

Known magnetometer measurements X:
   167.4          -242.4           91.7        1
   140.3          -221.9           86.8        1
   152.4          -230.4           -0.6        1
   180.3          -270.6           71.0        1
   190.9          -212.4           62.7        1
   192.9          -242.4           17.1        1

Solution vector BETA:
   311.5
  -478.3
    91.7
-81304.8

Hard iron vector V:
   155.7
  -239.1
    45.8
```

```
Geomagnetic field strength B: 47.2405 uT
> _
```

Looking at the results, the calculated solution vector BETA(3), and consequently the field strength, are very slightly different from the calculated values of the application note. This is simply because of the way the matrix inverse is computed in CoreBASIC, using single precision floating point.

For real...

To make this example real, I modified the program above to use a real magnetometer and display results on an LCD. To replicate this setup, you will require:

- a SolderCore, to run the algorithms
- a SenseCore shield, to mount the magnetometer
- a SolderCore LCD Shield, to display the results
- a CoreMPU, which contains a magnetometer to calibrate
- a Liquidware Lithium Backpack, for battery-powered operation
- a 2x2 flat-four base to mount it all on.

The only tricky part was connecting the Lithium Backpack to the 2x2 base. I soldered a couple of screw terminals from RS (part number 220-4260) onto the sites for them on the 2x2 base. Then I simply connected the 5V output of the Lithium Backpack to the 5V input on the base using some hook-up wire. With all the other components in place, you can flick the switch on the backpack to "Batt", and you have a 100% autonomous computing platform!

To calibrate the compass, run the program, disconnect any cables, and rotate the base 360 degrees slowly. What you'll see is some axes, and a plot of red circles corresponding to the uncompensated magnetometer readings.

After 10 seconds of rotating and collecting raw magnetometer readings, the hard iron effect is calculated and then another ten seconds of readings are taken, and a plot of green circles corresponding to the calibrated magnetometer readings. When I run it, I managed to see a plot like this:

And here is the code:

```
***../examples/corempu-compass-calibration.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "corempu-compass-calibration"` or `|corempu-compass-calibration`.

Digital spirit level

The following application uses a simple 16×2 LCD and an accelerometer to implement a digital spirit level.

```
***../examples/digital-spirit-level.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE `"digital-spirit-level"` or `|digital-spirit-level`.

Reading an MPL115A1 pressure sensor using SPI

This example shows how to use the SPI keyword and the SPI-DEVICE driver to communicate with a Freescale MPL115A1 pressure sensor. CoreBASIC has no direct support for the MPL115A1 (though that will change), so this is a realistic example of what you need to do to get a reading from a sensor from scratch.

Setup

The equipment uses a SenseCore and a SparkFun MPL115A1 Breakout that is wired to a CoreProto prototyping board. The CoreProto and mounted MPL115A1 sensor are then plugged into site D on the SenseCore. Of course, you could wire the sensor to the SolderCore SPI bus directly, and use one of the SolderCore digital I/Os as a slave select, but the SenseCore and plug-in sensors makes set up and test very convenient.

The code

Well, here is the code with plenty of explanation:

```
***../examples/spi-mpl115a1-demo.bas not found ***
```

You can load this into CoreBASIC using EXAMPLE "spi-mpl115a1-demo" or |spi-mpl115a1-demo.

Running the test code

With TEST set to TRUE, the substituted test data confirms correct implementation of the calculations:

```
> run

MPL115A1 Read Pressure Demo
===========================

Pressure offset coefficient:         2107.87
Pressure sensitivity coefficient:    -2.49512
Temperature offset coefficient:      -1.02069
Temperature sensitivity coefficient: 0.000866652

Raw temperature (Tadc): 513 counts
Raw pressure    (Padc): 415 counts

Compensated pressure (PCOMP): 733.293

Pressure is 96.5924 kPa
```

Running for real

When this is run for real in Dursley on an autumn day, the output looks like this:

```
> run

MPL115A1 Read Pressure Demo
===========================

Pressure offset coefficient:         2053.37
Pressure sensitivity coefficient:    -2.41748
Temperature offset coefficient:      -1.00311
```

```
Temperature sensitivity coefficient: 0.000843048

Raw temperature (Tadc): 519 counts
Raw pressure    (Padc): 380 counts

Compensated pressure (PCOMP): 780.383

Pressure is 99.5844 kPa

> _
```

Drawing the flag of the United Kingdom

This example shows how to use the graphics commands in CoreBASIC to draw the complex Union Flag, the flag of the United Kingdom.

```
***../examples/union-flag.bas not found ***
```

Dowloading

You can load this into CoreBASIC using `EXAMPLE "union-flag"` or `|union-flag`.

Downloading firmware using CoreBASIC

This example shows how to use the network, stream, and file capabilities of CoreBASIC to interact with a web server. The intent of the program is to download a binary file that contains the latest CoreBASIC firmware from the SolderCore server, and as such, replicates what `FIRMWARE GET` would do.

```
***../examples/firmware-download-demo.bas not found ***
```

Dowloading

You can load this into CoreBASIC using `EXAMPLE "firmware-download-demo"` or `|firmware-download-demo`.

Running

When you run the program, you see something like this:

```
> run
Opened a socket to www.soldercore.com
header- Date: Wed, 05 Dec 2012 17:58:23 GMT
header- Server: Apache
header- Last-Modified: Fri, 28 Sep 2012 16:24:02 GMT
header- Accept-Ranges: bytes
header- Content-Length: 491520
header- Content-Type: text/plain
header-
Received 491520 bytes...
Downloaded firmware.  Use FIRMWARE CHECK to verify.
> firmware check
Verifying integrity of upgrade firmware...
Verified 491,520 bytes of 491,520 (100%)...with good CRC.
Upgrade firmware is verified to work on soldercore-v1.
Installed firmware is 1.2.13; upgrade firmware is 1.0.0.
Installed firmware differs from upgrade firmware.
> _
```

Bouncing lines

The following application runs best on a SolderCore Arcade Shield to provide fast animation. You can run it on a SolderCore LCD Shield too, but the effect isn't quite as explosive. Running it on many other graphics shields will provide a dismal experience because they are designed to be cheap rather than perform well.

```
***../examples/bouncing-lines.bas not found ***
```

Dowloading

You can load this into CoreBASIC using `EXAMPLE "bouncing-lines"` or `|bouncing-lines`.

ITead Studio LCDs

ITead Studio offers a bewildering array of LCD components and shields, and this guide will to assist you in selecting a working combination on SolderCore. We compare only the LCDs that you can plug directly into the SolderCore, not the serial displays that require special wiring.

You can purchase LCDs as components that plug into the ITDB02 shields. There are a number of LCDs on offer, and two versions of the ITDB02 shield to choose from. In addition, ITead Studio has integrated the 2.4D and 2.8S displays and IDB02 shield into a single shield with a much enhanced appearance.

LCDs

You can choose from the following LCDs:

LCD Part Number	Model	Size	Mode	Touch?	Link
ITDB02-2.2	DIS025	2.2"	8-bit	Yes	2.2" TFT LCD Screen Module
ITDB02-2.4DWOT	DIS009	2.4"	8-bit	No	2.4" TFT LCD Screen Module
ITDB02-2.4D	DIS001	2.4"	8-bit	Yes	No longer sold, replaced by ITDB02-2.4E
ITDB02-2.4E	DIS001	2.4"	8-bit	Yes	2.4" TFT LCD Screen Module
ITDB02-2.8	DIS026	2.8"	8-bit	Yes	2.8" TFT LCD Screen Module
ITDB02-3.2S	DIS002	3.2"	16-bit	Yes	3.2" TFT LCD Screen Module
ITDB02-3.2WD	DIS012	3.2"	16-bit	Yes	3.2" TFT LCD Screen Module
ITDB02-4.3	DIS029	4.3"	16-bit	Yes	4.3" TFT LCD Screen Module (*)
ITDB02-5.0	DIS030	5.0"	16-bit	Yes	5.0" TFT LCD Screen Module

(*) This page has some problems at the time of writing: it makes reference to the 3.2-inch display, the ITB02-3.2S, which is incorrect.

ITDB02 versions

ITead Studio have made two versions of the ITDB02 shield. The first version, ITDB02 v1.2 (which we will refer to as v1 from now on), requires you to shunt links to select either an 8-bit or 16-bit LCD interface. The second version, ITDB02 v2, supports only an 8-bit LCD interface but offers selection between 3.3V and 5V interfaces.

So, what's the deal? Well, SolderCore will work correctly with either ITDB02 shield, but the IDB02 v2 will not support 16-bit-only LCDs such as the 3.2S, 3.2WD, 4.3, or 5.0, which is a shame.

The manual for the v1.2 shield is here:

ITDB02 Arduino shield v1.2

The manual for the v2 shield is here:

ITDB02 Arduino shield v2.0

Compatibility matrix

The following will show you which combinations of components will be successful with SolderCore, and which drivers to install to get going.

LCD Part Number	ITDB02 version	Mode	Touch?	Install
ITDB02-2.2		8-bit	Yes	Not supported
ITDB02-2.4DWOT	v1 or v2	8-bit	No	`ITDB02-2.4D`
ITDB02-2.4D	v1 or v2	8-bit	Yes	`ITDB02-2.4D`
ITDB02-2.4E		8-bit	Yes	`ITDB02-2.4E`
ITDB02-2.8	v1 or v2	8-bit	Yes	`ITDB02-2.8`
ITDB02-3.2S	v1	16-bit	No	`ITDB02-3.2S`
ITDB02-3.2WD	v1	16-bit	No	`ITDB02-3.2WD`
ITDB02-4.3	hand wire	16-bit	No	`ITDB02-4.3`
ITDB02-5.0	hand wire	16-bit	No	`ITDB02-5.0`

Hand-wiring and ITDB02 connections

In our experience, and that of others, the ITDB02-4.3 and ITDB02-5.0 LCD modules will not work reliably with the ITDB02 v1.2 shield. You can wire these LCD displays directly to the SolderCore using the 16-bit ITDB02 connections listed below as the SolderCore and LCD modules both work at 3.3V.

The following table is a memento, in case this information decays from the Internet. The ITDB02 v2 supports 8-bit mode only, the ITDB02 v1 supports both 8-bit mode and 16-bit mode.

Pin	8-bit mode	16-bit mode
D0	DB8	DB8
D1	DB9	DB9
D2	DB10	DB10
D3	DB11	DB11
D4	DB12	DB12
D5	DB13	DB13

D6	DB14	DB14
D7	DB15	DB16
D8	D_CS	DB0
D9	D_DOUT	DB1
D10	SD_CS	DB2
D11	SD_IN	DB3
D12	SD_OUT	DB4
D13	SD_CLK	DB5
A0	D_DIN	DB6
A1	D_CLK	DB7
A2	RST	RST
A3	CS	CS
A4	WR	WR
A5	RS	RS

ITead Studio reference

A concise reference for the displays offered by ITead Studio is found here:

http://iteadstudio.com/product/itdb02-tft-lcd-display-series/

Conway's Game Of Life

Here we present an implementation of Conway's classic Game of Life which requires nothing more than your SolderCore to run. It uses a text display to show the cells as each generation is computed.

Conway's Game of Life - Wikipedia, the free encylopedia

The code

```
***../examples/life.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "life"` or `|life`.

While showing life on the console is great, it's really neat displaying it on an array of LEDs. So, if you have a Jimmie Rodgers LoL shield, you can do just that:

```
***../examples/lol-life.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "lol-life"` or `|lol-life`.

Hangman

This is a version of Hangman with the computer choosing a word from the web server **http://www.randomword.net/**.

The basics of the game are fully implemented except you have you have unlimited guesses at the secret word. If you'd like to implement this, and some ASCII art or graphics that shows the current state of the gallows, great!

The code

```
***../examples/hangman.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "hangman"` or `|hangman`.

Sign a Twitter request with an OAuth signature

This example shows how you can use the power of CoreBASIC to sign a Twitter API call with an OAuth HMAC-SHA1 signature.

The example is taken from this page:

https://dev.twitter.com/docs/auth/creating-signature

When you run this, the correct OAuth signature is shown in hex and Base64 form:

```
> run
Raw hex signature:

["B6", "79", "C0", "AF", "18", "F4", "E9", "C5", "87", "AB", "8E", "20", "0A", "CD", "4E",
 "48", "A9", "3F", "8C", "B6"]

OAuth signature:

tnnArxj06cWHq44gCs1OSKk/jLY=
> _
```

The code

```
***../examples/twitter-oauth-signature.bas not found ***
```

You can load this into CoreBASIC using `EXAMPLE "twitter-oauth-signature"` or `|twitter-oauth-signature`.

Additional Information

CoreBASIC development history

CoreBASIC has its roots in a BASIC interpreter I wrote for the Texas Instruments MSP430, **Butterfly BASIC**. Butterfly BASIC created interest and some positive feedback, but many prospective projects required something more advanced than the simple Butterfly BASIC interpreter; and, in addition, these projects needed capabilities beyond the scope of the Butterfly BASIC project.

Writing an interpreter in assembly language certainly means that it can run quickly, but implementing more advanced, complicated features is both time consuming and difficult. Butterfly BASIC has many drawbacks: it is limited to a small number of variables, it is limited to integers, and it can't be ported to other architectures without complete recoding.

So, having completed Butterfly BASIC in MSP430 assembly language and knowing its problems, it was time to move on to a wider audience with a complete rewrite. I wanted to code a version in pure C so that I could use it on a wider range of hardware, with much wider capabilities than the MSP430.

I developed CoreBASIC over the Christmas holidays of 2004 and into the new year of 2005; it ran something like this:

- Having consigned my main Windows laptop to the cupboard for Christmas and New Year on my wife's orders, I retreated to my cute Apple PowerBook and began to code. CoreBASIC's roots lie in that intense development period under Xcode.
- Having completed most of the work on the PowerBook, certainly enough for it to run programs from the command line, I ported it to CrossWorks for the MSP430. Initially it ran using console I/O services over JTAG.
- To prove its portability I moved CoreBASIC to the AVR, again using CrossWorks. At that time I also found the AirDrop-A and made sure that it could work on an 802.11b wireless LAN.

CoreBASIC was then released to a few select customers for review, and the comments they provided were generally positive. However, I still wasn't satisfied with the result because both AVR and MSP430 were limited. And, in 2006, Revely Microsystems introduced a small computer that looked ripe for a CoreBASIC port.

Networking

One of the integral options in CoreBASIC, the "Net" part, is that you can use CoreBASIC in a network environment. To provide a simple network interface to CoreBASIC I originally chose Adam Dunkels' **uIP**. I cobbled together some network device drivers which opened up the possibility for users to telnet into network-enabled boards and start to write programs on the target interactively.

In time, however, uIP just didn't cut it on the microcontrollers I used. So, I integrated CoreBASIC with the CrossWorks Network Stack and it remains that way to this day.

Documentation

Because I'd like CoreBASIC to proliferate, it's important that the source code is documented well. I started off using Dimitri van Heesch's excellent **Doxygen** tool (I also use Doxygen internally to document **CrossStudio**, our

neat piece of development software). You can browse around CoreBASIC's documentation and, if you're going to extend it, you should at least try to document what you do.

However, Doxygen had some problems, and I didn't like some of the formatting. I didn't feel like hacking Doxygen to do what I wanted, so I now use a custom tool, DocGen, that I wrote and bolted into CrossStudio. In fact, all our API documentation is now written using DocGen, including the manual. However, DocGen is targeted at writing API documentation rather than internal interface and function documentation, so you've been warned.

Ports

I have ported CoreBASIC to a number of boards. The source distribution of CoreBASIC does not contain code for all these boards because they are either bound to proprietary hardware, are experimental, or are simply incomplete. To aid portability, I created the CrossWorks Platform API and the CoreBASIC code and many other examples are written to that API.

Who did all this?

All SolderCore products are designed and manufactured entirely within the United Kingdom. The project divides neatly into hardware and software:

- **Paul Curtis** at Rowley Associates in Dursley conceived, designed, wrote, documented, and eventually debugged CoreBASIC. And the firmware for the Arcade Shield and LCD Shield.
- **Iain Derrington** at K&I Design in Torquay designed, prototyped, reworked, tweaked, and finally built SolderCore. And all the accessories.

CoreBASIC runs on more than the SolderCore. Helping out with native ports:

- **Jonathan Elliott** at Rowley Associates reimplemented the file I/O functions and networking for the Raspberry Pi and built the Raspberry Pi CPU driver.

Thanks

The following deserve special mention:

- **Tony Kireluk** for constructing the excellent test jigs using his CNC expertise, for lending space to set out the SolderCore Production and Test area, and for being positive in the face of adversity.
- **Deborah Curtis** for keeping the home fires burning when her poor husband was chained to his keyboard at the office, late into the night, getting CoreBASIC to product status.
- **Kerry Derrington** (formerly **Kerry Keast**), who designed the SolderCore box, logo, and supporting graphics. During the SolderCore project, Kerry managed to get married to Iain—that's how long SolderCore has been in gestation.
- **Maureen Pugh** for keeping Paul in baguettes and dispatching SolderCore prototypes during development.
- **Ben Curtis**, **Bethany Curtis**, and **Thomas Derrington** for being part of the *SolderCore Production Team* at weekends when they could have been chilling with friends.

Production

The SolderCore products are manufactured by Camtronics Vale in the UK, an ISO 9001:2008 approved company:

http://www.camtronicsvale.com/

Geography

Some information from Wikipedia about Dursley and Torquay:

- **http://en.wikipedia.org/wiki/Dursley**
- **http://en.wikipedia.org/wiki/Torquay**

For those that have trouble with understanding the difference between England, the United Kingdom, and Great Britain, you might like to take a look at this fast-paced YouTube video on the subject:

- **http://www.youtube.com/watch?v=rNu8XDBSn10**

Hope that clears it up for you.