Support Group Application Note
Number: 202
Issue: 1.03CS
Author: James Bye

Writing Games for RISC OS

This application note outlines what is required by a game running under RISC OS and the considerations that need to be taken into account by the games writer. RISC OS computers, unlike Acorn's previous computers such as the BBC Model B and Master 128, run in a desktop environment where other applications are running and processing data. The games writer needs to take this into account and hence enable the user to start the game without the loss of any data or other disruption to the computer of any kind.

Applicable Hardware :	All Acorn RISC OS based computers	Related Application Notes:	None

Support Group
Acorn Computers Limited
Acorn House
Vision Park, Histon
Cambridge CB4 4AE
England

Index

1. Background

RISC OS is a multi-tasking environment, where many applications may be running at the same time. Any one or more of these applications may be actively editing data and it is important that the subsequent execution of a game does not result in the user losing valuable data.

On certain systems, other than those provided and manufactured by Acorn, the booting of such games is totally different. There are, for example, games for other machines where the manual actually states that to load the game you must first remove the system disc from the floppy drive, power down the machine, insert the game disc and then power up the machine to re-start the game. As you can see, doing such a thing in a RISC OS world could have adverse effects on the user which can result in the loss of data. In an environment such as RISC OS, more care and consideration needs to be used in the interests of the user.

2. The structure of a game application

Any application written for RISC OS should follow the structure that has been described in the RISC OS Style Guide and it is important that games follow the same structure. The game will be contained in an application directory and, for this example, we shall use the directory name !Game. The structure of the game application directory could be as follows :-

!Boot	*often found in the application directory but might well be unnecessary. The filer will automatically load !Sprites if the !boot is not present. You should only include the !Boot file if you wish to set run or load actions.*
!Sprites	*spritefile passed to *iconsprites by the filer (or the !boot file). This should contain the sprite to represent the application icon in the directory display.*
!Run	**Run by the filer when the application is double clicked on in the directory display. This should set environment variables as required and *iconsprites the !Sprites file.*
!Help	*the game application's documentation*
!RunImage	*the main executable part of the game*
Modules	*a directory containing any modules loaded by the game application (any Acorn modules should be inside !System and not inside the game)*
Sprites	*a directory containing the Game's own personal sprites used during the running of the game. Game's may have many different sprite files so it is useful to keep them in one place within the !Game.*
Code	*a directory containing other bits of code that are loaded and run by the !Game.*
Resources	*a directory containing assorted resources needed by the !Game (such as data files etc...)*

Of course, it is up to the games writer to decide on the structure of the !Game directory but it always a good idea to have a tidy and maintainable structure. The !Game directory must contain the !Sprites, !Help and !Run file but we recommend that the other items listed above are also included if they are necessary. It is up to games writers to decide whether they wish to add extra directories or files.

A game application is the same as a normal RISC OS application and help should be provided in the standard way. The standard RISC OS way is to have a text file inside the application directory called !Help. This provides the user with information about the application from the directory display menu. However, the game itself is not restricted to providing this as the only help available. Many games provide help when the !Game has been run, which is optional by user selection. Another common method under RISC OS is to provide a second application directory on the disc that, once run, provides the user with the documentation and information that he/she requires. This application is commonly called !ReadMe or !GameInfo (or similar).

It is worth noting that the use of a !Help file is the most common way a RISC OS application provides help to the user. Hence, using this method provides the user with a standard environment.

4. The execution and termination of a game

Any game should be in the form of a standard RISC OS application, as described earlier in this document. Hence, the user should be able to start the game as if it was any other application. This is carried out by double clicking on the !Game application in a directory display. The next steps that are taken are really up to the game entirely. It is most common for the game to start right away and take over the whole screen (if it needs to do so). Another method, that is currently being used by some games, is to install an icon on the iconbar like a normal application. The user can then set any configuration options, such as redefining keys or loading saved data, in the desktop environment. The game can then be started by selecting a menu option or clicking on the icon on the iconbar. The latter approach is preferred as it provides a similar environment to other applications.

Quite a few games in the RISC OS world still rely on Shift-Break to start them. This is a legacy from the BBC Model B world and this approach should no longer be taken. The user should start the game as if it were any other RISC OS application.

At any point in the game, the user should be able to terminate it and return to the desktop. A recommended method for this is the use of the Escape key whilst the game is running, or by selecting an option when the user is at the 'start-up' screen. Users should be able to return to a desktop that is in the state that they left it, if possible. Many users have plenty of free memory and in many cases no memory grabbing techniques are required. On no account should running the game result in the unexpected loss of data. The CMOS RAM settings must be in the same state as before the game was started. The CMOS RAM settings should not be touched under **any** circumstances. It is permissible to change the font cache, system sprite area or other transient data areas as long as the user is informed. In this case the only return to the desktop should be via a reset.

5. Ensuring there is enough space for execution

It is important that all games written for Archimedes computers work on the base machines. The base machine is currently a 1 Megabyte machine with a single floppy disc drive. On a base machine, the user will have about 720K (default) of free space for the game to use. The maximum amount of free memory for a default machine can be changed by shrinking the font cache and system sprite area to zero, and this can increase the amount of free memory to approximately 752K. This extra memory can be grabbed by the game under the consent of the user. Of course, a more complicated game may need a higher specification to run (e.g. a 2 Megabyte machine). If this is the case then the game should clearly state this to the purchaser. Please note that RISC OS 3 version 3.10 provides 24k less free RAM in a machine in its default state compared to that of an equivalent RISC OS 2 machine (696k compared to 720k).

In the absence of a suitable operating system interface in early Archimedes computer systems, some games developers used a variety of highly undesirable techniques for ensuring that host machines had sufficient free memory to run their games. As stressed elsewhere in this application note, actions such as changing CMOS RAM configuration or altering memory allocation are totally unacceptable in an application running in the RISC OS multi-tasking environment. However, since the majority of games run as single tasks a degree of configuration management is acceptable provided that it is :-

- necessary
- initiated with the user's consent
- transient in effect

As with any application, a game ought to be able to determine the amount of free memory it requires to start up in; if sufficient memory is available the game should start immediately.

If free memory is currently insufficient, the game should alert the user to the problem with a suitable dialogue box :-

If the user chooses to continue with execution, the game should first close the desktop (thereby offering the user the opportunity to save unsaved data); it is then at liberty to make any necessary changes to the machine's memory allocation. The author of the game application should use the Desktop Closedown protocol to ensure that any applications that have unsaved data give the user the chance to object. Example code on how to do this can be supplied on request via Developer Support.

A memory allocation module is available for the purpose of configuring memory areas, called MemAlloc. It enables a variety of memory areas to be altered dynamically (without recourse to CMOS changes or simulated resets) and it is available under licence by contacting Acorn Developer Support. Note that MemAlloc will not normally be licensed for use with products other than games. For more information on the functionality of MemAlloc, please read Appendix D.

In the event of MemAlloc being unable to free up sufficient memory it is also possible for game applications to retrieve memory from the Relocatable Memory Area (RMA) by RMKilling selected modules. It has to be said, however, that this technique is not totally reliable as its effect cannot be guaranteed across all variants of RISC OS. For a list of modules that can be RMKilled, then please see Appendix C. The RMA does suffer from fragmentation, so the killing of modules does not always ensure that the free memory will be returned for general use. To retrieve the memory after killing modules, you will have to issue the command *RMTidy.

It is recommended that a game which makes ***any*** change to the host machine's memory allocation (be it via MemAlloc or manipulation of the RMA, or both), should end its execution by instructing the user to perform a hard reset, e.g.

```
Execution of !Game required some changes to your machine's internal
memory allocation. Reset your computer to restore your machine's
normal configuration.
```

6. Saving the game state

It is quite often desirable, especially with tactical or adventure games, to allow the user to save the state of play at a particular point in the game. With RISC OS being a 'multi-disc' environment (i.e. having hard discs of different varieties - SCSI, IDE, ST506), it is important that the games writer does not rely on saving the game to a specific filing system or disc (e.g. save state to the root directory of the floppy disc). As explained earlier in this document, it is a good idea to have your game inside an application directory. In the !Run file you should set up the system variable 'Game$Dir' which will give you a pointer to your application directory (so that you do not have to be filing system specific). The game state can then be saved inside the application directory using a filename such as '<Game$Dir>.SavedState'.

However, a better method of saving the game state is to use the RISC OS filetype reserved for the purpose and allow the user to drag the file to any filing system that he feels fit. This method is only really possible when the game installs itself on the iconbar (explained earlier in this document). When the user terminates the game and returns to the desktop the game's icon should still be on the iconbar. The iconbar icon will have a menu from which the user can drag a standard RISC OS save box to save the state of the game when it returned to the desktop.

Acorn has a simple generic "saved-game" filetype that can be used by any game that is registered to use it. The format consists of a 16 byte header, which identifies the game with a name and unique number. This can then be dragged to the game icon on the iconbar and the game would enter at the stage that it left off.

Note that it is important that, when saving files, the game is never reliant on specific filing systems or filing system names.

If you do wish to use the saved game filetype, then please get in contact with Acorn Developer Support regarding its current status.

A full explanation of the saved game filetype can be found in Appendix A.

7. Interfacing to Joysticks

RISC OS 3 version 3.10 contains a module which allows third party software to interface to the built-in Joystick hardware (this is available on certain machines). Full information on the interface to this module can be found in Appendix B.

8. Software protection

It is important for the games writer to take into account that many users now have hard discs in their machines. Many owners of Acorn Computers have hard discs. Hence, for a game to insist on only running from a floppy drive due to software protection is not a route that most users would find acceptable. Of course, we do not mean that games writers should release un-protected software, but there are possible alternatives.

A common approach used by other application writers, as well as games writers, is to use an approach called 'KEY-DISCING'. This method permits the user to install the game onto the hard disc and then execute it from there. The protection comes when the game asks the user to insert the floppy disc supplied with the game to read a protected sector or track. If you do decide to use the 'KEY-DISCING' approach then it is important that you still allow the user to install the game onto his hard disc. Subsequent running of the game from the hard disc would ask the user to insert the original disc to read the protected sector or the like. This gives the user the benefit of speed by loading from the hard disc but also allows protection of the software.

Other methods that are worth considering are limiting the number of times they can install on his hard disc. Another possible method is to have a 'registration' scheme where, when the user purchases a copy, it will be protected and will run from floppy only. The user would then be able to send his disc back to the software house and have the disc un-protected and registered. The user would now be free to install the game on his hard disc, but every time he runs the software his registration name and number will appear on the screen. Hence, if the software is distributed without prior permission of the author, the tracing of the illegal distribution is made easier.

9. Screen mode independence

There is increasing diversity of monitor types being used with Acorn computers. VGA and Multiscan monitors have become more common. It is important that no assumptions should be made on the field refresh rate as monitors will vary and VSync may change. All timing should be derived from a stable base e.g. Timer1 or 100hz tick.

A special VGA compatible mode has been developed specifically for game style applications that take over the whole screen. Details of these modes can be found in appendix E.

10. Processor dependencies

With the increasing range of ARM processors being used, together with varying cache and bus clock speeds, it is important that a Game does not make any assumptions on processor performance to time the game. For example, timing through the use of software loops should not be used. The appropriate operating system calls should be used for timing purposes.

As many developers are already aware, there are some machine code sequences that should not be used with the ARM processor as they cause unstable states. An ARM Code check program with further information can be provided by Developer Support to enable developers to test their applications for the presence of these code sequences.

11. Direct addressing of the Video Controller (VIDC)

It is known that many games style applications need to poke the Video Controller (VIDC) directly to achieve certain special effects. These special effects include some of the following :-

- Palette Switching
- Hardware Scrolling
- Setting Video parameters

Although addressing the hardware for VIDC1 Video Controller works, the code used to address the controller directly will not work on new Video Controllers.

It has been recognised that it is necessary for games to achieve special effects, so a module has been developed to provide a SWI veneer to addressing the Video Controller. The module is called Screen Tricks and details of this can be found in Appendix F.

12. RISC OS

It is important for all software houses, including games software houses, to ensure that their software will work effectively of all versions of RISC OS. Hence, it is important that developers check that their software runs on all versions.

13. Pitfalls to avoid when writing games for RISC OS

Ever since the Archimedes range of Computer was first released, software houses have been writing innovative games for the machine. The aim of this section is to learn from previous mistakes that have either left the user's machine in an unuseable state or conflicted with desktop operation.

Altering CMOS RAM -	In the past, some games have re-configured the CMOS RAM to enable the application to run. Altering the CMOS RAM, even though it may be temporary, is very bad practice and should never be carried out.
Faking a reset -	This approach SHOULD NEVER BE TAKEN. If the game has been run in a low memory condition then it is up to the user to reset the machine and not the game application
Hardware addressing -	There are now many different hardware platforms in the Archimedes range of computers. Hence, we advise that games writers avoid addressing the hardware wherever possible and that they follow the standard operating system routes. In 1991, Acorn released the A5000, which has dramatically different hardware to previous machines.
Specific system reliance -	In the past, many games have relied on the user having a specific system (e.g. a 1 Megabyte A3000 with a single floppy drive) and then have designed their game taking into account this environment only. Many users now have machines with larger memory capacity, multiple floppy drives, hard discs in many flavours (the list is endless) and it is important that the game will run on any hardware platform in the Archimedes range, no matter how it has been configured.
Shift-Break -	Using Shift-Break to start the game should no longer be used. This method of starting a game is a legacy from the BBC Model B days and with the introduction of RISC OS and the desktop environment is no longer necessary. The games writer should follow the guidelines provided earlier in this document to find out how the game should start-up.

14. Summary

If the points in this document are taken into account then the games writer should be able to produce a RISC OS compatible item of software.

The basic rule when writing a game for RISC OS is :-

> *"Take the user into account - allow the user to start the game from the desktop, without the loss of any data, and then allow an easy return on termination of the game to a machine that is in the same state."*

If you are unsure about anything that has been described in this document then please get in contact with Acorn Developer Support (if you are a Registered Developer) otherwise you should contact Customer Services. It is important from Acorn's and all third parties points of view that only RISC OS Compatible or Compliant software is released for the Archimedes range of machines.

Appendix A: the "Saved Game" filetype

A generic filetype exists for data saved from a game (e.g. an adventure game). Hence, the loading of the saved game file will cause your game to re-start at the place that you left off.

FileType specification

The filetype is &AE8 and has the name 'SaveGame'. To allow this filetype to be used by many different games then the file must have a specific header so that the required game can recognise its own saved data. The file header will be as follows :-

Size	Description
4	A single word Identifier
12	Name of application that created it (null terminated)

This gives the file a 16 byte header followed by the data written by the application/game. The data will be at byte offset 16 in the file.

As with any generic filetype, you may encounter the case where this filetype will be loaded into your application and its data is for another. In this case, you should explain this to the user in an error box similar to the following :-

Allocations of this filetype

If an application or game wishes to use this filetype then it will obviously need two items reserved for its own use. These are as follows :-

A single word identifier (starting from &0000)
A 12 character name

These will be allocated by Acorn on request.

Appendix B: SWI Interface for Joystick Devices

The Joystick module provides a SWI interface for reading the state of a joystick. When the module initialises, it tests for the existence of built-in joystick hardware and if it does not find any then it will not initialise. Third parties can replace this module to provide support for different hardware. It is recommended that any such modules have version numbers greater than 2.00 so that Acorn can upgrade its own module without preventing its replacement.

SWI Joystick_Read (SWI &43F40)

Returns the state of a joystick.

On entry: R0 = joystick number

On exit: R0 = joystick state

Interrupts: Interrupt status is not altered

Processor mode: Processor is in SVC mode

Re-entrancy: Not defined

Use: This SWI is used to obtain the state of the requested joystick. The state is returned in the following format:

Byte 0: signed Y value (range -127 to 127)
-64, 0 or 64 for single switch joystick
(corresponds to Down, Rest, Up).

Byte 1: signed X value (range -127 to 127)
-64, 0 or 64 for single switch joystick
(corresponds to Left, Rest, Right).

Byte 2: Switches (e.g. fire buttons) starting in
bit 0, unimplemented switches return 0.

Byte 3: Reserved.

Note that this format allows both digital and analogue devices to be supported. Applications which are only interested in state (up, down, left, right) should not simply test the bytes for positive, negative or zero. It is recommended that the 'at rest' state should span a middle range, say from -32 to 32 since analogue joysticks cannot be relied upon to produce 0 when at rest.

Appendix C: Retrieving modules from the RMA

As discussed earlier in this document, in low memory conditions it is permissible for the game application to kill certain modules that are not used by the game itself, to be able to free as much memory in the machine as possible. This should, of course, ONLY be carried out in conjunction with the warning sequences, desktop closedown and non-return to the desktop as explained earlier in this document.

The list differs between machines running RISC OS 2 and RISC OS 3. When you are RMKilling modules in low memory conditions, you should ensure that none of the modules are needed by your application. Please note, that you should only RMKill modules and **never** UNPlug them.

The list of the modules is as follows. You should ensure that you *RMTidy the module area after you have killed the modules to truly free the space.

RISC OS 2

Desktop	SystemDevices	FileCore%RAM	Econet
NetFS	NetPrint	BBCEconet	International
InternationalKeyboard	Debugger	IIC	SoundScheduler
StringLib	Percussion	SpriteExtend	Draw
FontManager	WindowManager	TaskManager	PaletteUtil
Filer	ADFSFiler	RAMFSFiler	NetFiler
ShellCLI	Hourglass	NetStatus	SpriteUtils
Podule	SharedCLibrary	ARMBasicEditor	RamFS
MIDI	ColourTrans		

RISC OS 3

DOSFS	!Edit	!Paint	!Draw
!Configure	StringLib	Filer_Action	SharedCLibrary
Percussion	FilterManager	WindowUtils	WindowManager
TaskWindow	SystemDevices	SysInfo	SuperSample
Squash	SpriteUtils	SpriteExtend	SoundScheduler
ShellCLI	ScreenBlanker	RTCAdjust	ROMFonts
ResourceFiler	RAMFSFiler	PipeFS	Pinboard
PDumperDM	PDriverDP	PDriver	PaletteUtil
NetStatus	NetPrint	NetFiler	BroadcastLoader
NetFS	InternationalKeyboard	International	IIC
Hourglass	Free	FontManager	FilerSWIs
Filer	BBCEconet	Econet	Draw
DragASprite	Debugger	ColourTrans	BASICTrans
BASIC	ADFSFiler	Printers	!Help
!Chars	!Calc	!Alarm	TaskManager
Desktop	Podule		

Appendix D: Functionality of MemAlloc

Developer Support can provide a tool for games developers that allows specific memory areas in the machine to be reconfigured to the desired values (ie: the font cache). MemAlloc only provides 'soft' memory settings and all memory areas will return to their original state after a reset. MemAlloc comes in the form of a module and supports the following *Commands :-

`*SystemSize <size> [<*Command>]`	Allows the setting of the SystemSize in KBytes. SystemSize reserves an extra area of memory for the system heap after all modules have been initialized.
`*RMASize <size> [<*Command>]`	Allows the setting of the size of RMA in KBytes. RMASize is used to reserve an extra area of memory in the Relocatable Module Area after all the modules have been initialized.
`*ScreenSize <size> [<*Command>]`	Allows the setting of the size of the screen in KBytes. ScreenSize reserves an area of memory for the screen display.
`*SpriteSize <size> [<*Command>]`	Allows the sprite size to be set in KBytes. SpriteSize is used to reserve an area of memory for the sprite system.
`*FontSize <size> [<*Command>]`	Allows the font size to be set in KBytes. FontSize is used to reserve an area of memory for the Font Cache. Fonts are stored in the Font Cache to save reading them from the filing system; hence they can be displayed much more quickly.
`*RAMFSSize <size> [<*Command>]`	Allows the size of the RAM disk to be set in KBytes. RAMFSSize is used to reserve an area of memory for the RAM Filing System (when the RAMFS module is present). If this is set to 0 (zero) the the RAM filing system will be disabled.
`*RMAFree  <size> [<*Command>]`	Allows the amount of free space in the Module area to be set in KBytes. This will also increase the the RMASize in proportion.
`*SpriteFree  <size> [<*command>]`	Allows the amount of free space in the Sprite Area to be set in KBytes. This will also increase the SpriteSize in proportion (see the note on the command RMAFree).
`*FontFree  <size> [<*command>]`	Allows the amount of free space in the Font Area to be set. This will also increase the FontSize in proportion (See the note on the command RMAFree).

The optional *command (denoted above as [<*command>]) will be executed if the required amount of memory was not available.

A typical !Run file, that uses MemAlloc, would look something like the following :-

```
RMLoad     <Obey$Dir>.MemAlloc
RMASize 0k
FontSize 0k
SpriteSize 0k
RamFSSize 0k
SystemSize 0k
SpriteSize 80k
ScreenSize 80k
RMKill MemAlloc
```

Appendix E: The Games Modes module

The games modes module is a module that has been specifically developed for the use of game style applications which take over the whole screen. This module is intended to support the use of VGA monitors, but with lower demands on processor power and screen memory than the standard VGA mode 27 and 28. The games modes module provides two modes and these are as follows :-

Mode No.	Dimensions	BPP (no. Colours)	Screen Memory	Vertical Refresh
48	320 * 480	4 (16)	96k	60 Hz
49	320 * 480	8 (256)	160k	60 Hz

This module is available from developer support on request.

Appendix G: Other relevent documentation

This application note does not cover all the aspects of writing games applications for RISC OS and other technical documentation maybe relevant. Other relevant documentation includes the following :-

Title:	The RISC OS 3 Programmer's Reference Manual
Publisher:	Acorn Computers Limited
ISBN:	1 85250 110 3
Available:	For Registered Developer's, via the Developer Discount Scheme

Title:	Acorn Assember Release 2 Users Guide
Publisher:	Acorn Computers Limited
ISBN:	1 85250 096 4
Available:	For Registered Developers via the Developer Discount Scheme (documentation supplied as part of Desktop Assembler Pack)

Title:	Acorn C Release 4 User Guide
Publisher:	Acorn Computers Limited
ISBN:	1 85250 095 6
Available:	For Registered Developers via the Developer Discount Scheme (documentation supplied as part of the Desktop C pack)

Title:	BBC BASIC Reference Manual VI
Publisher:	Acorn Computers Limited
ISBN:	1 85250 103 0
Available:	For Registered Developers via the Developer Discount Scheme

Title:	Acorn RISC Machine Family Data Manual
Publisher:	Prentice Hall, Inc.
ISBN:	0 13 781618 9
Available:	VLSI Sales Office 486-488 Midsummer Blvd Saxon Gate West, Central Milton Keynes MK9 2EQ United Kingdom Phone: 0908 667595 Fax: 0908 670027

Some of the titles mentioned above depend on the language used in programming.