

# The User's Guide

Mel Croucher

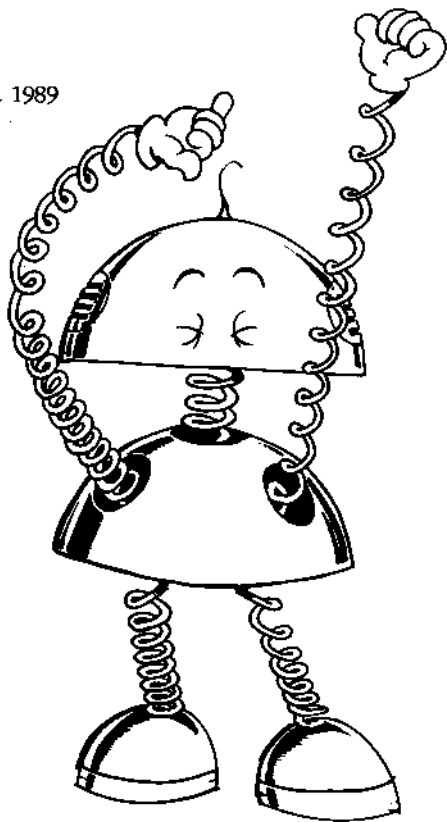
**SAN** *coupe*



# SAM Coupé Users' Manual

by Mel Croucher

© MILES GORDON TECHNOLOGY plc. 1989



This manual is something like the one I wanted when I started programming. I could have read through the Glossary and begun to understand what all the jargon obscured. I could have let SAM lead me by the hand for a bit of instant enjoyment. But in those days, computers were mysterious: as big as a bus and as daft as a brush. If you think I'm going to waste too much of the Manual's space on simple ideas and not devote enough pages to advanced concepts, you're probably right. Read through it because I think that the SAM Coupé is a wonderful machine. If only it had been around when I was you.

Mel Croucher, October 1989

This is Mel's manual, and as soon as you start reading, you'll see what a great job he's done. All credit to him and his team for making the unreadable readable - and especially for introducing us to SAM: a special word of praise to Robin Evans, the SAM cartoonist.

But there are other heroes too, and without them we'd never have managed to turn our Coupé dream into reality.

Ever since the Coupé was just a gleam in our eyes, Bob Brenchley and John Wase have encouraged and guided us. Simon Goodwin discovered MGT and SAM in Cambridge, told the world, and then contributed in so many ways. Andy Wright had faith in us in the early days too, and has gone on to write the best Basic we've ever seen.

Later, Bruce Everiss played a vital role in making it all happen: marshalling industry support and raising public awareness. Bo Jangeborg gave up a summer in Sweden to write 'Flash!' Fouad Katan, Keith Turner and David Whittaker not only got involved in developing material but also turned out to support us at public events. Dave Hood, who designed our original logo, now proves his graphic skills with the demonstration software.

Thanks too to Simon White, Mike Burns and Paul Bate for their invaluable support. Nor should we forget the MGT team: we can't mention one of you without mentioning all - let's just say that collectively, you're the best in the business. And finally, thanks to our loyal customers. We've often said that "our best ideas come from our customers". Now we hope you'll be able to see just how hard we've been listening to you!

Alan Miles, Bruce Gordon and Rob Collins

THIS SAM COUPE MANUAL IS WRITTEN AND DESIGNED BY MEL  
CROUCHER, LASER-TYPESET BY KATE CAMERON-DAUM AND  
ILLUSTRATED BY ROBIN EVANS FOR YOUR ENJOYMENT.  
FIRST PUBLISHED 1989, BY MILES GORDON TECHNOLOGY plc,  
LAKESIDE, PHOENIX WAY, SWANSEA ENTERPRISE PARK,  
SWANSEA SA7 9EH, U.K.  
© MILES GORDON TECHNOLOGY plc 1989. All rights reserved.

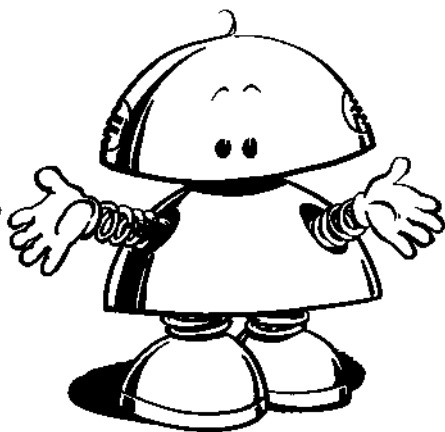
NO PART OF THIS MANUAL MAY BE REPRODUCED OR ADAPTED BY ANY  
MEANS, UNLESS FOR THE PURPOSES OF REVIEW OR FOR THE USE OF THE  
OWNER OF THIS MANUAL IN ENTERING PROGRAMS INTO THEIR SAM  
COUPE. EVERYTHING IN THIS MANUAL IS OFFERED TO YOU IN GOOD  
FAITH, AND MILES GORDON plc WILL NOT BE LIABLE IN ANY EVENT, FOR  
ANY LOSS, HOWEVER ARISING. WE RESERVE THE RIGHT TO ALTER THE  
SPECIFICATIONS OF THE SAM COUPE AND TO UPDATE THIS MANUAL AS  
NECESSARY.

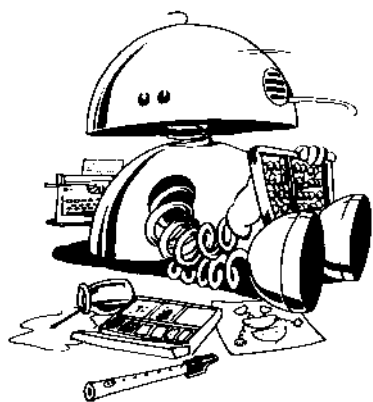
*'What's the use of computers?  
They will never play chess draw  
art or make music.'*

(Jean Genet)

*Play it SAM, play it!*

(Humphrey Bogart)





# CONTENTS

		Page
Chapter 1	<b>SETTING UP</b>	1
	testing video & audio	5
	First Aid checklist	6
Chapter 2	<b>WHAT'S WHAT</b>	7
	sockets & ports	9
	the keyboard	12
Chapter 3	computer care	15
	<b>SOFTWARE</b>	17
	ROM and RAM	18
	cassette recorders	20
	RUN and SAVE	22
	VERIFY and LOAD	23
	INPUT	24
	MERGE	25
Chapter 4	<b>SAM BASIC</b>	27
	◆ <b>Beginner's BASIC</b>	
	keywords and commands	28
	EDITing, DELETE and REM	30
	punctuation	31
	RUN, STOP, PAUSE	32
	INPUT, GO TO, CONTINUE	33
	PRINT statements	34
	IF and THEN	36
	◆ <b>Everyday BASIC</b>	
	strings, variables, arrays	37
	ELSE, LOOPS and STEPs	40
	subroutines	43
	calculations	44
	functions	45
	beautified BASIC	49
	◆ <b>Advanced SAM BASIC</b>	
	random numbers	51
	DATA	52
	DO, WHILE, UNTIL, IF	55
	AND, OR, NOT	56
defining functions	57	
handling strings	58	
procedures	59	
references and labels	61	
system variables	61	

# CONTENTS

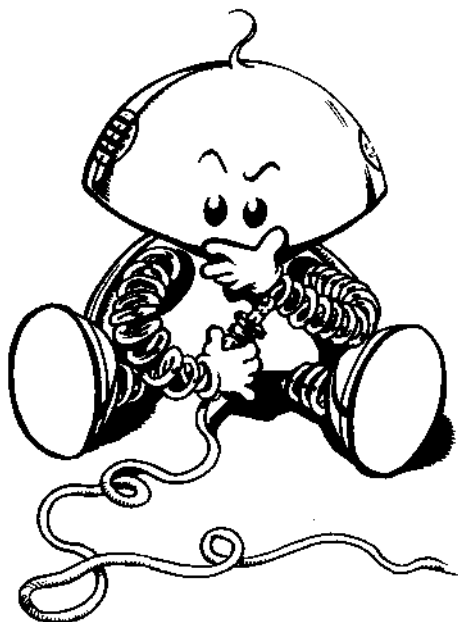
		<b>Page</b>
<b>Chapter 5</b>	<b>COLOUR &amp; LIGHT</b>	<b>63</b>
	the PALETTE	64
	the colour chart	66
	PEN and PAPER	67
	the four screen Modes	68
	attributes	70
<b>Chapter 6</b>	<b>TEXT &amp; GRAPHICS</b>	<b>73</b>
	graphic commands	74
	graphics scaling	77
	SCROLL, ROLL, WINDOW, GRAB	79
	animation	82
	the character set	86
	block graphics and UDGs	88
<b>Chapter 7</b>	<b>MAKING MUSIC</b>	<b>93</b>
	BEEP	94
	shortcuts	96
	MIDI	97
<b>Chapter 8</b>	<b>SOUND EFFECTS</b>	<b>99</b>
	ZAP, POW, ZOOM and BOOM	100
	SOUND	100
<b>Chapter 9</b>	<b>EXPANDING THE SYSTEM</b>	<b>103</b>
	add-ons	104
	printers	105
<b>Chapter 10</b>	<b>MEMORY</b>	<b>109</b>
	extra memory	110
	machine code	112
	disk drives	115
<b>Chapter 11</b>	<b>COMMUNICATIONS</b>	<b>117</b>
	education	118
	streams and channels	119
<b>Chapter 12</b>	<b>ERROR CODES</b>	<b>121</b>
	<b>GLOSSARY of SAM computing</b>	<b>131</b>
	<b>SAM Coupé Specifications and Appendix</b>	<b>169</b>
	<b>INDEX</b>	<b>181</b>



# Chapter 1

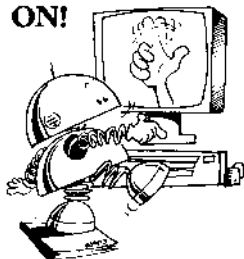
## SETTING UP

- unpacking the components
- setting up
- switching on
- testing - testing
- First Aid checklist



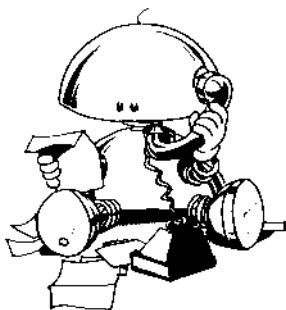
*'To begin at the  
beginning'*  
(Dylan Thomas)

## DON'T SWITCH ON!



## Your SAM Coupé

## TALK TO US



This User Manual is intended to help you enjoy all the benefits of the SAM Coupé micro-computer. Please read this chapter **BEFORE** you plug it in and turn it on. The manual is designed to assist and entertain those of you who are new to computing, as well as inform experienced users who will also benefit from the specialist information.

The Sam Coupé is an advanced machine, offering a large memory, brilliant graphics and superb sound. It's your vehicle for creative entertainment, education and above all, fun! But your new computer is as important to Miles Gordon Technology as it is to you, which is why we want to hear from everyone who enjoys using it, and keep you up to date with the latest news and advances concerning SAM.

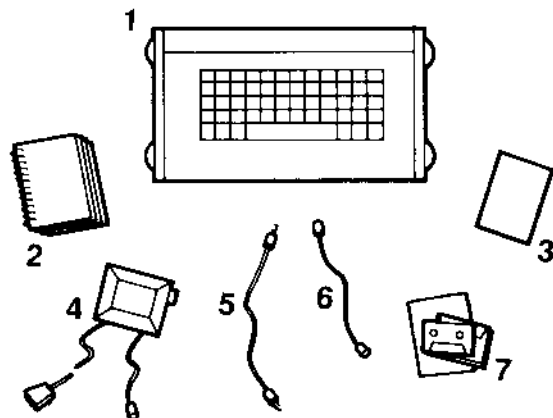
Please fill in your Registration/Guarantee form, and we'll do the rest. If you need any extra information or specialist details, we have prepared a Technical Manual for advanced users. Write to us at the address below or telephone the SAM Coupé Customer Care Line with any enquiries, ideas or achievements: 9 am to 7 pm Monday to Friday. MGT want to know what the most important component in our business thinks, and that means you.

MILES GORDON TECHNOLOGY  
LAKESIDE, PHOENIX WAY  
SWANSEA ENTERPRISE PARK  
SWANSEA SA7 9EH, UK

SAM CUSTOMER CARE LINE: tel. 0792 791100.

### What you get is what you see

Now that you have opened the SAM Coupé package, please check that everything is present and correct, and identify the following items:



- (1) your SAM Coupé computer.
- (2) this User Manual.
- (3) the Guarantee/Registration card.
- (4) the power supply unit, with a phono socket for connecting SAM to a TV set, a cable with a 6-pin DIN plug that delivers power to SAM, and another cable waiting for the fused plug (supplied) that will take power from the mains.
- (5) a connecting cable with 3.5mm jack plugs at either end, to transmit and receive computer data via a cassette recorder.
- (6) the connecting cable between the TV socket on the power supply unit and your own television set.
- (7) our data cassette and booklet of demonstration computer programs, including the spectacular graphic art package, 'FLASH!'.

**CONNECTING  
THE POWER  
CABLES**

*"It ain't what  
you do, it's the  
way that you  
do it."*

(Bananarama)

**CONNECTING  
TO AN  
ORDINARY  
COLOUR TV  
SET**

**Turn on  
Tune in**

The fact that you might have to fit the power supply unit (4) with the supplied mains power plug is not because MGT are lazy, but because cabling requirements and mains socket outlets vary in different regions and countries. For UK users, you'll probably have wall socket outlets that take 3-pin 240v fused mains plugs, so connect the power plug in the normal way, with the BROWN cable to the live terminal, and BLUE to neutral. The plug should have a 3 amp fuse.

Don't plug into the mains yet, and make sure the ON/OFF button at the back of the computer is in the OFF position. As you look at the back, it's the third item from the right on the control panel, and it sticks OUT for OFF, and stays IN for ON. Now identify the cable from the power supply unit with the 6-pin DIN plug on the end, and connect it to the power socket at the back of your SAM Coupé. As you look at the back panel there's only one socket that can take a 6-pin DIN, and it's the one on the extreme right hand side.

Next, identify the TV cable (6) and plug the correct phono plug into the aerial socket of your television set, with the other end plugged into the TV phono socket on the power supply unit. Now you can plug into the mains, switch on the TV set and push SAM's power switch ON, which is the 'IN' position.

The computer sends out signals that can be picked up by Channel 36 of a domestic TV set. If you are using specialist monitors, or stereo SCART-type sockets, connections are explained in Chapter 2 and Chapter 9. This Chapter applies to ordinary UHF televisions only. Fine-tune your set around Channel 36 until you see a test pattern with the following message appearing on the screen:

**'Welcome'  
screen****MILES GORDON TECHNOLOGY plc**

© 1989 SAM Coupé 256K

This is the 'welcome' screen that appears every time the computer is switched on or restarted from scratch, and the number at the end reports how much 'memory' your SAM Coupé has on board. If it has been fitted with extra memory, the end number will read 512K.

**Testing  
testing**

You can now test out SAM for colour and sound, to make sure that everything is in order. In this Manual, the symbol □ is used every time we want you to type something on the computer keyboard, so without any explanations, try typing this:

□ **BORDER 1**

You can type in capital letters or lower case letters, it doesn't make any difference. Now locate the large [RETURN] key near the centre-right of the keyboard, and press that. The **BORDER** around the TV screen picture should have changed to blue, with a message Ø OK, Ø:1 in the bottom left corner. Adjust the tuning if it's still a little bit out, then go for a red border, by typing:

□ **BORDER 2**

followed by a press of the [RETURN] key.

Make sure that the TV volume control is turned up a little, and type in the following instruction:

□ **BEEP 1,1**

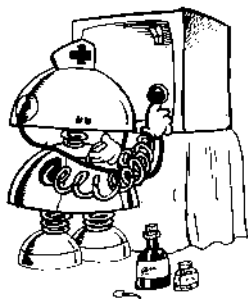
followed by [RETURN]. A short **BEEP** of sound should have squirted out of the TV speaker.

Congratulations! If the **BORDER** has changed colour and the speaker is **BEEP**ing, you've already started programming your new computer. Now read through the next couple of Chapters and find out what's going on.

*"I hate television  
as much as  
peanuts. But I  
can't stop eating  
peanuts!"*

(Orson Welles)

## FIRST AID CHECKLIST for domestic TV sets



*"Why pay money  
to see bad films?  
Stay home and  
see bad televi-  
sion for nothing"*  
(Sam Goldwyn)

There is probably a simple explanation for any problems experienced in setting up your computer. Identify what's wrong then go through this First Aid Checklist in order.

### PROBLEM

the TV screen is blank

the TV screen shows random white dots, or is broadcasting quiz shows!

the colour TV picture is washed out or black and white only

the TV picture is good, but there is no sound

there are strange geometric patterns or random characters on the screen, or the screen turns black.

### FIRST AID

Check computer ON-OFF switch is in ON position.  
Check TV is switched on.  
Check power supply plugs for loose or bad connections.  
Check fuse in power plug and any fused socket outlet.

Check computer is switched on.  
Check TV is tuned to Channel 36.  
Check cable between computer and TV is properly connected.

Check TV channel for fine tuning.  
Check TV colour adjustment.  
Check for loose cable connection between computer and TV.

Check volume control on TV.  
Check TV channel for fine tuning.

Press any key.  
Press the **BREAK** button, which is at the far left as you look at the back of the computer.  
Press the **RESET** button, which is on the left of the long connector socket as you look at the back of the computer.

If all else fails, 'phone the SAM Coupé Customer Care line.

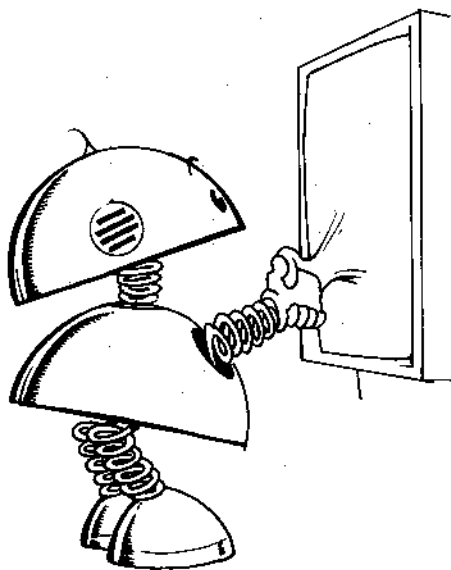
# Chapter 2

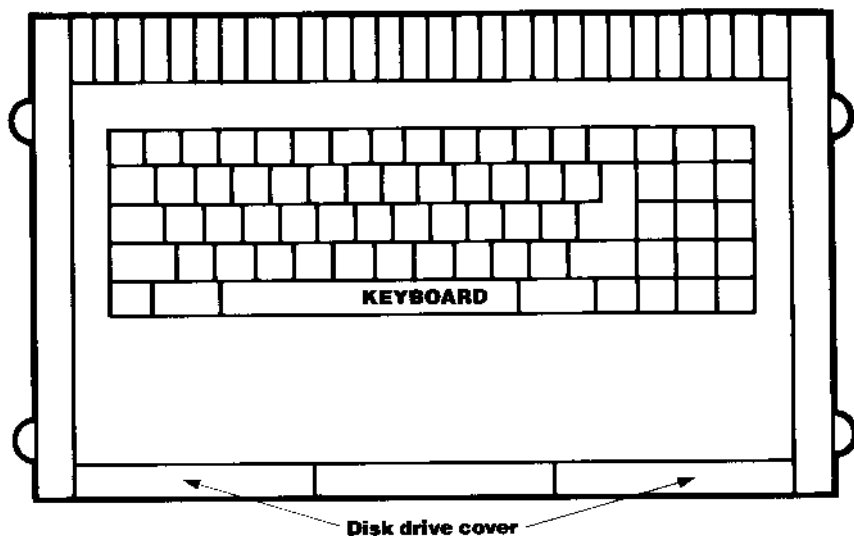
## WHAT'S WHAT

- what the sockets, ports and buttons do
- a trip round the keyboard
- caring for your computer and its software

*'This morning  
I abused my  
electric toaster.  
Tonight the  
elevator held  
me hostage in  
revenge.'*

(Woody Allen)

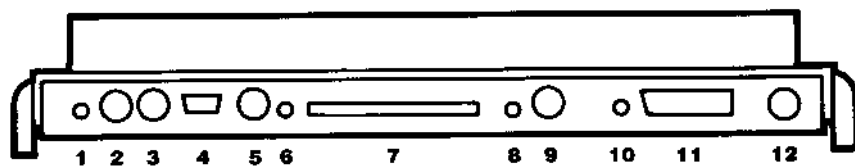




**TOP VIEW**



**FRONT VIEW**



**BACK VIEW**



## SOCKETS, PORTS AND BUTTONS

If you imagine your computer to be like an intelligent Mother-Ship, then all sorts of service modules and communications devices can dock with it, making the machine even more useful.

## THE FRONT OF THE MACHINE

The SAM Coupé has a large built-in memory and can call up information stored on magnetic tapes and disks. Memory and disks are dealt with in Chapter 10. MGT produce special slim-line DISK DRIVES that fit right inside your computer via the twin docking slots at the front of the machine. The slots are exposed by unclipping their protective covers, with access for DISK DRIVE 1 on the left and DISK DRIVE 2 on the right.

## THE BACK OF THE MACHINE

To connect your computer to the outside world, a bank of special sockets, ports and buttons is provided at the back of the machine. They are numbered on the diagram as follows:

### (1) BREAK BUTTON

When pressed, this button will halt whatever is going on in a computer program. Programmers can choose to use it as an **ESCAPE**, **BREAK** or **CRASH** button.

### (2) MIDI OUT

Musical Instrument Digital Interface Output, for sending musical information. The standard 7-pin DIN connector can also be used to talk to other computers in a **NETWORK**.

### (3) MIDI IN

Musical Instrument Digital Interface Input, or MIDI IN for short, can receive pulses of information from drum machines, synthesisers and other instruments as explained in Chapter 7. This interface socket



takes a standard 7-pin DIN connector, not supplied, and can also be used to listen to other computers as part of a **NETWORK**.

#### (4) JOYSTICK PORT

Socket for a standard 9-pin 'Atari'-type joystick plug. The joystick is used to control actions in game-playing. A second joystick can be connected for two-player options by using a special MGT dual joystick adaptor.

#### (5) MOUSE INTERFACE

Socket for an 8-pin DIN 'mouse' connector. The mouse is used to move a pointer around the screen and select on-screen options, and is available from MGT.

#### (6) RESET BUTTON

When pressed, this button resets the machine. Please see Glossary for details.

#### (7) EXPANSION CONNECTOR

This interface is a standard 64-pin Euroconnector. Expanding your computer system is explained in Chapter 9.

#### (8) CASSETTE JACK

A standard 3.5mm mono jack socket for connection to a cassette recorder. When taking information **IN** it should be connected to the **EAR** socket of the recorder. When giving information **OUT** it should be connected to the **MIC** socket. If your tape recorder allows it, **EAR** and **MIC** may be connected together.

**(9) LIGHT PEN & STEREO SOUND**

This standard 5-pin DIN connector acts as an input for a light pen or light gun and also functions as a stereo sound output when connected to the **AUX** input of a sound system.

**(10) ON-OFF BUTTON**

Even though power is supplied to the computer all the time its power supply is plugged in and switched on at the mains, this button interrupts the power when it is pushed to the **OUT** position for **OFF**. Power is allowed through to the machine when the button is pushed **IN** for **ON**.

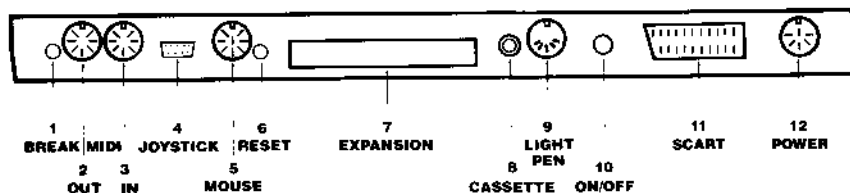
**(11) SCART SOCKET**

A special 21-pin socket capable of handling all video and stereo audio outputs. See Chapter 9, 'Expanding the System'.

**(12) POWER INPUT**

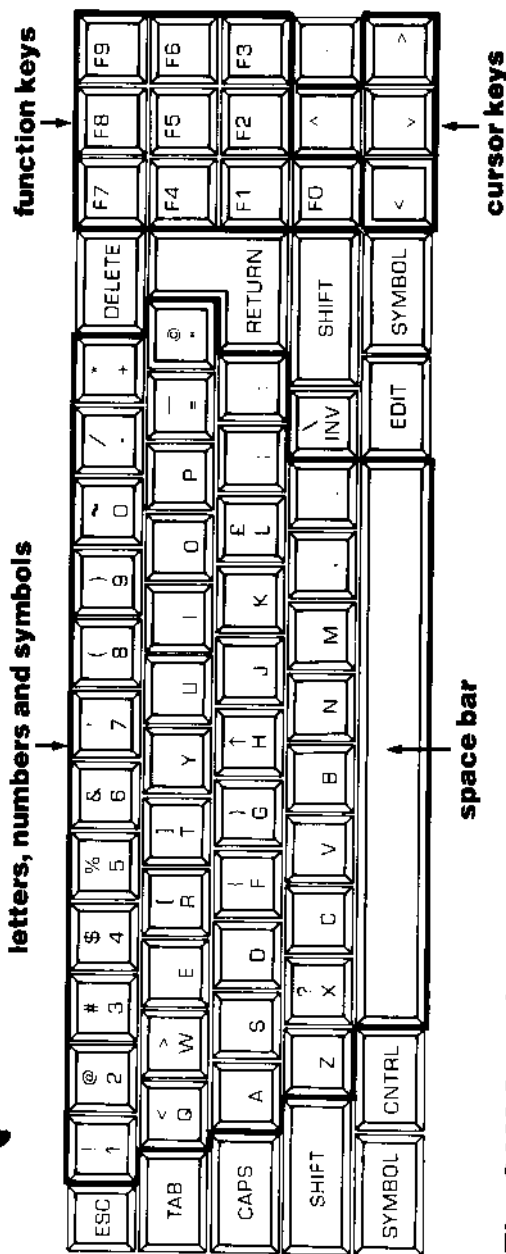
A dual-purpose 6-pin DIN connector which accepts 15 volts DC from the Power Supply Unit plug. It also carries TV video signals and mono sound back to the socket located in the power supply unit. See Chapter 1, 'Setting Up'.

Before you read any further, make sure that you know exactly what each of the back panel sockets and buttons is for.



*"Life is like a tin of sardines.  
We're all looking  
for the key."*

(Alan Bennett)



**The SAM Coupé Keyboard**

**THE TOP OF  
THE MACHINE**

Rest your hands over the top of the computer. It slopes like the top of a racing car, and that's one reason we christened it the Coupé. MGT have designed it to be practical as well as attractive, and if you have ever used an old-fashioned computer you will soon discover how much more comfortable your sloping keyboard feels. It's deliberately set back from the front edge, with a special wrist support panel for easy typing.

The whole range of SAM Coupé keys is used to communicate with the machine, but they are much better than those of an ordinary typewriter keyboard, because you can instruct most keys to do all sorts of things.

**a trip round  
the keyboard**

Let's suppose you are a beginner, and need to know what the keys do. The next section of this Chapter is a guided tour around the keyboard, so refer to the illustration on the opposite page which shows the SAM Coupé keyboard as it appears on your computer, with certain groups of keys outlined with thick border lines.

Throughout this Manual we refer to individual keys that must be pressed to achieve results by putting them inside square brackets, such as [A]. When two keys must be pressed at the same time, we write that as [SHIFT] [A], for example.

**characters  
and [SPACE]s**

The illustration shows a large group of keys inside a thick black line, and these are all the keys you would expect to find on a typewriter. The top row includes numbers and symbols, the middle three rows represent all the letters of the alphabet and some more symbols, and the bottom row features a long [SPACE] bar for typing in blank spaces.

**[SHIFT]**

There is a long **[SHIFT]** key at either side of the lower row of letter keys, and as you would expect, when this is pressed at the same time as a letter key it **SHIFTs** from lower case to upper case and makes a capital letter appear. So that pressing **[A]** on its own **PRINTs** the character 'a' on your TV screen, whereas pressing **[SHIFT] [A]** together displays the character 'A'. **[SHIFT]** also changes what appears when you use it with all the number and most of the symbol keys.

**[CAPS]**

You can lock everything into capitals by pressing the **[CAPS]** key, on the left hand side of the keyboard. This state is unlocked simply by pressing **[CAPS]** again, to go back to normal upper and lower case.

**[DELETE]**

Because you can't use an eraser or correcting fluid on a TV screen without making a filthy mess, we have provided a **[DELETE]** key near the right hand side of the top row. This will deal with any mistakes that get typed in accidentally. Press **[DELETE]** once, and the last character you typed gets rubbed out. If you hold down the **[DELETE]** key, characters continue to be erased very quickly, from right to left on your TV screen. **[SHIFT] [DELETE]** erases to the right.

The remaining keys have more specialised functions, but don't worry what they do yet, just locate them on the keyboard.

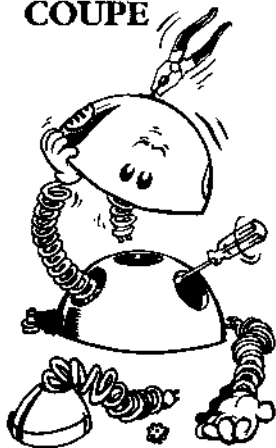
**[RETURN]** is the large corner-shaped key near the centre right on the keyboard. **[ESC]** is an **ESCape** key, at the extreme top left. Both of these keys are explained in the next Chapter.

The whole group of keys on the far right of the keyboard is explained in Chapter 4. It includes ten

special **function keys** numbered [F0] to [F9], a decimal point key and four keys at the bottom with directional arrow heads on them, known as **cursor keys**. You will also learn about the [EDIT] key on the bottom line.

Typing in text and graphics is covered in Chapter 6, where the remaining keys are explained: [TAB] (near the top left hand corner), [INV] (which is above the [EDIT] key), and finally, the two [SYMBOL] keys and [CNTRL] key, in the bottom row.

### CARING FOR YOUR SAM COUPÉ



### CARING FOR DISKS AND CASSETTES

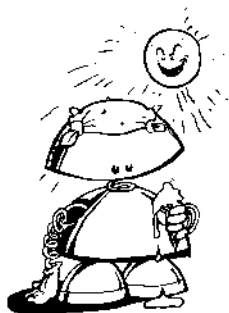
You can't hurt your SAM Coupé by bad programming or pressing the wrong key, but don't try to stick silly things in its sockets or ports, or mess about with its insides. You wouldn't like that and neither will your computer. The SAM Coupé prefers similar working conditions to you, so keep it clear of freezing cold and sweltering heat as well as harmful sunbathing, drizzle and thunderstorms.

Let fresh air circulate beneath and around all sides when in use, and your computer won't rise above blood temperature. The small amount of heat generated while SAM is switched on is quite normal.

Using the wrong power supply is dangerous, and beating up your machine or pouring drinks into it are not good ideas. Use a soft cloth to clean the case, but please avoid abrasive pads, powders and solvents like alcohol or benzine.

Programs are stored on disks and cassettes as magnetic recordings, so they may get damaged or made totally useless if they get exposed to magnets or radiation. Keep all disks and cassettes away from loudspeakers, and don't leave them on top of television sets or under telephones. If you are on the

## PROTECT AND SURVIVE



*'Out! Out!*

*Damn Spot!*

(Lady Macbeth)

## CARING FOR YOUR TV SET

move and carrying software, watch out for under-floor magnets in trains and baggage x-rays at airports.

Keep computer data out of direct sunlight and away from heaters, dust and high humidity. The back window of a car is a disaster zone!

Don't stand drinks next to software, they don't mix, and don't poke anything inside disk or cassette cases.

If you want to keep an important piece of software, then knock out the lug at the back of your cassette or push in the special tab on a 3.5 inch disk after you've **SAVED** the data, to prevent accidental erasure. Always make back-up copies of important software as explained in the next chapter.

Everyone should know by now that television screens can be tricky, but you can stay healthy and happy if you follow some simple advice. Because you may find it hard to drag yourself away from your SAM Coupé, sit as far away from the screen as you can and take a short break every couple of hours. Give the screen a wipe with a soft cloth to remove electrostatic dust every time you use it, and your skin will thank you for not touching the screen and fingering your face!

Television screens can burn out if the same image is displayed for too long, so a special safety feature will black out your screen if you don't press a key for about 22 minutes. Simply press any key to get your last image back. This is **NOT** a fault, it is a protection routine for your screen.



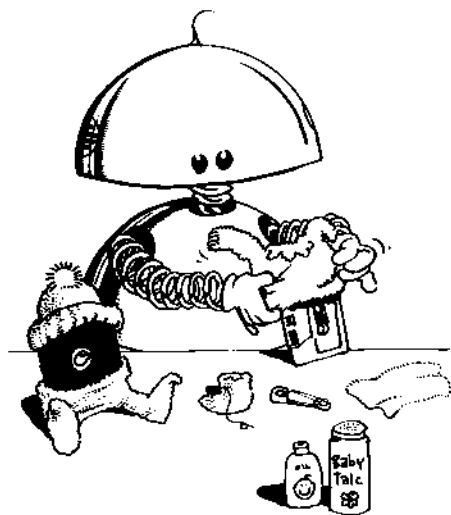
# Chapter 3

## SOFTWARE

- what software does and how it gets done
- writing a program
- **SAVE**ing
- **VERIFY**ing
- **LOAD**ing
- **MERGE**ing

*'When I was a kid I couldn't even spell PROGRAMMER, but now I are one'*

(graffiti, Sussex University)



**ROM**

The computer comes equipped with two types of brain. In the first type, information that has been taught in the SAM Coupé factory lies buried in special memory cells. You can read what's in these cells but you can't change the contents, so they are known as **READ ONLY MEMORY**, or **ROM** for short. The computer remembers what is in them even when switched off.

**RAM**

*"Brain; noun:  
an apparatus  
with which we  
think we think."*

(Ambrose Bierce)

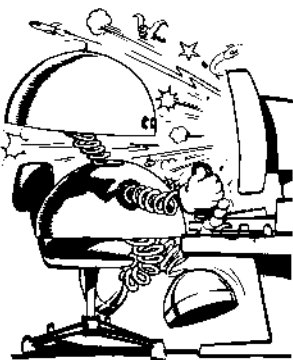
The second type of brain contains empty memory cells that are waiting to learn something new. You can plant all sorts of fresh ideas here and gain access to them as you choose, which is why they are called the **RANDOM ACCESS MEMORY**, or **RAM** for short. This type of memory gets erased every time the machine is switched off. So we need a method of teaching new ideas to the brain quickly and accurately, by **LOADING** computer programs already stored up somewhere else.

When you were setting up, you instructed your machine to change the **BORDER** colours of the screen and to **BEEP** a sound so that you could test the system. It only took a few seconds, so it wouldn't matter much if you had to do it again next time you powered up the computer. But what if you had spent all day slaving over a complicated piece of programming, only to have it forgotten when you pulled the plug?

Fortunately you can **SAVE** programs onto cassettes or disks for use later.

**Commercially  
available  
software**

If you are only interested in using your SAM Coupé to play computer games written by other people, you're in luck! There are thousands to choose from. Computer games, useful educational programs and special utilities that have been produced by private



individuals and companies can be purchased for your own use, and these packages come in the form of pre-recorded cassettes and disks. They are known as **software**. Apart from software written specially for the SAM Coupé, you are able to use a whole range of games and programs that have already been written for another less sophisticated computer called the Spectrum. The SAM Coupé will automatically examine this software using a special program that you will find on your 'FLASH' demonstration cassette and if all is well, convert it into a form that will **RUN** for your enjoyment. To be honest, dedicated games players can stop reading this Manual now, but you'll be missing out on creativity and home-grown fun.

All you have to do is follow the **LOADing** instructions that come with the games. If in doubt, follow the **LOADing** instructions in the next section of this chapter.

### Home-grown software

If you want to use the computer for creating and keeping your own ideas, you will need to use a few simple techniques for **SAVEing** them somewhere safe and **LOADing** them back into your machine next time you need them.

#### **SAVE**

is the command used to tell the computer to get ready to record your lists of instructions onto tape or disk.

#### **VERIFY**

is used to ask the computer to check that whatever has just been **SAVED** is all present and correct, and that whatever is now stored onto tape or disk is exactly the same as what the computer still has in its memory.

## STORING INFORMA- TION

### Setting up a cassette recorder

### Your first program

#### **LOAD**

tells the computer to empty its memory and get ready to receive new information which has already been stored on cassette tape or disk.

#### **MERGE**

commands the computer to **LOAD** in new information, but to keep any existing information in its memory, provided that it does not conflict with what's being **MERGED**.

If you are lucky enough to have an MGT Coupé disk drive, follow the instructions and fit it in **Drive 1** via the special slot in the front of your computer. All the instructions you need are in the Disk Drive Manual, and in Chapter 10 of this Manual. The following information is for cassette users.

Connect up a cassette tape recorder as detailed in Chapter 2, ready to send information **OUT** from the 3.5mm jack socket to the cassette's **MIC** socket. It is best to set any **TONE** controls to neutral, and you will have to use trial and error to discover the best **VOLUME** level for **RECORDing** and **PLAYing** back information. If your tape recorder has a 'VU-meter' that displays the level of recorded sound with a dial, a bar of light or a digital meter in decibels, the signal should not go much higher than **+1dB**. Make sure that the cassette tape is wound on past any non-magnetic leader tape, and that the knock-out lug is in place at the back of the cassette. Please use a new blank cassette, and **ON NO ACCOUNT** use the 'FLASH' demonstration cassette for recording over!

Now you are ready to test the system by writing a simple 'program'. A program is the word used for a collection of commands that you use to tell the SAM

### Line Numbers

*'Begin at the beginning', the King said, gravely to Alice, 'and go on till you come to the end; then stop.'*

(Lewis Carroll)

### [RETURN]

Coupe what to do. Don't forget how to use the [SHIFT] key, and where the [DELETE] key is, in case you make any mistakes.

Computers aren't very imaginative unless you encourage them, and they will obey all your commands one after the other, starting at the beginning right through to the end.

This works just fine until you decide to make changes in your programming and need to find the relevant part of your list of commands. To keep things in a neat and simple order, computer programs are broken up into lines, just like the lines in a printed book of instructions. Each line is given its own reference number at the beginning, so that the computer knows exactly which part of its program you are talking about, and we usually number lines in 'tens' to make it easier to add extra lines between the original line numbers.

Because you want the computer to obey each line of program, you will want to send your typed instructions to be stored in its memory, and this is where the corner-shaped [RETURN] key comes into play. [RETURN] has several uses, and the first one is to achieve just this.

Remember, the [key] symbol □ is used to mean 'type in whatever follows this', so have a go at typing the following program exactly as it appears below, press [RETURN] at the end of each line, and don't worry how it all works for the time being. You type a question mark by pressing [SYMB] [X] together.

```

□ 10 BORDER 3 [RETURN]
   20 PRINT "I HAVE AMNESIA" [RETURN]
   30 BEEP 3,0 [RETURN]
   40 BORDER 1 [RETURN]
   50 PRINT "What is my name?"

```

## RUNning a program

### [RETURN]

You will see your lines of program appearing at the bottom of the screen as you type them in, and as soon as they are committed to the computer's memory by pressing [RETURN], they are listed at the top of the screen as part of the computer program, with a little arrow character ▶ pointing out which line of program you have been concerned with. This indicator is called a **cursor**, and more will be explained about its uses later on.

Let's make the computer work by **RUNning** your program, and type in:

#### RUN [RETURN]

This makes the SAM Coupé **BEEP** a note, change the colour of the screen **BORDER**, and **PRINT** up your test successfully, ending with a little **OK** message at the bottom of the screen, to show at which line the program stopped. The program can **RUN** as many times as you like while the computer is switched on, by typing in **RUN [RETURN]** again, and your listing of lines can be examined by pressing [RETURN] on its own.

## SAVE

You can go away and make a cup of tea in a minute, but before you switch off the computer you'll want to **SAVE** your 'amnesia' program so that you can **LOAD** it in after the kettle has boiled. For **SAVEing** to cassette, just type in:

#### SAVE "amnesia" [RETURN]

and a message appears on screen, prompting you to start off the recording process on your tape recorder with the cable connected to its **MIC** socket for taking information **IN**, and then press a key on the keyboard. Any key will do. The screen's centre usually goes blank while waiting for the **SAVEing** process to begin, and the border will change colour.

Narrow coloured bars indicate that your program is being transferred to tape. After a short while the reassuring **OK** message appears, and you can stop the cassette recording.

To make doubly sure that what's on tape is the same as what's in the computer's memory, wind back your cassette and reconnect the cassette cable to the **EAR** socket, for sending signals **OUT** to the SAM Coupé. Now type in the checking command for your program:

□ **VERIFY "amnesia" [RETURN]**

and playback your **SAVED** program. If all goes well, more border patterns will appear as well as a recognition message at the top of the screen, **program:amnesia**, followed by the **OK** confirmation.

When the program is safely stored, and you've found the confidence to switch off your computer to wipe out its memory, and the kettle has boiled, you can switch back on and **LOAD** the **SAVED** information. Check that the cassette is wound back and ready to send information **OUT** via its **EAR** socket and give the command:

□ **LOAD "amnesia" [RETURN]**

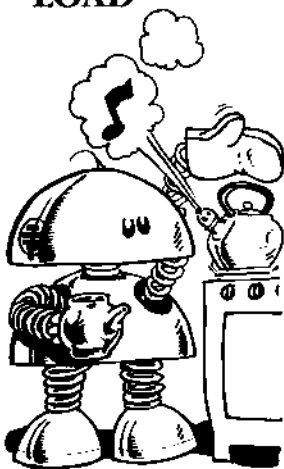
As you may have guessed, what you see on screen is the same as when you used **VERIFY**, with your program safely stored on tape, ready for use whenever you need to use it or change it.

If it doesn't work first time off cassette, check that the cables are connected correctly, and then wind back and adjust the volume until SAM is happy.

The SAM Coupé will let you know if it's having any problems in **SAVEing** and **LOADing** your information, by displaying the message "**Loading error**"

## VERIFY

## LOAD



## Error Messages

## Program names

on screen. And if you try to **SAVE** information to a device which doesn't exist, like a disk drive or another computer, the polite message "**Invalid device**" will appear.

If you don't specify a program name and just type:

**LOAD ""**

the computer will **LOAD** the first program that it finds. The reason for identifying individual programs with their own name is to make it easy for the computer to locate a particular set of instructions from a whole series of programs that may be stored on the same tape. You can now write another program and **SAVE** it onto cassette, leaving a gap of blank tape after your "**amnesia**" program. First type in:

```
 50 PRINT "please tell me your name"
      [RETURN]
60 INPUT n$ [RETURN]
70 BORDER 6 [RETURN]
80 BEEP 3,12 [RETURN]
90 PRINT "I remember now! Hello "
      ; n$ [RETURN]
      RUN
```

## INPUT

When you type in **RUN** this time, you will be able to talk to your SAM Coupé for the first time! This is because you have used a very special instruction in the program. **INPUT** tells the machine to wait until you type something in before responding. So when the screen asks you to tell it your name, it waits for your **INPUT** before recognising you. Type in your name and see how SAM recognises it. You don't have to use your real name, you can **INPUT** whoever or whatever you feel like.



You can call this or any other program anything you want, using a "name" up to 10 characters long. Identify the program you have just written by using your own name and **SAVE** it as before with:

□ **SAVE "my name" [RETURN]**

Now wind back the cassette to the beginning and try **LOADing "my name"**. The SAM Coupé will recognise the first program on the cassette but will ignore it, because it was called "**amnesia**", and continue searching until it finds the correct program title.

## MERGE

Switch off the computer, finish your tea, switch it back on and rewind the cassette. **LOAD "amnesia"** again and press **[RETURN]** to display the program. Now you are going to try something smart, by joining your two programs together and making them **MERGE**. Type in the instruction **MERGE "my name" [RETURN]**, and send the program into the SAM Coupé's memory in exactly the same way as a normal **LOADing** procedure. Now examine your listing.

You will notice that line **50** of "**amnesia**" has been replaced by line **50** of "**my name**", and your original program is now nine lines long. **RUN** it, and enjoy your first home-grown computer game. It may be primitive, but it's not a bad start.

## summary

What you have just done is to take your first steps in a computer 'language' called **BASIC**, and the next Chapter is devoted to explaining how **BASIC** works.

Before you plough into it, here's a summary of the four important instructions you have been using.

**SAVE "name"** christens a program and commands the computer to send a copy of that program from its memory to be recorded.

**LOAD ""** loads the first program recognised by the computer, wiping out the old memory.

**LOAD "name"** loads the program specified by its name, ignoring any other program that may be on the tape.

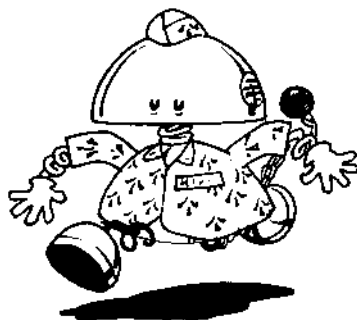
**VERIFY "name"** checks that the information just **SAVED** is exactly the same as the information in the computer's memory with the same name.

**MERGE "name"** adds new information from the named program to information already in the computer's memory, overwriting any existing program lines that conflict with the new program.

**CUP OF TEA** is a whole lot more hit-and-miss than any other procedure in this chapter.

[ESC]

Finally, if you ever get into difficulties while a program is **RUNNING**, you can [ESC]ape from trouble by holding down the key at the top left corner of the keyboard. This will stop the machine **RUNNING**, and then **BREAK** into its memory. After [ESC]aping, simply press the [RETURN] key to display the program listing.



# Chapter 4

## SAM Basic

### ◆ Beginner's BASIC

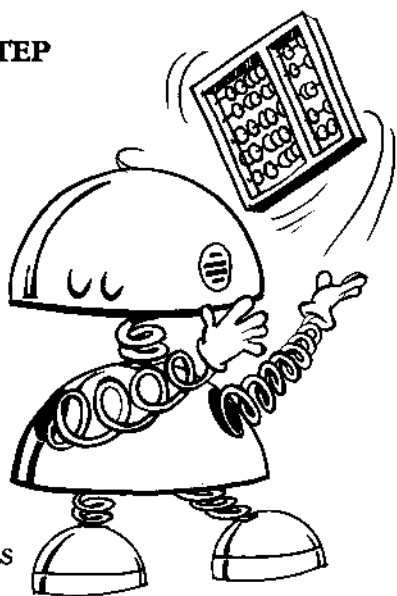
- keywords
- **REM**inders, starting and stopping
- **PRINT** statements
- making decisions, **IF** and **THEN**

### ◆ Everyday BASIC

- strings, variables and arrays
- **ELSE**, **LOOP**ing, **FOR-NEXT**, **STEP**
- subroutines
- calculations
- functions
- beautified BASIC

### ◆ Advanced SAM BASIC

- random numbers
- **DATA**, **READ**, **RESTORE**, **DIM**
- **DO**, **WHILE**, **UNTIL**
- **AND**, **OR**, **NOT**
- **DEF FN**
- string conversions
- **PROC**edures, **REF**s and **LABEL**s
- system variables



This is the Chapter that outlines how the SAM Coupé's own programming language works. The enhanced BASIC has been specially designed by Dr. Andy Wright, and anyone used to primitive BASIC is in for a pleasant surprise. There isn't enough room in this manual to explore all of its possibilities, but the Glossary section does give every command and function available to you, and advanced users can dip in and out of this Chapter. Games players might want to stop reading here, because this Chapter is for anyone who wants to experiment and create programs. It's in three sections: Beginner's BASIC, Everyday BASIC and Advanced BASIC, so if you're a beginner, here goes!

*"Let all things be done decently and in order."*

(Corinthians 14:40)

## Keywords

### BEGINNER'S BASIC

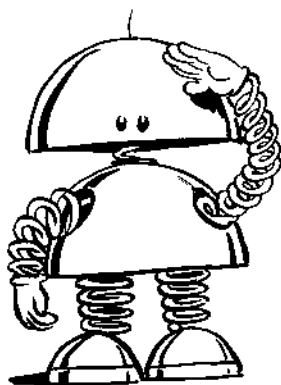
The SAM Coupé will obey your instructions (called **commands**) exactly, so you have to type them in very precisely.

The commands that you give your computer make use of some special words, and throughout this Manual they are printed in bold capital letters. We call them 'keywords' and you normally type them in full. There is a special way of programming single keys to generate complete keywords, which is dealt with later in this Chapter. Your SAM Coupé automatically recognises every keyword that it comes across and converts it to capital letters in your program listing, but you must be careful not to tack extra letters on if you don't mean them to be there. For instance, if you type in

□ 10 print x [RETURN]

this will become **PRINT x** in your listing. But if you forget to leave a space and type

### Giving commands



### Spotting mistakes

#### □ 10 printx [RETURN]

the computer will assume that you have given it the name of a special procedure. This only happens with letters, so you can get away with typing things like

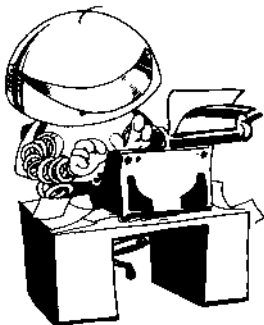
#### □ 20 print1 [RETURN]

which automatically becomes **PRINT 1** in the program.

Commands can be given directly to your computer for instant action: all you have to do is type one in and then press the [RETURN] key. They can also be used within program lines to work with other instructions later on. Whenever [RETURN] is used after you have written a line of program, that line becomes part of the 'listing'. From now on we'll assume you know that, and stop printing [RETURN] at the end of every line in our examples.

As lines of program are being written, they appear as a 'listing' in the top section of your screen if all is well, but any mistake in 'syntax' (the grammar of SAM BASIC) gets pointed out by a question mark, asking you to correct it before it can become part of the program. The bottom two lines of the screen are reserved for special reports that help you spot mistakes, and your computer will display all sorts of messages to pin down what the problem is and where it's hiding. These messages are called 'error codes' and they are all explained in Chapter 12. You may well need this sort of help, because each line of your program can be as long or short as you like. It can contain up to **127** individual 'statements', and each statement can consist of a command like **PRINT**, followed by something else. You can have up to **65279** lines in any one program, so the error messages can save hours of searching for mistakes.

## the [EDIT] key and Cursors



## DELETE

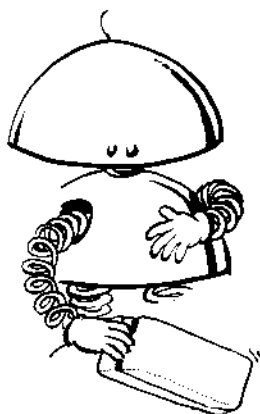
When writing your own computer programs, the most useful key on your keyboard is the [EDIT] key. If you press it, you jump into the part of your program that the computer is thinking about, marked by a special symbol on your screen that looks like an arrow head >. This marker symbol is known as a cursor, and it appears in front of the line number to be **EDIT**ed. If you already know which line number you want to **EDIT**, then all you have to do is type in that number and press the [EDIT] key, and watch it appear. You can then move around your **EDIT** line, up and down, right or left, by holding down any of the four cursor keys located at the bottom right corner of your keyboard. They are clearly marked with arrow-heads to show which direction they control.

You will notice that the SAM Coupé uses two other cursors on screen. An inverse \* points out where you are **EDIT**ing when the keyboard is in its normal state, but when [CAPS] lock has been engaged, upper case is indicated by an inverse +.

To change your program, rub out your old instructions before typing in new ones by using the [DELETE] key near the top right of the keyboard. Remember [DELETE] on its own rubs out characters to the left one at a time, and if you press [DELETE] and [SHIFT] together, characters on the right get erased.

### Delete n TO m

Apart from having its own key, **DELETE** is also a keyword that can be typed in to erase whole blocks of program lines. If the first line number to be **DELETE**d **n** is left out, everything from line 1 is deleted up to line **m**. If **m** is omitted, the last line of the program is assumed. A single line can be



## REMinders

## Program punctuation

## Colons

## Commas

**DELETED** by making **n** and **m** the same number. Neither **n** nor **m** actually has to exist, and any lines contained in their slice of the program will be deleted.

An alternative way of deleting a line you don't like is to type in its number followed by nothing at all, and press **[RETURN]**. **LOAD** up "amnesia" now, and experiment with the **[EDIT]**, **[DELETE]**, **[CAPS]** and cursor keys.

You can include short memos in SAM BASIC which will help to **REM**ind you what a particular section of your program is for. These statements or special comments are stuck in to jog human memories, and not for the computer to worry about, and they are called **REM** statements. It's good practice to leave yourself a note or a heading before each new section of your program, so just pop in a **REM** after a line number like this:

```
□ 10 REM amnesia
```

and you can call it up or refer to it later on.

When you want to use a number of different commands in a single line, they must be separated to allow the computer to clearly understand them. This is done by using colons, like this:

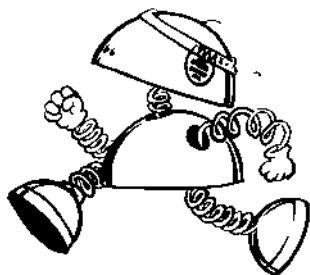
```
□ 10 PAPER 6: PEN 2: PRINT "red on yellow"
```

If you want to include several similar items in a line, like numbers to be examined or calculations to be made, they must be separated using commas. E.g.

```
□ 10 READ x,y,z
  20 PRINT x,y,z
  30 DATA 1,2,3,4,5
```

## Starting and Stopping

### RUNning



### STOPping

### PAUSEing

You are going to write a computer game in a while, but instead of just copying some lines from this Manual, there are a few BASIC concepts you need to know first.

There are several ways to manage the progress of your programs as you create them, but the first step is to get them going.

#### **RUN**

followed by **[RETURN]** of course, is all that needs to be typed to command the program to **RUN** from the start.

#### **RUN n**

instructs the program to get going from the particular line number that you choose.

#### **STOP**

is a command that does exactly what it says. It **STOPs** the program at any predefined point.

#### **PAUSE**

allows you to make part of a program take an exact amount of time. If you type in **PAUSE** by itself, or **PAUSE 0**, then your program will **PAUSE** for ever and a day until a key is pressed. For shorter timings, just type in **PAUSE** followed by a number. This will stop the program and freeze whatever is on the screen for as many 'frames' as you want, or until a key is pressed. Your screen displays 50 frames every second, so we can use numbers counted in frames to give a very accurate freeze-frame timing. For example:

□ **100 PAUSE 10**

will freeze an image for one-fifth of a second.



### Waiting for INPUT

#### INPUT

tells the computer to halt everything until something is typed in. We will be using **STOP**, **INPUT** and the next command **GO TO**, to write a guessing game at the end of this 'Beginner's BASIC' section.

### Jumping with GO TO

#### GO TO

is recognised as one keyword in BASIC, even though it is two words in English and may be typed 'go to' or 'goto'. It is used to command the computer to immediately **GO TO** a particular line of program, and can also be used to give the same instruction when a particular condition has been met. Try this example:

```

10 REM greetings
20 PRINT "happy birthday to you"
30 PAUSE 50
40 GO TO 20

```

When you run this, greetings are printed all the way down the screen every second. But before the program fills up the screen, press the [ESC]ape key to halt the process.

### [ESC]aping

### CONTINUEing

#### CONTINUE

Once a program has ground to a halt because of an error or a deliberate [ESC]ape break-in, it can be kick-started by typing in the keyword **CONTINUE**. Try using it now, and allow the greetings to **CONTINUE** until the screen is full, and asks you if you want to **scroll?**

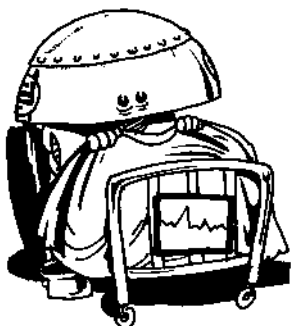
#### scroll?

appears when the computer discovers that the screen is full of characters, and wonders if you want to see more by scrolling up the text. It's not a command, which is why it is displayed in lower-case letters. Simply press [RETURN] to scroll up the screen.



## Clear Screen

### a NEW start



## PRINT statements

## PRINT statement punctuation

## PRINT commas

## CLS

stands for 'Clear Screen', and instantly erases all text and graphics that are currently displayed on your television screen.

## NEW

is a very powerful command which is the equivalent of giving the SAM Coupé a brain transplant, so think carefully before using it. **NEW** wipes out the entire computer program with all of its variables and the only thing left is a blank screen. It has the same effect as pressing the **[RESET]** button on the back panel of the computer, causing all your current work to be flushed to oblivion!

## PRINT

**PRINT** statements are used to display characters on screen. There are some simple rules of punctuation to learn, and the first of these is the use of quotation marks (" "). If there is anything in the SAM Coupé's memory now, get rid of it by typing **NEW**, followed by **[RETURN]**. Now carefully type in the following **PRINT** statement:

```
□ 10 PRINT "G'day: how are you, SAM?"
```

Now press **[RETURN]** and **RUN** it. You will see that what is contained inside the quotes appears exactly as you typed it, including the apostrophe, colon, comma and spaces.

However, it is very important to understand that punctuation used **OUTSIDE** of quotes in a **PRINT** statement can act as special instructions.

Commas (,) instruct the computer to start **PRINTING** at the left-hand margin, or to start **PRINTING** at the next **TAB** position, depending on which of these choices comes next. Try **RUNNING** this example:

**PRINT semi-colons****PRINT apostrophes****□ REM columns****10 PRINT 1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Semi-colons (;) tell the machine to **PRINT** whatever follows immediately after whatever precedes it.

Apostrophes (') cause whatever follows to be **PRINT**ed at the left-hand margin of the next line. Try out various combinations of these punctuation instructions to see how they operate. For example:

**□ 10 PRINT 1, 2, 3; 4, 5' 6' 7, 8; 9, 10**

Because **PRINT** can be used to show the results of calculations, as explained in the Everyday BASIC part of this Chapter, the numbers in the last two examples dutifully appeared on screen. But now try replacing them with letters or words, and watch the computer refuse to **PRINT**. However, as soon as you put any characters inside quotation marks, you can **RUN** the program, like this:

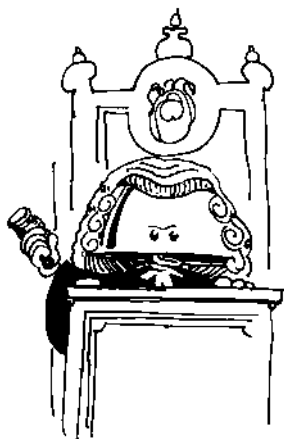
**□ 10 PRINT "wham", "bam" '  
"thank you"; "SAM"**

More information about **PRINT**ing on screen can be found in Chapter 6, and commands for **PRINT**ing out on paper are dealt with in Chapter 9.

So far, this Chapter has offered a lot of theory but not much fun. By taking on board a few more complex ideas, you can write your first computer game.

**MAKING DECISIONS**

If the computer just obeyed lists of instructions line after line, programs would be very limited and boring. The magic begins when your machine thinks for itself and starts making decisions!

**IF and THEN**

*Whoever can predict winning numbers has no need to let off crackers."*

(Kai Lung)

The easiest way to get the computer to make a decision is to show it a condition and offer it a choice between 'true' and 'false;', so that **IF** something is true **THEN** the computer will take one course of action. If it is false **THEN** the machine does something else. The SAM Coupé understands the following symbols which you can use as shorthand with **IF - THEN** statements:

- = means 'is equal to'
- <> means 'is not equal to'
- > means 'is greater than'
- < means 'is less than'
- >= means 'is greater than or equal to'
- <= means 'is less than or equal to'

Use **NEW** to wipe out any old programs in the computer's memory, then type in this guessing game and **RUN** it. The < and > characters are typed by pressing [SYMB] [Q] and [SYMB] [W]. Get a friend to think up and type in secret numbers, which the computer will help you to discover. See how the **GO TO**, **PAUSE**, **IF** and **THEN** work, how everything between the quotations marks "" gets **PRINTed** exactly on screen, and how the **STOP** brings the game to an end when the number is guessed correctly.

□ **10 REM secret numbers**

20 PRINT "Think of a number between 1 and 100 and type it in secretly"

30 INPUT a

40 INPUT "Find the secret number", b

50 IF b<a THEN PRINT "WRONG, go higher"

60 IF b>a THEN PRINT "WRONG... try lower"

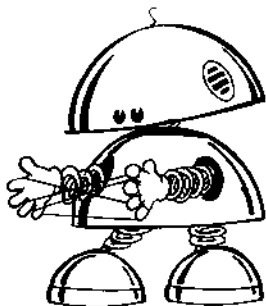
70 IF b=a THEN PRINT "RIGHT. Well done!":

STOP

80 GO TO 40

**EVERYDAY BASIC**

The following section contains the back-bone of this introduction to BASIC programming. In it you will find all of the every-day ideas and practices needed for writing programs, so make sure you understand and try out each concept before tackling the next one, and don't try to rush through them. First, you need to learn three definitions known as 'strings', 'variables' and 'arrays'.

**Strings**

A string is simply a number of characters all strung together. These characters can be letters, numbers, spaces or graphics, and a single string can be as short as you wish, up to **65520** characters long. (You might need that many to display a whole screen of graphic shapes that make up a complex picture.) In a program, each string is defined by giving it its own name and immediately sticking on a special string symbol after that name, and then **LET**ting it be equal to whatever follows in quotation marks. We use the 'dollar' symbol \$ to represent 'string', for example:

□ **10 LET sam\$ - "string along with me"**

**Substrings**

If you look at **sam\$**, you will see that it consists of 20 consecutive characters made up from letters and spaces. When a number of consecutive characters are taken from a string one after the other in the same sequence, this is called a '**substring**'. So whereas "**long wit**" is a substring of **sam\$**, "**bring a long whip**" is not! When you are sure you understand this idea, try the next concept.

**Strings\$(n, a\$)**

You can repeat a string for as many times as you like, by putting it inside a pair of brackets with the number of repeats you require. The special command **STRING\$** is used to achieve this. **RUN** this little joke:

**Using  
VARIABLES**

□ **10 PRINT STRING\$(8, "ha")**

Now you are ready for a more serious idea.

In a calculation, when numbers are represented by something else they are called **variables**. If you think of a variable as something that represents a value, then you will begin to understand one of the most powerful features of the computer. We have already come across variables in the simple form of:

□ **10 LET x=2**

**20 PRINT "THE ANSWER IS "; x+x**

which, when the program is **RUN**, results in 'THE ANSWER IS 4' appearing on the screen. Now try changing the value of **x**, and **RUN** it again.

**Arrays**

Very often, you will want to use a whole set of similar variables, for a weekly bank balance or a list of football results. An array is a group of such variables that are distinguished from one another by a number written in brackets after the name of the array. This number represents a **DIMENSION** or a space in the computer's memory that is reserved for the array to occupy. The name of an array can be up to ten characters long, excluding any spaces, and the number in brackets can be anything between 1 and 65535.

**DIM**

The statement for setting up the **DIMENSIONS** of an array is **DIM**, and is used like this:

□ **DIM a (100)**

We'll come across it again at a more advanced level.

**LET**

So far we have glossed over one of the most important processes in the computer's brain. The keyword **LET** is at the heart of all BASIC programming. It is used to define all variables, or to give them some sort of value, as in:

### numeric variables

#### □ 10 LET x=1

But your SAM Coupé allows you to make as many **LETs** as you want at the same time. You can use long string and array names and set up all sorts of variables in a single line, simply by dividing them with commas. For example:

#### □ 10 LET x=1, flag=2, alph\$="yummy"

A variable that represents a number is called, not surprisingly, a **numeric variable**. You can choose any name you want for it, providing you follow these simple rules: the name must begin with a letter, any letters or numbers or spaces can follow the first letter up to a total of 32 characters. Spaces don't count as characters. So the following numeric variable names are acceptable:

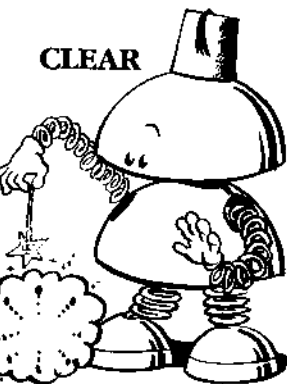
**x**, **SAM ENCHANTED EVENING** or **R2D2**. But **2R2D** is not acceptable and neither is the use of a keyword like **PRINT**. **FOR-NEXT** variables are treated like ordinary numeric variables.

### string and array variables

Variables can also be used to represent strings and arrays. The rules are the same, except the variable name is limited to a total of **ten** characters long, and it must be followed by the character **\$**. For example, **A\$, R2D2\$** and **GUITAR \$**

Variables can be ordered to vanish, by using a command telling them to **CLEAR** off. When **CLEAR** is typed in, it destroys every variable in your program, and frees up the space in the computer's memory which they occupied. If you have set up any variables, use **CLEAR** now, and get ready for the next important concept. If necessary, read through the last section of 'Beginner's BASIC' concerning **IF** and **THEN**, because things are about to get more complex!

**CLEAR**



**ELSE**

The SAM Coupé also understands the idea of **IF** qualified by **ELSE**, which must come at the start of a new statement in a line. Normally when the statement following an **IF** is false, the program jumps to the next line. But if the **IF** - **THEN** pair has an associated **ELSE** later in the line, the program continues with the statement following the **ELSE**. On the other hand, if the condition is true, the line will only be executed up to the **ELSE**. For example:

```
□ 10 REM liar
    20 LET x=1
    30 IF x=1 THEN PRINT "TRUE": ELSE
      PRINT "FALSE"
```

When this is **RUN**, truth is rewarded, but now change line 20 by giving **x** another value, and **RUN** into trouble.

You can think of that example as a 'short **IF**'. But a 'long **IF**' can work over a number of lines by omitting **THEN**. Try changing the value of **x** after **RUN**ning this:

```
□ 10 REM moody
    20 LET x=1
    30 IF x=1
    40 PRINT "HAHAHA"
    50 PRINT "snort"
    60 ELSE PRINT "BOOHOO"
    70 PRINT "SNIFF"
    80 END IF
```

**END IF**

So if **x=1**, "HAHAHA" and "snort" results, but if **x <> 1** we would get "BOOHOO" and "SNIFF". Of course the lines can contain more than one statement, but the **IF** routine must end with **END IF**.



**ELSE IF**

The condition **ELSE IF** has a special meaning. Try typing this in:

```

□ 5 REM vague numbers
  10 PRINT "type in a number between
    1 and 9"
  20 INPUT x: CLS
  30 IF x=1
  40 PRINT "one"
  50 ELSE IF x< 5
  60 PRINT "less than 5"
  70 PRINT "honestly"
  80 ELSE IF x=8
  90 PRINT "eight"

 100 ELSE
 110 PRINT "something else"
 120 END IF
 130 STOP

```

*"Disorder is  
always in a  
hurry"*

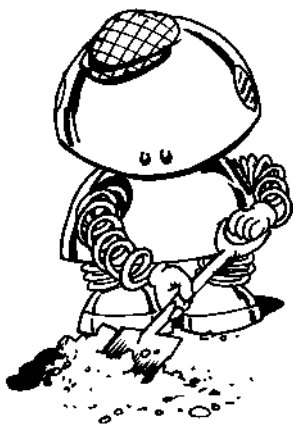
(Napoleon  
Bonaparte)

If  $x=1$ , line 40 is executed, followed by line 130.

If  $x=2,3$  or  $4$ , lines 60 and 70 are executed, and then line 130.

If  $x=8$ , line 90 is executed, then line 130.

If  $x$ = any other value, line 110 is executed, then line 130. So once the computer knows that any condition is true, the lines and statements after it are executed, and when another **ELSE IF** or **ELSE** is encountered the program jumps straight to a position immediately after **END IF**. Line numbers and the numbers of statements in any line are irrelevant. You are allowed to 'nest' long and short **IF**s inside a long **IF**, but you can't 'nest' a long **IF** between a short **IF** and its associated **ELSE**.

**FOR-NEXT**

If you feel like taking a **BREAK** at this point in the proceedings, **THEN** now is the time, **ELSE** get ready to make your computer work for a living! **END IF**.

Imagine you have to prepare a list of items for every day of the year. You could start by typing in a line for day 1, and keep typing until you reach 365 (unless it's a leap year!) This would be so tedious that there are four BASIC commands dedicated to making this sort of task as easy as possible, **FOR**, **NEXT**, **TO** and **STEP**, and they are used to make the computer perform the same job over and over again.

**FOR** and **NEXT** are always used together to cut down repetitive programming. **RUN** this example:

```

 10 REM calendar
      20 FOR day=1 TO 365: PRINT day:
      NEXT day
  
```

**TO**

As you scroll up the screen by pressing [RETURN], you can see that the use of **TO** in your **FOR-NEXT** loop has instantly set the limitations of the values in your **PRINT** statement. Naturally, the numbers go up one at a time, but by introducing the idea of handling values in **STEPS**, this can be varied.

**STEP**

For example, your SAM Coupé will instantly change its calendar from days to weeks if you tell it to work in **STEPS** of 7:

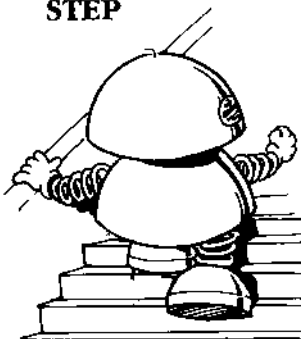
```

 20 FOR day=1 TO 365 STEP 7:
      PRINT day: NEXT day
  
```

Easy isn't it. **STEP**ping is not limited to whole numbers or positive values. Try out this sort of example:

```

 FOR n=1 TO 100 STEP 3.142
      FOR x=500 TO 250 STEP -25
  
```



**FOR-NEXT** variables are treated just like ordinary numeric variables, so there are no special rules to learn. However, if you want to do something more ambitious than writing lists, there are several commands that are ready and able to help you. Luckily they all behave in BASIC very much as you would expect in English. **DO**, **LOOP**, **WHILE**, **UNTIL** and **EXIT IF** are all explained in the last section of this Chapter, 'Advanced SAM BASIC'.

### subroutines

A collection of statements arranged in lines that all act together is known as a routine. In computer programs it is very common to want similar tasks to be performed in different places, but it is also very tiresome to type in the same lines over and over again. So these lines are isolated in what are known as 'subroutines', and they can be called up from anywhere in the program to do their job as many times as necessary. If you think of how the command **GO TO** works, then you will easily understand how to **GO** to a **SUB**routine, for example:

□ **GO SUB 500**

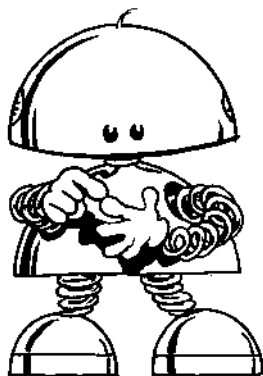
instructs the computer to leave its present place in a program and jump to line 500 in order to carry out whatever task awaits it there.

### RETURN

It may be helpful to mark the start of each subroutine with a **REM** statement, for your own convenience. This is optional, of course, but there is no choice in how you mark the end of a subroutine. The command **RETURN** is always used, like this (non working) example:

□ **500 REM pretend subroutine**  
**510 ...all the lines that form this**  
**subroutine**  
**600 RETURN**

## CALCULATIONS



+ - \* / ↑

### expressions

If you look at the multiple sound effects program that is featured on the demonstration cassette as well as in Chapter 8, you will see how useful subroutines are.

If all this theory seems concerned with complicated programs, BASIC allows some very common everyday operations. You can use the SAM Coupé as a simple calculator to perform arithmetic tasks. After typing **PRINT** followed by your calculation, all you have to do is hit the **RETURN** key. Nothing can be simpler than this:

□ **PRINT 2+2**

You don't even have to **RUN** it!

The computer will obey all of these commands:

- + the plus sign always signals addition
- in calculations, the conventional minus sign is used
- \* but for multiplication an asterisk is recognised.
- / and division is made using this symbol

↑ this is the exponentiation symbol. It means 'raise a given number to a given power' which is exactly the same as multiplying a number by itself, so that:

- **PRINT 3↑5** is the same as typing in:
- **PRINT 3\*3\*3\*3\*3**

A combination of calculations is called an expression, and you should be aware of the fact that your computer handles expressions in a strict order of priority. Any multiplication and division are calculated first, in order of appearance from left to right. Only after they have been dealt with will additions and subtractions be attended to, again in order from left to right. You can get round this rigid system by

MATHEMATI-  
CAL  
FUNCTIONS

using brackets ( ), and then anything inside the brackets is evaluated first and treated as a single number. For example:

□ **PRINT 10\*10-(25\*4)**

These days, an electronic calculator is expected to do a whole lot more than simple arithmetic, and your SAM Coupé has already been taught to handle mathematical and trigonometrical functions, using these abbreviations. We can't teach you maths or trigonometry here, so if you are not interested in these subjects you may as well skip the following section:

**EXP**

is an exponential growth function, defined by  $EXPx=e^{\uparrow}x$

**SQR**

calculates the **S**quare **R**oot of a positive number, **n**. That is to say, it finds out what number has to be multiplied by itself to give **n**.

**LN**

is the inverse of an **EXP**ponential function, and is the **L**ogarithmic **N**umber function. 'Common' logarithms use the numeric base 10, but **LN** calculates 'natural' logarithms which use the base **e**. To find logarithms to any other base, just divide the 'natural' logarithm by the natural logarithm of the base.

**INT**

Arguments that consist of a load of numbers either side of a decimal point can often give very messy results in BASIC programming. It would be much more convenient to use whole numbers, which are known as **INT**egers. This function always rounds down to the relevant whole number, which can be either a positive or a negative number.

*"I don't believe  
in mathematics"*  
(Albert Einstein)

## TRIGONOMET- RICAL FUNCTIONS

### SGN

is a function that shows whether a mathematical argument is positive or negative. It is short for 'signum' or 'sign', and can give three alternative results:

- 0 if the argument is zero
- +1 if it is positive, and
- 1 if the argument is negative.

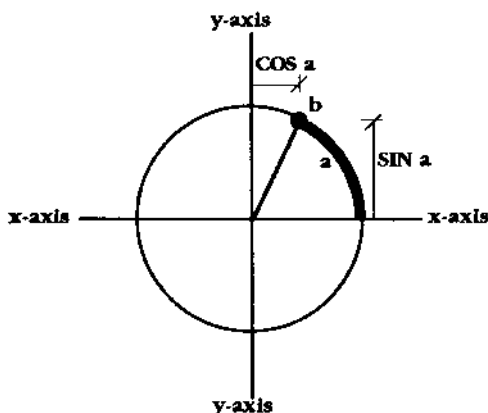
### ABS

converts an argument into a positive number, by completely ignoring any negative signs.

You would expect your computer to cope with the sort of trigonometrical functions to be found on a pocket calculator, and of course it can.

### PI

is the Greek letter  $\pi$ , and is used to summon up a useful number which begins 3.14159653 and goes on for ever. In trigonometry, **PI** is the tool for calculating aspects of circles and spheres, such as the perimeter of a circle ( $\text{PI} \times \text{diameter}$ ), and so on.





Supposing you want to know more about a circle, where a point has moved from the right-hand side of the  $x$ -axis up along the perimeter for a distance  $a$ , and stopped at the position  $b$ . We don't refer to  $a$  as the number of degrees in the angle between the  $x$ -axis and the line from the centre of the circle to point  $b$ , because the SAM Coupé uses **radians** instead of degrees.

To convert from radians to degrees you divide by  $\pi$  and then multiply by 180. And to convert degrees to radians, you divide by 180 and then multiply by  $\pi$ .

### SIN

is a function that calculates how far point  $b$  is above the  $x$ -axis, and this distance is known as its **SINe**.

### COS

calculates how far point  $b$  is to the right of the  $y$ -axis, the distance being known as its **COSine**. If  $b$  goes to the left of the  $y$ -axis, its **COSine** value becomes negative. Similarly, if it drops below the  $x$ -axis, the **SINe** results in a negative value.

### TAN

is the function that gives the **TANgent** of  $a$ , which is simply the **SINe** divided by the **COSine**.

### ASN, ACS and ATN

are short for arcsine, arcosine and arctangent, and these are the functions used to find values of  $a$  that have given a particular **SINe**, **COSine** or **TANgent**

## Functions

Those mathematical and trigonometrical key words act equally well inside computer programs, along with several others you have yet to come across. Collectively they are known as 'functions', and what they have in common is the fact that they all work

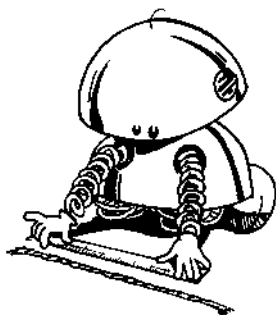
**FREE**

with numbers in order to give a result. Functions are used by typing them in and then giving them an argument to work on. They can be mixed in with other sorts of operations in the same expression, but they will always be worked out before those operations unless they are placed inside brackets.

Discover how much **FREE** memory is available for programming by using this function now:

**PRINT FREE**

and then type in a line or two of program. Check and see how much memory that has been taken up by using **FREE** again. You will discover how memory is calculated and what gobbles it up in Chapter 10.

**LEN**

Another function you can try out is the one that tells you what the **LEN**gth of a string is, like this:

**PRINT LEN "antidisestablishmentarianism"**

or if you prefer,

**PRINT LEN "SAM"**

Even though you may be in the early stages of learning to program your SAM Coupé, you have probably grown impatient by now and **LOAD**ed up some of the programs on the 'Flash' demonstration cassette, and I can't really blame you. Before we go on to more advanced SAM BASIC, choose one of the longer ready-made demonstration programs and **LOAD** it now, because you are about to take a few enjoyable liberties with it!

**LIST**

If you tell the Coupé to **LIST** the current program, it will do just that, and display the listings from beginning to end. What may be more useful is to ask for a listing from a particular line number, or for a part



of the program contained between two line numbers. For example:

- **LIST 500** will display everything from line 500 to the end of the program.
- **LIST 500 TO 1000** will give you a listing starting at line 500 up to and including line 1000.

If the line number you want to start from is left out, then the first line of the program is assumed. Similarly, if the line number you want to end with is omitted, the last line of the entire program is assumed. Try out a few **LIST** commands now.

## HOW TO BEAUTIFY YOUR BASIC

There are several tricks of the trade that can help make the layout of your BASIC programs easier to refer to and a whole lot more pleasing to the eye. If you want to highlight something for your own reference and a **REM** statement doesn't fit the bill, then use the **[INV]** key to make that part of the program stand out in **IN**verse video. The highlighting is turned off again by pressing **[SYMB][INV]** together.

## AUTOMATIC LINE NUMBERS

The first rule of BASIC is that every line of program must have its own line number, and your numbering system can get very messy after a while. The SAM Coupé is ready and willing to help out. It has a very convenient feature that **AUT**omatically numbers your lines of program. Enter **AUTO** by itself, and the computer takes the current line number where the program cursor is, then adds ten at every new line. This won't work if the current line number is less than **10** or more than **61439**. If you need to skip over a block of line numbers, just delete the **AUT**omatic number and type in a new one, followed by the rest of the line. **AUTO** will carry on from there, numbering in steps of ten.

Here are some variations:

- **AUTO** numbers after here in steps of ten
- **AUTO 150** numbers from line 150 in steps of ten
- **AUTO 150, 5** numbers from line 150 in steps of five

Of course you can choose any line number and any step you want. Direct commands, such as **STOP**, or deleting the line number offered will turn **AUTO** off.

## RENUMbering

### RENUM

This instruction will **RENUM**ber all your program lines in steps of ten, starting with the first line as **10**.

### RENUM n TO m

**RENUM**bers part of the program, starting from line **n**, up to and including line **m**. If **n** is omitted then it starts from the first line, and if **m** is left out, the last line is assumed.

### RENUM LINE L

makes number **L** the first new line number.

### RENUM STEP s

uses numbers in whatever step **s** you instruct. The slice of program, **LINE** and **STEP** can be used or left out as you wish, but they must occur in the following order:

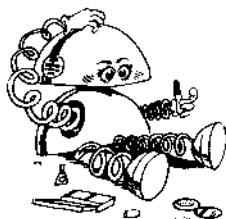
- **RENUM n TO m LINE L STEP s**

When **RENUM**bering, all your instructions like **GO TO**, **GO SUB**, **RESTORE**, **RUN**, **LINE**, **ON** etc. are dealt with, but any expressions such as **GO TO VAL "100"** will be ignored. The screen is used to hold temporary data during a **RENUM** operation, and is cleared at the end of the task.

*"Mathematics is the subject in which we never know what we're talking about."*

(Bertrand Russell)

## pretty LISTings



### **LIST FORMAT** or **LLIST FORMAT**

is a technique that creates an automatic 'pretty' **LISTing**, in which each statement of the program is automatically given its own line, and these lines are indented to help you read the program more clearly. The first six columns on the left-hand side of the screen have been reserved for line numbers and spaces, and programs can have up to 65279 lines.

#### **LIST FORMAT 1** or **2**

produces an automatically indented listing by one or two spaces every time a certain keyword is recognised, with other keywords restoring the original setting. These keywords are **IF**, **ELSE**, **END IF**, **ON**, **DEF PROC**, **END PROC**, **FOR**, **NEXT**, **DO**, **LOOP**, **EXIT IF** and **LOOP IF**. And if you wonder what some of those keywords mean, you are about to find out in the final section of this Chapter.

#### **LIST FORMAT 0**

returns the listing to its normal unindented state.

Try a 'pretty' listing of one of your programs, and see what happens!

## **ADVANCED SAM BASIC**

Imagine asking your SAM Coupé every morning, "How are you today?" and the answer was always the same, "Very well, thank you." What a predictable relationship that would be. The easiest way to introduce an element of chance or surprise into a program is to throw numbered options into an electronic pot and allow the computer to pull one out at random. After one has been selected and used, it gets thrown back into the pot. So all sorts of responses could result from your prompt, each with an equal chance of random selection.

## **RANDOM NUMBERS**

**RANDOMIZE**

is the keyword that kicks off this selection process. It uses the number of frames elapsed since the SAM Coupé was switched on to make its choice.

**RANDOMIZE n**

sets the random number 'seed' to your own choice, with **n** between 1 and 65535.

**RND**

is the function that pulls a number out of the pot from a fixed sequence of 65536 numbers, which has been thoroughly stirred up, and gives a result that is a floating point number. Floating point arithmetic keeps the digits of a number separate from the position of the decimal point. You can use the **INTE**ger function to cut off all those digits after the decimal point, but there is a much easier way to generate whole random numbers, by using

□ **RND(x)**

which returns a whole number between 0 and **x**. So you can now go back to the Secret Numbers game on page 36, and let the computer act as your playmate in thinking up the number without any help. Simply get rid of line 20, and replace line 30 with

□ **30 LET a = RND (100)**

By using the random numbers technique you can program the SAM Coupé to act as a pair of dice, a card dealer, an ever-changing galaxy or to choose from hundreds of unpredictable answers to the question, "How are you today?"

But what if you don't want a random number element, and need the computer to refer to a strict order of given information? This information, or **DATA**, can be put anywhere you want in a program, waiting to be used by the computer in much the

**DATA**

*It is a capital  
mistake to  
theorize before  
one has data"*

(Sir Arthur Conan  
Doyle)

## READ

same way as it waits for you to **INPUT** via the keyboard, and each **DATA** statement is made up of a list of expressions separated by commas. The way you call up these items of **DATA** is to **READ** them.

Try out this routine:

```

□ 10 READ w, x, y, z
   20 PRINT w, x, y, z
   30 DATA 1, 22, 333, 4444
   40 STOP

```

What you have done is to tell the machine to **READ** four items of **DATA** and allocate them with the variables **w**, **x**, **y**, and **z**. When the computer **READS** the list of **DATA** for the first time, it takes the first expression. Next time round it takes the second expression, and so on until all the **DATA** has been **READ**. Now change the order of your **PRINT** statement, and see what happens. Finally, remove the last item of **DATA** from line **30**, and watch the computer search in vain.

## RESTORE

You can make further use of the expressions in a list of **DATA** by instructing the computer to **RESTORE** or reset the variables. Repair lines **20** and **30** of that last routine, and then add the following lines:

```

□ 5 READ p, q
   6 PRINT p, q
   7 RESTORE 5

```

Run it once and note the effect. Now remove line **7** and see what effect the missing **RESTORE** has. Variables are also automatically **RESTORED** whenever a BASIC program is **LOAD**ed, or **RUN**.

**READ LINE**

String variables can also be **READ**, e.g.

□ **READ a\$, b\$, c\$**

**DATA "one" "fine" "day"**

**READ LINE** allows quotes in **DATA** statements to be omitted. So if

□ **DATA sam, "dog", 123**

was part of your program, **READ LINE a\$** would first make **a\$="sam"**, then **"dog"** and finally **"123"** with successive **READs**. Any words without quotes that are used in **DATA** statements must be legal numeric values. So **sam1** is fine, but **1x** is not.

**TRUNC\$ a\$**

Another useful function for taking **DATA** from string arrays is the ability to **TRUNC**ate them, by deleting any trailing spaces. So whereas

□ **DIM a\$(10,10): LET a\$(1)="sam":  
PRINT a\$(1); "my"**

will **PRINT "sam my"**, the use of **PRINT TRUNC\$ a\$(1)** results in **"sammy"**

**DIMensions**

It should be made clear that names up to 10 characters long are allowed when **DIM**ensioning arrays, not including any spaces. The **DIM**ensions that appear as numbers inside brackets can be between **1** and **65535**, and the total size of the array is limited only by the available memory. This example would use up 200K:

□ **DIM name\$(10000,20)  
DIM coords(100,2)**

**LENGTH**

Array **DIM**ensions can be returned by using **LENGTH**, for one or two dimensional arrays, and variable addresses. So for example, after **DIM**ensioning **a\$** as (4,6)

□ **LENGTH(2,a\$)** would give 6

The address of the first byte of a number can be found by using:

□ **LENGTH (0,n)**

which is useful for telling **CALL** the location of a variable, so that it can be modified. Brackets are required after the name of a numeric array, such as:

□ **LENGTH (1,n ( ))**

Apart from commands like **RESTORE** and **FOR-NEXT**, there are other ways of making certain sections of computer programs even more useful.

## DO -LOOP

A loop refers to any part of a computer program that is repeated. The command **LOOP** is always marked by the instruction **DO**, in much the same way as a **FOR-NEXT** routine, so that:

□ **10 DO**

**20 PRINT "PLAY IT AGAIN SAM";**

**30 LOOP**

will go round in circles for ever. So it is useful to qualify **DO** and **LOOP** commands if you don't want to repeat something for a specific number of times.

## WHILE and UNTIL

**DO WHILE (condition)** results in the part of the program between **DO** and **LOOP** being executed **WHILE** that particular condition remains true. Alternatively, you can use the computer to execute that part of the program between **DO** and **LOOP** while a condition is false, but to stop as soon as it becomes true, using **DO UNTIL (condition)**. Similarly, **LOOP WHILE** and **LOOP UNTIL** will allow conditional **LOOPing** at the end of a **DO-LOOP**. Conditions can be used at both the entry

and exit to the **LOOP** if you like.

**IF**

And finally, **IF** can be used to affect **DO-LOOPS** in two ways. **EXIT IF (condition)** can be used to leave the routine from somewhere in the middle rather than at the **DO** or **LOOP**. If the specified condition after the **EXIT IF** is true, the program will jump to the statement that follows **LOOP**, otherwise the routine carries on. **LOOP IF (condition)** branches from the middle of the routine back to the **DO** if that condition is true. If the condition isn't true, then again everything continues to the end of the routine.

Because these BASIC commands consist of words which have a similar meaning in English, their use should become clear very quickly. There is also a group of three functions that consist of short English words, whose use is equally obvious. **AND**, **OR** and **NOT** are used to combine two or more logical operations.

**AND, OR,  
NOT**

**IF** something is true **AND** something else is also true **THEN** that relationship can be exploited, for example:

```
□ 10 IF a$="true" AND x=1 THEN GO  
SUB 500
```

But a relationship can be made true whenever one thing **OR** the other is true, such as:

```
□ 10 IF x > 1 OR sky$="blue" THEN  
PRINT x
```

Perversely, the use of **NOT** makes false things true and true relationships false, when used on its own or in combination with **AND**, or **OR**. Try **LET**ting **x**, **y** and **a\$** have suitable values, then **RUN** this until you understand how it works:



- **10 IF NOT x <> y OR NOT a\$="sam"  
THEN NEW: GO TO 10**

These logical expressions can be put in brackets just like numerical expressions, and they are worked through with **NOT** having the highest priority, followed by **AND**, and with **OR** having the lowest priority. In actual fact, by changing relationships, you can dispense with **NOT** altogether.

**ON**

A particular statement can be selected from the rest of a line of program by using the keyword **ON**.

- **ON x: PRINT "one": PRINT "two":  
GO SUB 100**

will **PRINT "one"** if  $x=1$ , **"two"** if  $x=2$ , and **GO SUB** line 100 if  $x=3$ . If  $x$  has a value of zero, or is greater than the number of statements following **ON**, the program continues with the next line.

**FUNCTION  
KEYS**

The function keys on the right hand side of the SAM Coupé keyboards have been pre-defined as follows:

- [F0] **LIST [RETURN]**
- [F1] **RENUM [RETURN]**
- [F2] **PRINT**
- [F3] **MODE**
- [F4] **RUN [RETURN]**
- [F5] **CONTINUE [RETURN]**
- [F6] **CLS # [RETURN]**
- [F7] **LOAD " " [RETURN]**
- [F8] **LOAD " " code [RETURN]**
- [F9] **BOOT [RETURN]**

Please refer to Chapter 6 for redefining key codes.

**DEFining  
FuNctions**

Function names can have any length, must begin with a letter and continue with letters or numbers.

## String conversion



For example:

```
□ DEF FN charset = DPEEK SVAR 566
  + 256
```

would give the location of the character set, with the command **PRINT FN charset**. String function names must end with \$, such as

```
□ DEF FN Left$(a$,n)=a$( TO n)
```

Functions do not have to have brackets if there are no arguments, but any variable names used inside brackets will be local to the **FN**, and can only be one letter long.

Here are a few more useful functions that you should become familiar with. They are all designed to help with the handling of strings.

### STR\$

converts numbers and numeric expressions into strings, giving the same result as if the number was to be displayed on screen by a **PRINT** statement.

```
□ PRINT STR$(30*5)
```

### VAL

converts strings back into numbers, and the string which is used as its argument can contain any numeric expression. A similar function is also available called **VAL\$**, which uses a string as an argument and gives a string as its result. E.g.

```
□ PRINT VAL "69*2+3"
```

### CODE

can be used to give the first character in a string, and if the string is empty, the result 0 is given. E.g.

```
□ PRINT CODE a$
  PRINT CODE "x"
```

**CHR\$**

gives a single character string when it is applied to the code number of that character. Please see Chapter 6 for character codes. E.g.

□ **PRINT CHR\$ 65**

**Searching strings****INSTR (n, a\$, b\$)**

will search for **b\$** inside **a\$**, starting at position **n**. Its position is returned when found, or the result **0** is given if the search is unsuccessful. By omitting **n**, the search begins from position 1.

**PROCEDURES**

You will be pleased to discover that there are certain procedures that make programming easier! 'Modules' can be created that execute a specific task without affecting the main program, but first they have to be defined.

**DEFining PROCedures****DEF PROC name**

kicks off the **DEF**inition of a named **PROC**edure, and **DEF PROC** must be the first keyword in a program line (although preceding spaces and colour control codes are allowed). The procedure name must start with a letter, followed by a string of letters or numbers or "\$". Spaces are **NOT** allowed as any part of this name, and upper and lower case letters are treated exactly the same. But you can follow the procedure name by a list of parameters, which must be the names of variables, or you can use the **DATA** keyword. They automatically become **LOCAL** to the procedure. When **DATA** is used instead of a parameter list, no assignments are made, but the list of actual parameters that occur after the procedure has been called can be dealt with by using **READ** plus the function **ITEM**, which is explained below.

The location of a procedure in the program has no effect on the speed of the program whatsoever.

**LOCAL**

specifies a list of variables, including arrays, which you want to be local to any procedure anywhere in the program.

**ITEM**

is a function that relays information concerning the next **ITEM** to be **READ** in a **DATA** statement. The result **0** is given when there are no more **ITEMs**, **1** is given if the next **ITEM** is a string type, and if it is a numeric type, **2** is given.

**ENDING  
PROCEDURES****END PROC**

acts as you would expect, by marking the end of a named procedure. This allows the computer to avoid ploughing through all the statements between **DEF PROC** and **END PROC** unless that particular procedure is called up. **END PROC** erases any **LOCAL** variables, and restores their 'normal' values, if they exist.

**DEFAULTED  
VARIABLES****DEFAULT**

There are certain circumstances, usually to do with procedures, when you might want to create a variable only if it doesn't already exist. This is achieved by using the command **DEFAULT**. For example

□ **DEFAULT x=20**

will make  $x=20$  if  $x$  is not already existing, otherwise  $x$  will keep its current value.

**DISCARDED  
ADDRESSES****POP**

will discard a **GOSUB/PROCEDURE/RETURN** address, and **POP x** will assign a discarded **RETURN**

**REferences**

line number to a variable such as **x**.

In a **DEF PROC** list, variables (including arrays) are passed by **RE**ference if their parameter is preceded by **REF**. This means that the variable is renamed to the name given in the **DEF PROC** list, and any changes to it will persist outside of the procedure. Otherwise variables will have values assigned to them from the list in the procedure call. It is possible to omit any or all values, and supply them from inside the procedure, using **DEFAULT**. With non-**RE**ferenced variables, only the value is passed to the procedure, and the variable is **LOCAL**.

**LABELing**

*"Today there's  
law and order in  
everything"*

(Maxim Gorky)

Variables can be created by using **LABEL** at the start of a line, followed by a legal numeric variable name. When any part of the program is run via **GO TO** etc., the newly created variables are given the name after any **LABEL**s in the program, and their values are set to the line number with that **LABEL**. E.g.

```

□ 500 LABEL tea: PRINT "two sugars
  please"
  600 GO TO tea

```

The variables are re-created if **CLEAR** is used, or the program is **[EDIT]**ed.

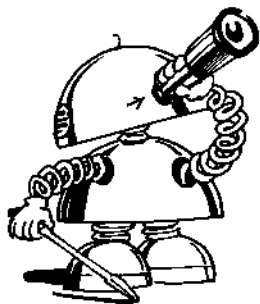
**System variables**

There are certain bytes of the SAM Coupé's memory that have been set aside for special uses, known as system variables. You can have a look at them by **PEEK**ing, and some of them can be **POKE**d without doing any harm to the system, but they are not the same as the variables used in BASIC and the computer won't recognise their names. The hundreds of system variables are detailed in the Technical Manual, and the most useful system variables are listed in the Appendix of this Manual. **SVAR** is used

to give the address of a system variable, when followed by its number. This can be used to customise system variables such as changing the lower-case cursor from " \* " to "x" in the following way:

□ POKE SVAR 1, "x"

Finally, if you've been looking for **PEEK** and **POKE** details, you can treat yourself to the examination of memory in Chapter 10.



# Chapter 5

## COLOUR AND LIGHT

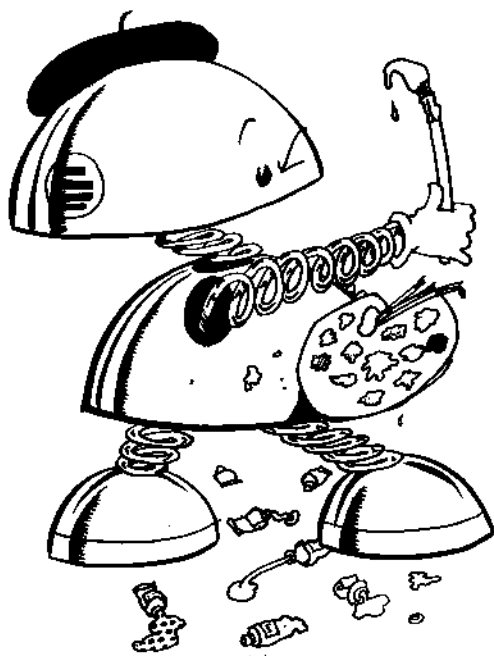
- using the **PALETTE**
- **PEN** and **PAPER**
- the four video **MODES**
- **BRIGHT** and **FLASH**
- **INVERSE** and **OVER**

*'Enjoy a rainbow without forgetting the forces that made it.'*

(Mark Twain)

*'You can have any colour you want, so long as it's black!'*

(Henry Ford)



**the PALETTE**

In the past, anyone wanting to enjoy a wide range of superb colours in computer programs had to invest in relatively expensive machines. The SAM Coupé changes all that! It has four video **MODES** built into it, and each of them offers no less than **128** colours to choose from.

Imagine yourself as an artist equipped with one of those gently-shaped boards on which you squeeze out colours from different tubes of paint. Artists call this board a **PALETTE** and the Sam Coupé provides you with just such a board for holding all the colours that you can use. The paint manufacturers make **128** different coloured tubes, and your **PALETTE** has room for 16 at a time, which it holds in numbered paint pots.

Every time you switch on, and each time you use the command **PALETTE**, you are given 16 pre-set colours to work with, and they are coded from **0** to **15** as follows:

0 black	8 black
1 deep blue	9 bright blue
2 deep red	10 bright red
3 magenta (purple)	11 bright magenta
4 deep green	12 bright green
5 cyan (light blue)	13 bright cyan
6 deep yellow	14 bright yellow
7 white	15 bright white

There are two tubes of black paint simply because your computer prefers things that way! Now imagine that you are choosing which extra colours you want to use for painting your own masterpiece, and you want to see the whole range available for use.



## displaying the Colour Chart

Switch on and type in this colour-chart program:

```

□ 10 FOR c=0 TO 127 STEP 8
    20 FOR p=0 TO 7
    30 PALETTE p+8,p+c
    40 PRINT p+c; TAB 3; PAPER p+8;
        STRING$(29," ")
    50 NEXT p
    60 PRINT
    70 PRINT "Press any key"
    80 PAUSE
    90 CLS
    100 NEXT c
    110 PALETTE
  
```

Make sure there is a space between the quotes in Line 40, and **RUN** the program.

What you should see on your screen now is a colour chart of the new tubes of paint on offer eight at a time, with their code numbers printed in front of them. It's possible that some users have only got a black-and-white television for displaying pictures and I apologise, but I can't think up names for 128 different shades of grey. For anyone enjoying glorious colour displays, you can see if you agree with my descriptions in the colour-chart table below.



## COLOUR CHART TABLE

0	pitch black	32	Burgundy	64	grass	96	damp straw
1	true blue	33	plum	65	slime	97	slug belly
2	brick red	34	hellfire	66	leaf green	98	wild honey
3	magenta	35	strike-me pink	67	sick parrot	99	weak tea
4	leaf green	36	butterscotch	68	lime	100	fern green
5	cyan	37	hog foot	69	jade	101	spring bud
6	banana	38	orange	70	apple green	102	cowardice
7	paper white	39	shrimp	71	mint dulep	103	papyrus
8	charcoal	40	sunset red	72	algae	104	sweetcorn
9	electric blue	41	mauve	73	sage	105	oyster
10	poppy	42	scarlet	74	kiwi fruit	106	apricot
11	lavender	43	shocking pink	75	olive green	107	piglet pink
12	emerald green	44	sandstorm	76	bowling green	108	envy
13	sky blue	45	doggy tongue	77	greenfly	109	China tea
14	primrose	46	pineapple	78	gooseberry	110	cold custard
15	snowflake	47	guava	79	lettuce	111	daffodil
16	navy blue	48	woad	80	cold steel	112	mushroom
17	peacock	49	misery blue	81	baby blue	113	glazed grape
18	deep purple	50	geranium	82	shark	114	salmon pink
19	Picasso	51	amethyst	83	lapis lazuli	115	shrunk violet
20	denim	52	moleskin blue	84	ghostly green	116	pistachio
21	bluebell	53	Pluto blue	85	turquoise	117	frost bite
22	Parma violet	54	terracotta	86	sea spume	118	parchment
23	bluegrass	55	pomegranate	87	duck egg blue	119	dandruff
24	Atlantic	56	aubergine	88	cold fish	120	turnip
25	Mediterranean	57	blueberry	89	pastel blue	121	iris
26	violet	58	Coupé rosé	90	pumice	122	caucasian
27	dusk blue	59	coral pink	91	purple haze	123	rose pink
28	dolphin	60	damp squid	92	blue bile	124	chlorophyll
29	sapphire	61	bilberry	93	Neptune blue	125	ice blue
30	powder blue	62	tuna	94	Sargasso sea	126	flax
31	Wedgewood	63	twilight	95	storm cloud	127	moonlight

**BORDER**

Don't forget that you can change the colour of the **BORDER** around the useable screen area, and the command **BORDER** changes the picture's surrounding frame to any colour in your **PALETTE** from **0** to **15**. For example:

□ **BORDER 1**

changes the screen **BORDER** to deep blue, with the 'normal' **PALETTE**.

**PEN and  
PAPER**

An artist needs something to paint on as well as tools to paint with, and you are provided with a **PEN** and **PAPER** to achieve this. Every character or graphic image that you **PRINT** onto your screen can have a different background or **PAPER** colour, and you can select a different colour **PEN** as well. You may use the command **INK** instead of **PEN** if you prefer, it will still appear in the programme listing as **PEN**! Try changing the colour of your screen background with:

□ **PAPER 6; PRINT "bumble bee"**

which will appear as black on yellow. Now change your **PEN** to red with:

□ **PRINT PEN 2; "well red"**

and experiment with different colour combinations. If you prefer a temporary state, the command **OVER** can be used. Please see Glossary.

**SCREEN  
MODES**

This computer can change its style of video display to suit your needs! There are four screen **MODES** available for your use, and each one uses different amounts of memory, and offers different attributes. A new screen **MODE** is selected simply by typing in:

□ **MODE n**

with **n** being the number from **1** to **4** as explained below.

*"The pen is  
mightier than  
the sword"*

(Cardinal Richelieu)

**Spectrum  
compatible  
MODE****MODE 1**

gives you 24 lines to use, with each line consisting of 32 'cells' made up of a pattern of 8 x 8 dots and spaces, like the 'A' illustrated later in this chapter. Your choice of colours for the **PEN** and **PAPER** of each 'cell' can be made from 16 'paint pots' selected from a range of 128 colours. Each screen will consist of 768 character cells. This is the screen **MODE** used by the Spectrum computer, so Spectrum users can feel at home simply by [**ENTER**]ing **MODE 1** for using their games library. The SAM Coupé will examine their software with the conversion utility included on the 'FLASH!' demonstration cassette.

**Alternative  
'cells'****MODE 2**

offers 192 lines with 32 cells each, giving a total of 6144 character cells of 8 x 1 dots, with colour selections the same as in **MODE 1**.

**Utility MODE****MODE 3**

allows up to 85 columns of characters, which is especially useful for applications like word-processing. Only four colours are used. **MODE 3** doesn't use cells at all, but lets you use the individual dots or 'pixels'. You are given 192 lines with each line having 512 pixels, giving 98304 dots to play with. The first time you select **MODE 3**, the first four **PALETTE** positions are redefined to **black**, **blue**, **red** and **white**. If you switch to another **MODE**, your previous selection of colours is poured back into these paint pots, and returning to **MODE 3** sets them back again to whatever was used last in this **MODE**. Of course you can change the pre-selected colours to any one of the 128 on offer.

**High  
resolution****MODE 4**

is the SAM Coupé's favourite, and is selected every time you switch on. There are 192 lines of 256 pixels

each, and you can use 16 colours out of the range of 128 over the 49252 dots on screen. There is a way to use all 128 colours on screen at once, which is explained later!

Because everything on a screen can alter instantly as the **PALETTE** is changed, some spectacular effects are easily achieved, as the demonstration programs on your **'FLASH'** cassette prove.

Experiment with a few lines of coloured text in each **MODE**. Now try out two special settings that can be used in **MODEs 1** and **2** only. **PEN 16** or **PAPER 16** means 'transparent', and **PEN 17** or **PAPER 17** means 'contrast'.

Here is a summary of the SAM Coupé's four screen **MODEs**:

#### **MODE 1**

32 cells \* 24 lines = 768 character cells, each cell has individual choice of **PEN** and **PAPER** colour.

256 \* 192 pixels, choice of any 16 screen colours from 128.

#### **MODE 2**

32 cells \* 192 lines = 5,444 cells, each cell has individual choice of **PEN** and **PAPER** colour.

256 \* 192 pixels, choice of any 16 screen colours from 128.

#### **MODE 3**

512 pixels \* 192 lines

each pixel has individual choice of colour, choice of any 4 screen colours from 128.

#### **MODE 4**

256 pixels \* 192 lines

each pixel has individual choice of colour, choice of any 16 screen colours from 128.

**Attributes**

These attributes are not just limited to **PEN** and **PAPER**. You can create a series of special effects by using two more attribute controls called **BRIGHT** and **FLASH**.

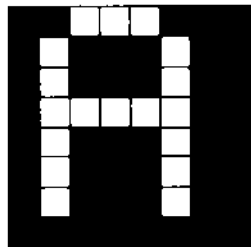
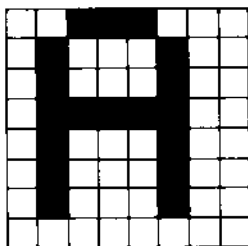
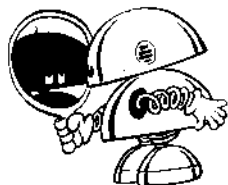
**BRIGHT**

If you want to make a colour more brilliant, **BRIGHT 1** turns the instruction on, and **BRIGHT 0** sets the colour back to normal. There is a special setting using **BRIGHT 8** or **BRIGHT 16**, which means 'transparent': in other words, the brightness on the screen is to be left unchanged when a character is printed there.

**FLASH**

The **FLASH** instruction works in **MODEs 1** and **2** only. This particular attribute of a given colour makes it appear to **FLASH** rhythmically. **FLASH 1** turns it on, and **FLASH 0** turns it off. **FLASH 8** or **16** sets a 'transparent' state that is left unchanged at any character position.

Whenever a character appears on the screen, it appears as a pattern of coloured dots chosen by your **PEN** on a background colour chosen by your choice of **PAPER**. So when you press the **[SHIFT]** and **[A]** keys together the symbol for a capital 'A' may appear as the pattern of dots in the left hand diagram:



**[INV] key**

Now look at the right hand diagram. By pressing the **[INV]**erse key before **[SHIFT][A]** the pattern of dots is reversed, with the **PEN** colour becoming whatever the **PAPER** colour was and vice versa.

**INVERSE**

There is also an **INVERSE** command, used to cause characters that follow it to be printed as **PAPER** colour on a background of **PEN** colour. This is set by **INVERSE 1**, and turned off by **INVERSE 0**.

**OVER**

**OVER** is a statement for controlling the pattern of graphic dots on the screen, and causes a sort of **OVER**printing of points, lines and characters. **OVER 1** turns the effect on, and **OVER 0** turns it off. In **MODEs 1** and **2** a character is repeatedly printed over itself in a rapid sequence that goes character/blank/character. In **MODEs 3** and **4** the sequence is character/colour/character, the usual colour being black. Other effects can be created when used with **DRAW**, **CIRCLE**, **PLOT** and **PUT**, which are examined in Chapter 6.

**POINT x,y**

If you want to find out the status or colour of a graphic dot or 'pixel' anywhere on the screen, you can use the function **POINT** which is explained in the Glossary.

Finally, here is a summary of all the ways you can use your **PALETTE**:

**PALETTE**

stops all changes to the colours you can use and resets the original colours.

**PALETTE position, colour**

sets the position of the 'paint pots' on your **PALLETTE** to any colour you want. There are sixteen different positions (**0** to **15**) and 128 different colours

available (0 to 127), so that:

□ **PALETTE 14, 43: BORDER 14**

will pour shocking pink paint into paint pot number 14, and display it on your screen.

**PALETTE position, colour a, colour b**

lets you choose two different colours to appear at the same palette position and swaps between them three times every second. You can create flashing special effects by using this command, for example, pour multi-coloured flashing paint into pot number 7 on your **PALETTE** with:

□ **PALETTE 7,8,36**

**PALETTE position, colour, LINE y**

is a command that sets a temporary palette position and colour when the **y** axis of the graphics coordinates system of a specified **LINE** on the screen is reached. The new setting continues to operate until the bottom of the screen, unless another setting is chosen. This is a way of displaying more than sixteen colours on one screen.

**PALETTE position, colour a, colour b, LINE y**

is a command that lets you choose temporary alternative flashing colour, at a given **y** coordinate with a maximum of 127 changes per screen.

**PALETTE position LINE y**

deletes any colour change that would affect **PALETTE** position **p** at line **y**.

*"She comes in  
colours every-  
where, she's like  
a rainbow"*

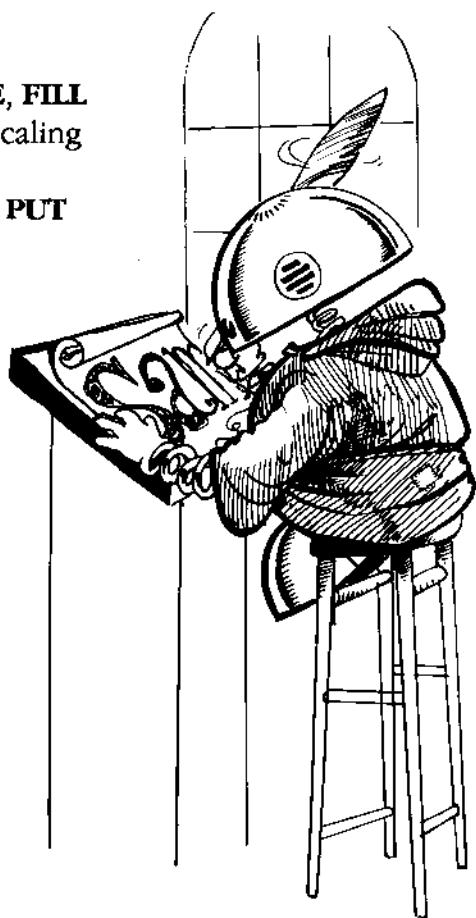
(The Rolling Stones)



# Chapter 6

## TEXT AND GRAPHICS

- graphic commands  
**PLOT, DRAW, CIRCLE, FILL**
- **FATPIX** and graphics scaling
- **SCREENs**
- **WINDOW, GRAB, and PUT**
- animation
- **INKEY\$** and **GET**
- the character set
- **CSIZE, TAB, AT**
- **BLOCK** graphics
- user defined graphics



*The difficulty is  
not to mean  
what you write,  
but to write what  
you mean.'*

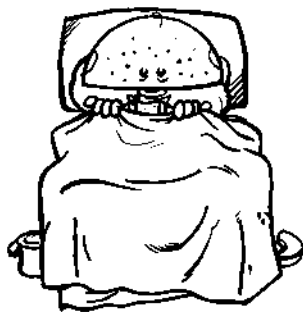
(Robert Louis  
Stevenson)

*'Art is a lie.'*

(Pablo Picasso)



## THE GRAPHIC COMMANDS



This chapter explains how to use the advantages of your computer for manipulating written text and graphic images. Old-fashioned typewriters use keys to print out pre-set characters, and more modern machines allow you to swap between a series of factory-made typefaces. But with the SAM Coupé you can make most keys give you any symbol you want, as well as use them for drawing complex pictures.

There is a whole host of special comands for creating shapes, patterns and works of art. We have included some very simple, but highly effective graphics routines on your 'FLASH' demonstration cassette, and you are welcome to use your [ESC] key or [BREAK] button to help examine their listings.

### PLOT $x,y$

is a statement used for pinpointing a graphic start position on your screen. This graphic process begins wherever you instruct the first picture element (pixel) to be located. It's done by typing **PLOT** followed by two numbers. The first number is the  $x$  coordinate, which tells the pixel how far it must be from the left hand side of your screen, and the second number is the  $y$  coordinate, saying how far it is from the bottom of the screen. So that

□ **PLOT 128,86**

will target a spot near the centre of your screen in **MODES 1, 2 or 4**, or left of centre in **MODE 3**. You can pick a colour before using **PLOT**, so try giving your computer red measles with this program in **MODE 4**.

```

□ 10 PLOT PEN 10; INT (RND*255),
  INT (RND*173); GO TO 10

```

## DRAWing

### **DRAW x,y**

Spots are all very well, but individual pixels are easily manipulated into lines. **DRAW** is used to draw a line relative to wherever you are on the screen. The value of **x** is the number of pixels to be moved to the right, and **y** determines the number of pixels to be moved upwards. If you want to move to the left or downwards, simply give **x** or **y** a negative value.

### **DRAW x,y,z**

works in the same way, but by adding the angle **z**, the line is turned and becomes a curve.

### **DRAW TO x,y**

is the command for drawing an absolute line from the current position to the point **x,y**. You can use **PAPER, PEN, OVER**, etc. after **DRAW TO** (and after **DRAW, PLOT** and **CIRCLE** commands). The following example draws random lines that never go off screen:

```

□ 10 FOR n=1 TO 100
  20 DRAW TO RND*255,RND*173
  30 NEXT n

```

### **DRAW TO x,y,z**

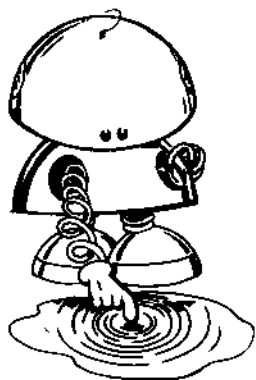
draws an absolute line from your current position to the coordinates **x,y**, while creating a curve by turning through the angle **z**. Draw a smile like this:

```

□ 10 PLOT 90,86
  20 DRAW TO 166,86,2

```

Leave the smile on your screen, and draw a circle round it.



### FILLing

#### **CIRCLE $x,y,radius$**

draws a circle on the screen, when it is followed by the position of the centre of the circle on screen (the  $x,y$  coordinates) and the radius of the circle. Now surround the smile with:

□ **CIRCLE 128,100,50**

If larger circles go far enough off the top or bottom of the screen, they will wrap around to the opposite edge of the screen.

#### **FILL colour, $x,y$**

is a graphics command for **FILLing** any enclosed shape with one colour. Colours are specified by using their code numbers, and the target area is specified by pinpointing anywhere within the boundary area to be **FILLED** by the usual  $x,y$  coordinates. So that, leaving your smiling circle on screen:

□ **FILL PEN 3,128,86: FILL PEN 6,1,1**

fills the shape around the target point, and any colour other than the one at  $x,y$  acts as the boundary. Special effects can be created by **FILLing** with patterns held in a string, so that:

□ **FILL USING a\$, $x,y$**

will use whatever pattern is specified by  $a\$$ . The most convenient way to scoop up such a pattern is to **GRAB** it from an existing screen, which is explained later.

#### **FATPIX**

In graphics **MODE 3**, the pixels are 'thin', only half the width used by the other **MODEs**, because there are twice as many of them horizontally. By using the command **FATPIX 1** you can double the width of **MODE 3** pixels, making vertical lines the same

## Graphics scaling system

thickness as horizontal lines and thin circles truly circular. To get back to the normal 'thin' state of affairs use **FATPIX 0**, which doubles the x-axis range in **MODE 3**. We can now examine ways of manipulating the x-axis and y-axis.

The scale and origins used by the **PLOT**, **DRAW**, **DRAW TO**, **CIRCLE**, **GET** and **FILL** commands can be changed, using four special variables.

**XOS** controls the **x** axis off-set of the graphic origin

**YOS** controls the **y** axis off-set

**XRG** controls the **x** axis range

**YRG** controls the **y** axis range

The off-sets have a value of **0** unless they are used with a **LET** statement, as in:

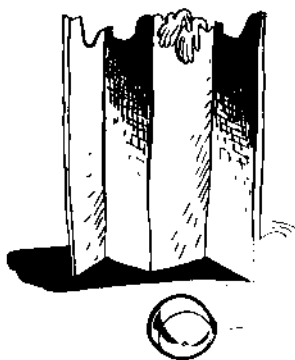
```
□ LET XOS=128,YOS=69
```

which moves the origin of all graphics coordinates to the centre of the screen. Similarly, changing **XRG** and **YRG** alters the scale to which **PLOT** and **DRAW** work. Combined with a **WINDOW** you can even create split-screen special effects.

Normally, position **0,0** of the x,y coordinates is at the lower left-hand corner of the screen, just above the bottom editing section, with coordinates **255,173** at the top right-hand corner. If 'thin' pixels are used in **MODE 3**, the **x** range is expanded so that the top right becomes **511,173**. The SAM Coupé also allows the area normally allocated for the two 'editing lines' to be used for **PLOT**ting by:

- **LET XOS=-18** if characters are 9 pixels high, or
- **LET XOS=-16** for those with a height of 8 pixels.

This makes coordinates **0,0** the extreme bottom

**BLITZing****USING  
SCREENS**

left-hand corner of the screen, and makes 0,191 the top left.

**BLITZ a\$**

is used to execute a string of graphics commands very quickly. Although it works in all **MODEs**, it's especially effective in **MODE 4**. **BLITZ** responds to the special graphics scaling variables **XOS** and **YOS**, which enable a graphic shape to be **BLITZed** anywhere on screen, and the size of the **BLITZed** shape can be altered by changing the special graphic variables **XRG** and **YRG**.

When you switch on the SAM Coupé, there is only one screen available called **SCREEN 1**. But you have access to a total of 16 screens to use in any way that you need.

**SCREEN number**

selects a particular **SCREEN** for use by **BASIC** commands. Its number must be between 1 and 16, and it must first be **OPENed** like this:

**OPEN SCREEN 2**

This allows commands like **PRINT**, **PLOT** and **DRAW** to use a second **SCREEN**, leaving **SCREEN 1** intact until it is selected again.

**OPEN SCREEN number, mode**

reserves memory for a screen, when qualified by the screen number, followed by the screen **MODE**. The screen number must be between 2 and 16, and **MODEs 1** to 4 can be selected. Each screen retains its own **MODE**, **PALETTE** details, **PRINT** positions, and so on.

**OPEN SCREEN n,m,0** acts the same as **OPEN SCREEN n,m**

**OPEN SCREEN n,m,1** opens a 'common' screen, so that if several programs are present in the

machine's memory, they can all use the screen when they are running.

**CLOSE SCREEN number**

releases the numbered screen's memory for another use. If you **CLOSE** a screen that doesn't exist, nothing will happen.

**DISPLAY number**

alters which screen is being **DISPLAY**ed, but not the screen used by BASIC. One screen can be building up invisibly while another is currently **DISPLAY**ed.

**DISPLAY** or **DISPLAY 0** shows the current screen.

**DISPLAY 1** to **DISPLAY 16** shows the required screen number.

**CLS**

stands for 'clear screen', and wipes off the display file from the screen. **CLS** or **CLS 0** clears the entire screen, whereas **CLS 1** clears the current **WINDOW** area of a screen only (see **WINDOW** below). There is a more powerful clearing command.

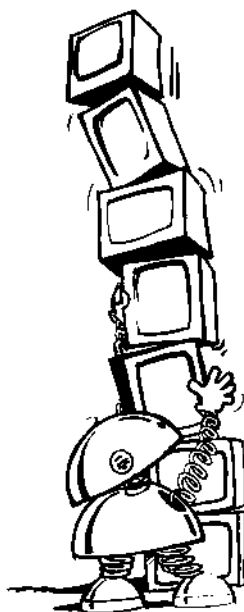
**CLS#**

not only clears the screen, but executes the commands **OVER 0: INVERSE 0: FLASH 0: PAPER 0: PEN 7: BORDER 0: PALETTE** at the same time.

Sometimes your screen isn't big enough to show everything you need, especially if you are producing complex documents and tables. When this happens, the screen displays the question '**Scroll?**' and waits for a response in order to scroll the screen upwards. But you can adjust what you see on the screen in a variety of other ways, either by moving the entire screen display vertically or horizontally, or by moving a particular section of the screen display, known as a **WINDOW**.

*"Obedience ....  
a mechanised  
automation!"*  
(Shelley)

**Scrolling  
and rolling**



## Windows

### **SCROLL CLEAR**

disables the 'Scroll?' prompt altogether when the screen becomes full, and makes the screen **SCROLL** upwards automatically.

### **SCROLL RESTORE**

turns the 'Scroll?' prompt back on.

### **ROLL direction,number of pixels**

moves the picture on your screen and wraps it around itself. Movement is controlled by direction (1=left, 2=up, 3=right, 4=down), followed by the number of pixels to be moved (1, 2, 4, 6, 8, 10, etc.).

### **ROLL direction,pixels,x,y,width,length**

only **ROLLS** the screen section designated by **x,y** coordinates marking the top-left corner, followed by the number of pixels wide and the number of pixels long. **X** is rounded to an even coordinate and the width is rounded to an even number of pixels. This technique only works in **MODEs 3 and 4**.

### **SCROLL direction,number of pixels**

works in exactly the same way as **ROLL** except that there is no wrap around effect. Blank background replaces the lost data.

### **SCROLL direction,pixels,x,y,width,length**

works in exactly the same way as its **ROLL** counterpart, except that there is no wrap around effect.

A screen window is rather like the picture-within-picture on some video recorders, and refers to a specified area that is to be manipulated independently from the rest of the screen.

### **WINDOW left, right, top, bottom**

sets up a text **WINDOW**. The limits are set by designating which left-hand column to use (**0** or above), followed by the right-hand column (usually



column **31**, but this can be up to column **84** in **MODE 3** with 6-pixel wide characters), followed by the top line to be used (**0** or above) with the fourth number giving the bottom line of the **WINDOW**. If the requested borders are not allowed in the current **MODE** or character size then 'Invalid **WINDOW**' is reported.

#### **GRAB a\$,x,y,width,length**

stores a screen area to a string, which can be **PUT** somewhere else. The screen area to be **GRABbed** is defined by **x,y** as the top left-hand corner, followed by its width and then its length in pixels. **GRAB** and **PUT** only work in **MODEs 3** and **4**, and the **x** coordinate for 'fat' pixels is rounded up to an even value. For **MODE 3**'s 'thin' pixels, the **x** and **y** values are rounded to a multiple of 4.

#### **PUT x,y,a\$**

Places a stored area on the screen which has already been **GRABbed**. The **x** coordinate is rounded up to an even value. **PUT** will respond to the following commands:

**INVERSE** as in **INVERSE 1: PUT x,y,b\$**

or as in **PUT INVERSE 1; x,y,ANY\$**

**OVER 0** which overwrites whatever is already there

**OVER 1** which gives **XOR**

**OVER 2** which gives **OR**

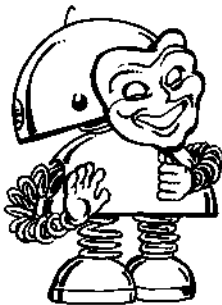
**OVER 3** which gives **AND**

but **PEN** and **PAPER** will not get a response, so you're stuck with the colours that you originally **GRABbed**.

#### **PUT x,y,a\$,n\$**

allows you to use a second string (which has to be the same length as the first string) to act as a graphic 'mask', determining which pixels will be used from

**masking**



## MOVEMENT and ANIMATION

*"Cinema is truth  
twenty-four  
times a second"*

(Jean-Luc Godard)

the first string. This allows a complex shape to be placed on a background as if it was a cut-out without a border. The 'mask' will not respond to **OVER**, but **INVERSE** will work perfectly.

There is another way of making a 'mask', by following these stages: (1) create the original shape on screen. (2) **DRAW** a box round it, slightly bigger than the area to be **GRABBED**. (3) **FILL** the box with **PEN 0**, to give it a border of zeros. (4) **GRAB** the area. (5) **FILL** the border area with **PEN 15**, to give a border of ones. (6) **PUT** the stored area over itself with **OVER 1**. (7) **GRAB** the area again. (8) **PUT** it back using **OVER 0**, **INVERSE 1**. (9) **GRAB** it for the last time to get a useable mask!

Movies and videos are really a rapid sequence of still images, each one a little different from the last. Computer animation works in exactly the same way to fool the eye. The simplest form of animation alternates graphic characters to give an impression of movement, such as:

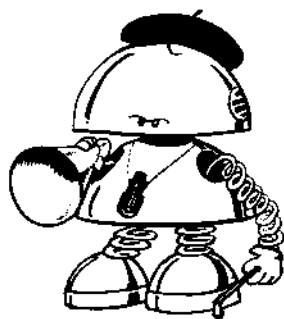
```

□ 10 PRINT "/" : PAUSE 3: CLS:
      PRINT "- " : PAUSE 3: CLS:
      PRINT "\" : PAUSE 3: CLS:
      GO TO 10
  
```

which is supposed to be a propellor. Pathetic isn't it. What is needed is a method to make our graphic images move around the screen, and to achieve this we must **PRINT** them in the right place, then wipe them off and **PRINT** them at the next position and so on. What we also need is a method of controlling their speed and direction of movement.

*"Television? No good will come of this device. The word is half Greek and half Latin"*

(C. P. Scott)



The next program creates an animated tapeworm:

```

□ 10 PEN 15: PAPER 0: BORDER 0: CLS
   20 LET tail=3: DIM aa(tail):
     DIM bb(tail)
   30 LET x=1: LET a=9: LET b=15
   40 PRINT AT a,b; PEN 12; "0"
   50 LET c=CODE (INKEY$):
     IF c<193 OR c>201 THEN GO TO 50
   60 IF c=197 THEN CLS: LET x=1
   70 LET aa(x)=a: LET bb(x)=b
   80 LET a=a+(c>192 AND c<196 AND
     a<18)-(c>198 AND c<202 AND a>1)
   90 LET b=b-((c=193 OR c=196 OR
     c=199) AND b>1) + ((c=195 OR
     c=198 OR c=201) AND b<30)
  100 IF x<tail THEN LET x=x+1:
     GO TO 40
  110 IF aa(x)≠a OR bb(x)≠b THEN
     GO TO 130
  120 GO TO 40
  130 LET aa=aa(1): LET bb=bb(1):
     LET z=1
  140 IF z=tail THEN PRINT AT aa,bb;
     " " : GO TO 40
  150 LET aa(z)=aa(z+1):
     LET bb(z)=bb(z+1): LET z=z+1:
     GO TO 140

```

Before you **RUN** that, try and understand how line 20 sets up the length of the tapeworm's tail and how line 40 creates its colour and shape. Line 30 sets up variables for screen coordinates. Lines 50 and 60 get ready to recognise certain key codes, and lines 80 and 90 set up 8-way direction controls if certain [F]

keys are pressed and their codes recognised. **[F4]** gives West, **[F7]** North-West, **[F8]** North, and so on clockwise. Line 60 gives **[F5]** a special function.

Try controlling the worm's travels by holding down various **[F]** keys. Now change its length by altering the variable **tail** in line 20. The longer it is, the more tired it becomes, so don't go much beyond **LET tail=10**. The shorter the worm the faster it moves, due to the speed of the **PRINTing**. Just try **LETting tail=1**. Now alter its colour and shape in line 40. Next, take a look at line 140 and see how the space between the **PRINT** quotation marks rubs out the last segment as the worm moves along. Finally change line 140 so that a full stop gets printed instead of a space, and watch the little devil leave a slimy trail!

## INKEY\$

You will have seen in line 50 of the last example how the **INKEY\$** function can be used to read which keys are being pressed, if any.

## GET

**GET** is a way of reading the keyboard without the use of **[RETURN]**. It is like **INKEY\$**, except that **GET** waits for a key to be pressed before continuing. When used with a string variable, **GET** acts like a type-writer, so that:

```
□ 10 DO: GET a$: PRINT a$; LOOP
```

prints keys as you hit them, and you can **[SHIFT]** between upper and lower case as normal. If **GET** is used with a numeric variable such as **GET x**, then the variable will equal 1 if **[1]** is pressed, up to 9 if **[9]** is pressed. **[A]** or **[a]** has a value of 10, **[B]** or **[b]** 11, and so on. **GET** is very useful in menu-driven programs.

**RECORD****RECORD TO a\$**

lets you record graphics commands as strings, so they can be executed a lot faster if you **BLITZ** them. While **RECORD** is on, the following commands will be recorded: **CIRCLE**, **CLS**, **DRAW**, **DRAW TO**, **OVER**, **PAUSE**, **PEN**, **PLOT** (but not **PLOT PEN** or other 'temporary' commands).

**RECORD STOP**

turns off the graphics recording. Both **RECORD TO** and **RECORD STOP** work in any **MODE**.

**HANDLING  
TEXT**

The second part of this Chapter explains how you can make the best use of the SAM Coupe's text-handling facilities.

**Pre-set  
keys**

Each key has been pre-set to deliver text or graphic symbols, and there is a table in the Appendix that shows what happens when you press each key using the following alternatives:

**NORMAL** simply pressing the key on its own.

**[SHIFT]** which is the same as pressing the [CAPS SHIFT] key on a typewriter or wordprocessor to give capital letters.

**[SYMBOL]** resulting in a series of alternative characters, graphic blocks or codes.

**[CNTRL]** which is a 'control' status, providing some specialised codes direct from the keyboard, as well as alternative graphic blocks.

Advanced programmers can make use of the keyboard 'map' in the Appendix, which shows the special number codes allocated to each key to indicate their position, but we don't want you to confuse these with the SAM Coupé's system for coding all the alpha-numeric and symbol charac-

**Character set**

ters. The command **KEY position, x** is explained in the Glossary.

You've got access to 256 possible characters, each one with a code between **0** and **255**. All the codes below **32** are used for special purposes, like controlling inverse video, and characters **32** to **168** have been predefined on your SAM Coupé. If you want to take a look at them now, **RUN** this small program:

```
□ 10 FOR a=32 TO 168: PRINT CHR$ a;;
   NEXT a
```

You can try that again, when you understand how to turn the special **BLOCKS** of graphics on and off, but for the time being, if you want to discover individual character codes, try:

```
□ 20 FOR a=32 TO 168:
   PRINT CHR$ a; " -";a: NEXT a
```

Another technique available for advanced programmers is the ability to define shorthand codes that can be automatically expanded when they get called up from your keyboard. The command **DEF KEY-CODE** is also explained in the Glossary.

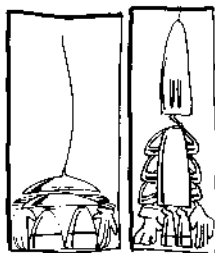
Anyone writing a novel, making a shopping list or cataloguing their record collection can use the SAM Coupé as a video typewriter, and there are several features on-board to help in the presentation of text.

Decide which of the screen **MODEs** is best suited for the particular document you want to write, and then experiment with the size and shape of the characters you can print using this command:

```
□ CSIZE width, height
```

which instructs the computer to change the size of the characters on screen to **x** pixels wide by **y** pixels high. In **MODE 3** your characters can be either **6** or

**Character size**



## TABulating

8 pixels wide, and you can adjust their height in any **MODE** from a squashed 6 pixels to a very stretched 32 pixels high, for use with headlines, early-learning displays or special needs. Any character height specified as being 16 or more results in genuine double height characters, rather than normal characters on widely spaced lines.

If you want to make your text look neat by indenting paragraphs, or need to create orderly columns of words and numbers, there is a built-in **TAB**ulation facility ready to help. **TAB**s are set by the usual method used by typewriters and wordprocessors, and each column is defined by its position from the left-hand edge of the screen using the **PRINT** statement. For example:

**PRINT TAB 2;"sam"**

will indent **sam** by two spaces in **MODE 1**, whereas:

**PRINT TAB 82;"!"**

is allowed if you are using **MODE 3** with characters 6 pixels wide, because **TAB 84** is the right-hand column.

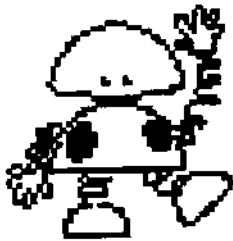
Don't forget that the use of a comma in your listing will cause the **PRINT** statement that follows it to be automatically tabulated by 16 columns. Its effect depends on the **MODE** you happen to be in, with **TAB**s at the centre or left-hand side of the screen in **MODES 1, 2 and 4** with 'normal' sized characters, and all sorts of permutations in **MODE 3**.

## AT

**AT** tells the computer where to **PRINT** anywhere on the screen, when qualified by a line number, followed by a column number. So that in screen **MODE 4**:

**PRINT AT 10,16;"!"**

**PRINT**s an exclamation mark near the centre of the



## BLOCKS

screen at line 10, column 16.

If you are not satisfied with the choice of pre-set characters in the computer's memory, there is plenty of scope to change them to your own needs, or to invent completely new designs. These special character positions have been allocated for your use in a group of user-defined characters. First, let's look at the group of 16 mosaic-shapes, known as block graphics.

These are called up by pressing keys [1] to [8] at the same time as the [SYMB] or [CNTRL] keys, as shown in the chart opposite, and can be used to build up diagrams and simple images in a pattern of several characters **PRINT**ed together.

Block graphics are made up from a pattern of pixels in exactly the same way as normal characters, as described in Chapter 5, and they can be turned on and off at will.

### BLOCKS 0

















turns off the predefined block graphics in the chart, allocated from **CHR\$ 128** to **143**, and frees up those characters for you to redesign as 'user defined graphics', or **UDGs** for short. When you type **BLOCKS 0** and take a look at them by **PRINT**ing their **CHR\$**, you will discover that they have already been redesigned as foreign characters.

### BLOCKS 1

turns the original block graphics back on again, but your redesigned **UDGs** are kept safe for later use, every time **BLOCKS 0** is used.



### Block Graphics Chart

	[SYMB] [1]		[CNTRL] [1]
	[SYMB] [2]		[CNTRL] [2]
	[SYMB] [3]		[CNTRL] [3]
	[SYMB] [4]		[CNTRL] [4]
	[SYMB] [5]		[CNTRL] [5]
	[SYMB] [6]		[CNTRL] [6]
	[SYMB] [7]		[CNTRL] [7]
	[SYMB] [8]		[CNTRL] [8]

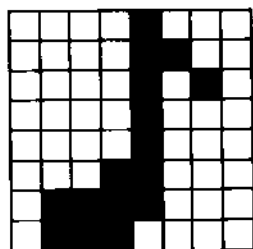
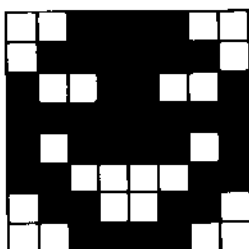
### User Defined Graphics

**CHR\$ 144 to 168** are also foreign character **UDGs**, and **CHR\$ 169 to 255** are available as **UDGs** for advanced users. You can find the address of any user defined character with **UDG**. So that

□ **UDG "A"**

will give the address of the character pattern for **A** in the main character set, and if the upper range of **UDGs** has been set up, then **UDG CHR\$ 170** etc. will work.

To store your own pattern for a home-made character, you will need to tell the computer how the character 'cell' is to be represented, by a grid representing every pixel that makes up that character. If you understand the concept of 'binary' numbers, this is easy, and if you don't, it is not too difficult. A pixel that is to be filled by the current **PEN** colour is represented by a **1**, and a pixel that has nothing in it except the current **PAPER** colour is represented by a **0**. Take a look at the following pair of **UDGs**.



We have chosen a simple 8 x 8 pixel grid from **MODE 1**, but the character cell will vary depending on **MODE** and **CSIZE**. Once you know what your own **UDG** should look like, choose an existing character that is available for you to overwrite, and store its new pattern in **BINARY** code. The smiling face graphic would be written as **BIN**, followed by these numbers representing **PEN** and **PAPER**:

```

□ BIN 00111100
   BIN 01111110
   BIN 10011001
   BIN 11111111
   BIN 10111101
   BIN 11000011
   BIN 01100110
   BIN 00111100

```



Now see if you can write out the equivalent numeric pattern for the musical **UDG** in our example. These numeric patterns can now be **POKE**d into memory, and the correct address will be **UDG "chosen character"** for the first byte (or group of eight digits), followed by **UDG "chosen character"+1** for the next group, down to **UDG "chosen character"+7** in our example. Please see Chapter 10, regarding **PEEK**ing and **POKE**ing.

Don't forget that you can change the **PEN** and **PAPER** colour for any **UDG**, so if you redefine a block graphic as a checkerboard pattern of alternate **0**s and **1**s, any two colours can be effectively mixed together!

We are very pleased to include a full range of options for several different type 'fonts' (as well as a user defined character set that you can customise) in our 'FLASH' demonstration cassette. You will be able to use the whole range of graphic options included in this art package for your own programs, and the 'FLASH' instruction booklet will introduce you to all the possibilities of computer art using your SAM Coupé.

*'Let no one say I  
have said nothing  
new;  
the arrangement  
of the subject is  
new.'*

(Pascal)



# Chapter 7

## MAKING MUSIC

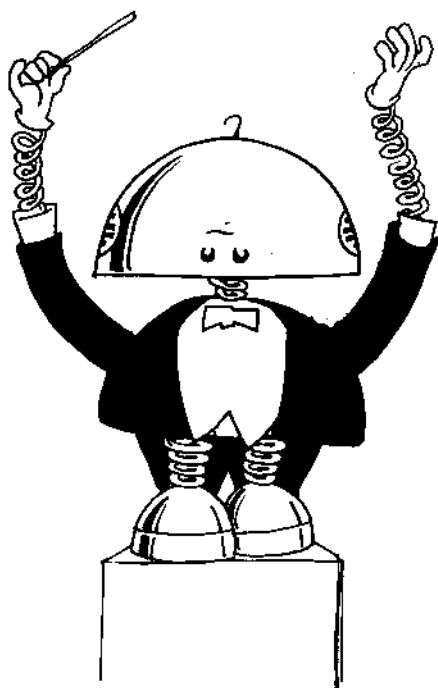
- **BEEP**
- hitting the right note
- frequencies and shortcuts
- volume and stereo
- SAM Coupé the sampler
- the wonders of **MIDI**

*'Music has  
charms to soothe  
the savage  
breast, to soften  
rocks, or bend a  
knotted oak.'*

(William Congreve)

*'Doobie-doobie-  
doo...'*

(Frank Sinatra)



**BEEP**

The Sam Coupé is an excellent music synthesiser! If you are using the UHF TV aerial socket, sounds are relayed through your television speaker in mono. A stereo signal is available via the SCART socket at the back of your machine, or you can enjoy top quality sound by connecting a hi-fi system to the light-pen/stereo socket outlet and use personal headphones if you want privacy.

**BEEP d,p**

The computer is instructed to produce a sound by typing in the command **BEEP** followed by two numbers, representing the duration in seconds (**d**) and the pitch (**p**) of a musical note. See if you are wired for sound by typing in the following routine, with apologies to Beethoven:

□ **BEEP 0.5,0: BEEP 0.5,0: BEEP 0.5,0:  
BEEP 1.5, -4**

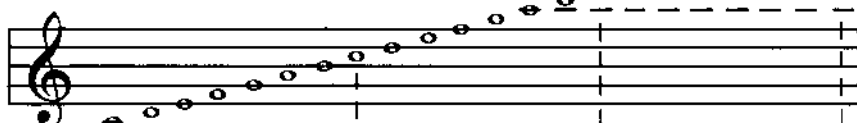
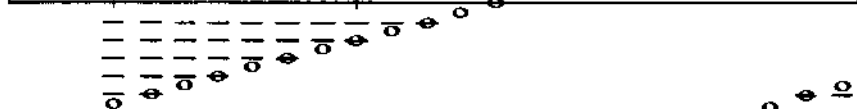
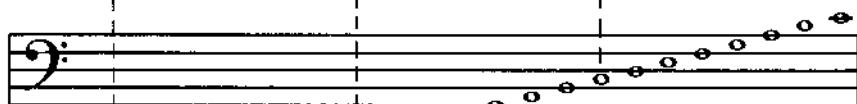
The duration and pitch of those first three notes was 0.5 (or half a second long) and 0 which is the equivalent to 'middle C' on a piano keyboard. Higher notes are created by making **p** a positive number up to 71 (which is the sort of high frequency that your dog will appreciate), and lower notes require a negative value all the way down to -60 (which can cause a small earthquake given enough amplification!) **BEEPs** can be as short as you like but the maximum duration is 16 seconds. Try this:

□ **FOR p=-60 TO 71: BEEP .3,p: NEXT p**

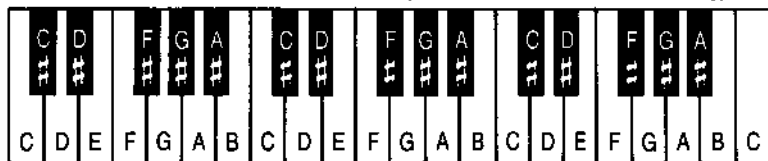
The following diagram shows all the **BEEP** pitch values that are equivalent to the black and white musical notes on a grand piano, along with their musical notation.

## BEEP PITCH VALUES

BEEP PITCH -35 -33 -30 -28 -26 -23 -21 -18 -16 -14 -11 -9 -6 -4 -2 MIDDLE  
 PITCH -36 -34 -32 -31 -29 -27 -25 -24 -22 -20 -19 -17 -15 -13 -12 -10 -8 -7 -5 -3 -1 C



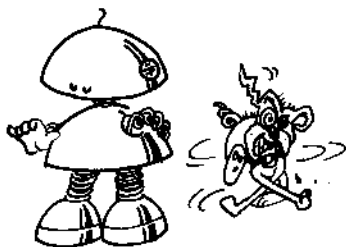
BEEP PITCH 1 3 6 8 10 13 15 18 20 22 25 27 30 32 34 1  
 PITCH 0 2 4 5 7 9 11 12 14 16 17 19 21 23 24 26 28 29 31 33 35 36



MIDDLE  
C

*'I played Mozart  
last night...  
Mozart lost!'*

(Les Dawson)



**Frequencies****Shortcuts**

If you want to use more complex scales for special effects or certain types of oriental music, smaller intervals are allowed such as:

□ **BEEP 1,0.5**

which will give a quartertone above middle C.

The computer has an eight octave output frequency range from 31Hz to 7.81kHz and for specialist notation, using frequencies in cycles per second, frequency can be calculated by typing in the following formula, where **P** equals pitch:

□ **PRINT 440\*2<sup>(P-9)</sup>/12**

There is no need to re-type your musical creations every time you want to fine-tune your computer to other musical instruments, or change the key of anything you have written. For instant key and timing changes, simply set up variables for duration and pitch before typing in your music. Now try this example:

□ **10 LET d=1**

**20 LET p=0**

**30 BEEP d, p+4: BEEP d, p+4: BEEP d,  
p+5: BEEP d, p+7: BEEP d, p+7: BEEP  
d, p+5: BEEP d, p+4: BEEP d, p+2:  
BEEP d,p: BEEP d,p: BEEP d,p+2: BEEP  
d, p+4: BEEP d, p+4: BEEP d, p+2:  
BEEP d, p+2**

If **p** has a value of 0, that piece of Beethoven will play in the key of C-major, but by **LET**ting **p=7** for example, the entire piece is transposed to the key of G. The bad news is that the 'Ode to Joy' sounds more like a funeral march because it is much too slow. The good news is that the tempo can be changed just as easily as pitch by **LET**ting **d=0.5** for example.





You can also use **d** as a shorthand method for writing exact note values, no matter what tempo you select. So if **d** represents a whole note taking up four beats to the bar, then:

**d** =  four beats

**d/2** =  two beats

**d/4** =  one beat

**d/8** =  one half beat, and so on.

To create silent notes or 'rests' in your music, the **PAUSE** command is used, just as in graphic **PAUSES**.

Because there are six separate frequency generators, musical power chords can be produced and two chords can overlap. Each frequency generator can produce 256 tones in every octave so that different musical instruments and special effects can be synthesised. The 'FLASH' demo cassette provides an excellent introduction to the audio aspects of your computer, and the Technical Manual contains full details on how to address and control the machine's sound chip, as well as writing data bytes to the **MIDI** ports.

## The wonders of MIDI

*Music is best understood by children and animals"*

(Igor Stravinsky)

**MIDI** stands for Musical Instrument Digital Interface and you can control a whole electronic orchestra, just by pumping information through the **MIDI** sockets at the back of your computer. Because **MIDI** is a world standard, there are thousands of manufacturers producing different lumps of musical kit that can all synchronise and play together through the computer. The SAM Coupé handles this data at the international standard rate of 31.25Kbaud, (31,250 bits of information every second) receiving it through the **MIDI-IN** port and transmitting it via its **MIDI-OUT** socket.

So the computer not only creates its own music, but thanks to **MIDI** you can use it to memorise, synchronise, learn, teach and control the musical wonders of piano keyboards, synthesisers, samplers, rhythm machines, special effects processors, audio-video units, real-time music clocks and dedicated sequencers. Of course, the SAM Coupé will also perform as any one of these items itself!

The next Chapter explains how to use the sound chip to create sound effects, and the use of volume and stereo.

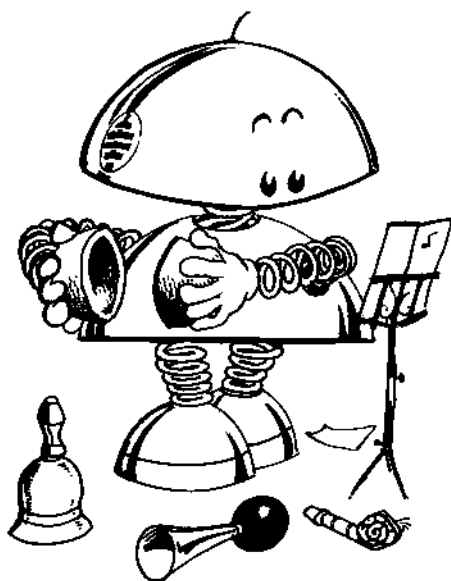
*"Is this plugged  
in?"*

(Bruce Gordon)

# Chapter 8

## SOUND EFFECTS

- sound effects in programs
- **ZAP, POW, ZOOM, BOOM**
- stereo and multi-channels



*'Noise is the most  
impertinent of  
all forms of  
interruption.'*

(Schopenhauer)

**Pre-set sounds**

*"Awopbop-  
aloobop  
Alopbam-  
boom!"*

(Little Richard)

The SAM Coupé is capable of producing wonderful sound effects which can enhance computer programs in any way you choose. They can create atmosphere, act as markers, add realism, soothe, shock and bring comic relief.

Sound effects often use mixed frequencies. Their manipulation is not easy to understand, but to start you off we have pre-programmed some special comic-strip effects that can be summoned and used in your programs simply by calling up their names, just as if they were ordinary commands.

**ZAP**

is exactly what it sounds like, the noise of a belligerent laser beam.

**POW**

provides an impact effect, a bit like dry flesh hitting wet fish.

**ZOOM**

can be used to illustrate fast movement of anything from a jet plane to a skateboard.

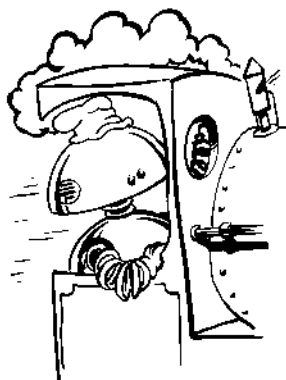
**BOOM**

is what happens when bombs, stock markets and gluttons explode.

Advanced programmers can use the command **SOUND** to send special codes to the sound chip. It routes data bytes **d** to individual sound chip registers **r**, and up to 127 pairs of such numbers can be used, like this:

□ **SOUND r, d; r, d** etc

Full details are to be found in the Technical Manual, but beginners in BASIC are welcome to cheat by copying in the following soundtrack. First, let me



set the scene. All is dark. An old steam locomotive moves from the West towards the dawn. Suddenly, a tourist flying saucer passes overhead, on its way to Alpha Centauri for breakfast. But the crew have forgotten to pick up one of the younger extra-terrestrials, who tries to catch up on her sonic scooter, and fails. Luckily, there is a public telephone nearby, and the wee alien calls home, reversing the charges. The first cuckoo of spring is heard!

- 10 REM sound effects
- 20 GO SUB 3000
- 30 REM steam locomotive
- 40 LET L-7: GO SUB 2000
- 50 LET p-20: GO SUB 1000
- 100 REM flying saucer
- 110 LET L-3: GO SUB 2000
- 120 LET p-20: GO SUB 1000
- 200 REM sonic scooter
- 210 SOUND 17, 4: SOUND 16, 64
- 220 LET p-5: GO SUB 1000:
- PAUSE 200
- 300 REM telephone
- 310 GO SUB 3000
- 320 LET L-7: GO SUB 2000
- 330 FOR n=0 TO 3
- 340 SOUND 2,255: PAUSE 18
- 350 SOUND 2,0: PAUSE 8
- 360 SOUND 2, 255: PAUSE 18
- 370 SOUND 2, 0: PAUSE 80
- 380 NEXT n
- 400 REM cuckoo
- 410 BEEP 0.2, 19: BEEP 0.4, 15
- 420 STOP
- 1000 REM stereo Left to Right
- 1010 LET a=0: LET b=0

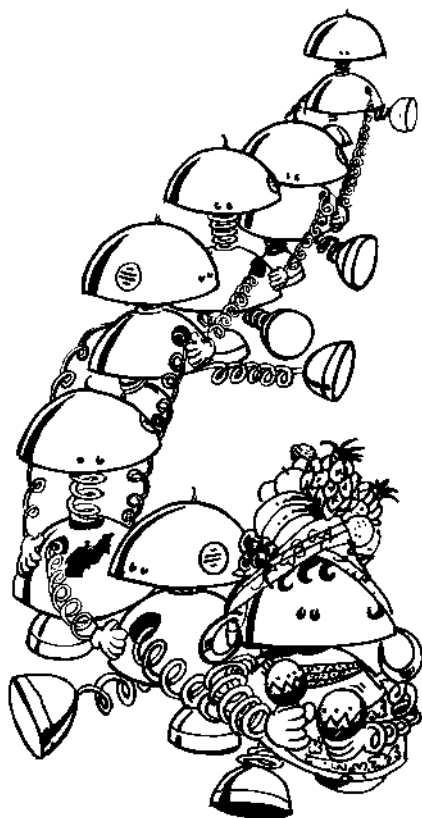
```
1020 LET L-1: LET R-0: GO SUB 1500
1030 LET L-0: LET R-16: GO SUB 1500
1040 LET L- -1: LET R-0: GO SUB 1500
1050 LET L-0: LET R- -16:
GO SUB 1500
1060 RETURN
1500 REM output subroutine
1510 SOUND 2, (a+b): PAUSE p
1520 FOR n-1 TO 15
1530 LET a-a+L: LET b-b+R
1540 SOUND 2, (a+b)
1550 PAUSE p: NEXT n
1560 RETURN
2000 REM output sound data
2010 DATA 9, 64, 16, 16, 17, 0, 24, 138,
28, 1, 21, 4, 20, 0
2020 DATA 9, 255, 17, 1, 20, 4
2030 DATA 24, 142, 20, 4, 10, 111, 17,
6, 9, 128, 16, 32, 28, 1
2040 FOR n-1 TO L
2050 READ a, d: SOUND a, d: NEXT n
2060 RETURN
3000 REM clear the sound
3010 FOR n-0 TO 31: SOUND n, 0:
NEXT n
3020 RETURN
```

The experts among you can detect that stereo and volume are one and the same thing, and that a value of 0 represents silence, all the way up to 16 for maximum noise. By changing volume levels over two channels, the stereo effect is created. See if you can locate which **SOUND** instructions control which effect, and alter their codes.

# Chapter 9

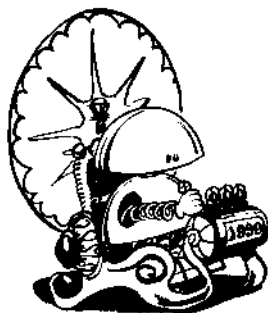
## EXPANDING THE SYSTEM

- the Euroconnector
- using a printer
- **LPRINT** and **LLIST**
- **DUMP**
- the SCART socket
- joysticks
- of mice and men
- light pens



*'Tis pleasant,  
sure, to see one's  
name in print.'*

(Lord Byron)



## EXPANSION CONNECTOR

## SCART SOCKET

One of the main design philosophies behind the SAM Coupé has been to build in as many features as we can that will allow you to expand your system and to upgrade the computer when new products come onto the market. So the machine will not fall out of step with innovations, because it's ready and waiting for the future!

The last two Chapters explained how the computer works with music and sound generators. This Chapter deals with the many other devices that can be linked to the machine, and the following Chapters deal with additional memory and communicating with other computers and networks.

The SAM Coupé's main expansion connector is the 64-pin Euroconnector at the back of the machine. It is capable of sending power to other devices, and handling data, colour and sound signals. The function of each pin is set out in a table that can be referred to in the Appendix. This is the socket that accepts devices such as **PRINTERS**, **SCANNERS** and **VIDEO DIGITISERS**.

This 21-pin socket can handle superb quality video as well as full-stereo audio outputs from the SAM Coupé, which results in better audio-visuals than normal composite signals output from the UHF socket to a television aerial input. With the **SCART** socket, the red, blue and green components of the video display are handled separately and synchronised, and the audio signals are delivered as separate left-hand and right-hand channels. This is particularly useful when using monitors, video digitisers and professional audio-visual recording equipment. The function of each pin is set out in the Appendix.



**JOYSTICK**

One of the simplest and cheapest add-ons for your computer system is used to control movement and action in game-playing. A **JOYSTICK** allows very fast reactions to be achieved by manipulating a short lever, tracker-ball or steering device, which acts in the same way as the cursor controls with the additional facility of an action controller or 'fire' button. Although the joystick port is a standard 9-pin Atari-type, two players can each have their own joystick control because an extra strobe line and power line has been included. This will accept a special **MGT** plug-socket adaptor, for dual joystick control.

**MOUSE**

*'Burn not your  
house to fright  
away the mice.'*  
~ (Thomas Fuller)

This little beast is connected to the mouse hole at the back of the SAM Coupé, and is used to control a screen cursor by sliding it over a non-slip flat surface or special mouse mat. Various on-screen options can also be selected by clicking buttons or tracker-balls built in to the mouse. The position of the mouse coordinates can be read by using the function **XMOUSE** to find its **x** coordinate, and **YMOUSE** to discover the current **y** coordinate. The status of a mouse button from 1 to 3 can be revealed by the function **BUTTON n**. Please see Glossary for details.

**LIGHT PEN**

This device is used to communicate directly with the screen, by physically pinpointing **x** and **y** coordinates with a so-called beam of light. It is especially useful when used with graphic art packages, and often operates with a 'graphics tablet' scratch pad. The light pen coordinates can be read by using the functions **XPEN** and **YPEN** to show the **x** and **y** coordinates.

**Using a Printer**

To use the SAM Coupé as an effective word processor, or simply to **PRINT** out computer programs and

screen images onto paper as 'hard copy', you will need one of the many printers that are commercially available. The most common kind (and usually the cheapest) is called a **DOT MATRIX PRINTER**, which uses a tiny grid of pins to strike a type ribbon and produce a pattern of dots. These patterns are similar to the grids of pixels that make up characters on screen, and they can be as flexible and numerous as anything you create on screen. **DAISY WHEEL PRINTERS** use a ribbed wheel with each 'petal' representing a pair of characters, but you have to change the wheel every time you want a different style of typeface. **LASER PRINTERS** use a laser beam to reproduce character codes sent from the computer onto paper.

Apart from the operating instructions that come with your printer, you will need to know how to use the following statements:

### **LPRINT and LLIST**

The pair of instructions **LPRINT** and **LLIST** act in exactly the same way as **PRINT** and **LIST**, except that they tell the computer to use the printer instead of the television screen.

### **DUMP**

You can use **TAB** in the same way as described in Chapter 6, with **AT** acting like **TAB** with printers.

To print graphic screens onto paper the instruction **DUMP** is used. Supposing that you've created a multicoloured graphic masterpiece that you want to keep on paper, but your printer only outputs in monochrome. **DUMP** deals with certain types of printer to stop complicated screens turning into garbage when they get printed out in black and white. It can be used in any screen mode, and will result in unshaded printed copy like this:

In **MODE1** and **2** all set pixels print out as black.

In **MODE 3** and **4** any colour other than **PAPER** prints out as black.

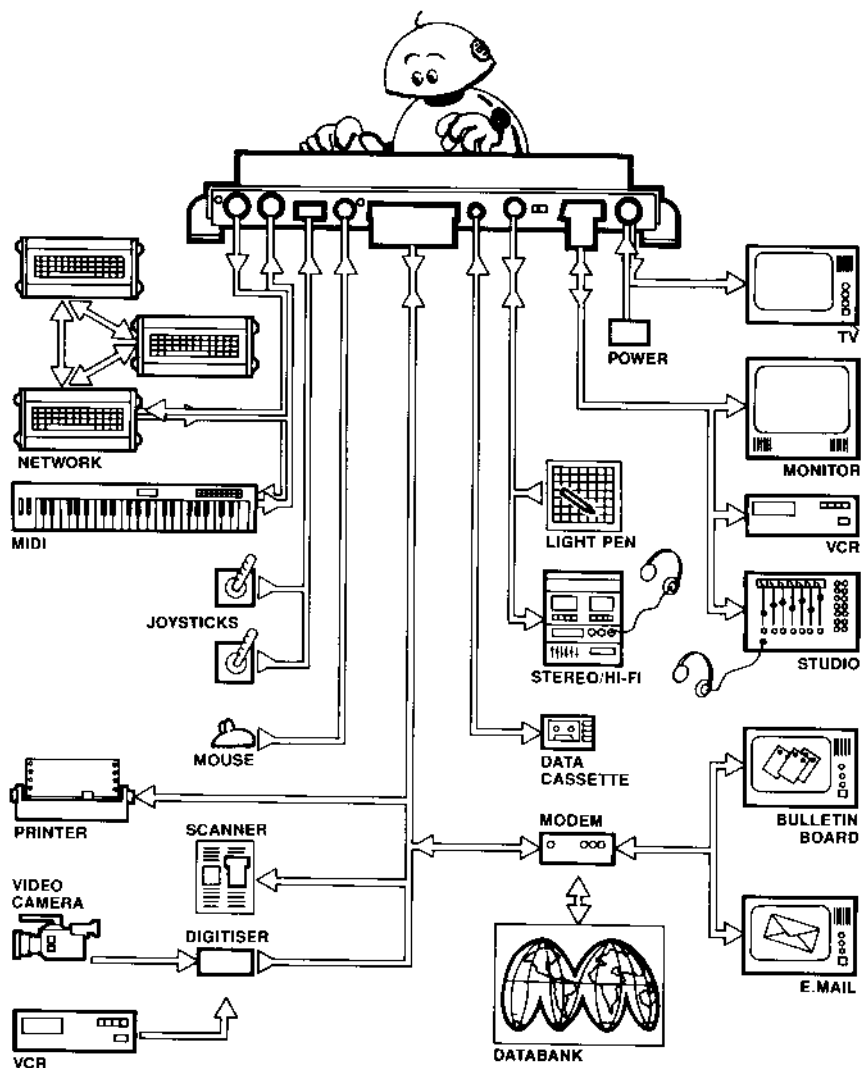
**DUMP** will not work with a listing that gets **CLEARED**. In the event of you changing your mind after the printer has begun to operate, you can rescue things by **BREAKING** into the program.

**DUMP** on its own does a graphics copy of whatever is on screen. Unless you tell it otherwise, this will be at the normal current width and height of the characters. If you want to shovel a screenful of text into your printer, then use the command **DUMP CHR\$**.

Experienced users should also note that **LPRINT** works via **stream 3**, which is normally open to **channel 'p'** (a parallel printer). There is a flexible system for translating characters with codes greater than 128 into sequences that allow alternative character sets to be selected while any of these characters is being **PRINTed**.

SAM Coupé

Ready to Expand



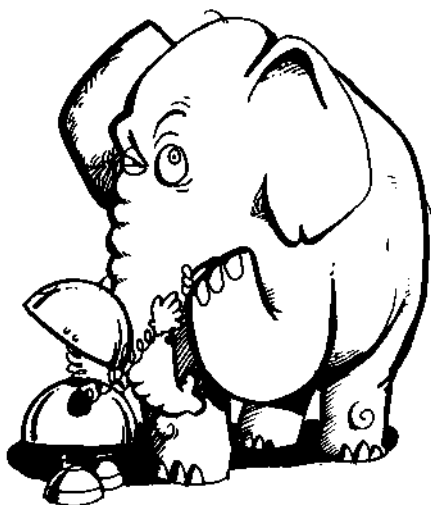
# Chapter 10

## MEMORY

- handling memory
- **FREE**, **RAMTOP** and **MEM\$**
- **OPEN** and **CLOSE**
- machine code
- **PEEKing** and **POKEing**
- **USR**, **CALL** , **CLEAR**
- **IN** and **OUT**
- using a disk drive

*'memory is the  
thing you forget  
with'*

(Alexander Chase)



**Available  
memory**

The more you write programs, the more you'll appreciate that your limitations are not restricted by imagination or creativity but by your computer's memory. So you must make the best possible use of the large but unforgiving memory provided.

Your SAM Coupé has 512 \* 1024 bytes of Dynamic Random Access Memory, with 256K already provided for your use. You can double this on-board memory to the full 512K by plugging in two extra memory chips, available from MGT. The simple fitting instructions come with the package.

**Bits and  
Bytes**

A byte is a unit of storage in memory, which can hold a number between 0 and 255, and each byte has an address which is a number between 0 and 528K. Each byte is made up of eight smaller units of memory called bits, and advanced users can refer to the discussion of machine code later in this Chapter.

**Memory  
gobblers**

*'Truth is the most  
valuable thing  
we have. Let us  
economize it.'*

(Mark Twain)

Complex BASIC routines use a surprisingly small amount of memory, but full colour screens of graphics will gobble up great chunks of the stuff. The largest possible program line can have up to 127 individual statements, consuming as many as 16,127 bytes, and your arrays can fill every aspect of available memory. Strings can be up to 65520 characters long. When the computer recognises a command it only takes the equivalent of one character's storage space, whereas all functions are stored as two characters in memory. Other types of information will occupy varying amounts of space.

**MIDI memory**

For example, each **MIDI** data item is made up of 8 bits (1 byte), plus one bit each for 'start' and 'stop'. Channel voice messages can carry information about the tones being used and the channel number using them, and each message takes up to 3 bytes, plus one bit each for start and stop.

**SCREEN  
memory**

32K of the SAM Coupé's memory is dedicated to the screen. If extra screens are opened, using **OPEN SCREEN**, each one will gobble up another 32K.

**Memory  
functions**

The following functions have been provided to help you handle available memory:

**FREE**

instantly informs you how much free memory is available for use with your **BASIC** program and its variables. As with other functions, just type

**PRINT FREE**

**RAMTOP**

starts at **81919** and tells you the top address available for use by **BASIC**, and is set by using the **CLEAR** command, e.g.

**CLEAR 70000: PRINT RAMTOP**

**MEM\$(n TO m)**

assigns a section of memory to a string. You can search it with **INSTR** and **POKE** it wherever you want. The length of the memory 'slicer' must be less than **65536**, both **n** and **m** must be included, and **n** can be **0**. **MEM\$** gives you the power to move large areas of memory very quickly when used with the SAM Coupé's ability to **POKE** strings. Another valuable use of the **MEM\$** function is to allow rapid searches of memory using **INSTR**, and although you can search a specified area of memory, the searching process is so fast that it may be simpler to search through the entire memory. To find every location of a particular string you could use:

**10 LET adr=1**

**20 LET adr=INSTR(adr, MEM\$(0 TO 65534), a\$)**

**30 IF adr <> 0 THEN PRINT adr: LET  
adr=adr+1: GO TO 20**

### Extra memory pages

You could also search and replace a string by using **POKE**.

Full details of memory locations and arrangements can be found in the Technical manual.

There are two commands that allow you to reserve and free up pages of memory.

#### **OPEN n**

reserves **n** extra memory pages of 16K each for use by BASIC, where possible. This allows **CLEAR** to set a higher **RAMTOP**.

#### **CLOSE n**

frees **n** 16K pages from use by the BASIC program and variables, but only if **RAMTOP** allows it. This is because BASIC insists that it keeps all pages up to and including the one where **RAMTOP** is located. All freed pages can be used for screens, DOS and other programs.

### Machine code

*"I never forget a  
face, but in  
your case I'll  
make an  
exception."*

(Groucho Marx)

### CODE

Large chunks of memory are saved and programs speeded up if **BASIC** keywords and routines can be bypassed, and you communicate directly with the computer's codes. 'Machine Code' is the set of instructions that the SAM Coupé's Z80B microprocessor chip uses, and you can write programs directly to it in 'assembly language'. This type of program has to be coded into a sequence of bytes using an assembler, and there are many ready-made software packages on the market. If you want to code them yourself there are several excellent books to teach you how, and this isn't one of them.

Information can be **SAVED** and **LOAD**ed in bytes by using the keyword **CODE**, without any need to define what the purpose of that information is.





PEEKing

POKEing

For example

- SAVE "name" CODE**
- LOAD "name" CODE** or [F8]

Machine coders will benefit from the following information.

**PEEK addr**

is the function that reveals the contents of the machine's memory at an address in the range from 0 to 528K. It returns a result from 0 to 255

**DPEEK addr**

allows a double **PEEK**, and is the equivalent to **PEEK addr+256\*PEEK (addr+1)**

**POKE addr,x**

is used to store a number directly into an address of memory in the range from 0 to 524287. Up to 32 numbers can be **POKEd** at once into successive addresses, like this:

- POKE addr, 255,129,129, 129, 129, 129,129, 255**

You can **POKE** strings into memory as well as numbers by typing:

- POKE addr, a\$** or, for example, **POKE addr, "testing"**

**DPOKE addr, value**

allows a double **POKE**. The value must be in the range from 0 to 65535.

**USR n**

is used to run machine code, with the given number as the start address. Values from 0 to 52487 can be used.

**USR\$ addr**

gives the string result of a machine code program at

a designated address. Machine code should end with **BC-length, DE-start, A-page of start** (if  $DE \geq 32768$ ).

**CALL n**

can be used to summon up a particular address with a list of numeric or string parameters, if desired. E.g.

□ **CALL, address,x,y,SAM\$**

This is explained in the Technical Manual.

**CLEAR**

is used to free up all of the space in the computer's memory occupied by variables, by **CLEARing** them away.

**CLEAR n**

acts like **CLEAR**, but it alters the system variable **RAMTOP** to a position specified by the number **n**. **N** must be within the 4 pages allocated to BASIC when the computer is switched on, and must be  $\leq 81919$  unless extra pages have been **OPENed**.

**IN and OUT**

The computer has 65536 **INput/OUTput** ports, or I/O ports for short, and these are used to communicate with internal devices such as the keyboard and sound chip, as well as external devices like printers. Each port can be controlled with the function **IN** and the statement **OUT**.

□ **IN address**

reads the byte from the addressed port, and gives that byte as the result.

□ **OUT address, value**

writes a given value (between 0 and 255) to the addressed port (between 0 and 65535)

**Disk drives**

A choice of one or two ultra-slim disk drive units can be slid inside the front panel of the SAM Coupé. These are removeable consoles, and each one can accomodate one megabyte of memory on a 3.5" floppy disk, offering 780K per disk after it has been formatted for use. Disks offer very fast **SAVE**ing and **LOAD**ing of programs, and are the most convenient way of storing large quantities of data. And if this massive capacity isn't enough, then you can connect up to gigantic networks of data as outlined in the next Chapter.

**Using a disk drive**

To take full advantage of the extra speed and memory available with the MGT removeable disk drives, a few more commands have to be learned. First of all the Disk Operating System, DOS for short, must be loaded into memory.

**BOOT**

loads DOS from disk, and allows you to use the special DOS operations:

**FORMAT**, to prepare a new disk for accepting data,

**DIR**, which examines the **DIR**ectory of files on disk,

**MOVE**, which shuffles files around, and

**ERASE**, to wipe out any files you don't need.

**DEVICE**

makes the commands **SAVE**, **LOAD**, **MERGE** and **VERIFY** work with the disk drive, or with a network of several computers. Please refer to the Glossary of this Manual for details, and rest assured that MGT disk drives are accompanied by their own illuminating manual!



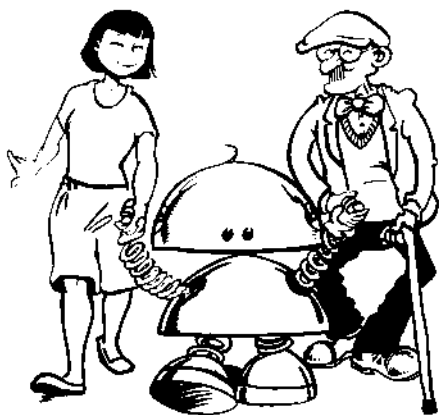
# Chapter 11

## COMMUNICATIONS

- networks
- education
- streams and channels
- piracy is theft

*"Soap and education are not as sudden as a massacre."*

(Mark Twain)

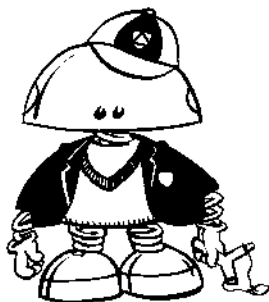


## NETWORKS

One of the most exciting features of your SAM Coupé is its ability to communicate with one or more similar machines. The **MIDI-IN** and **MIDI-OUT** connections at the back of the computer are ready to be used in setting up this network.

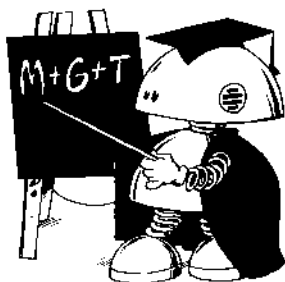
At its simplest level, there is no need for any sophisticated equipment or software, because everything is built in. Cassette-based systems work very well, and you can play multi-user games at two or more different locations. The only limit is your own imagination! More serious networking is achieved by using disk drives and printers, and there are even methods of communicating with other networks all over the world.

## EDUCATION



As a stand-alone machine, the Sam Coupé has been designed to excel as an educational computer. From primary school to university, its colour, sound and add-on facilities offer a complete range of options. Early learners and disabled users can take advantage of the MGT Mouse for simple control, as well as the MGT Light Pen for help with reading. The **CSIZE** command will create instant large-scale characters for users with impaired sight.

As many as 16 machines can be linked up in a classroom network, with obvious advantages when it comes to sharing the resources of disk drives and printers. Data is **LOAD**ed to every computer, and progress of the whole network can be monitored and **SAVE**d from the teacher's machine. Channel #0 is normally the 'broadcast' station in an educational network, with channels #1 to #15 used for the personal operator stations, allocated to the students' computers. There is a wide range of educational software for the SAM Coupé, and MGT are always happy to talk to parents, teachers and of



course younger SAM Coupé users, who want the latest information on educational programs and add-ons.

This next section is a little bit technical, and you can skip over it if you wish.

In a **NETWORK**, Channel #0 is normally the 'broadcast' station, with Channels #1 to #15 used for personal operator channels.

Streams 0 and 1 are normally **OPEN** to Channel 'K'.

□ **PRINT #0; "testing": PAUSE**

will **PRINT** at the bottom of the screen, whereas **PRINT #2** will **PRINT** on the upper part of the screen as usual (using Channel 'S'). **PRINT #3** acts like **LPRINT**, using Channel 'P', and you may choose to use it like this:

□ **OPEN #6; "P": PRINT #6; "hello"**

in order to send output to a printer via Stream #6.

**INPUT #2** is also allowed and can be used to gain access to the upper part of the screen as in the following example, for typing in a value next to a 'value' prompt:

□ **INPUT #2; "value: "; v**

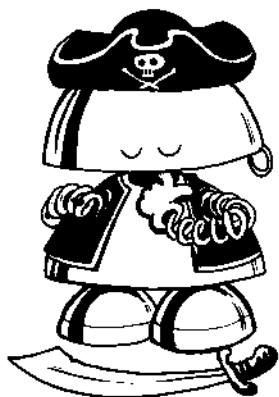
Unlike the lower screen, the line is left uncleared after you press [RETURN].

The **RECORD** command works via Stream 16. Stream 16 prints to the string variable specified by **RECORD**, and it will continue to do so after a **RECORD STOP**.

**PIRACY**

There's nothing clever about theft. If you make a copy of a program as a back-up for your own use, that's sensible, but if you steal by making pirate copies of commercial games, you hurt the writers and manufacturers. Piracy puts up the price of software and it can drive people out of business. If there is less and less original software produced, this harms everyone, including you.

By using special routines, the SAM Coupé's **BREAK** button can be used to transfer cassette-based software to disk, for your own convenience. Please don't abuse this feature and turn it into a 'public convenience'. MGT wants to make it clear that we will provide software producers with all the information we have on how to protect games that run on the SAM Coupé, and we expect help from responsible adults to discourage piracy using our products.

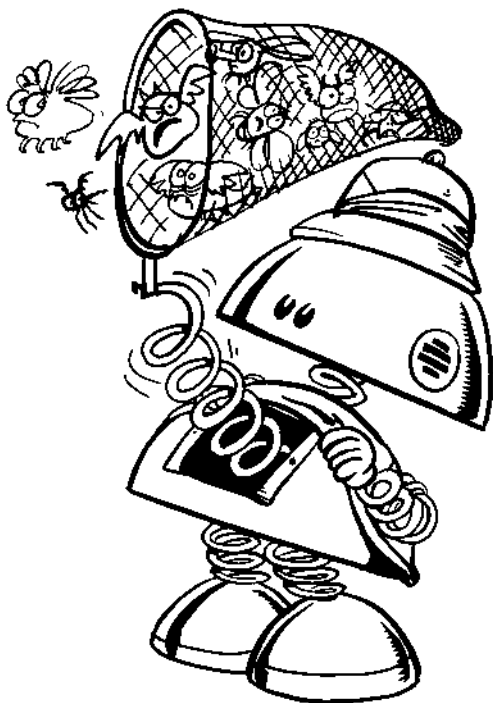




# Chapter 12

## ERROR CODES

- spotting mistakes
- error codes and messages
- using **ON ERROR**



*'A life spent  
making mistakes  
is more useful  
than a life spent  
doing nothing.'*

(George Bernard  
Shaw)



When a mistake is made in programming, or when you ask your computer to do the impossible, the SAM Coupé helps you spot the error and explain the problem. This results in special codes automatically appearing at the bottom of your screen to point out what's gone wrong and where the error is.

When a programming mistake is encountered, helpful messages are given in the following order: one of the code numbers and messages listed below telling you exactly what sort of mistake has been made, followed by the line number where the error is lurking, followed by the number of the statement in that line responsible for the foul-up. For example:

**5 NEXT without FOR, 700:3**

tells you that the type of error is code **5** meaning that you have written a **NEXT** command but forgotten the **FOR** command, and that the problem is at line **700** in statement **3**.

The code you want to see is **0** which means that everything is OK, but even the most experienced programmer can be expected to make mistakes. Here is a list of all the error codes and their error messages, with explanations of what they mean.

ERROR CODE  
NUMBER

ERROR MESSAGE

- |   |   |
|---|---|
| 0 | <b>OK</b><br>No problems, successful completion, everything is OK                                 |
| 1 | <b>Out of memory</b><br>There is not enough room in the computer's memory for what you want to do |

ERROR CODE NUMBER	ERROR MESSAGE
2	<b>'name of variable' not found</b> The computer cannot find a variable, either because it has not yet been loaded, not been assigned or set up, or you have not set its dimensions
3	<b>Data has all been read</b> You are trying to <b>READ</b> past the end of the existing <b>DATA</b> listing.
4	<b>Subscript wrong</b> Either the number of subscripts is wrong or the subscript is outside the dimensions of the array.
5	<b>NEXT without FOR</b> Even though there is an ordinary variable with the same name, the control variable has not yet been set up by a <b>FOR</b> statement.
6	<b>FOR without NEXT</b> Even though there is a <b>FOR</b> loop waiting to run, there is no <b>NEXT</b> statement to go with it.
7	<b>FN without DEF FN</b> A user-defined function is missing.
8	<b>RETURN without GOSUB</b> There is a <b>RETURN</b> statement without <b>GOSUB</b> to welcome it back.
9	<b>Missing LOOP</b> You have forgotten to follow a <b>DO</b> marker with the necessary <b>LOOP</b> .
10	<b>LOOP without DO</b> Even though there is a <b>LOOP</b> waiting to jump back, there is no <b>DO</b> to start it off.

ERROR CODE NUMBER	ERROR MESSAGE
11	<b>No POP data</b> You are trying to use <b>POP</b> , but there is no data on the stack.
12	<b>Missing DEF PROC</b> You want to end a named procedure with <b>END PROC</b> , but it is missing. This error can also result from a simple spelling mistake, such as 'PRUNT'!
13	<b>No END PROC</b> You have forgotten to mark the end of a named procedure.
14	<b>BREAK into program</b> <b>BREAK</b> has been hit or [ESC] pressed, in between two statements, and the line and statement number that are shown refer to the statement before <b>BREAK</b> was used. When you <b>CONTINUE</b> , the program goes to the statement that follows and allows for any program jumps that you have made.
15	<b>BREAK - CONTINUE to repeat</b> <b>BREAK</b> has been hit while a peripheral operation was taking place, so when you <b>CONTINUE</b> the last statement is repeated.
16	<b>STOP statement</b> When you want to <b>CONTINUE</b> after this, the program will start again at the next statement.
17	<b>STOP in INPUT</b> When you want to <b>CONTINUE</b> after this, the program will start again by repeating the last <b>INPUT</b> statement.

ERROR CODE NUMBER	ERROR MESSAGE
18	<b>Invalid file name</b> You are trying to <b>SAVE</b> a file but have forgotten to give it a name, or the name is longer than 10 characters.
19	<b>Loading error</b> The file you want to <b>LOAD</b> has been found but there is something wrong with it and it refuses to <b>LOAD</b> properly or fails to <b>VERIFY</b> . Check your cables, volume level, cassette tape and dirty play-back heads of the cassette player.
20	<b>Invalid device</b> You are trying to <b>SAVE</b> or <b>LOAD</b> data, but you are using the wrong thing for input/output (such as a disk drive instead of a cassette recorder), or have forgotten to plug it in.
21	<b>Invalid stream number</b> You are trying to use a stream number that is inappropriate. Streams <b>0</b> to <b>16</b> are the paths to the various channels. E.g. '5', 'P', 'K', 'N'.
22	<b>End of file</b> The end of a file has been reached, usually a disk file.
23	<b>Invalid colour</b> You have tried to specify a colour with a number that is not appropriate. Colours range from <b>0</b> to <b>127</b> .
24	<b>Invalid palette colour</b> The palette's 'paint pots' range from <b>0</b> to <b>15</b> .
25	<b>Too many palette changes</b> The maximum number of <b>PALETTE</b> changes is <b>127</b> per screen.

ERROR CODE NUMBER	ERROR MESSAGE
26	<b>Parameter error</b> Either you have used the wrong number of arguments, or the wrong type of argument, like a number instead of a string.
27	<b>Invalid argument</b> You are using an argument that is not suitable for the function you want.
28	<b>Number too large</b> Your calculations have resulted in a number that is too enormous for the SAM Coupé to handle.
29	<b>Not understood</b> The computer is confused by your (mis)use of BASIC.
30	<b>Integer out of range</b> A whole number (called an integer) is required, but the argument you are using has been rounded to an integer that is outside of a suitable range.
31	<b>Statement doesn't exist</b> The computer can't make a decision or obey an instruction without the necessary statements. For example, you may have deleted statements after a <b>GO SUB</b> and then <b>[RETURN]</b> ed.
32	<b>Off screen</b> The graphic requirements that you have asked for cannot fit on the screen.
33	<b>No room for line</b> There is not enough room in the available memory for the line you are trying to insert, or the line numbering requested in a <b>RENUM</b> is impossible.

ERROR CODE NUMBER	ERROR MESSAGE
34	<b>Invalid screen mode</b> There are four screen modes, numbered 1 to 4.
35	<b>Invalid BLITZ code</b> You have used <b>BLITZ</b> with a string that does not contain sensible graphics commands.
36	<b>Stored area too big</b> You cannot fit the area you want to <b>GRAB</b> into store.
37	<b>Invalid PUT block</b> The string that you want to <b>PUT</b> on a screen is not a real block of screen data.
38	<b>PUT mask mismatched</b> When creating a graphics mask, the second masking string must be the same length as the original string.
39	<b>Missing END IF</b> You are using a long <b>IF</b> statement over a number of lines and have chosen to omit <b>THEN</b> , but you have forgotten to end the statement with <b>END IF</b> .
40	<b>Invalid variable name</b> Your numeric variable name does not start with a letter, or is longer than 32 characters. Your string and string array variable name does not end with \$, or is longer than 10 characters.
41	<b>BASIC stack full</b> Please see Chapter 10 for dealing with memory stacks.
42	<b>String too long</b> Your string is more than 65520 characters long, or in the case of <b>STRING\$</b> , only 512 characters are allowed.

ERROR CODE NUMBER	ERROR MESSAGE
43	<b>Invalid screen number</b> When the computer is switched on, there is only one screen, numbered 1. You are trying to <b>OPEN</b> a screen with a number outside the range 2 to 16. This error message also appears if you select a screen with the <b>SCREEN</b> or <b>DISPLAY</b> commands that has not been <b>OPENed</b> .
44	<b>Screen is already open</b> You are trying to use <b>OPEN SCREEN</b> with a number that already exists.
45	<b>Stream is already open</b> You want to open a stream which is already occupied.
46	<b>Invalid channel</b> You are attempting to use an inappropriate channel.
47	<b>Stream is not open</b> You are trying to use a stream number that is <b>CLOSEd</b> .
48	<b>Invalid CLEAR address</b> You are trying to <b>CLEAR</b> with a number beyond the limits of memory allocated to BASIC.
49	<b>Invalid note</b> The note you want is too high or too low. You must stick between the range -60 to 71.
50	<b>Note too long</b> A note must be shorter than 16 seconds, so your command must be no greater than <b>BEEP 16</b> .



ERROR CODE  
NUMBER

## ERROR MESSAGE

- 51 FPC error**  
You have made an error in a machine coded floating point calculation.
- 52 Too many definitions**  
May be produced by the use of **DEF KEYCODE** or **DEF FN**.
- 53 No DOS**  
There is no DOS system to **BOOT** from disk, or no disk.
- 54 Invalid WINDOW**  
The borders of your **WINDOW** are not allowed in the current **MODE** and **CSIZE**.



You can lay plans for handling program errors by telling the computer that if an error occurs then it should go to a special error-handling routine. Error trapping is turned on by **ON ERROR** and is turned off by **ON ERROR STOP**. When this routine is turned on, some special variables are created, as follows:

**ERROR** (the error code number, as listed above)  
**LINO** (the line number where the error is located)  
**STAT** (the statement where the error is to be found)

Experienced users can devise their own error handling routines, which must be used within a program, such as:

□ **10 ON ERROR GOTO 100**

which tells the program to **GO TO** line **100** if it detects an error, and then execute any instructions placed there, or **GO SUB** a subroutine. **ON ERROR** is automatically turned off while you try to correct

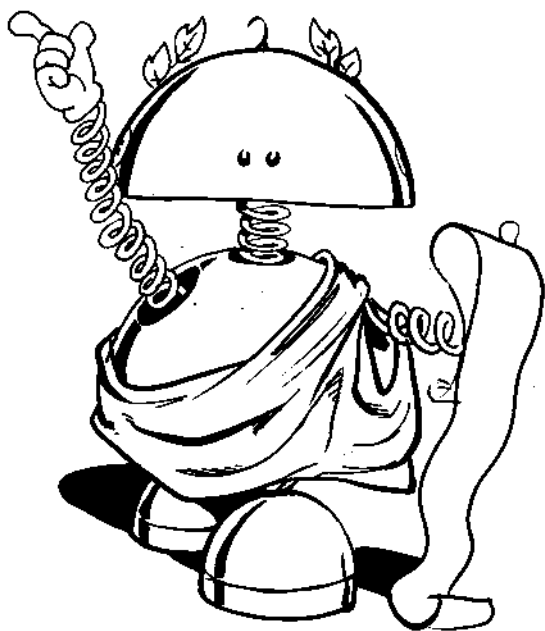
the error, so if there is another mistake within the error-handling routine it will be handled normally. Subroutines and procedures will return to the statement that comes after the rogue that caused the error, when they **RETURN** or **END PROC**. Now **ON ERROR** will activate itself once again, and go hunting for the next mistake.

You can use **CONTINUE** to go back to the statement that caused the error, without re-enabling **ON ERROR**, but if you **CONTINUE** while the error is **'BREAK'**, then you will go back to the next statement. To keep the Basic stack correct, you can use **POP** to get rid of a **RETURN** or **END PROC** address, should you decide to **CONTINUE** instead of **RETURNING**.

*"Intelligence isn't  
to make no  
mistakes, but to  
see how to make  
them good."*

(Bertolt Brecht)

# GLOSSARY



*"I like short  
words and  
vulgar fractions"*  
(Winston Churchill)

# Glossary

---

This section of the User Manual is your reference guide to all the words and expressions you are likely to need when programming the SAM Coupé. It includes commands, functions, keywords and definitions of technical terms and jargon. The easiest way to understand what each of these paragraphs means is simply to try out the techniques and see what happens. You can't harm your computer by experimenting.

Keywords that can be used in BASIC programs are printed in bold upper case letters only, such as **AND**.

Definitions and jargon are shown in upper and lower case letters, for example, **Array**.

Actual keys that can be found on the computer keyboard are represented as upper case letters enclosed by square brackets, like this **[RETURN]**.

## **ABS**

stands for **ABS**olute magnitude, and is a function for converting an argument into a positive number.

## **ACS**

stands for arccosine, and is a function for finding out the value of a number that has a given cosine.

## **Address**

is the numeric location of a unit of data in the computer's memory.

## **AND**

is used to combine conditions just like in English language, such as:

```
□ IF a$ = "sam" AND x > 0 THEN PRINT x
```

## **ARRAY**

An array is a set of variables which are only distinguished from one another by a number written in brackets after the name of the array. Before an array can be used, space must be made for it in the computer's memory by using a dimension statement, the command for which is **DIM**.

# Glossary

---

## **ASCII**

stands for American Standard Codes for Information Interchange, and is the widely-used system of character codes for transferring data between computers and various peripherals such as printers.

## **ASN**

stands for arcsine, and is the function for finding out the value of a number that has a given sine.

## **AT**

tells the computer where to **PRINT** anywhere on the screen, when qualified by a line number, followed by a column number.

## **ATN**

stands for arctangent, and is a function for finding out the value of a number that has a given tangent.

## **Attributes**

of a character or graphic block at any given position consist of its colour, brightness and in **MODEs** 1 or 2 its flash condition.

## **AUTO**

automatically numbers your program lines. If you enter **AUTO** by itself, the computer will take the current line number where the program cursor is, and add ten at every new line. It will not work if the current line number is less than 10 or more than 61439. You can skip over a block of line numbers while using **AUTO** by deleting the line number provided and typing in your own line number, followed by the rest of the line. **AUTO** will carry on from there, numbering in steps of ten. You can choose your own step values if ten doesn't suit you.

**AUTO**                    number after here in steps of 10

**AUTO 150**            number from line 150 in steps of 10

**AUTO 150, 5**        number from line 150 in steps of 5

## **BASIC**

stands for Beginners All-purpose Symbolic Code, and is the single most used computer language in the world. An advanced version of BASIC is built into the memory of your SAM Coupé.

# Glossary

---

## **BEEP duration,pitch**

is the instruction we use to make the computer produce sound.

## **BIN**

is short for binary, and is not a function or a command but more like a gear change, acting as an alternative notation for numbers. **BIN** is followed by a sequence of up to 16 digits made up of 1s and 0s, each different sequence representing a particular number.

## **BIN\$**

automatically gives the binary equivalent of a number as an eight character string if the number is less than 256, or as a sixteen character string if the number is between 256 and 65535. The following functions are also available, **n BAND m**, and **n BOR m**, which give binary **AND**, **OR**.

## **Bit**

is the smallest piece of data which can be represented in a computer's memory by **1** or **0**.

## **BLITZ a\$**

is used to execute a string of graphics commands very quickly. It works in all modes, but is especially effective in mode 4. **BLITZ** responds to the special graphics scaling variables **XOS** and **YOS**, which enable a graphic shape to be **BLITZ**ed anywhere on screen, and the size of the **BLITZ**ed shape can be altered by changing the special graphic variables **XRG** and **YRG**.

## **BLOCKS**

turns the block graphic characters on and off. **BLOCKS 0** turns the block graphics off and allows the pre-defined foreign character set **CHR\$ 128** to **143** to act as user-defined graphics (UDGs). **BLOCKS 1** turns the block graphics back on. Luckily UDGs don't get forgotten and you can call them up again by using **BGRAPHICS 0**.

## **BOOT**

loads the Disk Operating System (DOS) from the disk into the computer's memory.

# Glossary

---

## **BORDER**

changes the colour of the screen border to any colour in your palette from 0 to 15. For example, **BORDER 6** changes the screen border to yellow.

## **[BREAK]**

when pressed, this button at the back of the machine will interrupt the program. Programmers can use software techniques to change the function of this interruption.

## **BRIGHT**

is used to make a chosen colour more brilliant. **BRIGHT 1** turns the instruction on, **BRIGHT 0** sets the colour back to normal. There is a special setting **BRIGHT 16**, which means 'transparent': in other words, the colour on the screen is to be left unchanged when a character is printed there.

## **Bug**

is a slang expression that refers to a mistake in a computer program, which causes problems when the program is **RUN**.

## **Buffer**

is an area of memory set aside for temporary storage of data.

## **BUTTON n**

shows the status of the **MOUSE** button, with **n** being the number of the button from 1 to 3. When the specified button is pressed, 1 will be displayed, or 0 if it is not pressed. If you make **n=0** then 1 is displayed if any or all buttons are pressed.

## **Byte**

is a unit of memory made up of 8 bits, and is large enough to store one character or a whole number  $\leq 255$ .

## **CALL n**

summons up a machine code program at an address with a list of numeric or string parameters if desired. Please refer to the Technical Manual.

## **[CAPS] LOCK**

the **[CAPS] LOCK** key causes all key entries to appear in upper case, until it is disengaged by pressing it again.

# Glossary

---

## Character set

There are 256 possible characters, each with a code between 0 and 255. The pre-defined character set can be printed out by:

```
□ 10 FOR a = 32 TO 168: PRINT CHR$ a ; : NEXT a
```

## CHR\$

is applied to a number and it gives the single character string whose code is that number.

## CIRCLE x,y,r

draws a circle on the screen, when it is followed by the position of the centre of the circle on screen (the **x**, **y** coordinates) and the radius of the circle. If the circle goes far enough off the top or bottom of the screen, it will wrap around to the opposite edge of the screen.

## CLEAR

gets rid of all variables and frees up the space in the computer's memory which they occupied.

## CLEAR n

acts like **CLEAR**, but alters the system variable **RAMTOP** to the point specified by the number..

## CLOSE

is used to shut off streams, which are normally **OPEN** to specified channels.

## CLOSE n

frees up **n** pages of 16k each which are normally used by the BASIC program and variables, if **RAMTOP** allows this. After being **CLOSED**, the freed pages can be used for screens, DOS or other programs.

## CLOSE SCREEN n

releases a specified screen memory if it already exists, so that its memory can be used again, e.g.

```
□ CLOSE SCREEN 2
```



# Glossary

---

## **CLS**

stands for 'clear screen', and wipes off the display file from the screen. **CLS** or **CLS 0** clears the entire screen, whereas **CLS 1** clears the current **WINDOW** only, and **CLS#** not only clears the screen, but executes **OVER 0: INVERSE 0: FLASH 0: PAPER 0: PEN 7: BORDER 0: PALETTE**

## **CODE**

is applied to a string and it gives the code of the first character in the string. If the string is empty, it gives **0**.

## **Command**

is a word or short phrase like **OPEN SCREEN**, **DRAW** and **GO SUB**, used in a program to instruct the computer to perform a specific task.

## **Condition**

A condition is something that the program has to decide to be true or false before making a decision, and can be used after **EXIT IF**, **WHILE** or **UNTIL**. A condition always gives a numeric result, **1** for true or **0** for false (try **PRINTing 2=1**).

## **CONTINUE**

instructs the program to begin again after it has been stopped by an error report.

## **Control Codes**

are special characters that do not appear on the screen, but have actions when **PRINTed** or **PLOTted**. When certain codes are called up from the keyboard, the computer will search to see if there is an expanded definition for them. See **DEF KEYCODE**.

## **COS**

when followed by an angle in radians, is the function that calculates the cosine.

## **CPS**

stands for Characters Per Second, and refers to the theoretical top speed that a printer can **LPRINT** on paper.

# Glossary

---

## Crash

is the slang expression for a computer program blowing its own brains out, and getting splattered beyond rescue.

## CSIZE width,height

lets you change the shape of your characters by squashing them down or stretching them out. For example, in mode 3

□ · **CSIZE 6 , 9**

will make all your characters inhabit cells six pixels wide and nine pixels high. The width can be 6 or 8, and the height can be from 6 to 32 pixels.

## [CTRL]

the [CTRL] key is used to change the keyboard into the **CONTROL** status.

## Cursors

are position indicators, usually shown on screen in the form of an inverse symbol. The **EDIT** line cursor is represented by an inverse \* in the normal keyboard state, and by an inverse + when [CAPS] lock is engaged. The current line program cursor is represented by an inverse > symbol.

## Cursor keys

are the four directional arrow keys used to move the program cursor around the screen, as well as to control movement during game-play.

## DATA

**DATA** statements can include keywords like "go to" without them being changed into tokens. Other than this, words without quotes must be legal numeric variables. Thus **Sam 1** is acceptable, but **1x** is not.

## Debug

is the slang expression for tracking down programming mistakes, or 'bugs', and correcting them.

## DEFAULT

is used mainly with procedures, and creates a variable only if it does not already exist.

# Glossary

---

## **DEF FN**

is short for user-defined function. Function names can have any length and must start with a letter. They can continue with letters, numbers or underscore. String functions must end with \$. The location of the **DEF FN** statement has no effect on the execution speed of the program.

## **DEF KEYCODE**

is used to create shorthand codes which will be automatically expanded when they are called up from the keyboard. For example:

❑ **DEF KEYCODE 195: PRINT 123: PRINT "sam" [ENTER]**

will assign 'PRINT 123' etc to code 195, the code assigned to key [F3]. So whenever you press [F3] it will have the effect of **PRINTing**

**123**

**sam**

on the upper screen area. If you add a final colon to the **DEF KEYCODE** line, the automatic **[RETURN]** is suppressed, so

❑ **DEF KEYCODE 195: PRINT "sam":**

has the effect of **PRINTing "sam"** in the editing area. Instead of a line, you can also use this technique with a string, such as:

❑ **DEF KEYCODE 195, "TESTING:"**

A defined code can be cleared by omitting the expansion or whatever was between quotes as in either of the following examples:

❑ **DEF KEYCODE 195:**

❑ **DEF KEYCODE 195, " "**

## **DEF PROC**

is a statement that precedes the **DEFinition** of a **PROCedure** which has a name and a list of any variables that are being used to pass data to the procedure.

## **[DELETE]**

The **[DELETE]** key deletes to the left when pressed. **[SHIFT]** and **[DELETE]** pressed together will delete to the right.

# Glossary

---

## **DELETE n TO m**

deletes a block of program lines between a pair of chosen line numbers. If the line numbers are the same, a single line is deleted. If **n** is omitted, line number 1 is used. If **m** is omitted, the last line of the program is used.

## **DEVICE**

lets you use the commands **SAVE**, **LOAD**, **MERGE** and **VERIFY** with a disk drive or network of computers. You can use **DEVICE** in the following ways:

**DEVICE d** which tells the computer to use Disk Drive 1

**DEVICE d1** which does the same thing

**DEVICE D1** which also does the same thing

**DEVICE N5** use Network Station 5

**DEVICE T** use tape running at standard speed to **SAVE** data

**DEVICE T35** use tape running at fast speed to **SAVE** data

**35** is the top speed likely to prove feasible for saving data on to tape, **112** is a standard, relatively slow speed. Unfortunately, fast speeds are less reliable than slow ones. There is no need to remember at what speed you saved your data, because the **LOAD** command automatically copes with the full range of speeds.

## **DIM**

makes space for an array in the computer's memory. The name of the array can be up to 10 characters long, excluding spaces. The numbers that make up the **DIM**ensions follow in brackets, and they can be between 1 and 65535. The total size of the array is only limited by the available memory.

## **DIR**

only works with disks.

## **Disk**

is a magnetic device for the rapid **SAVE**ing, storing and **LOAD**ing of computer data. The SAM Coupé can use 3 1/2-inch 'floppy disks'.

## **Disk drive**

The SAM Coupe is fitted with two sockets for custom-made extra memory disk drives. Each drive can contain an additional 256K DRAM.

# Glossary

---

## **DISPLAY n**

alters which screen is being **DISPLAY**ed, but not the screen used by BASIC. One screen can be building up invisibly while another is currently **DISPLAY**ed.

**DISPLAY** or **DISPLAY 0** shows the current screen

**DISPLAY 1** to **DISPLAY 16** shows the required screen number.

## **DIV**

is the equivalent of dividing one integer by another, so that

□ **PRINT 241 DIV 24** results in 10

## **DO**

**DO** acts as a marker to which a matching **LOOP** statement can return. So:

□ **10 DO**

**20 PRINT "PLAY IT AGAIN SAM";**

**30 LOOP**

will go round in circles for ever, so it is useful to qualify a **DO** command. You can use **DO WHILE**, which will result in the part of the program between **DO** and **LOOP** being executed **WHILE** a specified condition is true. Alternatively, you can use **DO UNTIL**, which is the opposite condition, instructing the program to execute that part between the **DO** and **LOOP** while a condition is false, but to stop as soon as it becomes true.

## **DOS**

is the abbreviation for **D**isc **O**perating **S**ystem.

## **DPEEK address**

a double **PEEK** which is the equivalent to **PEEK address+256\*PEEK (address+1)**. See **DPOKE**.

## **DPOKE address**

executes a double **POKE** which is the equivalent to:

**POKE address, number - INT (number/256)\*256**

**POKE address + 1, INT (number/256)**

In other words, the least significant byte of the number is **POKE**d to the address, and the most significant byte is **POKE**d to the next higher address. The value must be between 0 and 65535.

# Glossary

---

## **DRAM**

is an abbreviation of **D**ynamic **R**andom **A**ccess **M**emory.

## **DRAW x, y**

is the command for drawing a line relative to the current position, by moving  $x$  pixels to the right and  $y$  pixels up. If  $x$  or  $y$  has a negative value then movement becomes leftwards or downwards.

## **DRAW x, y, z**

draws a relative line from the current position to the coordinates  $x, y$  while creating a curve by turning through the angle  $z$ .

## **DRAW TO x, y**

is the command for drawing an absolute line from the current position to the point  $x, y$ . You can use **PAPER, PEN, OVER** etc. after **DRAW TO**.

## **DRAW TO x, y, z**

draws an absolute line from the current position to the coordinates  $x, y$  while creating a curve by turning through the angle  $z$ .

## **DUMP**

is used with certain types of printer to stop complicated screens turning into a mess when they get printed out in black and white. **DUMP** can be used in any screen **MODE**, and will result in unshaded printed copy. Note that in **MODE 3** the result will be twice as wide as other **MODES**, because the screen employs 512 pixels instead of 256 horizontally, and that most screen **DUMPS** in this **MODE** won't fit on little printers!

## **DUMP CHR\$**

executes a text screen **DUMP** of the current screen/window.

## **[EDIT]**

If you press the **[EDIT]** key, the program line containing the  $\blacktriangleright$  cursor appears in the editing area. If you type a line number and then press **[EDIT]**, the desired line appears for editing. The  $\blacktriangleright$  cursor is moved left, right, up or down in the edited line by using the cursor keys, or up and down the listing.

# Glossary

---

## **ELSE**

is part of an **IF - THEN** structure, and must come at the beginning of a new statement. Normally, when the statement following **IF** is false, the program will jump to the next line. But if the **IF - THEN** pair has an associated **ELSE** later in the line, the program will continue with the statements following the **ELSE**. On the other hand, if the condition after the **IF** is true, the line will only be executed up to the **ELSE**.

## **ELSE IF**

has a special meaning so that once any condition is true, the lines or statements after it are executed, and when another **ELSE IF** or **ELSE** is encountered, the program jumps straight to a position immediately after **END IF**.

## **END IF**

must be used to end an **IF** condition.

## **END PROC**

is used to mark the end of a **PROC**edure already defined by **DEF PROC**.

## **ERASE**

only works with disks.

## **[ESC]ape key**

when pressed, allows you to **BREAK** into the program.

## **EXIT IF**

is part of the **DO - LOOP** structure, and is used to leave the structure from somewhere in the middle rather than at the **DO** or **LOOP**. If the specified condition after the **EXIT IF** is true, program execution jumps to the statement following **LOOP**, otherwise nothing happens.

## **EXP**

is a mathematical function to calculate exponential numbers, and can be defined by  $EXP\ x = e^{\uparrow}x$

## **[F] keys**

are the group of function keys, labelled **[F0]** to **[F9]**. Each of these keys has been pre-set to provide a specific function as shown in Chapter 4, allowing single-key entry during programming, and may be redefined.

# Glossary

---

## **FATPIX**

In graphics mode 3, the pixels are 'thin', only half the width used by the other modes. By using the command **FATPIX 1** you can double the width of **MODE 3** pixels, making vertical lines the same thickness as horizontal lines, and **XRG** will be halved. To get back to the normal 'thin' state of affairs use **FATPIX 0**, and **XRG** is doubled.

## **File**

is computerised data that can be **SAVED** and act like the magnetic equivalent to a paper document, self-contained and in its own named binder.

## **FILL**

is a graphic command for **FILLING** a shape with a colour. See Chapter 6. `

## **FLASH**

works in modes 1 and 2 only, and is an attribute of a given colour making it **FLASH** rhythmically.

**FLASH 1** turns it on. **FLASH 0** turns it off. **FLASH 8** or **16** sets a 'transparent' state that is left unchanged at any character position.

## **Font**

describes the style and appearance of the letters, numbers and symbols as they appear when **PRINTed** on screen or onto paper. Character fonts can be changed as described in Chapter 6.

## **FOR**

is a command which is always used in conjunction with **NEXT**, and it is used to cut down repetitive programming. For example, instead of entering a hundred lines for a hundred similar calculations, you can shortcut the procedure by writing:

□ **FOR times-1 TO 100: NEXT times**

## **FORMAT**

only works with disks.

## **FREE**

is the function that shows you exactly how much free memory you have left for **BASIC** programming and variables.



# Glossary

---

## Functions

work on numerical values (called arguments) in order to give another value (called the result), and are used by typing in the name of the function followed by the argument. Function names in SAM BASIC can have any length.

### GET

is a way of reading the keyboard without the use of [ENTER]. It is like INKEY\$, except that GET waits for a key to be pressed before continuing. When used with a string variable GET acts like a type-writer, so that:

□ **10 DO: GET a\$: PRINT a\$; LOOP**

prints keys as you hit them, and you can [SHIFT] between upper and lower case as normal. If GET is used with a numeric variable such as GET x, then the variable will equal 1 if [1] is pressed, up to 9 if [9] is pressed. [A] or [a] has a value of 10, [B] or [b] 11, and so on. GET is very useful in menu-driven programs (see ON)

### GO SUB n

is the statement that tells the program to GO to a numbered SUBroutine, up to 65279. It must be qualified by a RETURN statement.

### GO TO n

instructs the program to GO TO any line number, up to 65279. GO TO can be used with LABEL, for example:

□ **GO TO blazes**

Please see LABEL.

### GRAB a\$, x, y, w, L

stores a screen area to a string, which can be PUT somewhere else. The screen area to be GRABbed is defined by x,y as the top left-hand corner, followed by w for its width and L for its length in pixels. The x coordinate for 'fat' pixels is rounded down to an even value and the y coordinate is rounded up to an even number.. For mode 3's 'thin' pixels, the x and y values are rounded to a multiple of 4. GRAB and PUT only work in modes 3 and 4.

### Graphics coordinates

are normally 173 at the top to -18 at the bottom of the screen. Two lines of 9-pixel high characters are used for the editing area, with point 0,0 above them on the left of the screen. By selecting MODE 1, character height changes

# Glossary

---

to 8 pixels, and the vertical y-axis runs from 175 to -16. The horizontal x-axis runs from 0 to 255, except in **MODE 3**'s 'thin' pixels of 0 to 511. The coordinate system will change if you alter **CSIZE** directly.

## **HEX\$**

converts the human counting system based on ten fingers (decimals) to a form of computer binary counting, based on the digits 0 to 9 plus the letters A to F (hexadecimals), e.g.

□ **HEX\$ 10-"A", HEX\$ 32768-"8000", HEX\$ 100000-"0186A0"**

HEX numbers can be introduced by **&**, for example:

□ **&0A-10, &8000-32768**

## **Icon**

is a small graphic likeness of an object, concept, technique or message, like the □ symbol used in this Manual to represent 'type in the following data'.

## **IF**

is a conditional instruction, which is qualified by **THEN**. A 'long' **IF** that works over several lines is achieved by omitting the **THEN** and terminating the **IF** with **END IF**.

## **IN**

is a function in **BASIC** which acts a bit like **PEEK**. When followed by an address, **IN** reads the Input/Output port address and its result is a byte read from that port. **OUT** acts in a similar way, like **POKE**.

## **INK**

Please read the section titled **PEN**.

## **INKEY\$**

is the function that reads the keyboard to see which key, if any is being pressed.

## **INPUT**

Normally, what gets typed during **INPUT** is shown in the bottom two lines of the screen, but other streams can be used, e.g.

□ **INPUT #2; a\$** or **INPUT #2; "number"; x**

allows use of the upper screen.

# Glossary

---

## **INSTR (n,a\$,b\$)**

is a function that searches one string **a\$** for a target string **b\$**, starting from position **n**, and returns its position if found. If it fails to find the target string it returns **0**. **INSTR (A\$,B\$)** starts from position **1**, and you can use **#** as a 'match anything' character.

## **INT**

is short for **INTEger** part, and this function converts a fractional number into a whole number, known as an integer. It always rounds the fraction down.

## **[INV]**

is the key used to highlight screen characters, by throwing them into a 'negative' display. Pressing **[SYM][INV]** returns the display to its normal state.

## **INVERSE**

changes the way characters and graphics are displayed on the screen. **INVERSE 1** turns all new characters and graphics into a sort of negative of themselves, and swaps over their **PAPER** colour with their **PEN** colour. **INVERSE 0** changes them back to normal.

## **I/O ports**

is short for Input/Output ports, and these are used by the processor for communicating with the keyboard, printers, other computers, etc.

## **ITEM**

is a function that gives information about the next **ITEM** to be **READ** in the current **DATA** statement. **ITEM** has the following values and meanings:

**0** means there are no more **ITEMs** in the current **DATA** statement.

**1** means that the next **ITEM** is a string type.

**2** means that the next **ITEM** is a numeric type.

When used with procedures, **ITEM** can reveal the nature of the first **ITEM** in a **DATA** list, passed to the procedure.

## **K**

is the abbreviation for 'kilo' that is inaccurately used to represent 1,024 bytes of computer memory. So, for example, **128k** represents 131, 072 bytes.

# Glossary

---

## **KEY posn,x**

makes a **KEY** in a specified position in the key map produce a chosen character when it is pressed. **KEY** positions must be between **0** and **279**, and **x** must be between **0** and **255**. Codes **192** to **254** can be given expanded definitions, please see **DEF KEYCODE**. The key map is in the Appendix.

## **KEYIN a\$**

enters a string as if you had typed it in yourself, so that it allows programs to write themselves! For example:

```
□ KEYIN "100 DATA "+a$+", "+b$
```

will add a new line to the program after this is **RUN**.

## **Keyword**

A keyword is a word which can have a simple meaning in English, but is recognised by the computer as having a special meaning and treated as a command to do something very precise. When a keyword is recognised, it is converted to capital letters in the program listing.

## **LABEL**

can only be used at the start of a line, but preceding colour control codes or spaces are allowed. When any part of the program is run by **RUN**, **GO TO**, **GO SUB** etc., variables are created with the name given after any **LABELs** in the program, and their values are set to the line number with that **LABEL**. The name after **LABEL** must be a legal numeric variable name. These variables are recreated when **CLEAR** is used, or the program is **EDITed**.

## **Language**

is simply a collection of related commands that allow a computer to perform useful tasks. The most common language is **BASIC**, and an advanced version of this is built into the SAM Coupé's memory to make programming as easy as possible.

## **LEN**

is a function used to calculate the **LENGth** of a string.

## **LENGTH (d,array name)**

is used to tell you the size of an array. If **n=1**, the first dimension is returned,

# Glossary

---

and if **n=2**, the second dimension is returned. **LENGTH** can also be used to find the location of a string or array in memory by using for example:

□ **LENGTH (0,A\$)**

which gives the address of the first byte in the string or array data area. This is useful for telling **CALL** where the location is when you want to modify the data. For numeric arrays, you have to use brackets after the name e.g.

□ **LENGTH (1, x ( ) )**

## **LET**

is used to give a variable a value, such as:

□ **LET x=1**

The SAM Coupé allows long string and array names, and several **LETs** can be made at once.

## **Light pen**

When plugged into the correct port at the back of the SAM Coupé, this device can be used to communicate directly with a computer program by touching the screen. The **x** and **y** coordinates of the light pen may be read by using the functions **XPEN** and **YPEN**.

## **Line numbers**

All lines entered into a computer program must begin with their own number so that they can be stored away in order to be acted on when the program is **RUN**. It is usual to number lines in tens in case you want to add extra lines between the existing ones. You can tidy up messy line numbering with the **RENUMBER** facility, or the automatic **AUTO** feature, and line numbers can be up to 65279.

## **LIST or LLIST**

**LIST n** lists a specified line number.

**LIST n TO m** lists a specified block of lines. If the first line number is omitted, the first line after 0 is assumed. If the second line number is omitted, the last line of the program is assumed.

## **LIST FORMAT or LLIST FORMAT**

allows you to produce an automatic 'pretty' listing, with each statement given its own line and these lines indented to help program readability. The first six

# Glossary

---

columns on the screen are reserved for line numbers and spaces, and programs can have up to 65279 lines.

**LIST FORMAT 0** gives unindented listings.

**LIST FORMAT 1** or **2** gives an automatically indented listing by one or two spaces when certain keywords are recognised in the program.

**DEF PROC**, **DO** and **FOR** indent all following statements by one or two spaces until they are cancelled by **END PROC**, **LOOP** or **NEXT**.

**IF**, **ON ERROR** and **ON** indent all statements in the rest of the line by one or two spaces.

**ELSE** and **EXIT IF** cancel one or two indentation spaces for the current statement only.

**END IF** cancels indentations due to a long **IF**.

## **LN**

calculates natural logarithms. A logarithmic function is the inverse of an exponential function.

## **LOAD**

is used to **LOAD** information from cassette or disk into the computer. If you are using a cassette player:

**LOAD ""** loads the first program recognised by the computer, wiping out the old memory.

**LOAD "name"** loads the program specified by its name, ignoring any other program that may be on the disk or tape.

**LOAD "name" CODE** loads the bytes from the first address at which the bytes were **SAVED**, overwriting any bytes which were previously there.

**LOAD "name" CODE start** loads the bytes beginning at 'start', overriding the start address from which the bytes were originally **SAVED**.

**LOAD "name" LINE number** loads the program and goes to line number **n** automatically.

**LOAD "name" DATA a()** or **a\$()** deletes any specified array, then searches for a new one to load.

If you wish to **LOAD** from one of the internal disk drives, you must first tell the computer which drive to use with the command:

# Glossary

---

- ❑ **DEVICE d** or **DEVICE d1** or **DEVICE D1** for disk drive 1, and
- ❑ **DEVICE d2** or **DEVICE D2** for disk drive 2.

## **LOCAL**

specifies a list of variables which are to be **LOCAL**ised to a procedure. They can include arrays, if you want.

## **LOOP**

is used in conjunction with **DO**, with **LOOP WHILE** and **LOOP UNTIL** working in exactly the same ways as **DO WHILE** and **DO UNTIL**.

## **LOOP IF (condition)**

makes the program jump out of a **DO** loop from the middle, and carry on again at the statement after **LOOP**, only if a condition is true. If it's not true then the program carries on.

## **LPRINT**

instructs the computer to output to a printer instead of the television screen. Please see Chapter 6.

## **Machine code**

is the set of instructions that the SAM Coupé's Z80B microprocessor chip uses, and you can write programs directly to it in 'assembly language'. Such programs have to be coded into a sequence of bytes using an assembler, but if you want to code them yourself there are plenty of books to teach you how.

## **Memory**

The SAM Coupé comes with 32k ROM and 256k DRAM on-board. There are two internal sockets for an additional 256k DRAM to be installed inside the machine. Please see Chapter 10.

## **MEM\$(n TO m)**

assigns a section of **MEM**ory to a string. Both **n** and **m** must be included, **n** can be 0, and the length of the memory 'slicer' must be less than 65536. Large areas of memory can be moved by **POKE**ing the string to a new location, and you can use **INSTR**ing to search it.

## **Menus**

for computers are the same as menus for restaurants. They provide a table of

# Glossary

---

options on screen from which you make your choice.

## **MERGE**

**MERGE "name"** adds new information from the named program to information already in the computer's memory, overwriting any existing program lines or variables that conflict with the new program.

**MERGE "name" CODE** suppresses auto-running of BASIC programs that have already been **LOADed**. **MERGE** can't be used with any arrays that may have been **SAVEd**.

## **MGT plc**

stands for Miles Gordon Technology, possibly the most fascinating computer manufacturers ever to operate from the Swansea Enterprise Park.

## **MIDI**

is the international standard by which musical instruments talk to each other, and stands for **Musical Instrument Digital Interface**. The SAM Coupé is able to control and communicate with all sorts of music devices via its **MIDI-IN** and **MIDI-OUT** ports.

## **MOD**

is a mathematical function whose result is a load of left overs, e. g.

□ **PRINT 110 MOD 60** will result in 50.

## **MODE n**

clears the screen and selects the numbered screen mode.

## **Modem**

is the abbreviation for **MOdulator DEModulator**, a device for transmitting data between computers via telephone lines.

## **Mouse**

is a little device that is connected by a cable to the correct port at the back of the computer. When slid around over a flat surface it can control a screen cursor, and when its buttons are clicked it selects various options. The **x** and **y** coordinates of the mouse may be read by using the functions **XMOUSE** and **YMOUSE**.



# Glossary

---

## **MOVE**

only works with disks.

## **Nesting**

is the process of placing one or more program blocks inside each other.

## **Network**

Up to 16 SAM Coupés can be linked in a network, with Channel 0 as the broadcast station and Channels 1 - 15 as personal operator stations.

## **NEW**

gets rid of all old programs and old variables that are in the computer's memory. When you use the **NEW** command the computer forgets everything it has learned since you switched it on.

## **NEXT**

is a special counting command used in a loop. It is always used in conjunction with the **FOR** command, and you should refer to the **FOR** section to see how this works.

## **NOT**

is similar to the **AND**, **OR** conditions in an **IF - THEN** statement, except that it is completely perverse. The **NOT** relation is true when the relation is false, and false whenever it's true! **NOT** can be regarded as a low priority function which does not need brackets around it unless it contains **AND** or **OR** or both.

## **ON**

allows you to **GO SUB** or **GO TO** a particular line number in a list of line numbers, according to the value of the numeric expression immediately after **ON**.

## **ON ERROR**

intercepts mistakes in your programming to allow the program to cope without stopping and giving an error message. This is known as error trapping and is detailed in Chapter 12.

## **OPEN**

is used to **OPEN** streams that are normally closed.

# Glossary

---

## **OPEN n**

reserves **n** additional 16k memory pages for use by BASIC. This allows **CLEAR** to set a higher **RAMTOP**.

## **OPEN SCREEN n,m**

reserves memory for a screen, when qualified by the screen number, followed by the screen mode. The screen number must be between **2** and **16**, and modes 1 to 4 can be selected. There is only one screen (called 1) when your computer is switched on.

## **OPEN SCREEN n,m**

reserves memory for a screen, when qualified by the screen number, followed by the screen mode. The screen number must be between **2** and **16**, and Modes 1 to 4 can be selected. There is only one screen (called 1) when your computer is switched on.

**OPEN SCREEN n,m,0** acts the same as **OPEN SCREEN n,m**

**OPEN SCREEN n,m,1** opens a 'common' screen, so that if several programs are present in the machine's memory, they can all use the screen when they are running.

## **OR**

is a condition that can be used as part of a condition after **IF**, **WHILE**, **UNTIL** etc, so that an **IF** statement will work when one relationship **OR** another is true. Of course, it will still work if both relationships are true.

## **Origin**

is the term used for the screen graphic coordinates **0,0**

## **OUT p,x**

is a statement like **POKE**, and writes the given value to the designated port with the given address. **P** must be between 0 and 65535, **x** must be between 0 and 255.

## **OVER**

is a statement for controlling the pattern of graphic dots on the screen, and causes a sort of **OVER**printing of points, lines and characters. **OVER 1** turns the effect on, and **OVER 0** turns it off. In modes 1 and 2 the character repeatedly prints over itself in a character/blank/character sequence. In

# Glossary

---

modes 3 and 4 the sequence is character/colour/character, the usual colour being black. Other effects can be created when used with **DRAW**, **CIRCLE**, **PLOT** and **PUT**, where **OVER 2** gives **OR**ing of each pixel with the screen, and **OVER 3** gives **AND**ing. Using **OVER 1** for a **PLOT**, **DRAW** or **CIRCLE**, and then repeating the process, will result in the original screen being restored.

## **PALETTE**

stops all changes to the colours you can use and resets the original colours to their start-up values.

## **PALETTE position,colour**

sets the position of the palette to any colour you want. There are sixteen different positions (0 to 15) and 128 different colours available (0 to 127). Please see Chapter 5 for all other **PALETTE** commands.

## **PAPER n**

when followed by the selected code number, determines the background screen colour on which any particular character is to be **PRINT**ed, and can also be used by typing

- **PRINT PAPER n**; "whatever follows"

## **Parameter**

refers to a unit that is constant in a given procedure, but can vary in different arguments.

## **PAUSE n**

allows you to make part of a program take an exact amount of time. Your screen displays 50 frames every second, so we use numbers counted in frames to give a very accurate freeze-frame timing. For example:

- **PAUSE 125**

will freeze your screen for two and a half seconds, or until a key is pressed.

## **PEEK**

is a function used to reveal what's in the computer's memory at an address within the range 0 to 528K.

# Glossary

---

## **PEN**

is used to select a colour on screen for your text and graphics. The values **0** to **15** are the same as those on your **PALETTE**. If you prefer, you can use the command **INK** instead of **PEN**, it will do the same thing.

## **PI**

is the Greek letter that is used to denote the ratio of the diameter of a circle to its circumference. In fact, **PI** is an infinite non-recurring decimal, but you can always try to confuse your Sam Coupé by typing **PRINT PI**

## **Pixel**

is supposed to stand for 'picture element', and refers to a single addressable dot of the computer's screen display.

## **PLOT x,y**

is a statement used for setting the start position for drawing pictures and diagrams on your screen. The drawing begins wherever you instruct the first pixel to be located. This is done by typing **PLOT** followed by the **x** coordinate, which tells the pixel how far it must be from the left hand side of your screen, and the **y** coordinate, saying how far it is from the bottom of the screen.

## **POINT (x,y)**

gives a pixel's colour at coordinates **x,y**. In **MODEs 1** and **2** it can be **0** or **1**, (showing if a pixel is **PAPER** or **PEN**), in **MODEs 3** and **4** it can be from **0** to **15** (to indicate the pixel's colour).

## **POKE**

is an impolite statement that barges its way past the rules of BASIC and stores a number directly into a memory location, which we call an address. For the memory that comes with your SAM Coupé, addresses can be from **0** to **528K**. **POKEs** in the range **0** to **16384** are not normally effective as this area is occupied by **ROM**.

## **POP**

takes away a line number from the **GO SUB/PROC/DO** stack, and throws it away. When **POP** is followed by a variable, then the discarded line number becomes assigned to that variable, so you can jump out of **GO SUB** routines, **DO** loops and **PROCedures** without messing things up with unused line

# Glossary

---

numbers. **POP x** assigns the discarded return line number to a variable such as **x**.

## **PRINT**

is the command that instructs the computer to **PRINT** out specified characters, graphics or results on screen. See Chapter 4 for full details of how this works.

## **Printer**

is a machine that can produce images of text and graphics on paper, known as 'hard copy', when connected to the appropriate port on the SAM Coupé.

## **Procedures**

make programming easier! By using the statements **DEF PROC**, **END PROC**, **DEFAULT**, **LOCAL** and **REF**, program 'modules' can be created that do a specific task without affecting the main program. Please see Chapter 4.

## **Program**

is simply a collection of commands used to instruct the computer.

## **PUT x,y,a\$**

places a stored area on the screen which has already been **GRABbed**. The **x** coordinate is rounded up to an even value.

## **PUT x,y,a\$,m\$**

makes use of a second string (which has to be the same length as the first string) to act as a graphic 'mask' which determines which pixels will be used from the first string. This allows a complex shape to be placed on a background as if it was a cut-out without a border. **OVER** has no effect in this 'masking' mode, but **INVERSE** will still work.

## **Radian**

A radian is the angle subtended by an arc whose length is equal to the radius of a circle.

## **RAM**

stands for **R**andom **A**ccess **M**emory. Data stored in **RAM** is lost when the **RESET** button is pressed, or the computer is switched off.

# Glossary

---

## **RAMTOP**

is the function that shows you the top address in the computer's **Random Access Memory** which is usable by **BASIC**.

## **RANDOMIZE**

uses a 'seed' number to calculate 'random' numbers. **RANDOMIZE** or **RANDOMIZE 0** sets it to the number of frames generated since the computer was switched on. **RANDOMIZE n** sets the random number 'seed' to a specified number.

## **READ**

is a statement that is followed by a list of the names of variables separated by commas. It works like an **INPUT** statement, but saves you the trouble of typing in the values of the variables by looking them up itself from **DATA** statements. You can also use **READ LINE** with string variables, allowing you to omit quotes in **DATA** statements.

## **RECORD STOP**

turns off the graphic recording. This command works in any mode.

## **RECORD TO a\$**

lets you record graphic commands as **STRINGS**, so they can be executed a lot faster when you **BLITZ** them. While **RECORD** is on, the following commands will be recorded: **CIRCLE**, **CLS**, **DRAW**, **DRAW TO**, **OVER**, **PAUSE**, **PEN**, **PLOT** (but not **PLOT PEN**).

## **REM**

is simply a **REMI**nder statement in a program, put there for your use not the computer's. **REM** can be followed by anything you like, and the computer will ignore it. You can use it to title blocks of information, such as:

□ 10 **REM** Income tax routine

## **RENUM**

automatically **RENUM**bers an entire program in steps of 10, starting with line 10.

## **RENUM n to m**

automatically **RENUM**bers part of a program, starting from line **n** and includ-

# Glossary

---

ing line **m**. If line **n** is left out, the first line is assumed. If line **m** is omitted, the last line is assumed.

## **RENUM STEP s**

makes the new line numbers go up in steps of **s**, if there is enough space. **STEP**, **LINE** and the program 'slicer' can be used or left out as you wish, but must occur in the following order:

- **RENUM n TO m LINE l STEP s**

## **[RESET] button**

when pressed, this button at the back of the machine DOES reset the CPU memory, the floppy disk controllers, the memory page registers, MIDI-IN, MIDI-OUT and the BORDER registers. It does NOT reset the colour look-up table, sound chip registers or the LINE INTERRUPT.

## **RESTORE n**

RESTORES the **DATA** pointer to the first **DATA** statement in line **n** or the nearest line after the designated number. If the number is omitted, **RESTORE** will be 0, i.e. set to the first **DATA** statement, if any. A **RESTORE** also happens every time a **BASIC** program is **LOADed**.

## **[RETURN]**

is the key used to commit a line of instructions into a program. Technically, **[RETURN]** terminates a subroutine.

## **RETURN**

a **RETURN** address gives a line and statement that is jumped to after a **GO SUB**.

## **RGB**

is the abbreviation for the system of coloured dots that make up any colour on your screen by combining **Red**, **Green** and **Blue**.

## **ROLL d,p**

moves the picture on your screen and wraps it around itself. Movement is controlled by direction **d** (1=left, 2=up, 3=right, 4=down), followed by the number of pixels to be moved, **p**. **P** can be 1, 2, 4, 6, etc, but not 3, 5, 7 etc.

# Glossary

---

## **ROLL d,p,x,y,w,L**

only **ROLLS** the screen section designated by **x,y** coordinates marking the top-left corner, followed by **w** pixels wide and **L** pixels long. **X** is rounded to an even coordinate and **w** is rounded to an even number of pixels. **ROLL** commands only work in modes 3 and 4.

## **ROM**

stands for Read Only Memory, and is the unchangeable part of computer's brain that has been educated in the factory.

## **RND**

generates so-called random numbers, starting at a point in a fixed sequence of 65536 numbers. Please see Chapter 4.

## **RUN**

commands the program to be **RUN**. To **RUN** a program from any particular line, you should use **RUN n**.

## **SAVE "name"**

christens a program and commands the computer to send a copy of that program from its memory to be recorded.

## **SAVE "name" LINE number**

is a neat way of getting the program to **GO TO** a particular line when it gets **LOADed**.

## **SAVE "name" CODE start address, length**

records information without any reference to what that information is to be used for. The first number (**start address**) must be the address of the first unit of storage memory known as a **byte**. The second number (**length**) is the number of **bytes** to be stored. The address is a number between 0 and 528K. Length can be from 1 byte to 512K.

## **SAVE "name" CODE start, length, execute-address**

will **SAVE** an auto running **CODE** file.

## **SAVE "name" SCREEN\$**

**SAVES** a screen with its **PALETTE** and **MODE** information.



# Glossary

---

## **SAVE "name" DATA a\$( )**

saves an **array** or **string** designated by its own particular name. Quotes are optional for strings or string arrays.

## **SAVE string LINE number**

tells the computer to record a program in such a way that when it is **LOADed** it jumps to a particular line number and **RUNs** itself.

## **Screen dump**

is the process of reproducing what is displayed on screen to a printer.

## **SCREEN n**

selects a **SCREEN** for BASIC commands to use. **N** must be between **1** and **16**, and the **SCREEN** must first be **OPENed**. So that:

□ **OPEN SCREEN 2: SCREEN 2**

makes commands like **PRINT**, **LIST**, **PLOT** and **DRAW** use a second **SCREEN** leaving **SCREEN 1** intact until **SCREEN 1** is typed again. Please also see **DISPLAY**.

## **SCREEN\$( r,c)**

is a function that works in any mode and identifies user-defined graphics in the range **CHR\$ 32** to **168** as well as normal characters. It does not recognise block graphics and it makes no difference if **BLOCKS 0** or **1** has been specified.

## **SCROLL d,p**

works in exactly the same way as **ROLL d,p** except that there is no wrap around effect. Blank background replaces the lost data. **SCROLL** commands only work in Modes 3 and 4.

## **SCROLL d,p,x,w,L**

works in exactly the same way as **ROLL d,p,x,y,w,L** except that there is no wrap around effect.

## **SCROLL CLEAR**

disables the '**Scroll?**' prompt on screen, so that when a screen becomes full it will scroll without any message or the need for a key to be pressed.

# Glossary

---

## **SCROLL RESTORE**

turns the '**Scroll?**' prompt back on.

## **SGN**

is the sign function, which is sometimes known as the 'signum'. So if you want to know the result of an argument, **SGN** will give **+1** if the argument is positive, **0** if it is zero, and **-1** if the argument is negative.

## **[SHIFT]**

when pressed simultaneously with other keys, the **[SHIFT]**ed keys are used to **PRINT** upper case letters and alternative characters.

## **SIN**

is a trigonometrical function that calculates a **SINE**.

## **Slicers**

are specified strings or substrings. A slicer can be a numerical expression, an optional numeric expression **TO** another optional numeric expression, or it can be empty.

## **Software**

is a list of instructions used to control the computer, which is stored to and **LOAD**ed from cassette or disk.

## **SOUND r,d**

sends data byte **d** to sound chip register **r**. Up to 127 pairs of numbers can be used, like this:

□ **SOUND r,d;r,d;r,d** etc

## **Space**

The long **[SPACE]** bar at the bottom of the keyboard acts the same as any conventional typewriter or wordprocessor.

## **SQR**

commands the computer to calculate the square root of a number **n**. The result is a number that when multiplied by itself gives **n**.

## **Stack**

refers to a number of data items 'stacked' in order of usage, with a specific

# Glossary

---

purpose in a computer program, such as a **GO SUB** stack or a calculator stack.

## Statement

consists of an individual instruction in a program line. Multiple statements in a single line are separated by colons.

## STEP n

is used in a **FOR - NEXT** loop to qualify the control variable which normally goes up 1 every time, like this:

❑ **FOR** control variable = first value **TO** last value **STEP n**

The **STEP** need not be a positive or a whole number, for example:

❑ **FOR x=100 to 1 STEP -1.5**

## STOP

halts the program and displays the '**STOP**' code on screen. When the program is started off again with **CONTINUE** it resumes at the next statement.

## Strings

in theory can be 65520 characters long, and their maximum legal names may be up to 10 characters long.

## STRING\$ (n,a\$)

repeats a string for the number of times you choose.

## STR\$

converts numbers into strings. The function ends in a \$ symbol to show that its result is a string. Please see **VAL**.

## Subroutine

is a block of numbered statements which perform a specific duty, and can be called up many times by the **GO SUB** statement.

## SVAR n

gives the address of System **VARIABLE n**.

## [SYMB]

when pressed, the **[SYMB]** shift key gives access to alternative characters and operations from the keyboard.

# Glossary

---

## Syntax error

in BASIC is similar to making mistakes in the English language when rules are broken. The computer is unforgiving if an error is made in the use of punctuation marks or a keyword is misspelled! Certain errors will be automatically corrected such as **gosub** and **RANDOMISE**, and the easiest method of writing programs is to type in lower case, and observe key words turning into capital letters as they are displayed in their lines.

## System variables

control all sorts of operations, and each has its own **SVAR** number. The most useful system variables are listed at the end of this Manual. There are hundreds more listed in the SAM Coupé Technical Manual!

## TAB

is a **PRINT** qualifier used to indent text and set up columns of text or numerals into orderly columns.

## TAN

is a trigonometrical function that calculates a **TAN**gent (a sine divided by a cosine).

## TO

is used to set the limitations in a **FOR - NEXT** loop, for example:

```
□ FOR play=1 TO 69: NEXT play
```

## TRUNC\$ a\$

gives **a\$** with any trailing spaces deleted. This is useful when you take data from string arrays, so that normally:

```
□ DIM a$ (10,10): LET a$(1)="sam": PRINT a$(1)
```

will result in "sam" being **PRINT**ed, whereas you can use:

```
□ PRINT TRUNC$ a$(1)
```

which results in "sam"

## UDG" "

gives the address of any user-defined graphic character from **CHR\$ 32** to **168**. The range from **CHR\$ 169** to **255** is available to advanced users.

# Glossary

---

## **UNTIL**

is a condition used with **DO** and **LOOP**. Please refer to these commands.

## **USR n**

When a numeric argument is a start address, **USR** is employed to run machine code, and allows values from **0** to **524287**

## **VAL**

is a very powerful function that converts strings back into numbers. The string can be any numeric expression. **VAL** takes away the quotes from a string, evaluates the arguments and finally evaluates whatever is left as a number.

## **VAL\$**

is similar to **VAL**, with a string for an argument but giving a result which is also a string.

## **Variable**

There are four types of variables, which are the names given to identify certain values in programs.

**Numeric variable** names must start with a letter and be followed by numbers, letters, underlining and spaces up to 32 characters long. Spaces don't count as characters.

**FOR-NEXT** variables are similar to numeric variables, but are treated differently internally.

**String variable** names and **array variable** names are similar, but can be no longer than 10 characters long and must be followed by **\$**.

## **VDU**

stands for **Visual Display Unit**, and generally refers to any unit housing a screen for displaying computer generated images.

## **VERIFY "name"**

checks that the information on tape or disk is exactly the same as the information in the computer's memory with the same name.

# Glossary

---

## **VERIFY "name" CODE**

checks the bytes on tape or disk, starting at the address from which the first byte was saved.

## **VERIFY "name" CODE start**

checks the bytes on tape or disk against those in memory starting at the address number given.

## **VERIFY "name" CODE start, length**

checks the bytes on tape or disk against the bytes in memory starting at the address number given. If the file on tape or disk is longer than the designated length, the "Loading error" message will be given.

## **VERIFY "name" DATA**

checks the named array on tape or disk against the one stored in memory.

## **WHILE**

is a condition used with **DO** and **LOOP**. Please refer to these commands.

## **WINDOW left, right, top, bottom**

sets up a text **WINDOW**. The limits are set by designating which left-hand column to use (**0** or above), followed by the right-hand column (maximum **84** in Mode 3 with 6-pixel wide characters), followed by the top line to be used (**0** or above) with the fourth number giving the bottom line of the **WINDOW**. If the requested borders are not allowed in the current mode or character size then 'Invalid **WINDOW**' is reported.

## **x-axis y-axis**

are nominal reference lines used in trigonometry, cutting a circle from side to side (**x-axis**) and from top to bottom (**y-axis**).

## **x-coordinate y coordinate**

coordinates locate an exact point on the screen. **X** is the distance from the left-hand side of the screen, and **Y** is the distance from the bottom of the screen.

## **XMOUSE YMOUSE**

please see **MOUSE**

# Glossary

---

## **XOS XRG YOS YRG**

are special functions that control the graphics scaling system in terms of scale and origin. Please see Chapter 6.

## **XPEN YPEN**

please see **PEN**

The following symbols have specific meanings when used within a SAM Coupé BASIC program, which may be different from their meanings in conventional English.

- +** plus
- minus
- \*** multiplied by
- /** divided by
- =** equal to
- <>** not equal to
- >** greater than
- <** less than
- >=** greater than or equal to
- <=** less than or equal to
- ↑** raise to the power of
- &** is used to introduce hex numbers
- ()** act as conventional brackets in numerical expressions
- [ ]** is used in this Manual to indicate a specific key to be pressed
- #** can be used to introduce a stream or channel number, or as a 'match anything' token. It is sometimes referred to as "hash" or "number".

# Glossary

---

- ' apostrophes in a PRINT statement cause whatever follows to appear at the left hand margin of the next line
- " quotation marks enclose strings, all characters inside quotation marks will be PRINTed exactly as typed. Please see Chapter 4 for other details
- , commas tabulate the following PRINT statement by 16 columns
- : a colon is used to separate commands in a line of program
- ; semi-colons cause PRINT statements to appear concurrently
- . the full stop can be used instead of the decimal point key
- ? appears in your EDITing lines whenever you manage to confuse the SAM Coupé



# SAM Coupé

## Appendix

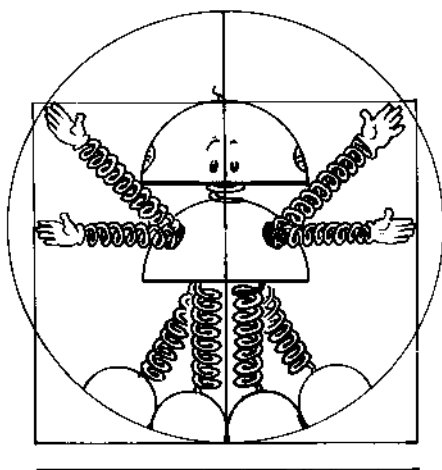
- General Specifications
- Lead Connections
- Pin Connections
- System Variables
- Predefined Keys
- Keyboard Map
- Control Codes

*"Yoghurt is very  
good for  
appendicitis"*

(Eugène Ionesco)

*"Man is still the  
most extraordi-  
nary computer  
of all"*

(John F. Kennedy)



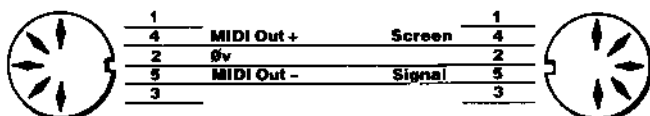
**MILES GORDON TECHNOLOGY, SAM Coupé**

<b>ENGINE</b>	Z80B microprocessor running at 6MHz.
<b>CONTROL</b>	Customised VLSI 10,000-gate ASIC chip.
<b>ROM</b>	32K including SAM BASIC, disk bootstrap, BIOS.
<b>RAM</b>	256K upgradeable to 512K (256K x 4 100nS DRAM).
<b>SOUND</b>	Philips SAA 1099 Synthesizer: 6 channel, 8 octave, stereo with envelope and amplitude control, plus wave form choice.
<b>GRAPHICS</b>	Motorola MC 1377P Video Chip. ASIC serves as graphic processor. All Modes allow 128 colours to be displayed on-screen by redefining line interrupt: <b>Mode 1:</b> 32 x 24 character cells per screen, each cell with 2 colour capability; 16 colours selectable from 128; Spectrum attribute compatible. <b>Mode 2:</b> as Mode 1, but 32 x 192 cells, each cell with 2 colour capability; 16 colours selectable from 128. <b>Mode 3:</b> up to 85 column text display, 512 x 192 pixel screen with each pixel colour selectable; 4 colours per line selectable from 128. <b>Mode 4:</b> 256 x 192 pixel graphics screen; each pixel colour selectable; 16 colours per line selectable from 128.
<b>INTERFACES</b>	UHF TV Channel 36, through power supply unit. Colour composite video, digital and linear RGB, all through SCART. Atari-standard joystick (dual-capacity with splitter cable). Mouse, Coupé standard. Light-pen/Light-gun, Coupé standard. Domestic cassette recorder. MIDI-In, MIDI-Out (MIDI-Through via software switch). Network, screened microphone cable via external MGT interface to Expansion Port. 64-pin Expansion Port for further peripherals.
<b>DISK DRIVES</b>	1 or 2 removeable and internally mounted 3.5" ultra-slim Citizen drives, 1Mb unformatted, 780K formatted.
<b>DC POWER</b>	Power Supply 4.75 volts to 5.25 volts.
<b>CONSUMPTION</b>	Power Consumption 11.2 Watts.
<b>SHOCK</b>	Operating 3 G, non-operating 60 G.
<b>VIBRATION</b>	Operating 5 to 500Hz / 0.5 G, non-operating 5 to 500Hz / 2 G.
<b>ENVIRONMENT</b>	Ambient Temperature, operating 5 to 45C, storage -20 to 50C.
<b>HUMIDITY</b>	Relative Humidity Wet Bulb Maximum 29.4C, nil condensation.
<b>RELIABILITY</b>	MTBF: 10,000 POH. MTTR: 30 Mins. Component Life: >5 years.
<b>WEIGHT</b>	2.26Kg = 4.97lb.

**MIDI lead**

MIDI SOCKET PLUG DIN 5

MIDI DEVICE PLUG DIN 5

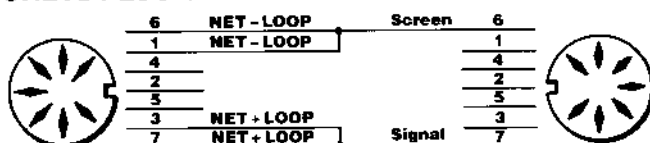


Cable Type: 2-core with Screen, length 2m

**NETWORK lead**

MIDI SOCKETS PLUG DIN 7

FROM NETWORK PLUG DIN 7

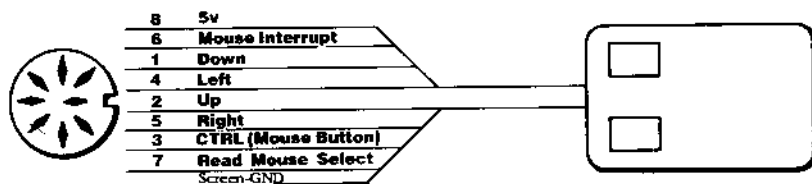


Cable Type: 1-core with Screen, length 2m

**MOUSE lead**

COMP. END PLUG DIN 8

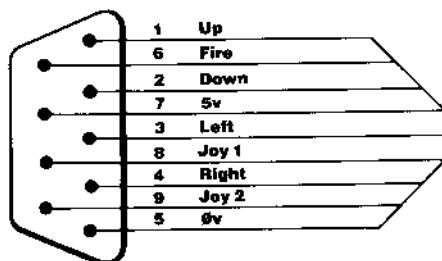
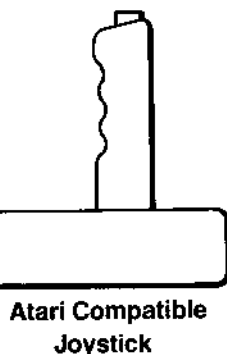
MGT Mouse



Cable Type: 8-core flex, length 1.25m

**JOYSTICK lead**

JOYSTICK PLUG DB9

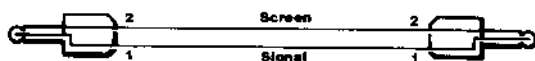


Cable Type: 8-core with Screen, length 1.25m

**CASSETTE lead**

COMP.END 3.5mm Jack

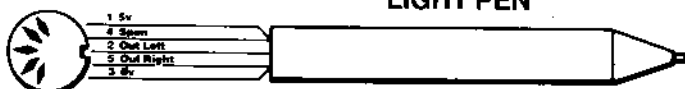
EAR/MIC 3.5mm Jack



Cable Type: 1-core with Screen, length 0.75m

**LIGHT PEN lead**

LIGHT PEN



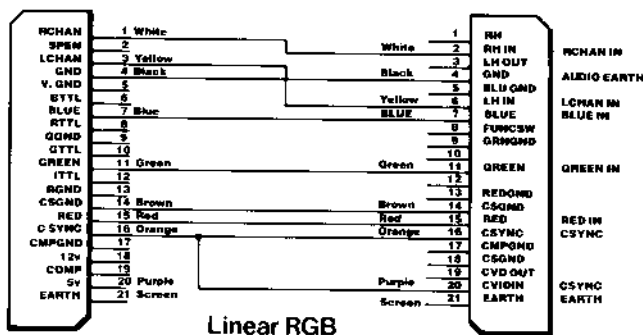
LIGHT PEN PLUG DIN 5

Cable Type: 4-core with Screen, length 1.5m

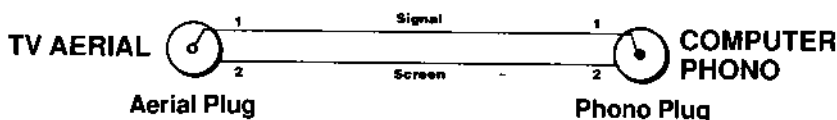
**SCART lead**

SAM SCART

MONITOR STD. SCART

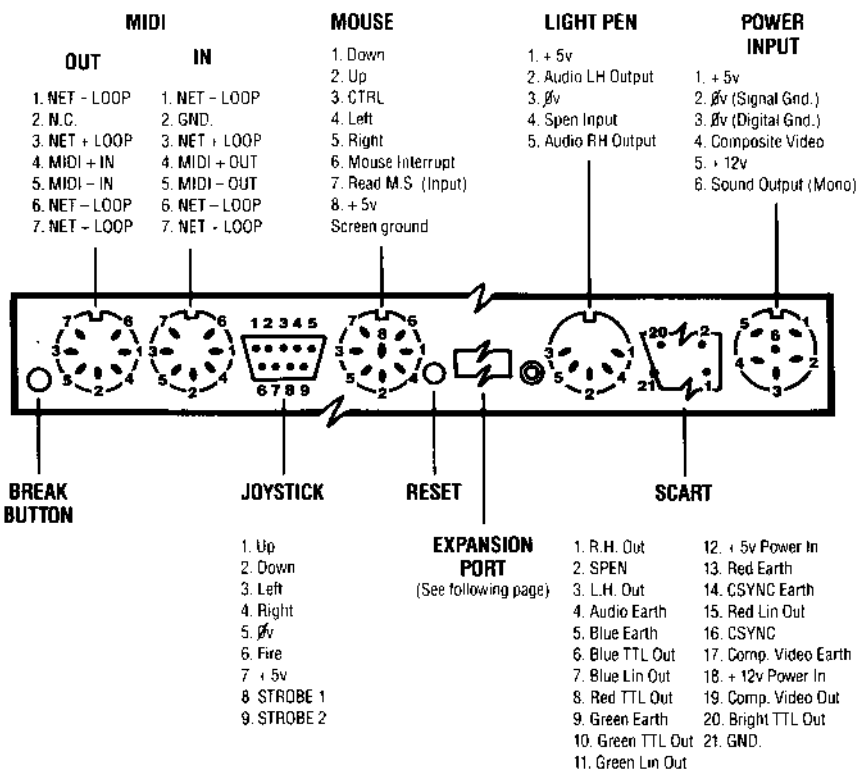


Cable Type: 8-core with Screen, length 2m

**UHF TV lead**

Cable Type: 1-core with Screen, length 2m

Pin connections. All plugs and sockets are viewed from the back of the SAM Coupé. Please see the following page for Expansion Connector details.



## EXPANSION CONNECTOR

Standard 64-pin Euroconnector socket with rows A-C fitted. Note that row A is at the bottom of the Euroconnector and row C is located at the top, thus:

1C	32C
1A	32A

PIN	SIGNAL	PIN	SIGNAL
1A	<u>DBDIR</u>	1C	<u>IORQ</u>
2A	<u>RD</u>	2C	<u>MREQ</u>
3A	<u>WR</u>	3C	<u>HALT</u>
4A	<u>BUSAK</u>	4C	<u>NMI</u>
5A	<u>WAIT</u>	5C	<u>INT</u>
6A	<u>BUSREQ</u>	6C	CD1
7A	<u>RESET</u>	7C	CD0
8A	<u>CM</u>	8C	CD7
9A	<u>REFRESH</u>	9C	CD2
10A	0 VOLTS	10C	+5 VOLTS
11A	A0	11C	CD6
12A	A1	12C	CD5
13A	A2	13C	CD3
14A	A3	14C	CD4
15A	A4	15C	CPU CLK
16A	A5	16C	A15
17A	A6	17C	A14
18A	A7	18C	A13
19A	A8	19C	A12
20A	A9	20C	<u>A11</u>
21A	A10	21C	<u>DISK 2</u>
22A	NO CONNECTION	22C	<u>ROMCS</u>
23A	<u>XMEM</u>	23C	<u>EARMIC</u>
24A	8MHz	24C	<u>DISK 1</u>
25A	RED 1	25C	<u>PRINT</u>
26A	<u>GFREN 1</u>	26C	<u>BLUE 1</u>
27A	<u>C SYNC</u>	27C	<u>ROMCSR</u>
28A	SPEN	28C	AUDIO RIGHT OUTPUT
29A	BLUE 0	29C	AUDIO LEFT OUTPUT
30A	RED 0	30C	COMP VIDEO
31A	BRIGHT	31C	GREEN 0
32A	+5 VOLTS	32C	0 VOLTS

There are over 600 bytes of system variables available for the advanced programmer. The full list is contained in the SAM Coupé Technical Manual available from MGT. The following list is of the most useful system variables, which are altered by, e.g.

❑ **POKE SVAR 8,10** or

❑ **POKE SVAR 1,"LU"**

They can be examined by, e.g.

❑ **PRINT PEEK SVAR 8**

NAME	SVAR NUMBER	
LNCUR	0	CURRENT LINE CURSOR CHAR (NORMALLY '>')
LCCUR	1	CURSOR CHARACTER WHEN CAPS LOCK IS OFF (NORMALLY '*')
UCCUR	2	CURSOR CHARACTER WHEN CAPS LOCK IS ON (NORMALLY '+')
BIN1DIG	3	CHARACTER USED BY BIN\$ AS '1' (NORMALLY '1')
BIN0DIG	4	CHARACTER USED BY BIN\$ AS '0' (NORMALLY '0')
INSTHASH	5	CHARACTER USED BY INSTR AS 'MATCH ANYTHING' CHARACTER (NORMALLY '#')
SLDEV	6	CURRENT DEVICE LETTER (USUALLY 'T' ON A TAPE SYSTEM)
SLNUM	7	CURRENT TAPE SAVE SPEED, OR DEFAULT DRIVE NUMBER WHEN A DISK DRIVE IS IN USE
SPEEDINK	8	SPEED OF FLASHING COLOURS (SEE PALETTE COMMAND)
PRRHS	14	RIGHT-HAND TEXT COLUMN FOR PRINTERS (1 LESS THAN THE MAXIMUM LINE LENGTH DESIRED)
AFTERCR	15	CHARACTER CODE SENT TO THE PRINTER AFTER CHR\$ 13 IF CHANNEL 'P' IS IN USE. (NORMALLY 10, TO GIVE AUTOMATIC LINE FEED.) POKE SVAR 15,0 TO STOP ANY CHARACTER BEING SENT.
DMPTL	16	(2 BYTES) ADDRESS IN SCREEN OF TOP LEFT CORNER FOR GRAPHIC DUMPS

DMPLEN	18	GRAPHIC DUMP LENGTH (IN 8-PIXEL UNITS; NORMALLY 22)
DMPWID	19	GRAPHIC DUMP WIDTH (IN 8-FAT-PIXEL UNITS; NORMALLY 32)
DMPWM	20	GRAPHIC DUMP WIDTH MULTIPLIER (1=NORMAL; 2 OR 3 CAN BE USED FOR DOUBLE OR TRIPLE WIDTH)
DMPHM	21	GRAPHIC DUMP HEIGHT MULTIPLIER (1=NORMAL, NOT 1=DOUBLE)
GCM1	22	(9 BYTES) INITIAL MESSAGE SENT TO PRINTERS BEFORE A DUMP
GCM2	31	(8 BYTES) MESSAGE SENT BEFORE EACH DUMPED ROW
GCM3	39	(7 BYTES) FINAL MESSAGE SENT TO PRINTERS AFTER A DUMP
TABVAR	47	0 IF THE PRINT COMMA SHOULD TAB BY 16 COLUMNS, NOT 0 IF AN 8 COLUMN TAB IS TO BE USED
SOFE	50	FLAGFOR SCREEN OFF ENABLE/DISABLE, NORMALLY 0 (ON); THE SCREEN WILL GO BLACK IF YOU DO NOT USE THE KEYBOARD FOR 22 MINUTES. POKE SVAR 50,1 TO DISABLE THIS FEATURE
TPROMPTS	51	BIT 0=1 TO SUPPRESS PRINTING OF FILE NAMES DURING LOAD; BIT 1=1 TO SUPPRESS PROMPTS DURING SAVE TO TAPE
BGFLG	52	BLOCK GRAPHICS FLAG. 0 IF BLOCKS 1, NOT 0 IF BLOCKS 0
FL6OR8	53	0 IF MODE 3 IS USING 6-PIXEL WIDE CHARACTERS, OTHERWISE NOT 0
CSIZEH	54	CHARACTER HEIGHT
CSIZEW	55	CHARACTER WIDTH
UWRHS	56	UPPER WINDOW RIGHT-HAND COLUMN
UWLHS	57	UPPER WINDOW LEFT-HAND COLUMN
UWTOP	58	UPPER WINDOW TOP ROW
UWBOT	59	UPPER WINDOW BOTTOM ROW



LWRHS	60	LOWER WINDOW RIGHT-HAND COLUMN
LWLHS	61	LOWER WINDOW LEFT-HAND COLUMN
LWTOP	62	LOWER WINDOW TOP ROW
LWBOT	63	LOWER WINDOW BOTTOM ROW
MODE	64	SCREEN MODE (MINUS 1)
YCOORD	65	CURRENT GRAPHICS Y COORDINATE, 0 AT THE TOP
XCOORD	66	(2 BYTES) CURRENT GRAPHICS X COORDINATE
LASTH	513	LAST KEY PRESSED
KBHEAD	520	KEY AT HEAD OF KEYBOARD BUFFER QUEUE
REPDEL	521	DELAY BEFORE KEYS AUTO-REPEAT
REPSPD	522	SPEED AT WHICH KEYS AUTO-REPEAT
CHARS	566	(2 BYTES) ADDRESS 256 BYTES BELOW MAIN CHARACTER SET
ERRSOUND	568	LENGTH OF ERROR SOUND
CLICK	569	LENGTH OF KEYBOARD CLICK
KPFLG	615	FUNCTION KEYS IF EVEN, NUMBER PAD IF ODD
KLFLAG	618	8 IF CAPS LOCK ON, OTHERWISE 0
FRAMES	632	(3 BYTES) TV FRAMES SINCE COMPUTER TURNED ON
UDG	635	(2 BYTES) ADDRESS OF CHARACTER PATTERN FOR CHR\$ 144
HUDG	637	(2 BYTES) ADDRESS OF CHARACTER PATTERN FOR CHR\$ 169

Numbers in brackets are character codes. Characters with codes of 128 and above can have several interpretations, depending on **BLOCKS 0** or **1** being used, or if they are enclosed by quotation marks. E.g. **[SYMB] [L]** gives the key word **LET** if used outside of quotes, or **&** if used inside them. **[SYMB] [RETURN]** toggles the keys **[F0]** to **[F9]** to give numbers. **[CNTRL] [P]** gives the **PAPER** control code and **[CNTRL] [I]** gives the **PEN** control code: press **0** to **7** to select a colour number. **[CNTRL] [B]** gives the **BRIGHT** control code: press **0** or **1** to force **PEN** and **PAPER** to be in the range **0** to **7** or **8** to **15**. Obviously, some characters occur in several places on the keyboard, and this is for the convenience of users familiar with different keyboard layouts.

### NORMAL + [CAPS] + [SYMBOL] + [CNTRL]

#### top line of keys

ESCApe	ESCApe	ESCApe	ESCApe
1	!	(129)	(142)
2	@	(130)	(141)
3	#	(131)	(140)
4	\$	(132)	(139)
5	%	(133)	(138)
6	&	(134)	(137)
7	'	(135)	(136)
8	(	(128)	(143)
9	)		
0	~	-	
-	/	/	
+	*	*	
DEL Lt	DEL Rt	DEL Rt	
F7 (199)	(209)	7	
F8 (200)	(210)	8	
F9 (201)	(211)	9	

#### second line of keys

TAB	TAB	TAB	
q	Q	<	
w	W	>	
e	E	é	
r	R	[	
t	T	]	
y	Y	¥	
u	U	ü	
i	I	INVERSE	(PEN CC)
o	O	ö	
p	P	PRINT	(PAPER CC)
=	-	-	
"	©	COPY	
F4 (196)	(206)	4	
F5 (197)	(207)	5	
F6 (198)	(208)	6	

**NORMAL + [CAPS] + [SYMBOL] + [CNTRL]****third line of keys**

CAPS Lk	CAPS Lk	CAPS Lk	
a	A	STOP	
s	S	SAVE	
d	D	DRAW	
f	F	{	
g	G	}	
h	H	↑	
j	J	—	
k	K	+	
l	L	£ LET	
:	:	*	
:	:	*	
RETURN	RETURN	RETURN	
F1 (193)	(203)	1	
F2 (194)	(204)	2	
F3 (195)	(205)	3	

**fourth line of keys**

SHIFT	SHIFT	SHIFT	SHIFT
z	Z	?	
x	X	?	
c	C	¿ CSIZE	
v	V	VERIFY	
b	B	BORDER	(BRIGHT CC)
n	N	ñ BRIGHT	
m	M	Ñ INVERSE	
,	,	<	
.	.	>	
INV	\	TRU-VID	
SHIFT	SHIFT	SHIFT	SHIFT
F0 (192)	(202)	0	
UP	UP	UP	
.	.	.	

**bottom line of keys**

SYMB	SYMB	SYMB	SYMB
CNTRL	CNTRL	CNTRL	CNTRL
SPACE	SPACE	SPACE	
EDIT	EDIT	toggle	
SYMB	SYMB	SYMB	SYMB
LEFT	LEFT	LEFT	
DOWN	DOWN	DOWN	
RIGHT	RIGHT	RIGHT	

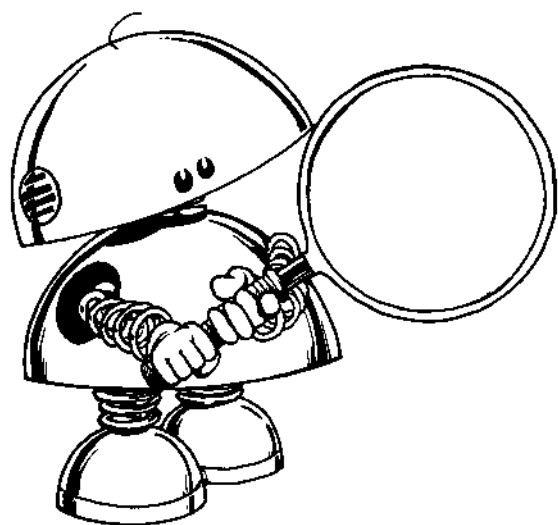
6	When printed: PRINT COMMA (TAB) When coming directly from the keyboard: CAPS LOCK TOGGLE
7	When printed: '?' When coming directly from the keyboard: EDIT
8	CURSOR LEFT
9	CURSOR RIGHT
10	CURSOR DOWN
11	CURSOR UP
12	DELETE LEFT
13	RETURN
14	When printed: SPACE When coming directly from the keyboard: DELETE RIGHT
15	When printed: '?' When coming directly from the keyboard: KEY PAD TOGGLE
16	PEN
17	PAPER
18	FLASH
19	BRIGHT
20	INVERSE
21	OVER
22	AT
23	TAB

## Keyboard Map

23	68	59	50	41	32	31	40	49	58	67	22	13	4	24	15	6
14	69	60	51	42	33	30	39	48	57	66	21	12		25	16	7
5	70	61	52	43	34	29	38	47	56	20	11	65		26	17	8
+70	62	53	44	35	28	37	46	19	10	1	+70		3	54		10
+140	+210	64										2	+140	36	45	27

Add 70 to obtain map position of a key plus [SHIFT]  
 Add 140 to obtain map position of a key plus [SYMB]  
 Add 210 to obtain map position of a key plus [CNTRL]  
 E.g. 'a' is at position 70, 'A' is at position 140.

# Index



# Index

- |                     |                   |                    |                 |
|---------------------|-------------------|--------------------|-----------------|
| <b>ABS</b>          | 46, 132           | <b>CODE</b>        | 58, 112, 137    |
| <b>ACS</b>          | 47, 132           | colons             | 31, 168         |
| addition            | 44                | colour chart       | 65              |
| address             | 60, 113, 114, 132 | commands           | 137             |
| <b>AND</b>          | 54, 132           | commas             | 31, 34, 87, 168 |
| animation           | 82                | conditions         | 36, 137         |
| apostrophes         | 35, 168           | <b>CONTINUE</b>    | 33, 137         |
| arrays              | 38, 132           | control codes      | 137, 180        |
| array variables     | 39                | <b>COS</b>         | 47, 137         |
| ASCII code          | 133               | <b>CSIZE</b>       | 86, 118, 138    |
| <b>ASN</b>          | 47, 133           | cursor             | 30, 138         |
| <b>AT</b>           | 87, 133           | cursor keys        | 12, 138         |
| <b>ATN</b>          | 47, 133           | Customer Care Line | 2               |
| attributes          | 70, 133           | <b>DATA</b>        | 52, 59, 138     |
| <b>AUTO</b>         | 49, 133           | <b>DEFAULT</b>     | 60, 138         |
| <b>BEEP</b>         | 94, 134           | <b>DEF FN</b>      | 57, 139         |
| <b>BIN</b>          | 90, 134           | <b>DEF KEYCODE</b> | 139             |
| <b>BIN\$</b>        | 134               | <b>DEF PROC</b>    | 59, 139         |
| bit                 | 110, 134          | <b>DELETE</b>      | 30, 140         |
| <b>BLITZ</b>        | 78, 134           | [DELETE] key       | 14, 30, 139     |
| block graphics      | 89, 178           | <b>DEVICE</b>      | 115, 140        |
| <b>BLOCKS</b>       | 88, 134           | <b>DIM</b>         | 38, 54, 140     |
| <b>BOOT</b>         | 115, 134          | <b>DIR</b>         | 115, 140        |
| <b>BORDER</b>       | 67, 135           | disks              | 15, 140         |
| brackets            | 45, 167           | disk drives        | 9, 115, 140     |
| [BREAK] button      | 9, 135            | <b>DISPLAY</b>     | 79, 141         |
| <b>BRIGHT</b>       | 70, 135           | <b>DIV</b>         | 141             |
| <b>BUTTON</b>       | 135               | division           | 44              |
| byte                | 110, 135          | <b>DO</b>          | 55, 141         |
| cables              | 3, 4, 10, 171     | <b>DPEEK DPOKE</b> | 113, 141        |
| calculations        | 44                | <b>DRAW</b>        | 75, 142         |
| <b>CALL</b>         | 113, 135          | <b>DUMP</b>        | 106, 142        |
| [CAPS] lock         | 14, 135, 178      | <b>DUMP CHR\$</b>  | 142             |
| cassettes           | 15                | [EDIT] key         | 30, 142         |
| cassette recorder   | 10, 20            | education          | 118             |
| channels            | 107, 118          | <b>ELSE</b>        | 40, 143         |
| character set       | 88, 136           | <b>ELSE IF</b>     | 41, 143         |
| <b>CHR\$</b>        | 59, 89, 136       | <b>END IF</b>      | 40, 143         |
| <b>CIRCLE</b>       | 76, 136           | <b>END PROC</b>    | 60, 143         |
| cleaning            | 15                | <b>ERASE</b>       | 115, 143        |
| <b>CLEAR</b>        | 39, 114, 136      | error codes        | 23, 29, 129     |
| <b>CLOSE</b>        | 112, 136          | error trapping     | 129             |
| <b>CLOSE SCREEN</b> | 79, 136           | [ESC] key          | 26, 33, 143     |
| <b>CLS</b>          | 34, 79, 137       | equals             | 36              |
| [CNTRL] key         | 85, 138, 178      | Euroconnector      | 10              |

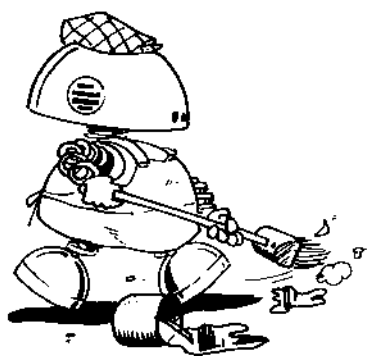
# Index

<b>EXIT IF</b>	56, 143	<b>LENGTH</b>	54, 148
<b>EXP</b>	45, 143	less than	36
expansion connector	10, 104	<b>LET</b>	37, 149
exponentiation	44, 45	light pen	11, 105, 149
expressions	44	line numbers	21, 29, 149
<b>FATPIX</b>	76, 144	<b>LIST</b>	48, 149
<b>FILL</b>	76	<b>LIST FORMAT</b>	51, 149
<b>FLASH</b>	70, 144	listing	51
floating point	52	<b>LLIST</b>	106, 149
<b>FOR</b>	42, 144	<b>LLIST FORMAT</b>	51, 149
<b>FORMAT</b>	115, 144	<b>LN</b>	45, 150
<b>FREE</b>	48, 111, 144	<b>LOAD</b>	20, 22, 26, 115, 150
functions	45, 47, 145	<b>LOCAL</b>	60, 151
function keys	57, 143	<b>LOOP</b>	55, 151
<b>GET</b>	84, 145	<b>LPRINT</b>	106, 151
<b>GO SUB</b>	43, 145	machine code	112, 151
<b>GO TO</b>	33, 145	memory	110, 151
<b>GRAB</b>	76, 81, 145	<b>MEM\$</b>	111, 151
graphics coordinates	75, 77, 145	<b>MERGE</b>	20, 25, 26, 115, 152
graphics scaling system	77	<b>MIDI-IN/MIDI-OUT</b>	9, 97, 110, 152
greater than	36	<b>MOD</b>	152
<b>HEX\$</b>	146	<b>MODE</b>	67, 152
<b>IF</b>	36, 40, 56, 146	mouse	10, 105, 152
<b>IN</b>	114, 146	<b>MOVE</b>	115, 153
<b>INK</b>	67	multiplication	43
<b>INKEY\$</b>	84	nesting	41, 153
<b>INPUT</b>	24, 33, 119, 146	networks	10, 118, 153
<b>INSTR</b>	59, 111, 147	<b>NEW</b>	34, 153
<b>INT</b>	45, 147	<b>NEXT</b>	42, 153
<b>[INV] key</b>	49, 71, 147	<b>NOT</b>	54, 153
<b>INVERSE</b>	71, 81, 147	numeric variables	39
<b>ITEM</b>	60, 147	<b>ON</b>	57, 153
joysticks	105	<b>ON ERROR</b>	129, 153
joystick port	10	<b>[ON/OFF] button</b>	4, 10
key positions	148, 178	<b>OPEN</b>	112, 153
keyboard	12	<b>OPEN SCREEN</b>	78, 111, 154
keyboard map	180	<b>OR</b>	54, 154
<b>KEYIN</b>	148	<b>OUT</b>	114, 154
keywords	28, 148	<b>OVER</b>	71, 81, 154
<b>LABEL</b>	61, 148	<b>PALETTE</b>	64, 71, 155
lead connections	171	<b>PAPER</b>	67, 155
<b>LEN</b>	48, 148	parameter	155
		<b>PAUSE</b>	32, 97, 155
		<b>PEEK</b>	113, 155

# Index

- PEN** 67, 156  
**PI** 46, 156  
pin connections 173  
pitch values 95  
pixels 68, 74, 86, 156  
**PLOT** 74, 156  
**POINT** 71, 156  
**POKE** 113, 156  
**POP** 60, 156  
power supply unit 3, 4, 11  
**PRINT** statements 34, 157  
printers 105  
procedures 59, 157  
program names 24  
punctuation 31, 34, 167  
**PUT** 81, 157  
  
radian 47, 157  
**RAM** 18, 157  
**RAMTOP** 111, 158  
random numbers 51  
**RANDOMIZE** 52, 158  
**READ** 53, 54, 158  
**RECORD** 85, 119, 158  
**REF** 61  
**REM** statements 31, 49  
**RENUM** 50, 158  
[RESET] button 6, 10, 159  
**RESTORE** 53, 159  
**RETURN** 43, 159  
[RETURN] key 5, 21, 29, 159  
**RND** 52, 160  
**ROLL** 80, 159  
**ROM** 18, 160,  
**RUN** 22, 32, 160  
  
**SAVE** 19, 22, 25, 115, 160  
SCART socket 11, 104  
**SCREEN** 78, 161  
**SCREEN\$** 161  
screen modes 67, 69  
**SCROLL** 80, 161  
scroll? 33, 79  
semi-colons 35, 168  
**SGN** 46, 162  
[SHIFT] key 13, 85, 162  
**SIN** 47, 162  
  
**SOUND** 100, 162  
sound effects 100  
[SPACE] bar 13, 162  
specifications 170  
**SQR** 45, 162  
stack 162  
statements 29, 163  
**STEP** 42, 50, 163  
streams 107, 119  
**STOP** 32, 163  
**STR\$** 58, 163  
strings 37, 58, 163  
**STRING\$** 37, 163  
string variables 39  
subroutines 43, 163  
substrings 37  
subtraction 44  
[SYMB] key 21, 36, 85, 163, 178  
system variables 61, 164, 175  
**SVAR** 61, 163, 175  
  
**TAB** 34, 87, 164  
**TAN** 47, 164  
television sets 4, 6, 16  
**THEN** 36, 146  
**TO** 42, 164  
trouble shooting 6, 16, 122  
**TRUNC\$** 54, 164  
  
user defined graphics 89  
**UDG** 89, 164  
**UNTIL** 54, 165  
**USR** 113, 165  
**USR\$** 113  
  
**VAL** 58, 165  
variables 38, 60, 165  
**VERIFY** 19, 22, 26, 115, 165  
  
'welcome screen' 5  
**WHILE** 55, 166  
**WINDOW** 79, 80, 166  
  
x-axis y-axis 46, 77  
x-y coordinates 77, 166









The logo for Miles Gordon Technology (MGT) features the letters 'MGT' in a bold, white, sans-serif font. The letters are set against a blue background that is shaped like a downward-pointing triangle. The 'M' and 'G' are connected, and the 'T' is separate. There are small red squares at the bottom of the 'M', 'G', and 'T' respectively.

**MILES GORDON TECHNOLOGY plc**

Lakeside  
Phoenix Way  
Swansea, SA7 9EH  
United Kingdom

ISBN 1-872589-00-6



9 781872 589008