# 2: What's It All About?

RISC OS is a clever piece of software. Considering that its first incarnation was over 30 years ago, it hasn't changed that much. This is all the more remarkable when you consider how far computers have come and have themselves changed in that same period of time.

Much of the reason for this is possibly because RISC OS was designed with the ARM chip in mind and makes use of its unique features. The ARM chip started as an innovation and has continued to be developed and innovative and to set the pace at the forefront of technology. Just about every smart phone and tablet device on sale today has an ARM chip at its core, and for example these apparently modest devices are being used to control budget satellites being placed in Earth orbit. And in theory all could run RISC OS!

In this chapter we'll look at some of the features that make RISC OS what it is on the Raspberry Pi — RISC OS Pi. Many of the concepts introduced in this simple overview form the basis of forthcoming chapters that deal with these topics in much greater detail.

## Operation Process

The operation of RISC OS is based largely around the processing of interrupts and events. Take these two actions away and not a lot goes on. At the core of this RISC OS lies the Kernel (the part of the computer software that has control over everything it does) and everything that happens is directed through it. This is what holds everything together; everything else is attached to it.

Interrupts are largely driven by the ARM chip and for the most part are provided in order to deal with attached hardware, with tasks such as making sure that important peripherals such as disk drives and keyboards get serviced as quickly as possible. Because these devices are hardware connected RISC OS doesn't have to worry about looking for them; it simply has to provide the software for dealing with them. The ARM looks after interrupts right from the start, long before RISC OS is loaded.

Events are essentially sanitised versions of interrupts. They are used to inform the user whenever RISC OS performs or detects that some significant task has taken place. The user in this case may also be an application. These events may be ‘captured’ and acted upon. Many events are produced in response to RISC OS detecting an interrupt from the hardware and the event is ‘issued’ to users who can identify what it is and act accordingly.

Software and applications are written in much the same way, and use their own interrupt or polling system to manage what is going on. The Desktop that RISC OS Pi boots into provides a complete WIMP environment whereby you can use windows, menus, icons, pointers and a range of other features. This is all managed by a part of RISC OS called the Window Manager and it is continually issuing its own series of events to all the elements of the Wimp using a code number to tell or poll the various Desktop elements that certain actions have taken place. The RISC OS application programming interface (API) is extremely simple and accessible — to the point that it can be mastered by anyone with time to spare.

## Routine Library

RISC OS, probably more than any other operating system available, provides an extensive library of routines which any programmer can use to interact with the hardware and applications. This means that the programmer doesn’t have to spend time and effort writing their own – all that is needed is to provide the SWIs (Software Interrupts) with the relevant information and set the process in motion. This is perfect for programmers who, for example, do not have to worry about reading input from the keyboard, as there is a RISC OS routine to do just that (and just about everything else). This has the added advantage of ensuring that the interface appears standard and consistent, and for example provides an almost universal drag and drop facility for file save, file load, and operations such as copy, cut and paste. These are covered extensively.

Because RISC OS is so organised it needs programmers to be likewise when presenting information. This can be quite daunting as in some instances — thankfully few — a lot of details are required. These need to be provided in memory blocks of buffers which themselves remain consistent within applications, so it is often possible to let information run within the buffers themselves.

## Extendibility

RISC OS itself is quite small and light in terms of memory usage. This is why it is so incredibly quick to boot-up. It is also infinitely expendable by adding bolt-on extras called modules. In fact RISC OS itself is largely composed of modules that provide additional functionality, making it very easy to adapt to changes and development in hardware and software. Not surprisingly, modules follow a standard format and are not difficult to write. This means that as new hardware becomes available, modules to handle their attachment can be developed very quickly.

As a programming environment RISC OS has always sought to make things as easy as possible, and in a bid to ensure everything looks and acts the same all the input/output is handled by RISC OS — programs and applications just piggy-back onto it. This is not just the keyboard and screen; it includes items such as filing systems and the opening and closing of files. Writing a filing system is simplified because RISC OS provides all the relevant i/o interfaces for the programmer to use.

The inclusion as standard of an established structured language such as BBC BASIC as part of RISC OS is a bonus. The fact that this and its built in ARM assembler are seamlessly integrated makes it amazingly simple to quickly produce effective code.

## Well Supported

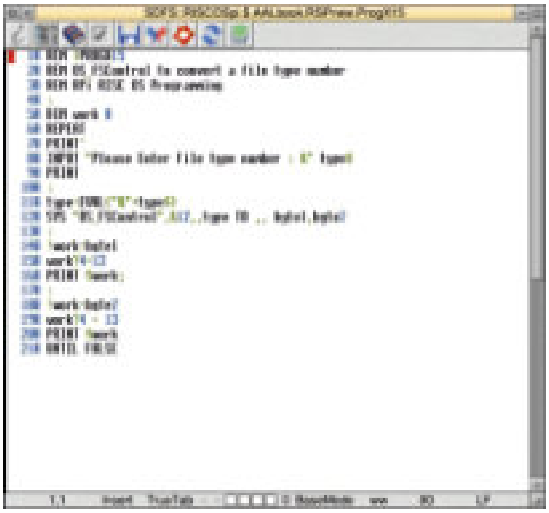
In addition to BBC BASIC, RISC OS Pi comes with a wide variety of compilers and programming languages – all free. Charm and Risc Lua are supplied with the standard distribution, while GCC and a variety of support libraries can be downloaded and installed simply in a matter of minutes.

There is also an abundance of educational software for a child that is easy to use, even for a person that has never touched a computer before. Scratch is a kids programming language that a four-year old would appreciate. Add to this a reasonable selection of games, both the good old ones and several new titles and you begin to understand what RISC OS really brings to the Raspberry Pi.

## Programmers Tool Chest

Organising your own programming tool chest will not take too long, and you will probably already have your own likes and dislikes. Much of what you will need is already present in the Raspberry Pi RISC OS Pi distribution. If you are not sure or want to investigate other alternatives then here are a few items I would suggest you get to know. Some of these you will find on the Desktop; others in the Apps directory.

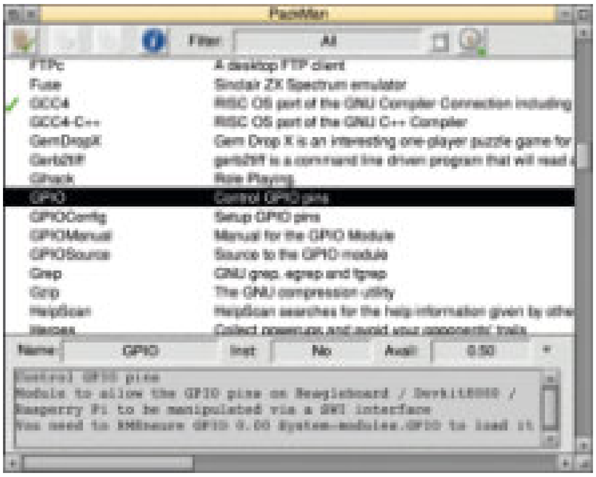
**StrongEd** is a full-feature programmer's text editor that is surprisingly easy to use. It can be used to edit just about anything and can do this in the context of the filetype being edited. This means that you can create a BASIC file and StrongEd will treat it like a BASIC program, saving it in the correct format. In this way you can also run the program from within the Editor so that you don't have to go to a BASIC task window, load and run it, before retreating back to make any corrections.



```
   10 REM >PROG2-1
   20 REM OS_FSControl to convert a file type number
   30 REM RPi RISC OS Programming
   40 :
   50 DIM work% 8
   60 REPEAT
   70 PRINT'
   80 INPUT "Please Enter file type number : &" type$
   90 PRINT
  100 :
  110 type=EVAL("&"+type$)
  120 SYS "OS_FSControl",&12,,type TO ,,byte1,byte2
  130 :
  140 !work=byte1
  150 work?4=13
  160 PRINT $work;
  170 :
  180 !work=byte2
  190 work?4=13
  200 PRINT $work
  210 UNTIL FALSE
```

Figure 2a. A BBC BASIC program being edited in StrongEd.

**PackMan** is a simple, clever and easy way to add software to your set-up and ensure you have the latest releases and updates. It does this by maintaining lists of software, and you should ensure that these lists are up-to-date by selecting the ‘Update Lists’ option from its icon-bar menu.



```
  FTPc          A desktop FTP client
  Fuse          Sinclair ZX Spectrum emulator
✓ GCC4          RISC OS port of the GNU Compiler Connection including
  GCC4-C++      RISC OS port of the GNU C++ Compiler
  GemDropX      Gem Drop X is an interesting one-player puzzle game for
  Gerb2RIP      gerb2rip is a command line driven program that will read a
  GHack         Role Playing
  GPIO          Control GPIO pins
  GPIOConfig    Setup GPIO pins
  GPIOManual    Manual for the GPIO Module
  GPIOSource    Source to the GPIO module
  Grep          GNU grep, egrep and fgrep
  Gzip          The GNU compression utility
  HelpScan      HelpScan searches for the help information given by othe
  Heroes        Collect powerups and avoid your opponents' trails

Name: GPIO        Inst: No        Avail: 0.50

Control GPIO pins
Module to allow the GPIO pins on Beagleboard / DevkitE000 /
Raspberry Pi to be manipulated via a SWI interface
You need to RMEnsure GPIO 0.00 System-modules.GPIO to load it
```

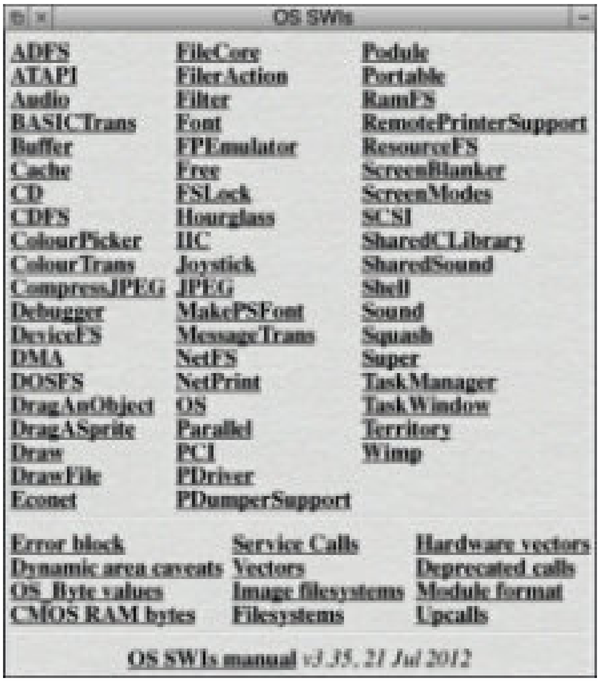
Figure 2b. The Packman window.

Once PackMan has checked their status you can choose to 'Update all' if you wish and Packman will take care of the process for you. By using Packman you should never have to download and install software by hand.

By scrolling through the alphabetical software list in Packman you can get descriptions of the programs and can see if updates are available. If you are considering C then you can get the latest version of GCC and libraries such as OSlib here too.

PackMan is based on an earlier packaging system called RiscPkg, but with a friendlier user interface (for example, the ability to move installed programs wherever you like). It is still in its infancy and you can expect to see new features as it is developed.

**StrongHelp** for most is an access point to StrongHelp manuals. A good majority of RISC OS software comes with hypertext manuals that invariably contain an incredible amount of information. Some examples of these are given later in this book. StrongHelp is also a simple hypertext authoring program which you can use to create your own documentation.



OS SWIs

ADFS
ATAPI
Audio
BASICTrans
Buffer
Cache
CD
CDFS
ColourPicker
ColourTrans
CompressJPEG
Debugger
DeviceFS
DMA
DOSFS
DragAnObject
DragASprite
Draw
DrawFile
Econet
FileCore
FilerAction
Filter
Font
FPEmulator
Free
FSLock
Hourglass
IIC
Joystick
JPEG
MakePSFont
MessageTrans
NetFS
NetPrint
OS
Parallel
PCI
PDriver
PDumperSupport
Podule
Portable
RamFS
RemotePrinterSupport
ResourceFS
ScreenBlanker
ScreenModes
SCSI
SharedCLibrary
SharedSound
Shell
Sound
Squash
Super
TaskManager
TaskWindow
Territory
Wimp

Error block
Dynamic area caveats
OS_Byte values
CMOS RAM bytes
Service Calls
Vectors
Image filesystems
Filesystems
Hardware vectors
Deprecated calls
Module format
Upcalls

OS SWIs manual v3.35, 21 Jul 2012

Figure 2c. StrongHelp manuals abound – here OS SWI calls are arranged by function.

StrongHelp manuals are stored in the !Manuals directory in the Utilities folder on the SD Disc. The riscos.info website hosts a collection of StrongHelp manuals for everything RISC OS. A link to this can be found on the book support pages on my website.

**Director** can be found in Apps. It installs on the icon bar and from here you can interrogate the status of your RISC OS setup. It is a tool that allows you to customise your desktop and create quick-access menus to anything within RISC OS. With it you can create menus and place icons on the icon bar. It also provides easy access to files and a lot more. You can look at system settings and also delve into things like modules and SWIs.

Director includes a scripting system which gives you the ability to do things such as map keyboard shortcuts and custom menus to almost action.

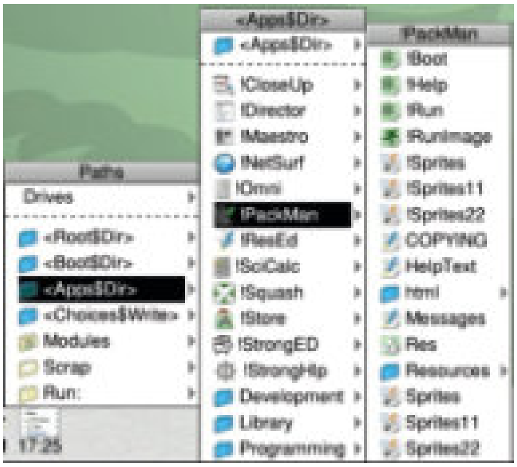


Figure 2d. Director menus allow you to interrogate features such as modules.

**SparkFS** is a compression filing system that gives you more disc space by saving and managing your files in a compressed — zip — format. It does so seamlessly without the need to resort to compressing and uncompressing processes – this all happens in the Filer. The bonus of this is that it maximises the use of your SD card space which can become quite tight if you are not using alternative storage options. There can be a slight overhead in some additional access time for compressed files, but for files and data that you are not accessing on a regular basis this should not be an issue. There are free and paid for versions.

**!MultiTask** is a multi-tasking desktop utility which allows you to examine the contents of files and to load, run and edit BASIC programs from the desktop. It is available from !Store (free of charge) and clicking on the icon installs it on the icon bar. Dragging and dropping a BASIC file onto this will load the program into its own window, from where you can type RUN to fire it up as though it was a multi-tasking program. This app is covered in more detail in Chapter 15 where it is used for a specific purpose.

This is only a small sample of my favourite apps that I consider essential for effective RISC OS *System Programming*. There are plenty of others; have fun discovering and utilising new ones.

## Fitting RISC OS Together

Figure 2e illustrates diagrammatically how RISC OS slots together all its major components. We shall discuss each of these in some detail as we go forward, so you may wish to come back and refer to this figure from time to time.

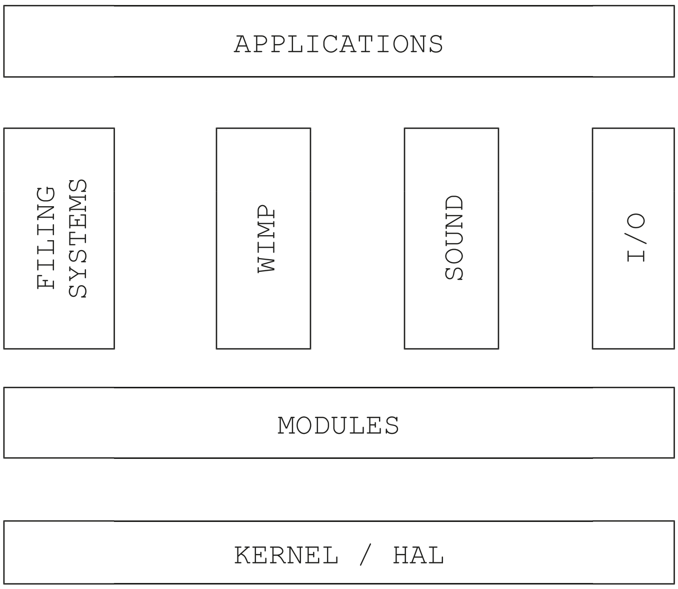


Figure 2e. RISC OS structural composition.

Holding everything together is the Kernel and Hardware Abstraction Layer. The HAL was introduced to RISC OS 5 as a way of making interfacing to different hardware setups and system chips easier. This means that RISC OS is no longer dependent on a fixed set of core chips. In the case of the Raspberry Pi the HAL has been configured to work with Broadcom's BCM2835 System-On-a-Chip. The Kernel supplies the basic core routines that make RISC OS function and interface with both the HAL and the outside world. Everything goes to the hardware via the Kernel.

Sitting on top of the Kernel/HAL are the modules. Modules provide a way of extending RISC OS giving it additional functionality. Anyone can write a Module and interface it to RISC OS by using a standard that we'll discuss later in this book. One of the first Raspberry Pi specific modules to be developed and released was one to control the GPIO port.

From the user point of view the four main operating components of RISC OS are the Filing System, the WIMP (Desktop), the Sound System and the Input/Output interfaces. These all sit alongside each other and much of their functionality is provided with module support.

Finally applications run across all these and will invariably require support from each of them. Each application will call on many of the components underneath it, indeed applications may also be implemented as modules, so it shows how integrated RISC OS is a system being

## Soft Interaction

Figure 2f shows diagrammatically how the command/call hierarchy fits together in RISC OS. We shall discuss each of these in some detail as we go forward, so you may wish to come back and refer to this figure later on.

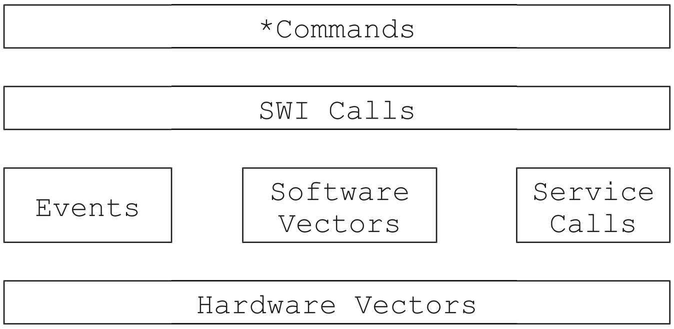


Figure 2f. The RISC OS Command/Call structure.

* Commands provide the standard CLI method of accessing operations in RISC OS. By definition they are language-independent so can operate across all software platforms. In almost all instances ( but not every one) *commands have underlying SWI calls, which are provided by either the Kernel or an associated module. These in turn are supported by a trio of calls that provide three different types of RISC OS functionality. Virtually all operations are directed through Software Vectors — these provide direct control to specific locations in the Kernel or associated module. Service Calls allow RISC OS to inform all the components of RISC OS that they may need to perform

housekeeping operations. Events are actions that can be configured to occur at pre-defined times in relation to certain conditions being met.

Underlying all this are the hardware vectors; these are hard wired into the ARM chip and control access to critical operations. When you first turn on the Raspberry Pi it starts its boot-up process by accessing functions through the hardware vectors.