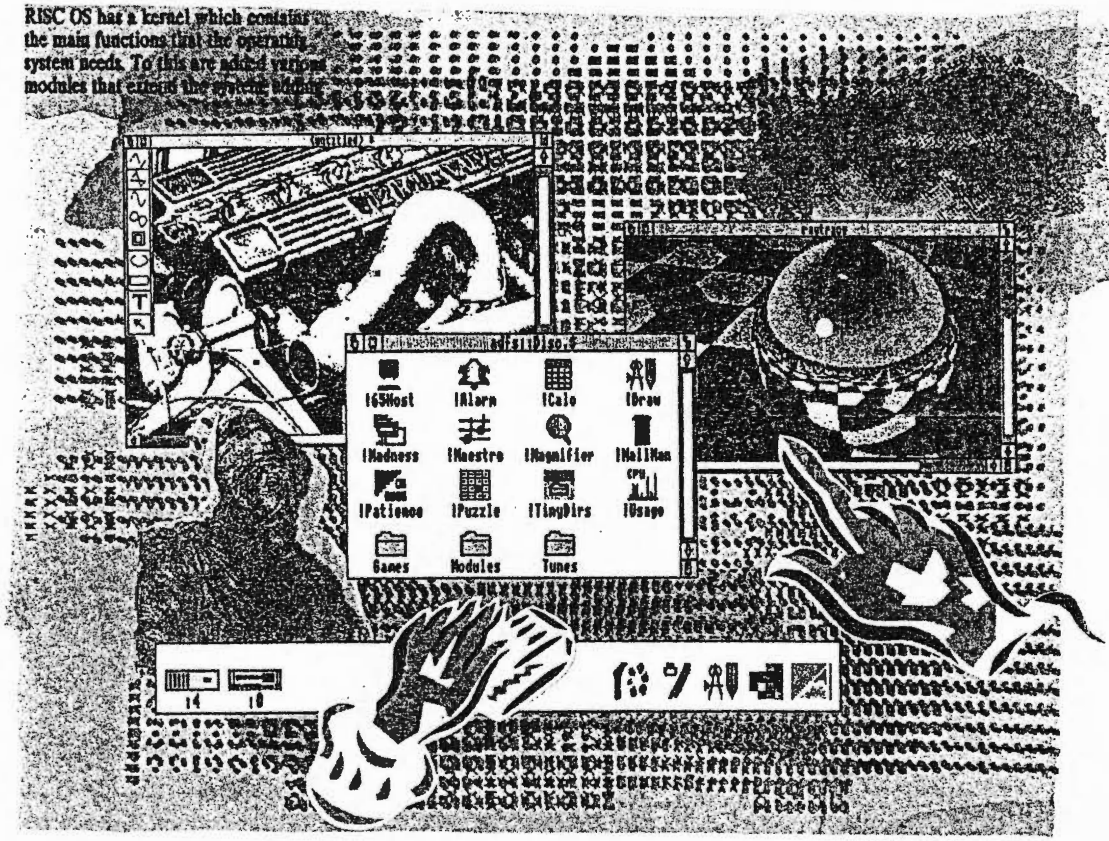
RISC OS STYLE GUIDE



RISC OS has a kernel which contains the main functions that the operating system needs. To this are added various modules that extend the system's ability
!65Host
!Alarm
!Calc
!Draw
!Madness
!Maestro
!Magnifier
!MailMan
!Patience
!Puzzle
!TinyDirs
!Usage
Games
Modules
Tunes

Acorn
The choice of experience.

If you have any comments on this Guide, please complete and return the form at the back of the Guide to the address given there. Any other correspondence should be addressed to:

Customer Services
Acorn Computers Limited
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN

Information can also be obtained from the Acorn Support Information Database (SID). This is a direct dial viewdata system available to registered SID users. Initially, access SID on (0223) 243642: this will allow you to inspect the system and use a response frame for registration.

Published January 1990 by Acorn Computers Limited

Issue A

Contents

About this Guide

About this Guide

This Guide tells you the standards of *'look and feel'* to which you should write a RISC OS application. It is split into several chapters:

- The first chapter introduces you to this Guide, explaining its scope, and why a standard *'look and feel'* is desirable.
- The next chapter defines terminology that is used in this Guide, and in RISC OS in general. If you use this standard terminology in your applications, and in their manuals, users will find them easier to use.
- The remaining chapters cover specific areas of RISC OS applications, explaining how they must conform to the standard 'look and feel', and how they should conform.

Finding out more

You will find a certain amount of relevant information in the *Welcome Guide* and *User Guide* supplied as standard with all RISC OS computers.

The *Programmer's Reference Manual* gives full documentation of RISC OS, and all the calls to the operating system that you may need to use in your code. The chapter entitled *The Window Manager* is especially relevant, and tells you how to implement many of the standards defined in this Guide.

Languages

If you wish to write your applications in C, you will find the manual supplied with Acorn's ANSI C compiler useful. If you have Version 2 or earlier of the compiler you are strongly recommended to upgrade it to the current version, as this contains far more extensive support for writing RISC OS applications.

- The chapter entitled *How to write desktop applications in* C is especially relevant, as it details the RISC OS library.
- The chapter entitled *How to use the template editor* is also useful; you can use the template editor to interactively design windows for the desktop.

Other specialist programming languages are available from Acorn suppliers for your RISC OS computer; you will find their manuals useful if you are using them.

If you wish to write BASIC programs on your RISC OS computer you will find the *BBC BASIC Guide* useful.

If you wish to write your applications in assembler, you will find the manual supplied with Acorn's ARM assembler useful.

Introduction

Why have a standard?

One of the most important and powerful aspects of RISC OS is that applications within the desktop world should present a consistent and reliable interface to a user. Multiple applications can then work together in a uniform and powerful environment, in a way that is easy enough to learn and to use that a user will want to work with it instead of being forced to.

This is to the advantage of you, the software vendor, as well as the software user. Studies show that the average Macintosh user buys and can use considerably more packages than the average PC user. The most frequently quoted reason for this is that it is easier to learn a new package if the environment for a user is standardised.

The scope of this Guide

This Guide is aimed at those specifying, planning, or implementing software that is to work within the RISC OS desktop environment. It tells you how such applications must look and feel to a user. A lot of the points it raises are ones you will have to bear in mind quite early on when considering the writing, or porting, of applications packages.

Much of what is said below is to do with consistency and standards. It tells you what you must do to conform to the RISC OS standard, and what you should do where possible.

The scope of this Guide is so large that in places it is necessarily imprecise. We believe that it provides enough specific cases that its aims and spirit can be clearly understood. The effect of this Guide should be to ensure you give a minimum level of service to a user, and never to limit your innovative and imaginative ideas.

The criteria described in this document are extremely demanding, and in places require significant effort to implement. Some of the Applications Suite itself does not strictly conform, in places. The standard to which we all aspire will evolve continuously as RISC OS evolves and improves.

Terminology

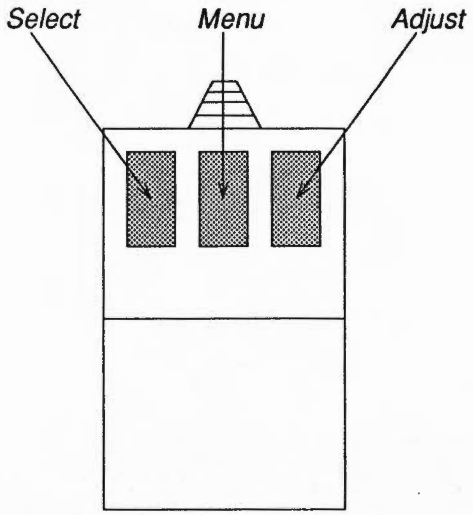
Introduction

You must always bear in mind that the main users of RISC OS are users and not programmers. If you use consistent terminology, and avoid jargon, it will make RISC OS more friendly to them. Your application's prompts and documentation should use the terminology outlined in this chapter, and used in the rest of this Guide

Because RISC OS is not solely a desktop operating system (eg a user has access to the command line interpreter, and non-windowing applications such as BASIC) you will inevitably have to let some jargon slip through, but you should always minimise this.

Mouse buttons

The mouse has three buttons:



The buttons have these names because of the actions they perform:

- *Select* is used to make an initial selection
- *Adjust* is used to toggle elements in and out of this selection and to add extra selections without cancelling the current ones
- *Menu* is used to call up a menu.

The mouse moves a *pointer* on the screen.

Mouse operations

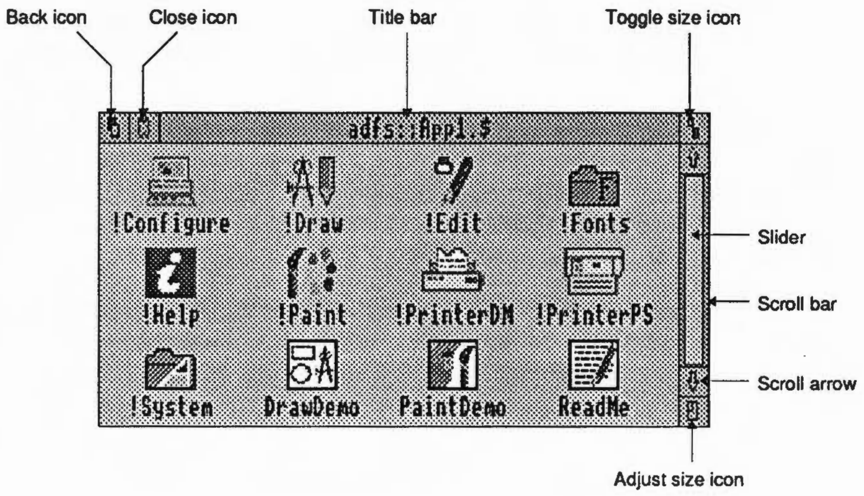
These are the terms you should use for mouse operations:

Press	press a button down
Release	release a button
Click	press and release
Drag	press and move the mouse, or press for more than 0.2s
Double-click	press, release, press, release within 1s without moving the mouse
Choose	what you do to a menu option
Type	what you do to keys on the keyboard
Select	change an object's state by clicking on it.

It is a common fault to confuse *press* and *click*, and to talk about *selecting* menu options.

Parts of a window

The icons around a window have the following names:



Use these names with initial capitals in running text.

Other parts of the desktop

You must use this terminology to refer to other parts of the desktop:

- A *menu* has *options* or *menu items*, some of which lead to *submenus* (no hyphen).
- A chosen menu option is shown *highlighted* (no need to say 'in inverse video').
- A box to which a user has to respond by clicking on an icon is a *dialogue box*.
- The bar at the foot of the screen is the *icon bar*.
- A menu that appears when you press **Menu** over an icon on the bar is an *icon bar menu*.
- Directories are shown in *directory displays*.

Editors

An *editor* is an application that presents files of a particular format as abstract objects which a user can load, edit, save, and print. Text editors, word processors, spreadsheets and drawing programs are all examples of editors (in this context). Their data files are referred to as *documents*.

Each document being edited must be displayed in a window. Such windows are referred to as *editor windows*.

An editor must record, for each document currently being edited, whether a user has made any adjustments yet to the document. This is done using a *modified flag*.

Most editors are capable of editing several documents of the same type concurrently; these are known as *multi-document editors*. Others can edit only one object at a time; these are known as *single-document editors*.

General principles

Ease of use

Ease of use is what Wimp systems are about most of all. All of the various elements described here are ultimately designed to make the computer easier and more pleasant to use, over a wide range of user experience and practice. An application should be:

- easy to learn
- easy to relearn
- easy to use productively.

These things can conflict with each other, and with other things (eg system cost, program size, program development time, backwards compatibility). Design is not easy, and not all users agree.

Consistency

The multi-tasking Wimp emphasises that applications work together for a user of the machine:

- they cooperate in sharing the machine
- they look harmonious
- their user interfaces are similar
- the whole is more important than a single application.

If you port an application into the desktop environment, check that it works well with the existing applications and utilities. Strive to ensure that a habitual user of the desktop environment and the Appplications Suite programs will find your program easy to use, and natural to learn.

Quality

It is much better that you write a small program that does something simple, and does it well, than a sprawling mass that crashes occasionally. With a view to this:

- do not bypass operating system interfaces or access hardware devices directly
- do not peek and poke page zero locations (the hardware vectors etc), or kernel workspace.

Such tricks may well not work on future machine and operating system upgrades. Acorn will pursue a policy of continuous improvement and expansion for its product lines; build your software to last.

Using the Wimp

You must use the RISC OS window system for functions it can perform, such as window manipulation, menu construction, etc. You must use the mouse to set screen positions.

Different configurations

Your application must work with any reasonable hardware configuration that runs RISC OS. It must run from ADFS, NetFS or any other filing system.

You must not make any unnecessary assumptions about:

- filing systems
- peripherals
- precise processor speed
- precise ROM or module versions (at or beyond RISC OS 2.00)
- video refresh rate
- video/palette hardware
- peripheral controller hardware
- memory management hardware
- screen mode
- fonts available
- native language
- alphabet / character set.

When your application cannot run on the current configuration, it must tell a user why not in a comprehensible way, rather than crashing or misbehaving. Your application must not reconfigure the machine or unplug modules so that it can run.

Use of memory

In a multi-tasking environment, you must use memory as sparingly as possible. If your application needs varying amounts of memory (for instance as documents are loaded and unloaded) you must claim extra memory as you need it, and free it when you have finished using it.

If your application runs out of memory then it must degrade gracefully, and not crash. If there is not enough memory for it to start, then it must say so (and leave the machine cleanly) rather than crashing the machine in an attempt to start. Try to work out your application's total memory needs (including any modules it needs to load) before you load anything, rather than loading a lot of modules and then finding that your main application doesn't have enough memory.

If your application allows a user to dynamically change how much memory it has, it should do so by using the Task Manager display.

File handling

The model of files and filing systems presented within the RISC OS Desktop is that files are always manipulated by their full pathname, including the filing system name, disc title, etc. This gives each file in the system a unique name. There is no concept of 'current directory', so you should not refer to, or rely on, its being set. Every effort is made to ensure that users never have to type a full pathname, but they do have to see and (more or less) understand them.

RISC OS makes heavy use of file types. You must give all files you create a file type and date-stamp, rather than using the older load/exec address form (but you must be prepared to encounter these, and respond correctly).

Never build absolute drive numbers, file names or filing system names into your program.

Supporting !Help

You must support the !Help application to help new users. The technical details you need to do so are included in an appendix at the end of this Guide.

What help you should provide

The text should consist of simple complete English sentences, each starting on a new line and ending with a full stop. The sentences should usually be simple imperatives or information such as:

- Click SELECT to set the alarm.
- The pointer is at pixel (47, 215) within the sprite.
- You are in Select mode.
- Click ADJUST to change to path edit mode.
- This is the icon for Edit.

In general you need not mention menu entries, except when specific ones interact with pointer operations. As a general rule present information of interest to the beginner near the top, and expert tips or information lower down.

You must use the terminology we've already defined. For mouse operations you must use initial capitals (eg Click). The mouse buttons must be in capitals (eg SELECT), as must key names (eg ESC, RETURN, SHIFT, CONTROL, A, B, F1, COPY). Miss out speedups and shortcuts – just provide enough to help a beginner without drowning him in information.

Provide interactive help thoroughly – include the icon bar, and the work area of all your windows. If no actions are possible in a window, just

> This window shows ...

is better than nothing.

What you should assume

You should assume a user knows:

- what the MENU key is
- how to navigate menu trees and choose entries
- what the icon bar is
- how to move/size/toggle/close windows, and so on

- what 'dragging an icon' means
- what 'filling in a field' (writable icon) means.

Dragging

The Wimp's drag operations are specifically for drags that must occur outside all windows. As well as using the cycling dashed box form, you can define your own graphics to drag arbitrary objects between windows.

If you build drag operations within your window, check that redraw works correctly when things move in the background (the Madness application is useful for testing this).

If the drag works with the mouse button up then menu selection and scrolling can happen during the drag, which is often useful.

If the drag works with the button down, then you can implement it so a user can drag the object out of the window with the button still down. Alternatively you can restrict the pointer to the visible work area, and automatically scroll the window if the pointer gets close to the edge.

Time

There are two clocks that keep track of real time in the system, the hardware clock and a software centi-second timer. The two can diverge by a few seconds a day, but are resynchronised at machine reset. For consistency, always use the centi-second timer.

Exiting your application

When your application exits, it must leave the machine in an undisturbed state. It must not leave modules unplugged, fonts claimed, or files open. You may leave additional modules in the machine if they are generally useful to other packages.

Screen handling

Modes

Your application must work in any screen mode that the window system can use. Your application must read the current screen mode when it is loaded (and any associated information such as resolution and aspect ratio) rather than setting it. You must handle changes of screen mode on the fly. All this may seem troublesome when you write your application, but it allows an end user to use a wide price-range of monitors, and to choose between resolution and cost. It also means you can easily move your application to new better screens and modes when they become available.

At the very least, your application must not crash in inappropriate modes, but must instead display an error message. Mode 0 is not usually useful, but it is worth making it work if you possibly can. You should make Mode 23 work for users with big monochrome screens. Also, try a square pixel mode (eg mode 9), and check in modes 13, 15, 16, 18, 19 and 20.

Mode 16 is highly non-square – ie the aspect ratio is wrong. Do not try to correct for this automatically; it is an inevitable consequence of trying to fit a great deal of text onto a standard monitor. Some monitors can in any case be adjusted to make the pixels square.

Screen size

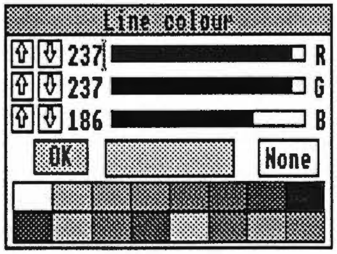
Likewise, you must not rely on the size of screen your application is using. Work using OS graphic units; think of them as a constant unit of measurement, rather than a fraction of the width of the screen. The standard assumption is that there are 180 OS units to the inch, even though this may in fact vary between physical screens. If your application is to be device-independent, it must be the same size in OS units in any mode, rather than the same fraction of the screen.

Colours and the palette

Just as with screen modes and sizes, you must not set or rely on the palette, but instead must read and use the existing one. You must also cope with palette changes on the fly. When you set colours, use one of these methods rather than the older GCOL mechanism:

- Use the standard Wimp palette if you are just using colour to give a contrast between different objects you are drawing.
- Use 'true' (RGB triplet) colours if you need to display a particular colour. Then use the ColourTrans module to give the closest possible approximation in the current palette, so you don't restrict yourself to the limitations of today's hardware.

  Use a dialogue box like the ones in the Applications Suite to set 'true' colours:



Animated bright colour graphics can help make your application easier to understand and to use. Even if your program doesn't use many colours, you must check it works correctly in 256-colour modes. If you have used any EOR (exclusive OR) operations for screen handling, check these with particular care; an EOR can give different results in a 256-colour mode from those it gives in 16-colour modes, because the palette is arranged differently. Similarly, check two-colour modes carefully; these use ECF patterns (stippling) for different shades of grey, and again using EOR may give unexpected results.

Responsiveness

RISC OS runs on extremely fast machines, and you can use this speed to make your application easier to use and more productive. The system software has been written very carefully so that all of this performance is delivered to be used by applications, rather than being swallowed up within the operating system. Fast, smooth scrolling and redraw are worth striving for as they make it easier for a user to make effective and productive use of your application.

Redrawing speed

Some systems remember the bit-map behind a menu or dialogue box when they pop it up; to remove the menu, they just redraw the bit-map. You can't do this in a multi-tasking environment like RISC OS, because a window from a separate task may be changing in the background. Instead you must concentrate on making redraw fast.

One technique you can use for a window that is difficult to redraw quickly is to store its image as a sprite – of course you can only do this if it won't change. Another important technique for speeding up redraw is the use of source-level clipping. During redraw and update, the Wimp will always inform your application of the current clipping rectangle. Don't waste processor time redrawing bits of your window if you don't need to. (For an example of how to use this technique, see the Patience program.)

If you make extensive use of icons within dialogue boxes, this means that RISC OS does most of their redrawing for you. You should only need to process redraw events for dialogue boxes when they contain complex user graphics.

Taking over the screen

Some program developers feel very strongly that a program should be able to take over the entire screen, without any scroll bars etc. You can do this and still benefit from the multi-tasking environment, so long as you treat this as a specific mode of operation (chosen by a menu entry saying 'Fill screen', for instance), and your application can also operate in a window. You can easily implement this by opening a window the size of the screen on top of all others. If you set its 'backdrop' bit, then this will also stop any windows from going behind yours. Your application may even have special properties that only operate when in this mode, such as animation implemented using direct writing to the screen. If you need this mode of operation, however, it should not alone lead you to abandon the multi-tasking world entirely.

Application directories

Application resource files

You must place your RISC OS applications in a directory whose name begins with `!`, such as `!Draw`. When you refer to these applications, however, you leave the `!` off the name. The Filer modules provide various mechanisms to help such applications, so you can treat your program and its resources as a single unit, and its installation is straightforward.

You can hold any form of resource within an application directory. There are several standard ones; a given application may not need all of them, but for those it does use, it must use the filename(s) given below:

`!Boot`	`*Run` by the Filer when it first displays the application directory
`!Sprites`	Passed to `*IconSprites` by the `!Boot` file, or the Filer
`!Run`	`*Run` by the Filer when a user double-clicks on the application directory
`!RunImage`	The application's executable code
`Templates`	The application's window template file
`Sprites`	The application's private sprite file
`Messages`	The application's text messages
`ReadMe`	The application's release notes
`!Help`	The application's documentation
`Choices`	User choices/preferences

The only file you must provide is the `!Run` file.

Most of these resources are discussed in more detail below. In each case we assume that the application is called `!Appl`.

The !Appl.!Boot file

This is the name of a file which is *Run when the application directory is first 'seen' by the Filer. It is usually an Obey file, ie a list of commands to be passed to the command line interpreter (see documentation of the *Obey command in either the *User Guide* or the *Programmer's Reference Manual* for details).

You might typically use a `!Boot` file to set up the icons, file types and corresponding system variables that RISC OS needs so it can show your application in a directory display and run it when you double-click on its icon.

The Filer only runs `!Appl.!Boot` if the sprite called `!appl` does not already exist in the Wimp sprite pool (sprite names are lower case). This prevents repeated delays from re-executing `!Boot` files (or even re-examining application directories). However, it relies on the various applications seen by the Filer having unique names – so, for example, if you have more than one `System` directory, only the first one 'seen' will be used.

The !Appl.!Sprites file

This is the name of a sprite file that provides sprites for the Filer to use to represent your application's directory, in both large and small form. For an application *`!Appl`* these must be named *`!appl`* and `sm`*`!appl`* respectively. The *`!appl`* sprite is also used when the application is installed on the icon bar.

`!Sprites` can also provide sprites for data files that your application controls, in both large and small form. These sprites must be named `file_`*`ttt`* and `small_`*`ttt`*, with *`ttt`* being the hex identity of the file type.

See the chapter entitled *Sprites and icons* for rules about the appearance of these sprites.

Note that all these sprites are merged into the Wimp's shared sprite pool using *IconSprites. If your application uses any private sprites, you must instead put them in the *`!Appl`*`.Sprites` resource file, and your application must load them into a private sprite area.

Standard icons provided

If your application creates or uses one of the following standard file types, you will not have to provide a `file_ttt` icon for it, as they are already provided in the Wimp sprite ROM area:

Sprite		Type
`file_bbc`		BBC ROM
`file_feb`	†	Obey
`file_fec`		Template
`file_fed`	†	Palette
`file_ff6`		Font
`file_ff7`		BBC font
`file_ff8`	†	Absolute
`file_ff9`	†	Sprite
`file_ffa`	†	Module
`file_ffb`	†	BASIC
`file_ffc`		Utility
`file_ffd`	†	Data
`file_ffe`	†	Command
`file_fff`	†	Text
`file_dir`	†	Non-application directory (folder)
`file_xxx`	†	Un-typed (load/exec address) file

Sprites marked with a † also have small format versions in the Wimp sprite area. Those which haven't can be scaled to half size if small icons are needed. There are also two sprites named `application` and `small_app`, which are used for applications which don't have a sprite called !appl.

The !Appl.!Run file

This is the name of a file which is *Run when the application directory is double-clicked. It is usually an Obey file.

It should be emphasised that the presence of multiple applications with the same name should be thought of as an unusual case, but should not cause anything to crash. Also, you should complain 'cleanly' if you can no longer find your resources after program startup.

The !Appl.Messages file

This is the name of a file that is used to store all of an application's textual messages. If you use such a file, it makes it easy for you to replace your messages with ones in a different language should you come to sell your application on the international market.

You should preferably read in every textual message when your application starts. You must not read them one by one, as this forces a user of a floppy disc-based system to have your application disc permanently in the drive. As a minimum standard you must read in all error messages when the application starts up, so that producing an error message does not cause a `Please insert` *`disc title`* message to appear first.

The !Appl.Choices file

This is the name of a file used to store user-settable options so they are preserved from one invocation of the program to the next. If you save them within the application directory, then a user does not have to worry about separate files containing such data. You must always use a `Choices` file rather than reading an environment string, so that users don't need to understand how to set up a boot file in order to set their preferences.

Shared resources

Some resources are of general interest to more than one program. Typical examples include fonts, and modules that provide general facilities. Such resources should be placed in the System application (whose `!Boot` sequence sets a variable `System$Path`) or in a separate application such as Fonts.

You should note that the use of shared resources makes applications slightly harder to install, so check carefully that error messages are helpful if the shared resources cannot be located.

Large applications

The rules above may break down for large applications. Some applications occupy more than one floppy disc, with swapping required during operation. It is difficult to give precise guidelines for such programs, because their requirements vary so widely. The rules above, however, will be used for many smaller programs and so will be reasonably familiar to users. Larger programs should be designed and organised to fit within the same general philosophy, so that users find them easy to install, understand and operate.

For an example of splitting a large application, see *Acorn Desktop Publisher*.

Sprites and icons

Introduction

RISC OS uses sprites to represent a variety of different objects:

- applications (including editors)
- files (including editors' documents)
- devices.

Most of these can be shown in two different sizes:

- Large sprites are used on the icon bar, and in directory displays that show *Large icons*.
- Small sprites are used in directory displays that show *Small icons* or *Full info*.

This chapter outlines what rules these sprites must follow. For information on how these sprites must be 'made known' to RISC OS, see the chapter entitled *Application directories*. For information on using sprites as icons within dialogue boxes, see the chapter entitled *Menus and dialogue boxes*.

Defining sprites

Sprites are normally defined in mode 12 – but if you can use a mode with less colours or resolution (such as mode 9) then do so. You must not define them in 256-colour modes, as RISC OS currently has limitations in how it translates colours from these modes to ones that support fewer colours. Check the appearance of your sprites in two, four, sixteen and 256-colour screen modes; the Wimp will do its best to translate from mode 12 colours to those available.

Appearance of sprites

Sprites you use to represent an application should not have a square or rectangular border; they should instead have an irregular outline. This means they must have a transparency mask. You may use any colours you like for them.

Sprites you use to represent a file should be square, with a black (Wimp colour 7) border. If the file is a document that 'belongs' to a particular editor, then the editor's icon and the document's icon should be visually related to each other.

Sprites you use to represent a device will often have an irregular outline. If they do, then they must have a transparency mask. They should have a grey (5) outline on a cream (12) background.

Size of sprites

Sprites must conform to the following rules about size. If a sprite has an irregular outline, then you should consider any size given below as that of a bounding box containing the sprite's transparency mask, within which the sprite itself is centred.

Large sprites

A large sprite must be 68 OS units high. You should preferably use a square sprite (ie 68 OS units wide). For mode 12 this size converts to 34 pixels wide by 17 pixels high. If you have to make your large sprite wider, you can make it:

- up to 160 OS units wide if it will be used in directory displays – although 100 OS units is a more practical limitation if you want the corresponding small sprite to have the same proportions
- as wide as is necessary if it will only be used on the icon bar.

The border of a large file (or document) icon must be four OS units wide. In mode 12, that makes vertical borders two pixels wide and horizontal ones one pixel high.

Small sprites

A small sprite must be half this size – that is, 34 OS units high. Again, it should preferably be square (ie 34 OS units wide). For mode 12 this corresponds to 17 pixels wide by 9 pixels high (rounding up halves). If you have to, you can make a small sprite up to 50 OS units wide.

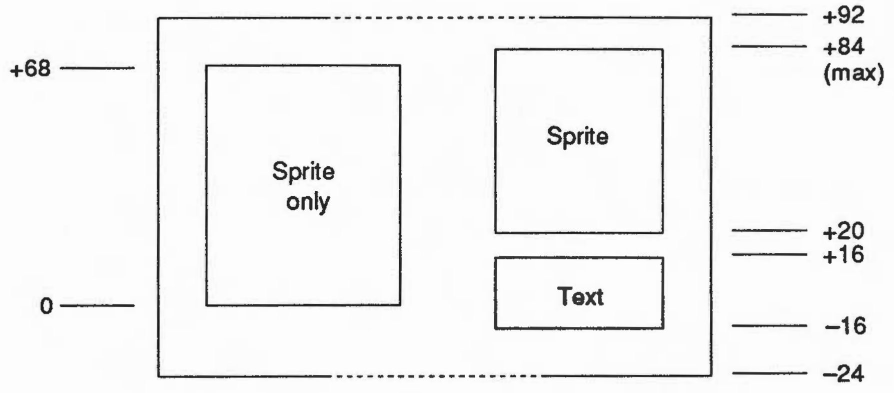
The border of a small file (or document) icon must be two OS units wide. In mode 12, that makes all borders one pixel wide.

You should define small versions of any sprites that are to be used in directory displays. If you do not, then RISC OS will scale the corresponding large sprite to half size. This may be adequate for your purposes.

Positioning icons on the icon bar

When you place an icon on the icon bar, RISC OS uses the icon's width to position it horizontally as it sees fit. If there are so many icons on the icon bar that it fills up, RISC OS automatically scrolls the bar whenever a user moves the mouse pointer close to either end of the bar.

However, it is your responsibility to position the icon vertically. There are two main types of icon which you can put onto the icon bar: those consisting simply of a sprite, and those consisting of a sprite with text written underneath. The diagram below summarises how you must position icons vertically on the icon bar:



y coordinates are given in terms of the icon bar work area origin; lower coordinates are inclusive, and upper coordinates are exclusive.

As laid down earlier, all icon bar sprites must be 68 OS units high. You must position ones with text underneath them 16 OS units below the icon bar's work area origin, and ones without text level with it.

Use of sprite pools

Do not use the system sprite pool in your application; build a user one, or use the Wimp area if appropriate. The system sprite pool is present under RISC OS for backwards compatibility with previous products, and to help the construction of very simple programs.

Menus and dialogue boxes

Basic menu operation

Your application must provide a single menu tree which is displayed when a user presses Menu. This is preferable to using a collection of short menus, each of which requires a user to point at a specific place in the window before pressing Menu. The former approach means that a user can quickly guess what your program can do, and discover fairly rapidly what it can't do, without having to search everywhere for hidden menus.

You can, however, make entries in the menu context-sensitive, so that they depend either on the object beneath the pointer (eg in the Paint file window or Filer directory displays), or on what object(s) are selected (eg in Edit, and again Filer directory displays).

Just like windows, all menus, submenus and dialogue boxes must be movable.

Shading menu entries

You must shade any leaf items of the menu tree that are not available due to the context, rather than omitting them. You must also shade any item that leads directly to a dialogue box, but that is unavailable.

You must not shade an item that leads to a submenu, even if all items on the submenu are unavailable. This is because if a menu item is shaded its submenu is not displayed, thus preventing a user from quickly seeing all the available options.

Menu colours

Standard colours you must use for a menu are:

- black (7) on a grey (2) background for the title
- black (7) on a white (0) background for unshaded menu items
- light grey (2) on a white (0) background for shaded menu items.

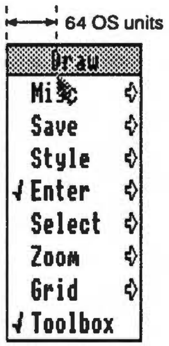
Menu size and position

Each item on a menu must be 44 OS units high. Try to keep the width of non-leaf menus as small as possible; this reduces the amount of mouse movement needed to reach a leaf, and makes choices fast and easy for a user to make.

You must open a menu 64 OS units to the left of the pointer's position when Menu was pressed. This further reduces the amount of mouse movement a user needs to make.

- In C Release 3, RISC_OSLib uses a value of 48 OS units. If you are using RISC_OSLib, it is acceptable for you to use this incorrect value; it will be changed to 64 OS units in a future release.

The bottom of the menu title must normally align with the pointer:



Sometimes the vertical positioning must be different from this, though:

- You must open Icon bar menus so that their base is 96 OS units from the bottom of the screen. This stops the menu from obscuring the icon bar sprites.

Other points

Other rules for menus are:

- The title says 'Appl' (the application name) rather than 'Appl Menu'.
- Items have their first initial letter capitalised (ie 'Set type', not 'Set Type') and are in lower case otherwise.
- Items are left-justified (except for keyboard equivalents – see the chapter entitled *Handling input*).
- Items use the system font, rather than using the Font Manager.

Making menu choices

If a user presses Select or Menu on a menu entry, you must choose the current menu item, perform any associated activity, and close the menu tree.

If a user presses Adjust on a menu entry, you must choose the current menu item, perform any associated activity, and leave the menu tree displayed.

If a user presses a button on a non-leaf item, you should either do nothing, or you should do some sensible default available on that item's submenu – for example, clicking on a Save item commonly saves a document using the pathname that would be used in the Save dialogue box.

Types of dialogue box

There are three basic types of dialogue box you can use:

Ordinary dialogue boxes

An *ordinary* dialogue box appears as a submenu, and functions in the same way – for example a Save dialogue box. It has at least one action icon (such as 'OK' or 'No') but no Close icon. It is typically small, to make it easy to browse through the various submenus an application offers.

Always try your hardest to implement any submenus as ordinary dialogue boxes rather than detached ones.

Detached dialogue boxes

A *detached* dialogue box also appears as a submenu, but suspends its parent application until it is filled in – for example large dialogue boxes in *Acorn Desktop Publisher*.

A detached dialogue box has at least one action icon (such as 'OK' or 'No'). It may also have a Close icon, but this is usually replaced by a 'Cancel' icon. It appears after a user clicks on its parent entry in the menu tree, which must have an ellipsis '`...`' after it to show that it leads to a detached dialogue box – so `Style...` would be a typical such entry.

Try to avoid detached dialogue boxes wherever possible. However, you will have to use one if technical restrictions prevent you from doing what you need to with an ordinary dialogue box – for example if you want the dialogue box:

- to have menus of its own
- to have panes of its own
- to act when icons are dragged onto it.

You may also prefer to detach a dialogue box if it would be so large as to normally obscure its parent menu tree, and it cannot easily be split into smaller sections. In this case, open it so that it is centred on a mode 12 screen.

Static dialogue boxes

A *static* dialogue box remains when the menu disappears, but still allows you to use any application, including its own parent. There are two variations:

- A *static pane* dialogue box is attached to a particular window – for example Draw's tools.
- A *static non-pane* dialogue box is not attached to a particular window (although it might be associated with one) – for example Paint's tools.

Use static dialogue boxes to provide such things as tool boxes, and palettes. Choose the most appropriate type of static dialogue box based on these factors:

- A pane (such as Draw's tool box) often has a menu entry to toggle whether it is displayed or not, which is preceded by a tick when you are displaying the pane. The pane must disappear when a user closes the window it is attached to.
- A non-pane must be implemented as a standard RISC OS window. It must have a menu entry to initially display it. A non-pane dialogue box is typically associated with something such as an application (for example Paint's tool box), or a window (for example Paint's colours). You must close a non-pane when the object it is associated with is no longer displayed, or if a user clicks on the dialogue box's Close icon.

Dialogue box colours

Standard colours you must use for a dialogue box are:

- black (7) on a grey (2) background for the title, whether or not you have the input focus (ie don't highlight the title)
- black (7) on a grey (1) or white (0) background for the body
- black (7) on a white (0) background with a black (7) border for writable icon fields
- black (7) on a cream (12) background with a black (7) border for action buttons.

Dialogue boxes match the colouring of menus, to show that they are part of the menu tree. If the dialogue box is large and has fill-in fields then use colour 1 as the window background rather than 0. Large expanses of white background can make fill-in fields harder to see.

Dialogue boxes and keyboard short cuts

A dialogue box must work in exactly the same way whether it was opened from a menu or using a keyboard short-cut.

For full details on using keyboard short-cuts, see the chapter entitled *Handling input*.

Standard icons used in dialogue boxes

There are various standard forms of icon that occur within dialogue boxes, which are outlined below.

Writable icons

Writable icons are used for various forms of textual fill-in field. You should use either validation strings or your own filtering code to ensure that only legal strings are entered.

You must handle the following keystrokes within a dialogue box with writable icons:

↓	move to the next writable icon within the dialogue box, or to the first if currently at the last.
↑	move to the previous writable icon within the dialogue box, or to the last if currently at the first.
↵Return	move to the next writable icon within the dialogue box, or perform the default 'go' operation for this dialogue box if currently within the last writable icon
Esc	cancel the operation and remove the dialogue box.

When you move to a new writable icon, place the caret at the end of any existing text.

Action icons

An action icon is a 'button' on which a user clicks in order to cause some event to occur – typically that for which he has just entered parameters in the dialogue box. An example is the OK button in a 'Save as' dialogue box.

An action icon must invert while the pointer is over it (like a menu item). Do this by setting an appropriate button type – see the *Programmer's Reference Manual* for details.

As well as the keystrokes outlined above, you may wish to provide keyboard equivalents for any action icons. This is especially useful if the dialogue box itself can be popped up by a keystroke, so that the entire dialogue box can be driven from the keyboard.

If you wish to do so, you should arrange your action icons in a row (preferably horizontal). Assign F2 to the leftmost (or top) action icon, F3 to the next one, and so on until you reach F10. Note that:

- F1 must always provide help if it does anything.
- F11 is reserved for future use by Acorn.
- F12 must always remain a route to the CLI.

See the chapter entitled *Handling input* for further details.

Option icons

An option icon is a 'switch', and can either be on or off. You must use the standard option icons available in the Wimp's sprite pool for such icons:

▣ opton ☐ optoff

Any associated text must be to the right of an option icon. Pressing either Select or Adjust over an option icon must toggle its state.

Radio icons

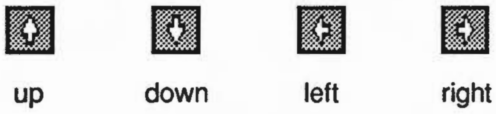
A radio icon is one of a group of 'buttons' only one of which may be selected at once. You must use the standard radio icons available in the Wimp's sprite pool for such icons:

◈ radioon ◇ radiooff

Any associated text must be to the right of a radio icon. Pressing either Select or Adjust over a radio icon must select it, and deselect any other radio icon in the group that was previously selected.

Arrow icons and sliders

An arrow icon is used to increase or decrease a numeric value (such as when setting a Zoom value in Draw or Paint). It is sometimes used in conjunction with a slider (such as when setting a Palette entry). You must use the standard arrow icons available in the Wimp's sprite pool for such icons:



Pressing Select must adjust the value or move the slider one way; pressing Adjust must do the opposite. So if pressing Select on a left icon moves a slider to the left, pressing Adjust would instead move the slider to the right.

[illegible] [illegible]

Handling input

Gaining the caret

You may gain the caret if:

- a user clicks inside your window
- a user calls up a menu or dialogue box.

In the latter case, you must give the caret back to the previous owner when you close the menu or dialogue box. Normally RISC OS automatically does this for you.

When you gain the caret, you must not automatically re-open your window on top of all others. The reverse also applies; if you pop a window to the front of the window stack, do not automatically gain the caret.

Unknown keystrokes

If you receive a keystroke that you do not understand or use, do not claim it – pass it on to other applications using Wimp_ProcessKey. This allows other windows to provide hot key operations that work anywhere; it also allows the Wimp to do function key expansion in the last resort.

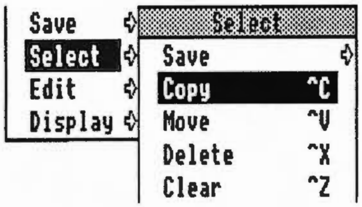
Abbreviations

Keyboard speedups for menu operations are useful to expert users. You must place reminders of their existence right-justified in the relevant menu entry. The following are examples of the abbreviations that you should use:

^X	control character
F3	function key
⇑F3	shifted function key
^F3	control function key
^⇑F3	control shifted function key

The character code for '^' is &5E and the code for '⇑' is &8B, assuming you're using the system font – which you must do for menus and dialogue boxes.

Typical menu entries would look like this:



Selections

Many applications support the concept of a *selection* of data within a document – that is, a portion of data on which a user can perform operations. If you support selections then you must use:

- Select to select an object or (by dragging) an object range
- Adjust to extend or reduce the selection, either by pointing or by dragging.

Selecting text

If a user is selecting text in your application, then you must follow these rules::

- Clicking Select sets the caret.
- Dragging Select selects a range of text.
- Clicking or dragging Adjust adjusts the extent of the selection.

You should also use these conventions, which will make your application more powerful and consistent:

- The caret should be separate from the selection.
- A double-click when setting/dragging a selection should select words.
- A triple-click when setting/dragging a selection should select lines or paragraphs, as appropriate.
- If a user holds down Ctrl while setting a selection, you should not move the caret.

Operations with selections

You should always aim to make copying, moving or deleting a selection a single operation, rather than the separate cut/paste required on other systems.

Keyboard shortcuts

Using a mouse and pointer to choose items from a menu is not always the quickest way to use an application. Many users, particularly experienced ones, like to have keyboard shortcuts to particular operations. To save confusion, common commands should have consistent shortcuts across different applications. These are listed below.

In general, you should try to use function keys for shortcuts, and provide a keystrip to use with them.

Porting applications

If you are porting an existing application from another operating system (or are writing an emulation of one) we recognise that there will be a strong case for not changing the keystrokes it uses, so that existing users of the package do not need to learn new keystrokes. In such cases we leave it to your discretion whether you use the shortcuts below, or the ones your application originally used.

Preferred shortcuts

This list shows the keyboard shortcuts you should try to use. The left column gives the standard abbreviation for the shortcut – use this in your menus where applicable – and the right column a description of what it does. Obviously you do not have to implement all of these, but where a function corresponds to one you do provide, you should use this shortcut rather than any other:

Abbreviation	Action
F1	Help
F2	Load named document
⇑F2	Insert named document
^F2	Close window
F3	Save document
F11	Reserved for use by Acorn – do not use this key *at all*
F12	Give access to * Commands using CLI – do not use this key *at all*
Print	Print document
Delete	Delete left if there is a caret (as backspace), or delete selection if there is not a caret (as ^X)
Copy	Delete right if there is a caret, or copy selection to cursor if there is not a caret (as ^C)
Esc	Cancel operation

←	→	Move by a character
↑	↓	Move by a line
⇑←	⇑→	Move by a word
⇑↑	⇑↓	Move by a page (like clicking on the scroll bar background)
^⇑↑	^⇑↓	Move window by a line (like clicking on the scroll bar arrow icons)
^←	^→	Move to start/end of line
^↑	^↓	Move to start/end of document
^U		Delete line
^Z		Clear selection
^C		Copy selection to cursor
^X		Delete selection
^V		Move selection to cursor

International support

RISC OS already has some facilities for international use (eg multiple alphabets/keyboards). It will be extended in the future to allow translation of ROM messages, to provide such extra facilities as international lexical sorting. Even without these facilities RISC OS computers are sold in many non-English-speaking countries. Every step you make towards helping a user understand programs in their native language helps your sales in the international market. Accordingly:

- Use system facilities for datestamps etc.
- Use pictorial icons rather than text/picture combinations.
- Use Alt as a shifting key rather than as a function key. Different forms of international keyboards have standardised the use of Alt for entering accented characters.
- Do not forbid the use of top-bit-set characters in your program – again this will interfere with a user who wants to use accented characters.
- Don't assume that Latin1 is the current character set.

Editors

Introduction

An editor presents files of a particular format as abstract objects which a user can load, edit, save and print. You must always remember that an editor is a special type of application, and as such, it must comply with all the rules laid down in other chapters within this Guide. Any editor you write should also be consistent with the editors (Draw, Edit and Paint) provided in the applications suite.

Multi-document editors

Wherever possible, you must write an editor so that it can edit multiple documents concurrently. This removes the need for multiple copies of the program to be loaded. Edit, Draw and Paint are all multi-document editors.

Matching standard dialogue boxes

Many of the dialogue boxes that you will need to use for your editor should match standard ones already used by other editors (such as Draw, Edit and Paint) that are shown as illustrations in this chapter.

The best way to ensure they match is to define them using templates. If you do so, you can use FormEd to:

- copy templates that closely match your needs from Draw, Edit, Paint and other established RISC OS editors
- edit the templates to make any necessary changes, such as changing the title of the editor
- save the modified templates to your editor's template file.

Editor windows

Like any other RISC OS application, an editor must run in a window.

Title

The title of an editor window must be the full pathname of the current document, centred in the title bar. If the document has not yet been saved or loaded, then its title is instead `<untitled>`.

- If the document has been modified, you must append a space followed by a * to the title.
- If there are multiple views of the same document, you must lastly append a space followed by a number n to the above title, where n is the number of existing views of the document.

You can set the window's minimum size field so that the title length does not restrict the window's minimum size.

Colours

Standard colours you must use for the editor window are:

- black (7) on a grey (2) background for the title when it is not highlighted (ie you do not have the input focus)
- black (7) on a cream (12) background for the title when it is highlighted (ie you do have the input focus)
- dark grey (3) for the outer colour of the scroll bar
- light grey (1) for the inner colour of the scroll bar.

The user interface

The user interface that RISC OS provides to load and save documents is rather different from that of other operating systems, because directory displays are always available. This means that there is no need for a separate 'mini-Filer' which presents access to the filing system in a cut-down way. Although this may feel unusual at first to experienced users of other systems, it soon becomes natural and helps the feeling that applications are working together within the machine, rather than as separate entities.

Starting an editor...

You must start your editor if a user:

- double-clicks on the editor's icon within a directory display using either Select or Adjust
- double-clicks on a document icon within a directory display using either Select or Adjust, where the document 'belongs' to the editor, and the editor has not already been started
- drags a document to the printer icon using either Select or Adjust, where the document 'belongs' to the editor, and the editor has not already been started.

...by double-clicking on the editor's icon

In the first case you must always load a new copy of your editor. You must also put an icon containing your editor's `!appl` sprite onto the icon bar. This applies even if you have already loaded one copy of your editor.

You would typically do this by running your editor's `!Appl.!Run` file.

...by double-clicking on a document's icon

In the second case, if your editor isn't already running you must start up a new copy of it and put its icon on the icon bar. You must then open the document that was double-clicked, as described below.

You would typically do this by using the run-type of the document file, which in turn will invoke the application by name with the pathname of the document file as its single argument.

If your editor is already started, then a double-click on a document that 'belongs' to it doesn't start a new copy of the editor – it just loads the document, as described below.

...by dragging a document's icon

In the last case, if your editor isn't already running you must start up a new copy of it and put its icon on the icon bar. You must then print the document that was dragged, as described below.

You would typically do this by using the print-type of the document file, which in turn will invoke the application by name with the pathname of the document file as its single argument, followed by a –print option flag.

If your editor is already started, then dragging a document that 'belongs' to it to a printer driver doesn't start a new copy of the editor – it just prints the document, as described below.

Creating a new document

You must create a new document and open a window on it if a user:

- clicks on the editor icon on the icon bar using Select.

If your editor needs arguments to create a new document, you may also use a dialogue box during the course of this process. If a style sheet is required (eg for a DTP program) then you may instead use a static dialogue box, and drag the style sheet from a directory display.

Opening the window

The first window your editor opens must be horizontally and vertically centred in a mode 12 screen. It must be no larger than 700 OS units wide by 500 high, so it does not occupy the entire screen. This emphasises that the application does not replace the existing desktop world, but is merely added to it. Open subsequent windows 48 OS units lower than the previous one, but if this would overlap the icon bar then return to the original starting position. The initial size and position of windows should be user-configurable, by editing a template file.

Single-document editors

In a single-document editor, if a user clicks on the editor icon on the icon bar you must create a new, blank document only if a document is not already loaded. If a document is already loaded, you must instead move the document window to the front of the window stack, in case it has been obscured by other windows.

Loading a document...

You must load a document and open a window on it if a user:

- double-clicks on a document icon within a directory display using either Select or Adjust
- drags a document icon from a directory display to your editor's icon on the icon bar using either Select or Adjust
- drags a document icon from a Save dialogue box to your editor's icon on the icon bar using either Select or Adjust.

...by double-clicking on a document's icon

In the first case you may have to start your editor (see above).

...by dragging a document's icon

In the latter two cases, the editor must already have been started for its icon to be on the icon bar. This way of loading a document allows a user to specify exactly which editor to use. For example, you can drag a file of type PoScript onto the Edit icon to edit a PostScript program.

In all cases, the size and position of the window you open must be as laid down above.

Inserting one document into another

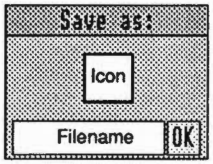
You must try to insert a document into the one you are editing if a user:

- drags a document icon from a directory display to an open editor window using either Select or Adjust
- drags a document icon from a Save dialogue box to an open editor window using either Select or Adjust.

If the document is not of a type that your editor can import, it must ignore the operation and not generate an error.

Saving a document

You must provide a dialogue box as follows for a user to save a document:



The dialogue box consists of a sprite icon, a writable icon, and an action icon. This is the standard equivalent of the 'mini-Finder' in other systems. If there is no pathname (ie the document is a new one that has not yet been saved), then invent a simple leaf name, eg `TextFile` in Edit, so that dragging the icon to a directory display will not cause an error.

Once the dialogue box is displayed, a user must be able to:

1 press Return or click on OK to save in the already named file

  (clicking on the menu entry that leads to this dialogue box should have the same effect)

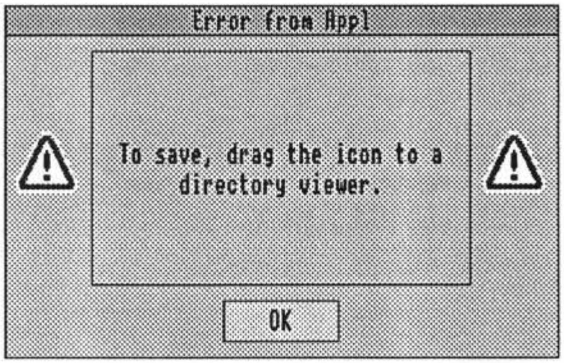
2 edit the pathname as desired using the keyboard

3 drag the icon into a directory display, to save in that directory with the given leaf name

4 press Escape to cancel the operation.

A user will typically use (1) to save an existing document that has already been saved, (2) to give the leaf name of a file when it is first saved, and (3) to specify the directory when it is first saved. There is, of course, nothing to stop a user specifying the directory by typing the complete pathname of a file, instead of dragging it to a directory display.

When (1) happens, you must check that the proposed name does at least contain one '.' character. This prevents a common error in beginners, who just see the proposed leaf name, and attempt to select OK immediately. If there is not a '.' character, you must generate an error window like this:


Error from Appl

To save, drag the icon to a directory viewer.

The writable icon you use for (2) must be able to accommodate pathnames up to 255 characters long, and have a validation string of 'a~ ', so that spaces cannot be included in the pathname. Your application must not crash if a user gives a longer pathname.

When you save the document, you must:

- make sure the document's datestamp is unchanged if the document was unmodified; otherwise you must update it
- check any return codes and errors from saving the document, and take any appropriate action, such as displaying an error in a window
- mark the document as unmodified, unless the save was to a scrap file

- update your stored name for the document and the window title (if necessary)
- remove the Save dialogue box and the rest of the menu, unless Adjust was used to do the save, in which case they must remain on the screen.

Save should be interpreted as being like 'save and resume' from some other systems, ie after the operation a user is still editing the same document.

Saving a selection

You should also provide a similar dialogue box so that a user can save a selection from a document. If you do so, the default leaf name offered must be `Selection`. Its menu entry may be grouped either with other selection operations, or with the 'Save file' operation. If there are several possible selection save formats, putting it on Save may be more appropriate. Balancing submenus may also be an issue. Edit and Paint, for instance, group Save selection with other selection operations; Draw (which has several different forms of Save selection, and many other operations on the Selection submenu) groups it with Save file.

Other uses of the icon in a Save dialogue box

The icon in a Save box should be treated in the same way as an icon in a directory display. So as well as dragging the icon to a directory display to save the document (or part of it), a user can also drag the icon from the save box:

- to the same editor's icon, which creates a new (cloned) copy of the document – see the section on *Loading a document*
- to a different editor's icon, which loads a copy of the document into that other editor – see the section on *Loading a document*
- to another document, which inserts your document into the other document – see the section on *Inserting one document into another*
- to a printer driver, which then prints your document – see the section on *Printing a document*.

Printing a document

You must print a document if a user:

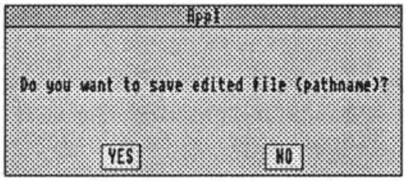
- drags a document icon from a directory display to a printer driver using either Select or Adjust
- drags a document icon from a Save dialogue box to a printer driver using either Select or Adjust.

See the chapter entitled *Printer Drivers* in the *RISC OS Programmer's Reference Manual* for full details of how the printer drivers work.

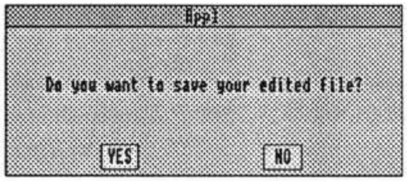
Closing document windows

If a user clicks with Select on the Close icon of a document window, you must:

- close the document immediately if it is unmodified
- pop up a dialogue box similar to one of the following if the document has been modified:



Do you want to save edited file (pathname)?



Do you want to save your edited file?

You should use the one on the left if the document has previously been saved, or the one on the right if the document is new and has never been saved.

If the user then clicks on `Yes` you must pop up a Save dialogue box (see above); if the document is successfully saved then close its window. If the answer is `No`, or any cancel-menu (eg Escape) occurs, then you must not close the window.

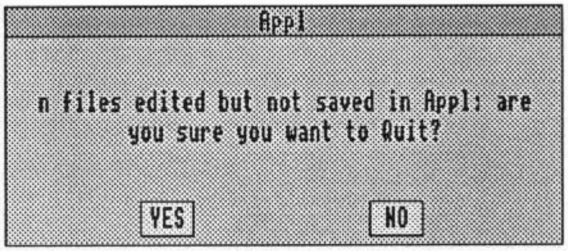
Closing document windows using Adjust

If a user clicks with Adjust on the Close icon of a document window, you simply have to:

- close the document window if the document is unmodified and the Shift key is not depressed
- open the document's home directory display, if it has one.

Quitting editors

You must supply a Quit option at the bottom of an editor's icon bar menu. If a user chooses it when there is unsaved data, then you must display a dialogue box like this:



n files edited but not saved in Appl: are you sure you want to Quit?

where n is the number of modified files that have not been saved. (*Files* should say *file* if n=1.)

You must also use this dialogue box if the user has used another method (such as the Task Manager) to quit the editor, and there is unsaved data.

Matching documents to editors

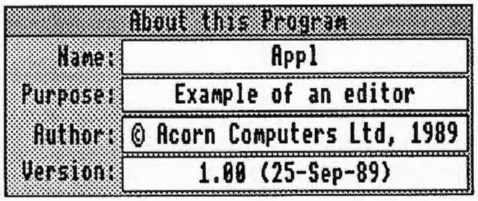
Editors use RISC OS file types to distinguish which files belong to them. Your editor's `!Boot` file must define any of the following that are relevant:

- `Alias$@RunType_ttt`, `Alias$@PrintType` and `File$Type_ttt` variables
- `!appl` and `sm!appl` sprites
- `file_ttt` and `small_ttt` sprites.

For further details, see the chapter entitled *Application directories*.

Providing information about your editor

The 'About this program' dialogue box provides useful information about your editor. For example:

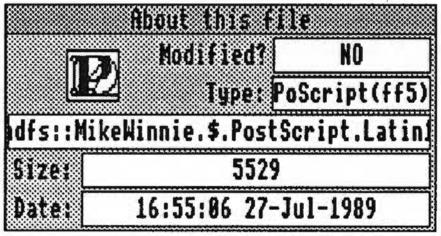


Name:	Appl
Purpose:	Example of an editor
Author:	© Acorn Computers Ltd, 1989
Version:	1.00 (25-Sep-89)

You must provide it at the top of your editor's icon bar menu, but it doesn't have to match the one above – we encourage creativity here.

Providing information about documents

The 'About this file' dialogue box provides useful information about a document being edited. For example:



You must provide it at the top of a document window's menu, or within a 'Misc' submenu if there are other miscellaneous menu entries to collect. Again, it doesn't have to match the one above.

Data transfer between editors

One of the aims of RISC OS is to encourage the free circulation of data between a number of cooperating applications. The following points are all relevant to this:

- You must thoroughly document any data formats that your editor uses, and make such documentation available to third parties.
- Your editor must be able to read in data formats that are in common use and are relevant to its specific application area.
- Your editor must implement both the RAM Transfer and the Scrap Transfer protocols for data transfer between applications. For full details of these protocols, see the *RISC OS Programmer's Reference Manual.*
- Your editor must be able to export the same formats of data that it can include or import, even if that format is normally processed by another editor (such as plain Text, a Sprite or a Draw file).
- If you use Draw files you must render them accurately, as Draw itself does. Draw files should be used as the standard form for structured graphic data interchange.

Appendix – !Help

Technical details – introduction

These technical details are included here because they were unintentionally omitted from the *Programmer's Reference Manual.* We consider !Help to be important, and want to make sure that you have the information you need to support it.

Messages

For an application to use interactive help, two Wimp messages are employed. One is used by Help to request the help text, and the other is used by the application to return the text message.

To request help, the Help application sends a message of the following form:

```
block   +16   &502 – indicates request for help
        +20   mouse x co-ordinate
        +24   mouse y co-ordinate
        +28   mouse button state
        +32   window handle          (–1 if not over a window)
        +36   icon handle            (–1 if not over an icon)
```

Locations 20 onwards are the results of using Wimp_GetPointerInfo.

The Wimp system will pass this message automatically to the task in charge of the appropriate window/icon combination. If the application receiving the message wishes to produce some interactive help, it should respond with the following message:

```
block   +16   &503
        +20   help message, terminated by 0
```

The help text

The help text may contain any printable character codes (including top-bit-set ones). If the sequence |M is encountered, this will be treated as a line break and subsequent text will be printed on the next line in the window. If !Help needs to split a line because it is too long, it does so at a word boundary (space character).

The help text is terminated by a null character.

The Help application

The help application issues message type &502 every 1/10th of a second to allow applications such as Edit and Draw to change the help text according to the current edit mode. To avoid flicker, the display is only updated when the returned help string changes.

With certain applications, such as the Filer, no interactive help is supplied and the Help application supplies some default messages in instances like this.

Reader's Comment Form

RISC OS Style Guide

We would greatly appreciate your comments about this Guide, which will be taken into account for the next issue:

Did you find the information you wanted?

Do you like the way the information is presented?

General comments:

If there is not enough room for your comments, please continue overleaf

How would you classify your experience with computers?

☐	☐	☐	☐
Used computers before	**Experienced user**	**Programmer**	**Experienced Programmer**

Cut out (or photocopy) and post to:

Dept RC, Technical Publications
Acorn Computers Limited
645 Newmarket Road
Cambridge CB5 8PB.

Your name and address:

This information will only be used to get in touch with you in case we wish to explore your comments further.