RISC OS 3
Style Guide

Acorn

RISC OS 3
Style Guide

Acorn

Contents

About this guide

About this Guide

This Guide describes the standards of 'look and feel' to which you should write a RISC OS application. It covers aspects of designing a new application, and implementing the design:

- The introduction explains why this Guide was written and how to use it. It explains the scope of the Guide, and why a standard look and feel is desirable. It also looks at the issues you need to consider when you begin designing a new application.

- Chapters 2–14 deal with design issues, concentrating on the user interface. This includes the design of menus and dialogue boxes, how to load an application and other issues that are part of the design of an application.

- Chapters 15–18 deal with the implementation of the issues covered in the earlier chapters and of the application's functionality. They concentrate on programming issues such as the structure of application directories and how to construct the elements of dialogue boxes.

- Finally, there is a glossary of terms used in this Guide.

This is the second edition of the RISC OS *Style Guide*. The Guide has been reissued to take account of the new features and benefits of RISC OS 3 and to reflect the developments in application design both within Acorn and amongst the developer community. The main aim of the Guide is to help all developers to give their applications a common and consistent look and feel so that users will be able to find their way around new programs easily and will be able to use applications together when appropriate.

Finding out more

You will find a certain amount of relevant information in the *Welcome Guide* and RISC OS 3 *User and Applications Guide* supplied as standard with all RISC OS computers.

The RISC OS 3 *Programmer's Reference* Manual gives full documentation of RISC OS, and 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.

Where this Guide refers you to Acorn for more information, Registered Developers should contact Developer Support in the usual way; other developers should **write** to Acorn Technical enquiries.

1 Introduction

The scope of this Guide

This Guide will help you to specify, plan, and write software to work within the graphical user interface (or GUI) used by RISC OS. It describes the 'look and feel' a user expects from RISC OS applications. Because it is concerned with the design of applications on all levels, you will have to bear in mind many of the points raised here quite early on in development whether you are writing a new application or porting one from another platform.

The Guide is concerned primarily with maintaining consistency and standards in all areas of the style of applications. It describes the standards which we hope you will adhere to so that all developers can move towards a high level of consistency in look and feel, helping users to learn new applications quickly.

By 'style' we mean not only the look of an application on the desktop but also features of its functionality, how well it integrates with other applications and the extent to which it can use common conventions (such as consistency in keyboard shortcuts). These are not issues that can be sorted out at the last moment, but areas you need to consider from the very start of the development process. Some of the points raised are simple rules that are easy to follow – setting the distance between icons in a dialogue box, for example. Others require you to interpret guidelines in the context of your applications.

The scope of this Guide is so large that in places it is necessarily imprecise. Wherever possible and helpful, we have given examples to help make points clearer. Sometimes, the Guide has to enter uncharted waters, and here we can only make recommendations and indicate the direction we expect developments to take. In some areas there may be few or no models to follow, but we hope the guidelines will enable all developers – including Acorn – to move towards a common goal. If you follow the guidelines given in this Guide, you should be able to help users learn and make the most of your applications without limiting what your applications can do.

The main difference between this and the last issue of this Guide is that we now feel the time is right for RISC OS applications to move towards a 3D look and feel. There is advice on how to implement this, and detailed descriptions of how to produce the icons you will need to use in your dialogue boxes and windows. We will be interested to receive feedback from developers who follow the guidelines.

Who should use this Guide?

You should read this Guide if you are involved in the design or writing of applications to run from the RISC OS desktop. This includes designers and writers of games or other applications that may take over the screen completely; there are some guidelines on this in the section entitled *Taking over the screen* on page 33.

The structure of this Guide reflects the process of designing or specifying an application and the implementation of the design in programming the application. Chapters 2–14 will be most useful to the person designing an application. They enable the designer to specify how the menu tree should be structured, what should appear on each dialogue box, how error messages should be worded, and so on. Chapters 15–18 will be most useful to the person or people responsible for writing the program. They cover such details as precise placing of buttons on dialogue boxes and where to put resources.

You should read all the way through the Guide once; we have kept it short so that this is not too time-consuming. You can then use it as a reference work whenever you design or write an application.

Why have a standard?

One of the most important aspects of developing applications to run under RISC OS is to make sure that applications within the desktop world present a consistent and reliable interface to a user. This applies both to how an application looks and to how it behaves. This is to the benefit of all users and developers. If the RISC OS world is a consistent and coherent environment, users will feel confident and at ease even with a new application because it will use a familiar interface and structure. This is to your advantage as well as the user's advantage. A user who has found your package easy to use, following the styles and procedures that are already familiar from other applications and the operating system, will feel happy using your applications and is likely to buy more of them in the future.

Into the future

The requirements set out in this Guide are demanding, and in places require significant effort to implement. Some of the Applications Suite itself does not conform in all respects; as the operating system and Applications Suite change relatively infrequently, they cannot be the main means of introducing change. As new applications are developed or existing applications are updated, we should all aspire to a close match to the current style guidelines.

The standard to which we all aspire will evolve continuously as RISC OS evolves and improves, so 'style' is not static. No doubt future issues of this Guide will be able to be more precise in some recommendations as Acorn and its developers together discover the standards and conventions that work best.

As we have mentioned already, the most noticeable change from the previous edition will be the implementation of a 3D desktop. As the operating system is not updated very frequently, there will be many 3D applications before the desktop itself becomes 3D as standard. It is vitally important that we all develop 3D along common lines. Users will be confused and irritated by a proliferation of different versions of 3D look and feel, so we must work together to ensure that all applications continue to look good together on the desktop and share a consistent implementation of 3D. The chapter entitled *Implementing the design* on page 91 explains in detail how to create the standard 3D icons.

2 Starting a new application

Thinking about a new application

When you begin to think about developing a new application, you will take many considerations into account. The most important of these will be the functionality you want for your application, and the market you are targetting. At the same time, though, you should begin to think about the style you will give the application. If you begin considering this at the earliest stages of development, it will be fairly easy to make sure your new application fits in with the RISC OS desktop and the applications that use it.

Try to bear the following considerations in mind early on. There is more information on the first two in this chapter, and more on each of the others in later sections of this Guide.

- Your applications should be easy to learn and easy to use.
- Your applications should fit in well with others that use the desktop: aim for consistency by following the design guidelines given in this Guide.
- Use the RISC OS Window Manager module (the programming interface to the RISC OS desktop, commonly known as the Wimp) so that your applications will look right and work properly with future releases of the Wimp.
- Use memory efficiently; remember that some users have only a 1MB machine, and users with larger machines will want to run several applications at once.
- Support all reasonable configurations of hardware and software that a user might have.

It goes without saying that you will decide at an early stage in development what you want your application to do and which language you will use to program it. There are some comments on these issues later in this chapter (see the section entitled *Quality* on page 8 and the section entitled *Choice of programming language* on page 91).

Ease of use

The principal aim of this Guide is to help you produce applications that make the computer easy and pleasant to use, for users with varying levels of experience and different requirements. A user should find an application

- easy to learn

- easy to re-learn
- easy to use productively.

You can help the whole developer community and Acorn to achieve these aims by following the guidelines that work towards consistency and common standards. To help the user to learn, relearn and use applications you should:

- Make it easy for users to see all the options and actions available within your application.
- Make each user action perform a well-defined task.

  A good guideline is whether you can describe the task using a single noun-verb combination, such as 'File-delete'. Typically the user will select something (the noun) and then choose an action (the verb).

- Break down complex tasks so they can be performed as a series of simpler tasks.
- Give your application a clear and logical structure so that users don't have to remember complex sequences or too many details.
- Provide clear feedback to each action, so the user feels in control.

  This includes things such as using the hourglass when your application is busy in the foreground, putting status words under icons on the icon bar or changing icons to show the state of a file, application or tool.

- Provide a way for users to undo the mistakes they'll inevitably make.

  Usually this will mean offering an Undo function that can reverse either a single action or a sequence of actions. You might also consider providing a Redo function to reverse an Undo.

- Warn and ask for confirmation when an action may be destructive.

  It's especially important to do this if you don't provide an Undo function. Issue a warning by default, although you may want to provide a facility for experienced users to turn off such warnings.

Multi-tasking

A multi-tasking interface requires that applications work together for a user of the machine. This means that

- they co-operate in sharing the machine
- they look harmonious
- their user interfaces are similar
- files are transferable between applications
- the whole is more important than a single application.

A habitual user of the desktop environment and the Applications Suite programs should find your program easy to use and natural to learn.

Data interchange

One of the most elegant features of the RISC OS desktop is that users can easily move files between compatible applications. If your own applications integrate well with others, including the Applications Suite, users will be keen to use them. A primary requirement is that data file formats are compatible or interchangeable. Aim either to use common data file formats (such as CSV files, for example) or make provision for importing from or exporting to different filetypes. There is a list of filetypes in the RISC OS 3 *Programmer's Reference Manual*, and further information on filetypes in the section entitled *Standard icons provided* on page 105.

An advantage for developers of the RISC OS environment is that new applications don't need to duplicate functionality already offered elsewhere. For example, printing should use the standard RISC OS printer drivers. This saves you effort and means that users already know how to set up printing.

Consistency on the desktop

Before users begin to use your application, or even load it, it will have a presence on the desktop as its icon is visible in a directory display. How this icon looks, and the appearance of the icon(s) it uses on the icon bar and to represent its files, should be harmonious with other items on the desktop. Users must be able to load your application by double-clicking on its icon and, if it uses files, to be able to open a file by dragging it to the icon on the icon bar. All these are common desktop activities which users will expect your application to support. The look of the windows, menus, dialogue boxes and error boxes your application uses should also be consistent with those used by other desktop applications so that the user always feels secure and at ease using the familiar interface, even if it is for a new purpose.

If you are writing a game, your application may not usually be run on the desktop, but may take over the whole screen while running. Even so, its icon will be visible in directory displays and on the icon bar and it must be consistent in these respects with other applications. There is more about single-tasking applications in the section entitled *Taking over the screen* on page 33.

The appearance and wording of menus, dialogue boxes, error boxes and the appearance of sprites and windows are described in the following chapters.

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. In general, a simple program is often an elegant and efficient one; quality and simplicity frequently go hand in hand. Design applications carefully and don't duplicate functionality that is already provided by the Wimp, the operating system or the Applications Suite.

Terminology

You must always bear in mind that your application will mainly be used by the general public, not just by programmers. If you use consistent terminology, and avoid jargon, it will make the application more friendly to them. There is an established vocabulary for referring to parts of the desktop, the mouse buttons and mouse operations. You should use this when communicating with the user. This includes menu and dialogue box text, error messages, help text, manuals or guides and any other user documentation. The names of action buttons, menu items and other very specific items are covered in the appropriate chapters of this Guide. There is more information on terminology and how to present documentation on your application in the *Acorn Technical Publications Style Guide* (AKJ17).

Versions of RISC OS

There have been two major versions of RISC OS which you need to support, and one intermediate version:

- RISC OS 2 was the original version of RISC OS, released in May 1989. This has been superseded by RISC OS version 3 and most users are expected to upgrade. Your applications should continue to run under RISC OS 2 if possible. If an application **requires** RISC OS 3, this should be stated clearly on the packaging.

- RISC OS 3, v3.00 was supplied with early A5000 systems and has been replaced by v3.10. You don't need to offer support for v3.00 as users are encouraged to upgrade to v3.10.

- RISC OS 3, v3.10 is the general release of RISC OS 3 and has been provided as standard on production machines and as an upgrade for users with RISC OS 2. This Guide assumes that you will be using and supporting RISC OS 3, v3.10.

A later version, RISC OS 3, v3.11, is essentially the same as v3.10.

There will be later versions of RISC OS released in the future; it would not be prudent to write applications to match RISC OS v3.10 too precisely.

3　The desktop

Using the desktop

All applications will need to use the desktop, even if only briefly in the case of some games usually run in a single-task mode. It is important that while your application uses the desktop, it looks and behaves in the same way as other desktop applications. This will make your application easy for users to learn and use and will help to maintain the consistency and harmony that we should all be aiming for in developing applications for RISC OS.

If your application usually operates from the desktop, all the guidelines here on designing the user interface will be relevant to you. They cover

- loading, starting and leaving applications
- using the mouse
- icons
- windows
- menus
- dialogue boxes and toolboxes
- handling input
- selection
- using colour and sound
- editors
- user preferences
- internationalisation.

From this chapter to chapter 14, the Guide deals with the design of these items, not the implementation through programming. There is some guidance on implementation issues in chapters 15–18, but for detailed information on how to create and manipulate desktop items you will need to look in the RISC OS 3 *Programmers' Reference Manual*.

If your application takes over the whole screen, or has an option which users can choose to allow it to do so, it will still need to start up from the desktop and appear on the icon bar as an icon in the same way as applications which operate wholly within the desktop environment. The PC Emulator, which may be run as a single task replacing the desktop, is an example of an application that does this. Much of

this Guide will be relevant to you, even if your application is not normally going to use the desktop while it is running. Some of the advice in the section entitled *Taking over the screen* on page 33 will help you to integrate your application into the RISC OS world.

The pinboard

RISC OS 3 provides a 'pinboard' facility, allowing the icons of open or closed files or directories to be 'pinned' to the backdrop of the desktop. Users can quickly access these files and directories and open a window onto any of them by double-clicking on the icon. This saves time opening sequences of directories, and saves space on the desktop by allowing users to 'iconise' an open file or leave directories accessible without being open. You should provide a sprite that can be used to represent iconised windows from your application. See the section entitled *Sprites for iconised windows* on page 95.

Multi-tasking

Remember that users are likely to want to run your application alongside others. Multi-tasking is one of the most important benefits that RISC OS offers users, so make sure your application does not impair the computer's ability to multi-task and do not reset any configuration options or other settings that will affect other applications.

Terms for desktop items

Consistent use of terminology is important for users. To avoid confusion and difficulty, you must use these terms to refer to parts of the *desktop*:

- The bar at the foot of the screen is the *icon bar*.
- The background to the desktop is called the *pinboard*.
- A *window* may be a main window, a menu, a dialogue box, an error box, an info box or a pane window.
- A *main window* may be a document window, a control window or a directory display.
- A main window showing the contents of a directory is called a *directory display* and **not** a *directory viewer*.
- A *menu* has *menu items*, some of which lead to *submenus* (no hyphen). You may shorten *menu items* to *items*, providing the context is clear. The main menu from which submenus may be accessed is called the *root menu*.

  In manuals, menu items and the names of action buttons (see page 52) such as **Save** should be in bold text.

- A menu that appears when you press Menu over an icon on the bar is an *icon bar menu*.

- A menu that appears when you press Menu or Select over a button in a dialogue box is called a *pop-up menu*.

- A chosen menu item is shown *highlighted* (no need to say 'in inverse video').

- A window used for a dialogue between user and computer within an application or on the desktop is a *dialogue box* if there is a delayed effect - that is, the user must click on a button to initiate an action. A dialogue box allows the user to give some details of an action and initiate the action, or close the dialogue box taking no action.

  A dialogue box that remains on screen if the user clicks outside it is a *persistent* dialogue box; a dialogue box that disappears if the user clicks outside it is a *transient* dialogue box.

- An *error box* is a special type of dialogue box that gives information to the user, and requires acknowledgement that it's been read.

- An *info box* is a window that displays information for the user to read. It may be transient, in which case it has no control icons. If it is persistent, it may have a control icon allowing the user to display more information, and will have a Close icon or action button to remove it.

- A *pane window* may be a toolbox or a scrolling list of options.

Other special terms are explained as they occur in this Guide; there is also a glossary.

4   The mouse

Introduction

Although the mice supplied with different systems vary in design, their function
and the function of each button is the same across Acorn systems. It is important
that you support the established standards of mouse activity, and use the
established vocabulary when describing the mouse and mouse activity.

Mouse buttons

The mouse has three buttons:
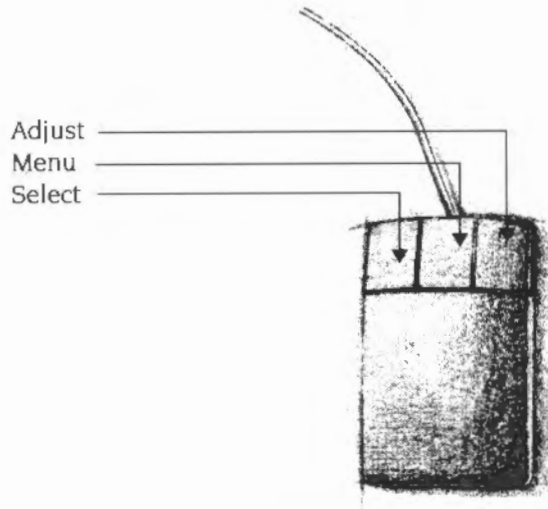
Adjust
Menu
Select

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 a button
Double-click	click twice quickly, without moving the mouse
Triple-click	click three times quickly, without moving the mouse
Drag	press a button and move the mouse, then release the button
Choose	click on a menu item
Select	change an object's state by clicking on it.

Here are some examples of these terms in use:

Type Ctrl-Z or choose **Clear** from the menu.

Triple-click Select to select the whole line of text.

Press Select, drag the icon to a directory display and then release Select.

Select the object you want to delete.

Common faults include confusing *press* and *click*, and talking about *selecting* menu items.

Remember that the mouse speed and double-click speed are configurable; you can't rely on users configuring their mouse to particular settings.

Mouse functions

Do not replace the established functions of the mouse buttons with anything new. Use:

- Select to choose items from a menu, select objects, click on window parts or icons to choose or use them, indicate positions in the window, or drag objects.

- Menu to display a menu anywhere within the window or choose an item from a menu. If you are using menu buttons in dialogue boxes, a user must be able to use the Select or Menu button to call up the menu.

- Adjust to alter selections, reverse the direction of movement brought about by clicking on an icon (such as an adjuster arrow) or scroll arrow, choose an item from a menu leaving the menu displayed, open a directory while closing its 'parent', or open a 'parent' directory while closing the 'child'.

Where possible, Select should be used for all the main functions in you application; Adjust should not be needed by new users, but be used for shortcuts and alternatives to other procedures.

In addition, text editors should support the following mouse shortcuts:

- click to position the caret
- double-click to select a word.

If it is appropriate, triple-click should be supported to select a line or paragraph, depending on the context.

5   Icons

Introduction

The first that users see of your application is its icon in a directory display. Make it attractive and intelligible; if you can, give a hint of its function. The Edit and Paint icons are good examples of this. However, you need to bear the following guidelines in mind when designing icons for your application. It is difficult to design good icons; consider enlisting the help of a graphic designer.

There is more precise information on the sizes for icons in the section entitled *Size of sprites* on page 93.

When to use icons

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

- applications (including editors)
- files (including editors' documents)
- devices (such as discs and printers)
- iconised windows.

For an application, you will certainly need an application icon and may also need a file icon. However, if your application uses files of an existing filetype, use the standard icon for its files.

Appearance of icons

Application icons will appear on the icon bar and should have an irregular shape. Square or rectangular icons look dull on the icon bar and are confusing in directory displays where they tend to look like file icons (see below). You may use any colours you like for application icons. Use a background mask (transparent) rather than a grey background for sprites representing applications.

File icons should be square sprites, with a black border (Wimp colour 7). If the file is a document that 'belongs' to a particular editor, try to make the editor's icon and the document's icon look related to each other, even though the editor's icon has an irregular shape.

Device icons will often have an irregular outline. They should have a grey outline (Wimp colour 5) with cream (Wimp colour 12) as the major foreground colour. Device icons with an irregular outline must have a transparent mask.

Large and small icons

Icons that can appear in a directory display will need large and small versions. If you don't define a small icon, RISC OS will display the large icon at half size. As the appearance of a scaled-down icon is unlikely to be as pleasing as a specially designed small version, it is best to design your own large and small icons. Large icons are used on the icon bar and in directory displays that show **Large icons**. Small icons are used in directory displays that show **Small icons** or **Full info**. There is detailed information on the size the icon should be in the section entitled *Size of sprites* on page 93.

Icons and screen resolution

You will need to provide versions of the icons your application uses for standard (low) resolution and high resolution screen modes. If you think it is likely that your application will be used with the high resolution monochrome screen mode 23, you should also supply an icon for this. There is more about icons for different screen modes in the section entitled *The !Sprites file* on page 104.

Loading an application

When a user double-clicks on an application icon in a directory display, the application must load, installing its icon on the icon bar. The icon on the icon bar may include some text as well as a sprite. This may give details about the state of a device, for example. The section entitled *Positioning icons on the icon bar* on page 94 explains how to position the icon and any text attached to it.

6 Standard operations

Introduction

It is important that all applications behave in the same way when performing standard operations such as starting up and closing down. In addition, there are some common procedures used by many editors which must also be standard.

Starting an application

You must start your application if a user:

- Double-clicks on its icon in a directory display using either Select or Adjust. This should load a new copy of your application, putting its icon on the icon bar.

- Drags your application's icon to the icon bar (under RISC OS 3 only; this is handled by the Wimp).

- Double-clicks on a file icon in a directory display using either Select or Adjust, where the file 'belongs' to the application, and the application has not already been started. If the application isn't already loaded, it must start up and open the chosen document. If it is already loaded, it must just load the document.

- Drags a file to a printer icon using either Select or Adjust, where the file 'belongs' to the application, and the application has not already been started. If the application isn't already running, it must start up and print the file. If it is already started, it must print the document.

Applications should put an icon on the icon bar; only very small applications, which may be better described as utilities, do not need to do this.

Providing information about your application

The 'About this program' dialogue box is accessed from the Info item which you must provide at the top of the application's icon bar menu. The dialogue box provides useful information about your application. For example:

About this program	
Name	Edit
Purpose	Text editor
Author	© Acorn Computers Ltd, 1993
Licence	Single User
Version	1.45 (09-Jul-93)
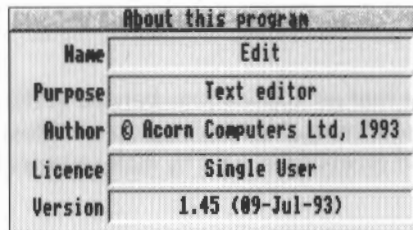
You can make some modifications to this basic design as long as the dialogue box does not become too large. It is a good idea to include a line showing the licence type. This helps a user identify the limits of allowed use.

Closing windows

If a user clicks with Select on the Close icon of a window, the application must

- close the window immediately if no work will be lost: for example, if it is unmodified, or if a view of the file is still open.

- display the dialogue box described and illustrated in the section entitled *Closing windows* on page 60 if the information in the window is not safe.

For more information on closing windows, see the section entitled *Closing a window* on page 28.

Quitting applications

The last item in an application's icon bar menu must be **Quit**. If a user chooses this item, the application must first close all windows belonging to it. If any windows contain unsaved data, the application should display a dialogue box like this:

Edit	
2 files modified	
Discard	Cancel
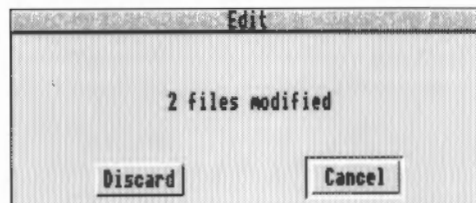
The application may only quit once all the user's information is safe or has been explicitly discarded by the user.

You must follow this procedure even if the user has used another method (such as the Task Manager) to quit the editor, or has used **Shutdown**. Your application must be able to handle Message_PreQuit messages from the operating system and warn the user of any unsaved data which will be lost if Shutdown takes place immediately.

Editors

An *editor* is an application which can create, load, display, edit and save *documents* of a particular type. A document is usually stored as a file, with a particular filetype. Most editors can load several documents at once; these are called *multi-document editors*. Examples include Draw, Edit and Paint. Editors must comply with all the rules laid down in other chapters within this Guide; this section gives some extra guidance for designers of editors.

It is important that you consider how data may be transferred between your own application and other applications. Wherever possible, allow the user to save data from the document in a standard file format (such as text, or a Draw file) to allow transfer between editors.

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

● chooses a new document option from the application's icon bar menu (this may be available if you offer several document types, for example).

If your editor needs arguments to create a new document, such as a page size, you may also use a dialogue box during this process. If a style sheet is required (for a DTP program, for example) then you may instead use a persistent dialogue box, and drag the style sheet from a directory display.

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 editor 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, first starting the editor if necessary

● 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.

In the last 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 PostScript file onto the Edit icon to look at or edit it.

It is normal for a new document to gain the input focus without the need for the user to click in the window. There is more about gaining the input focus in the section entitled *Gaining the caret* on page 67.

If the user tries to load a document which is already open on the desktop, bring the window containing the document to the front rather than opening a new copy of the document. If your application supports multiple views of the same document, this is best offered through a **New view** menu item.

Your editor should be able to load and edit multiple documents concurrently.

As soon as a user makes any changes to a new document or a document that has been loaded, the title bar must show an asterisk to indicate that the document has been modified. This is only removed when the whole document (not a selection) is saved.

Matching documents to editors

Editors use RISC OS filetypes to decide which files 'belong' to them.

An editor may only claim filetypes for which it is likely to be the primary editor. This means that it will open a window for a document if the user double-clicks on the file icon in a directory display. Your application may only claim files 'belonging' to other editors if it provides a superset of that editor's functionality; for example, you may only claim Draw files if your editor does all that Draw can, and more besides. If your application is not the primary editor for a filetype, it may still open and process a file of a type it can handle if the user drags the file icon onto your application's icon on the icon bar.

Your editor must not claim filetypes that are mainly used to exchange information between different editors, such as CSV files.

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.

22

If the document is not of a type that your editor can import, it should display a suitable error message.

Saving a document

The dialogue box you should use to save a document is described fully in the section entitled *Save* on page 56.

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
- to a different editor's icon, which loads a copy of the document into that other editor
- to another document, which inserts your document into the other document
- to a printer driver, which then prints the document.

The writable field you use in a Save as dialogue box 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. The field must not accept a pathname longer than 255 characters.

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 with the current date. This ensures that the timestamp reflects when the document was last meaningfully updated.
- Check any return codes and errors from saving the document, and take any appropriate action, such as displaying an error in an error box
- 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.

It is becoming increasingly common for applications to save current status information with a document. This may include, for example, the view scale, document-specific choices, and window position.

Holding unsaved documents in memory

Don't allow users to close document windows but retain documents and their unsaved changes in memory; clicking on the close icon must always remove a document from memory. It is very easy for users to lose information in documents that aren't currently displayed if they turn off the computer without first quitting all applications or using **Shutdown**.

The pinboard provides a way for users to keep open documents with unsaved work on the desktop in an iconised form, and this should remove the need for any other method of doing this (see section entitled *Iconising a window* on page 29). If your application needs to provide a way of hiding unsaved documents, supply it by some other method, such as a menu option.

Printing a document

You must print a document if a user

● chooses **Print** from a menu in your application

● presses the Print key on the keyboard while your application has the input focus and a suitable document is loaded

● 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.

Before printing, your application will need to display a dialogue box for users to set printing options. Sample dialogue boxes and guidelines on designing print dialogue boxes are described in the section entitled *Print* on page 55.

If your application supports printing, it must show print borders, or have an option to show them. The print borders show the user what will be printed on the page, and where page breaks fall. Your application can retrieve information about the margins set if there is a printer driver active, or use default values if no printer driver is active.

If your application supports printing, the chapter entitled *Printer Drivers* in the RISC OS 3 *Programmer's Reference Manual* gives full details of how the printer drivers work and the protocols involved.

Providing information about documents

The 'About this file' dialogue box is accessed from the item **Info** in the File menu. The dialogue box provides useful information about a document being edited. It must include the full pathname of the document. For example:

```
           About this file
        ┌───┐  Modified?    NO
        │ ⬜ │    Type Text    (fff)
        └───┘
        │  ADFS::IDEDisc4.$.TextFile  │
   Size │          1024              │
   Date │    11:41:15 24-Jun-1993    │
```
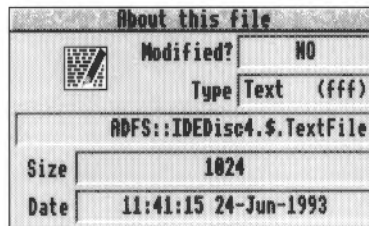
Data transfer between editors

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 support the Scrap Transfer protocol and should implement the RAM Transfer protocol for data transfer between applications. For full details of these protocols, see the RISC OS 3 *Programmers Reference Manual*.

● Your editor should 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 wherever possible be used as the standard form for structured graphic data interchange. Remember that a Draw file can include sprites, and so be used to transfer them.

Think about how users may want to use your own application with others, and try out data transfer between them to make sure you provide the kind of support users will actually need.

If you need code to render data formats, ask Acorn if it is available already to avoid duplicating the efforts of other developers.

7   Windows

Introduction

This chapter describes how windows behave on the RISC OS desktop. Much of this behaviour is enforced by the Wimp; the information is provided here for completeness. For more details see the chapter entitled *The Window Manager* in the *RISC OS 3 Programmer's Reference Manual*.

The chapter entitled *Dialogue boxes and toolboxes* on page 47 of this Guide has some extra information that is specific to dialogue boxes; the chapter entitled *Standard operations* on page 19 has some specific recommendations for editors.

The section entitled *Colours* on page 95 describes the standard colours that you must use for windows.

Parts of a window

The icons around a window have the following names:
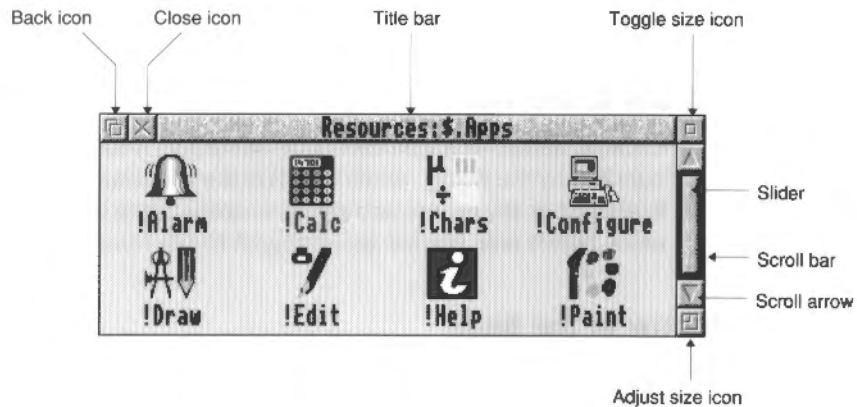
In running text in manuals and help information, use these names with initial capitals (with the exception of *slider* and *scroll bar*, which should be in lower case throughout).

RISC OS now has a 3D desktop which can be set up with an application NewLook which is freely available. The use of 3D is now common on other platforms, and many RISC OS applications are already implementing 3D. Although RISC OS 3 does not itself use 3D icons, we expect that an increasing number of applications, and finally the operating system itself, will do so.

Title bar

The title information of a window is handled by the Wimp. At present, the window title is centred, but in later versions it will be right-justified so that the leafname is always visible. Windows that represent directories use the full pathname; windows that represent files show just the leafname.

If the document in an editor window has not yet been saved or loaded, its Title bar should show a suitable default document name, such as Text file. If the document has been modified, you must append a space followed by a * to the title. You may also show the view number (if there are multiple views) and the window scale. Avoid using the Title bar to show other information about documents or files; try to use a pane or other method of showing additional information (such as whether a grid is locked on).

You can set the window's minimum size field so that the title length does not restrict the window's minimum size. If the title will fit in the title bar you should centre it; if it won't fit the window manager will right-justify it, so that at least the end of the title is visible.

Bringing a window to the front

Clicking Select on a window's Title bar brings it to the 'front' of the desktop. This is handled by the Wimp, which reorders the windows in the stack so that your window is in front of any others occupying or overlapping the same area. Resizing a window using Select (see below) also brings it to the front.

Sending a window to the back

Clicking Select or Adjust on a window's Back icon sends that window to the 'back' of the desktop, hiding it behind any windows it currently hides or overlaps. This is handled by the Wimp.

Closing a window

The effect of clicking on the Close icon of a window depends on which mouse button is used and whether the *Shift* key is pressed at the same time:

Mouse button	Without Shift	With Shift
Select	Close the window	Reduce the window to an icon and pin it to the pinboard
Adjust	Close the window; open its parent window at the front of the desktop	Open its parent window; don't close the original window

The functions associated with Select are handled by the Wimp. Your application needs to supply the functions associated with Adjust.

Whenever a window is closed and there is unsaved data, the application must offer the user the chance to save the data before closing the window. For details of what to do with editor windows containing unsaved data, see the section entitled *Closing windows* on page 60.

There is more about the pinboard in the section entitled *The pinboard* on page 10.

Iconising a window

If a user clicks on the Close icon with Select while pressing the Shift key, the window is reduced to an icon stuck on the pinboard. The file name is shown beneath the icon, which looks like this:

TextFile

The small icon within the border is application-specific.

Double-clicking with Select on the icon on the pinboard must reopen the window, preserving any unsaved edits and displaying the same area of the file as when the window was iconised.

Iconising a window is handled by the Wimp, but your application can respond to a user iconising a window, and may supply an alternative, application-specific iconised sprite. The small icon within the iconised window icon must be application specific. See the section entitled *Sprites for iconised windows* on page 95.

Resizing a window

A window will always open with a standard size. This is some sensible default that you set. Thereafter, window resizing is handled by the Wimp. If the user subsequently resizes the window by dragging the Adjust size icon, this becomes the new 'standard' size.

Dragging the Adjust size icon with either Select or Adjust resizes the window, subject to the constraints of a 'maximum' size. If Select is used, the window is first brought to the front. If possible, the maximum size shows everything over which the window can be scrolled. If this will not fit on the screen, the maximum size fills the screen instead.

Clicking Select or Adjust on a window's Toggle size icon toggles its size between a maximum size and a standard size. If Select is used to toggle the size of the window to its maximum it is also brought to the front, but its old depth is remembered. If the window is subsequently toggled back to its standard size, it resumes this depth.

Clicking on the Toggle size icon with the *Shift* key held down resizes the window so that it occupies the whole of the screen except the icon bar (or as much of the screen as it needs to show its full extent, if it isn't large enough to fill the screen).

If the window reaches the edge of the screen during resizing it grows in the opposite direction if possible – so if the window reaches the right-hand edge of the screen, but there is space on the left, the window grows to the left to increase its width. If your application window has a toolbox attached, it will have to handle repositioning the toolbox itself (see the section entitled *Toolboxes* on page 64).

Moving a window

In RISC OS 3, a window can be dragged not just to the edge of the screen but almost off the edge of the desktop. The position of the pointer on the Title bar determines how far off the desktop a window can be dragged, as the pointer can't move outside the desktop. Moving a window is handled by the Wimp.

Moving a window has implications for any tool panes attached to the window.

Scrolling a window

Normally, the Wimp handles window scrolling. This is what happens:

- Clicking Select on a window's scroll arrow scrolls the window (effectively scrolling through the contents of the window) in the direction the arrow points, and by an appropriate amount. For example, text files scroll by one line of text, taking account of the font size in use. Clicking Adjust on a scroll arrow scrolls the window in the opposite direction.

- Clicking Select on a window's scroll bar (not its slider) scrolls the window by approximately the height/width of the window, as appropriate. Again, clicking with Adjust scrolls the window in the opposite direction. There is a small

overlap between successive window views, so that it is clear how the new view relates to the previous view. For example, if you scroll a window down over some text, the last line of the old view will be the first line of the new view.

- Dragging a window's slider with Select scrolls the window in the direction of the drag, and by an appropriate amount. The amount reflects the proportion of the whole document visible in the window and how far the slider is dragged. Dragging the slider with Adjust scrolls the window in both dimensions at once (sideways and up and down).

A window cannot scroll past any natural limit, such as paper limits or the end of a file.

The length of the slider represents the proportion of the whole file that is currently visible in the window.

Context-sensitive pointers

You may want to change the shape of the pointer while it is in a window, or over some type of item in a window, to indicate to the user the sort of activity that is available. Examples of this include changing the pointer to a caret in a text window, or to a menu shape when it is over a button that leads to a menu. Used in moderation, context-sensitive pointers can help users to find their way around your application. However, over-use can be confusing and looks messy. If you want to use context-sensitive pointers, use the following established set of pointer shapes:

- *Caret* for writable icons and text areas.

- *Hand* for moving objects such as frames and windows – but not for dragging objects such as icons when the precise drop point is important, since icons are obscured by the hand shape.

- *Alter* to show that the pointer is over a handle that can be used for resizing an object (such as the handles on a Draw object).

There is an additional pointer shape associated with the 'drag and drop' method of selection which will eventually complement the cut and paste method. Drag and drop is described in the section entitled *Drag and drop* on page 78.

Dragging objects that are within a window

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 objects between windows.

If you allow drag operations within your window, check that redraw works correctly when windows move in the background.

- You can choose whether to allow the user to drag the object out of the window, or to restrict the dragged object to within the window. Normally the context will tell you which is the sensible choice.

- If you restrict the dragged object to within the window, you must automatically scroll the window if the object gets close to the edge of the window's visible area, and if more of the window lies in the direction of the drag (see below).

- If the drag works with the mouse button released then menu selection and scrolling can happen during the drag, which you might find useful.

You may use the Shift key to modify the effect of the drag from a move to a copy, or vice versa. A drag between windows without the Shift key should perform a copy. A draw within a window without the Shift key should perform a move.

The user may regret beginning a drag; pressing Escape during a drag should cancel the drag and restore the object to its original position.

Automatic scrolling

Within an editor window, selection by dragging over text or other objects should cause the document to scroll if the user holds down the Select button and moves the pointer near the edge of the window. So dragging near the right-hand border of the window should move through the document to show objects to the right (unless the window is already at its limit in that direction), and dragging over the lower border should move forwards through the document to display objects lower down (until it meets the end of the document). Dragging with Adjust to increase a selection should also give automatic scrolling.

Once automatic scrolling has started, the user should be able to increase the speed of scrolling by moving the pointer. Scrolling stops when the user releases the mouse button, stops moving the mouse, or meets the natural limits of the document. The pointer is never artificially repositioned.

This mechanism allows users easily to select material that extends over more than one windowful of information. It also allows users using the 'drag and drop' method of moving and copying selections to move through a document easily to the destination for the selection. Drag and drop is described in the section entitled *Drag and drop* on page 78.

Taking over the screen

You may feel very strongly that your application should be able to take over the entire screen, without any scroll bars or other window paraphernalia. Usually, there is no need for this to be the only method of operation, and you should make it possible to run the application in a window, perhaps with an option to run it as a single task. There is a model of how this can be handled in Acorn PC Soft: single screen operation can be chosen from the application's icon bar menu.

All applications should install an icon on the icon bar, even if they may be run as a single task taking over the screen. The application should start up when the user clicks on the icon on the icon bar. You may allow the user to make any settings specific to your application from an item in the icon bar menu. This may include redefining keys and loading saved data, for example.

The application needs to provide an easy way of returning to the desktop. Pressing either of F12 or Escape should return the user to the desktop, and your application should support both of these unless you have given Escape an application-specific function. If you offer a menu option to return the desktop, this should be called **Return to desktop** and not **Quit** or **Exit**. The application should operate in a window once it has returned to the desktop. The desktop should be in the state in which the user left it. Your application must not alter any configuration settings without first asking the user to confirm that it may, and must **never** change CMOS settings.

If there is insufficient memory available to run your application, it must display a suitable error box and not try to free memory by altering the configuration of the computer or doing anything which may cause the user to lose data. Always give the user the opportunity to save any data which may otherwise be lost.

If you are writing a game, you may want to allow users to save the state of play. It is best to offer this as an option from the icon bar menu, allowing the user to drag the file icon to a directory display in the usual way.

There is an Application Note for games writers, called *Writing games for* RISC OS. It is available from Acorn.

8   Menus

Introduction

When different developers use very different menu structures, it is difficult for users to find their way around new applications. To help overcome this, this Guide suggests a basic, general-purpose structure for main menus which can be adapted to suit many types of application, and offers some guidelines on menu design in general. You should follow these to help users come to grips with your applications. There is also a new standard way of choosing fonts from menus.

The Wimp enforces some aspects of menu behaviour, so some of the information included here is provided for completeness. For more details, see the chapter entitled *The Window Manager* in the RISC OS 3 *Programmer's Reference Manual*.

Basic menu operation

Your application must provide a single menu tree for each window type that needs menus, and an icon bar menu. A menu must be displayed when a user presses Menu within one of the application's windows that has a menu tree. This is better than using a collection of short menus, associated with different places in the window: a single menu is easier to learn about than lots of small ones, and users can quickly discover what your program can and can't do without having to search everywhere for hidden menus.

You can, however, make menu items context-sensitive. Context-sensitive menus have one or more items that change according to the object beneath the pointer when the menu is displayed or according to what object(s) are selected when the menu is displayed.

The user must be able to move all menus, submenus and dialogue boxes by dragging them. The Wimp handles the movement of menus.

Displaying menus

A menu must appear at the position of the pointer when the user presses the Menu button. The main menu must appear when a user presses Menu with the pointer inside a window belonging to your application. The icon bar menu must appear when the user presses Menu with the pointer over your application's icon on the

icon bar. Options that lead to submenus are indicated by an arrow to their right. A user must be able to display a submenu by moving the pointer to the right over an item that has an arrow beside it.

If a menu item is not available because of the context in which the user has displayed the menu, it must be greyed out (not omitted). Also, grey out any item that leads to an unavailable dialogue box.

Grey out any items that don't do anything in the current context. Don't grey out a menu item that leads to a submenu but show it in black, even if all items on the submenu are unavailable. This allows users to see quickly all the options your application offers, even if they aren't currently available.

Choosing menu items

A user must be able to press with any mouse button on a menu item to choose that item, unless it leads to a submenu. The application must perform any associated activity, which may be a task, or it may be to open a dialogue box. If the user presses Select or Menu to choose the item, the menu must then disappear. If the user presses Adjust, the menu tree must remain displayed so that the user may choose extra items.

If a user presses any mouse button on a menu item that leads to a submenu, the application should either do nothing, or should do some sensible default available on the item's submenu – for example, clicking on a **Save** item should save a document using its existing pathname (if it has one). The application may display a dialogue box if necessary: for example, if a user clicks on a **Save** item for a document that has not yet been saved, the Save dialogue box may appear.

A menu entry should have an ellipsis rather than an arrow if it leads to a persistent dialogue box (one that remains on screen until explicitly dismissed by the user). This means that the user has to select the item to display the dialogue box. There is more on dialogue boxes in the next chapter. A menu item should never have both an ellipsis and a submenu arrow.

Removing menus

If a user clicks anywhere outside a menu, the menu is removed from the screen and the click is obeyed. The user may also press Escape to remove the menu tree without making a choice; this function is provided by the Wimp.

Menu structure

Very long main menus and submenus are cumbersome and complex for users to deal with. As a general guide, you should strive for a balanced overall structure, with items that are needed frequently not hidden deep within a system of submenus.

The following example of a main menu structure might be suitable for some applications; it is intended as a guide only, not a set of rules you must follow.
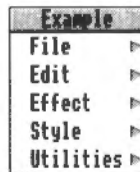
Many applications will need to offer the items **File**, **Edit** and **Utilities**; many will need one or both of **Effect** and **Style**, or suitable application-specific alternatives. The main menu items will probably lead to submenus offering tasks of related type. Some menu items that appear in many applications should be handled in standard ways to increase consistency and ease of use. These are described in the section entitled *Standard menu items* on page 40.

File menu

The File menu will typically look like this:

These are all standard items that are described in the section entitled *Standard menu items*. You may want to include some other options that relate to the whole document. Any options that relate to the application (that is, to all documents) should be included in the icon bar menu and not the File menu.

Edit menu

The Edit menu should contain all the main functions of your application. From it, users should be able to see what the application can do. Obviously the entries will be determined by the functionality of your application; the illustration below is just a guide.

```
          Edit
Cut              ^X
Copy             ^C
Paste            ^V
Delete           ^K
----------------------
New Header/Footer
Alter pages...   ⇧^A
Alter graphic... ^F11
----------------------
Select all       ^A
Clear            ^Z
```

An **Undo** function is very useful to users and you should include this if possible (and appropriate). **Undo** and **Redo** should be the first two entries in the menu.

Effect menu

The Effect menu should offer simple functions to change the appearance of text (or whatever). A typical example is shown below; if you use these items, you should use the same names and give them in the same order as far as possible.

```
          Effect
Text font            ▶
Text size      ⇧^S ▶
Text colour...
Line spacing   ⇧^L ▶
Alignment            ▶
----------------------
Bold             ^B
Italic           ^I
Underline        ⇧^U
Superscript      ⇧^J
Subscript        ⇧^K
----------------------
Paragraph border     ▶
```

If there is a selection current when the user picks an effect, the effect should be applied to the selection. If there isn't a selection, it should be turned on to apply to subsequent input.
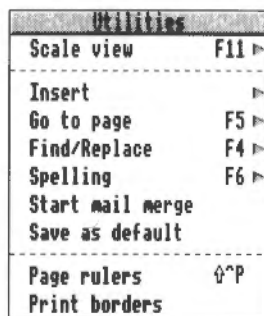
Style menu

The Style menu should offer more complex settings such as combinations of effects and additional features. Typically, it will offer some general operations and some user-defined styles.

```
    Style
Line width     ▷
Line colour    ▷
Fill colour    ▷
Line pattern   ▷
Join           ▷
Start cap      ▷
End cap        ▷
Winding rule   ▷
```

Utilities menu

The Utilities menu generally holds items that don't fit naturally elsewhere in the menu structure. However, check carefully that each item you think of putting in Utilities should really be there; don't use it as an alternative to devising a proper, logical menu structure.

```
      Utilities
Scale view      F11 ▷
- - - - - - - - - - - -
Insert              ▷
Go to page      F5 ▷
Find/Replace    F4 ▷
Spelling        F6 ▷
Start mail merge
Save as default
- - - - - - - - - - - -
Page rulers     ⇧^P
Print borders
```

Toggling menu items

Sometimes you will want to offer in the menu structure a setting that can be toggled on and off. There are three ways of doing this. In order of preference, they are:

1   Have one menu item that is ticked when set on and not ticked when set off (Use Palette, for example).

2   Have a context-sensitive menu item that changes to show the state that can be selected (Show graphics when graphics are hidden, and Hide graphics when graphics are shown, for example).

39

3   Have two menu items with opposite functions (Show graphics and Hide
     graphics, for example) and grey out the option that is currently in use.

It is worth considering whether the option would be better included in a dialogue
box. There are guidelines on presenting choices like these in dialogue boxes in the
section entitled *Standard components in dialogue boxes* on page 50.

Dialogue boxes and writable fields

Writable fields in menus are not particularly easy to use, especially for users who
have difficulty controlling the mouse. Use a small dialogue box in place of a
writable field coming from a menu item. Dialogue boxes are easier for the user
than writable fields as it is possible to click to place the caret. A small dialogue box
should typically have a field for text and an action button, such as **Save** or **Modify**;
OK is acceptable as long as the context makes its meaning absolutely clear. Where
sensible, try to maintain the text that has been previously entered in the text field.
There is more detail about the design of dialogue boxes in the next chapter.

Adapting the menu structure

Even if your application can't use exactly the structure we have outlined here for its
menu tree, try to keep close to it, particularly for the main menu and the common
features of the File menu. Users will come to expect this as the model for the main
menu in all new applications, and they will find your application easy to learn if it
follows the model.

Standard menu items

File menu

Info should lead to a dialogue box showing information about the current file; it
should not display application information, which is displayed from **Info** on the
application's icon bar menu.
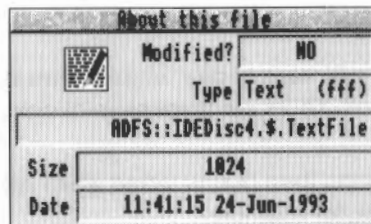
Save should lead to a Save as dialogue box which offers saving a selection (if there
is one) as an option, and shows a file icon for the application's natural file type.
The Selection button is greyed out if there is no selection, and is always turned off

when the dialogue box appears; this prevents users accidentally overwriting the whole file with a selection. Dialogue boxes are described fully in the next chapter. The Save as dialogue box with a selection button available should look like this:
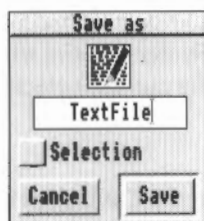
Incorporating the selection option into this dialogue box saves space in the menu tree and helps to rationalise the structure of menus. The new button replaces the **Save** item in a Selection submenu and the **Select** item in a Save submenu, which are a source of inconsistency in many existing applications.

If it is possible to save a file or selection using other filetypes, the menu should include an **Export** item, leading to an Export dialogue box or a submenu if there are several alternative file formats available. A menu of filetypes will typically look something like this:
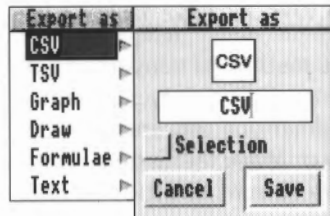
Each filetype item should lead to an Export dialogue box with the appropriate file icon shown. You should put the more commonly used filetypes at the top of the menu, and any unusual ones at the bottom, separating the two groups with a dotted line if appropriate. Only include a submenu **Other** for more unusual filetypes if your application offers so many file formats that the Export submenu becomes unwieldy. Each filetype item should lead to an Export dialogue box with the appropriate file icon shown.

There is more about the design of Save as dialogue boxes in the section entitled *Save* on page 56.

If a user clicks on the **Save** item in the File menu, the file should be saved using the default filetype. If it has been saved in that format previously, it should be saved with the same name; if it has not, display the appropriate Save as dialogue box. Some users like to be warned before overwriting an existing version of a file with a

new version; others find such warnings irritating. If your application issues warnings, make it an option that users can set on or off as one of the application's choices.

Print should lead to a dialogue box allowing the user to make choices such as how many copies to print. By default, the whole document will be printed, but you can offer options to print the current selection (if there is one) or a page range.

Edit menu

If your application uses the Cut/Copy/Paste method of moving and copying selections, you will need to include each of these items in the Edit menu.

The cut and paste method requires an application to keep a clipboard on which cut or copied items are stored until the user pastes them back in or closes down the application. Instead of a single operation, two stages are needed to cut or copy and then paste a selection.

Cut removes the selection from the document and stores it on the clipboard.

Copy makes a copy of the selection and stores it on the clipboard. The selection in the document remains in place.

Paste pastes the current contents of the clipboard into a document at the position of the caret, or replacing a selection current in the document.
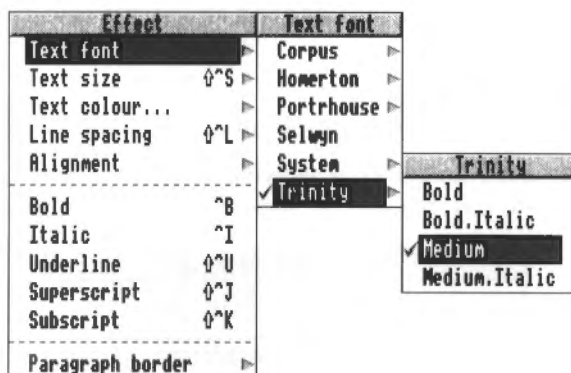
The cut and paste method of copying and moving objects or text has now established itself as the industry standard. You should use it in preference to the Edit model of highlighting text and then positioning the caret at the input point and using a **Copy** or **Move** menu item. This has been replaced because retaining a current caret and a current selection confuses many users.

In the future, cut and paste is likely to evolve into a 'drag and drop' method which will allow the user to make a selection and then drag it with the pointer to its destination. This is outlined in the section entitled *Drag and drop* on page 78; there is more about cut and paste in the section entitled *Cut and paste* on page 77.

Effect/Style menu

Font selection has been adapted to make it easier for users who want to make a change to text in several fonts, perhaps changing it all to italic or all to bold.

The following menu structure gives users the option of changing just the weight or style of the font, or of making a fully controlled font change.

```
    Effect                    Text font
 Text font          ▶    Corpus      ▶
 Text size      ⇧^S ▶    Homerton    ▶
 Text colour... ▶        Portrhouse  ▶
 Line spacing   ⇧^L ▶    Selwyn
 Alignment          ▶    System      ▶          Trinity
 ‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑     ✓Trinity    ▶     Bold
 Bold            ^B                         Bold.Italic
 Italic          ^I                        ✓Medium
 Underline      ⇧^U                         Medium.Italic
 Superscript    ⇧^J
 Subscript      ⇧^K
 ‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑
 Paragraph border   ▶
```

The menu items **Bold** and **Italic** will work over a selection that includes several fonts.

● The menu item **Bold** will be ticked if any of the selected text is in bold, or if there is no selection but bold is turned on at the position of the caret.

● The menu item **Italic** will be ticked if any of the selected text is in italic, or if there is no selection but italic is turned on at the position of the caret.

The menu items have the following effects:

● If **Bold** is not ticked and the user clicks on it, all selected text in any font will be changed to bold (retaining the same typeface(s) and angle(s)); if there is no selected text, bold will be turned on at the position of the caret.

● If **Bold** is ticked and the user clicks on it, all selected text in any font that is currently emboldened will be changed to medium (retaining the same typeface(s) and angle(s)); if there is no selected text, bold will be turned off at the position of the caret.

● If **Italic** is not ticked and the user clicks on it, all selected text in any font will be changed to italic or oblique (retaining the same typeface(s) and weight(s)); if there is no selected text, italic will be turned on at the position of the caret.

● If **Italic** is ticked and the user clicks on it, all selected text in any font that is currently italic or oblique will be changed to upright (retaining the same typeface(s) and weight(s)); if there is no selected text, italic will be turned off at the position of the caret.

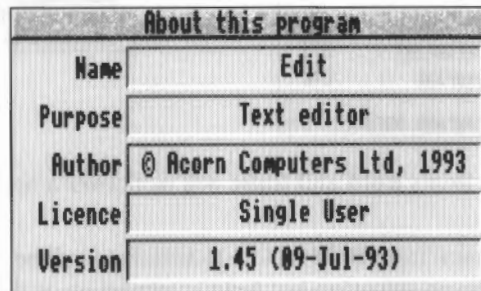An Application Note explains how to implement these menu items.

Colour selection may be of two types. Selection of 'true' colours which may be precisely defined and stored and printed with a suitable printer, but not necessarily accurately displayed on screen, should be from a dialogue box. The standard

dialogue box for picking colours is described in the section entitled *Selecting colour* on page 58. Selection of desktop colours for display may be from a dialogue box or a submenu of colours (as Edit's text and background colours are selected).

Icon bar menu

The icon bar menu for your application must have at least two items: **Info** and **Quit**.

Info displays a dialogue box showing information about your application. It should be similar to this, but you can make slight modifications:

About this program	
Name	Edit
Purpose	Text editor
Author	© Acorn Computers Ltd, 1993
Licence	Single User
Version	1.45 (09-Jul-93)

Whenever you release a new version of your application, make sure it has a new version number shown in its Info box.

Quit removes your application from memory. It must first check that there are no unsaved documents, and use the dialogue box illustrated in section entitled *Closing windows* on page 60 if there are.

The icon bar menu may include other items, including Choices. The handling of choices is described in section entitled *User choices* on page 85.

Appearance of menus

Text in menus

The following rules govern the use of text in menus:

- Items must have an initial capital, with the rest in lower case (ie 'Set type', not 'Set Type').

- Items must be left-justified (except for keyboard shortcuts – see the section entitled *Keyboard shortcuts* on page 68).

- Items must use the system font, rather than using the Font Manager. This will change in the future as RISC OS will adopt a proportionally-spaced font for menus and dialogue box text. However, this will be handled by the Wimp and need not affect your application designs at present.

- Items must (where relevant) use ticks to show they have been selected, whether by the application as a default, or by the user as a conscious choice.

- Items may be split into groups within a menu by separating them with a dotted line.

- A menu entry may be a sprite instead of text where this is appropriate.

Keyboard shortcuts

You may well want to offer keyboard shortcuts for some menu items. Details of the keyboard shortcuts used for various functions are given in the section entitled *Keyboard shortcuts* on page 68.

Pop-up menus

Sometimes you may wish to include a list of alternative choices within a dialogue box, using a pop-up menu. The procedure for including a pop-up menu in a dialogue box is described in the section entitled *Pop-up menus* on page 55. The icon to show a pop-up menu looks like this:

A pop-up menu looks and acts in all respects like an ordinary menu. It has a Title bar, by which it can be dragged around, and uses the same colours as an ordinary menu and may have items greyed out.

Size and position of menus

For the details of the size a menu should be and how it should be positioned when displayed, see the section entitled *Menus* on page 95.

9 Dialogue boxes and toolboxes

Introduction

Dialogue boxes are an important way in which users communicate with the computer and give instructions to your application. It is vitally important that the wording, function and layout of dialogue boxes is clear and easy to use. This chapter gives guidance on aspects of dialogue box design and tells you how you can make your dialogue boxes consistent and harmonious with other RISC OS dialogue boxes. You will need to give careful consideration to how to present your application's functions and options within the protocol described here.

The dialogue boxes for some standard functions – such as Save, and selecting colours – are described and illustrated. Use these standard dialogue boxes whenever appropriate.

Toolboxes are another method of interaction, but one which has been under-used in the past. They can cover a wide range of functions; the tools in Paint and Draw are examples. Generally, they can remain on screen while an application is being used, and do not disappear until dismissed by the user clicking on a Close icon or choosing a menu item.

The Wimp enforces some of the behaviour of dialogue boxes. For more details of how to construct and use dialogue boxes in your application, see the chapter entitled *The Window Manager* in the RISC OS 3 *Programmer's Reference Manual*.

3D and dialogue boxes

As RISC OS moves towards a 3D look and feel, dialogue boxes are necessarily going to change in design. This chapter sets out the appearance of dialogue boxes in 3D. Shifting an existing 2D application to 3D may involve considerable work in redesigning templates, but you will find that this is worthwhile as 3D gives a much enhanced and more sophisticated appearance to applications. It isn't generally possible to make dialogue boxes support both 2D and 3D, changing according to the user's choices, as 3D dialogue boxes often have to be slightly larger than their 2D equivalents to accommodate the larger icons and borders needed to render 3D.

RISC OS may at some point in the future display a proportionally spaced font in dialogue boxes rather than the current fixed-pitch system font. This will have further implications for dialogue box design. The new font will be handled by an

extension to the window manager and need not affect your design of dialogue boxes at the moment, except that you should avoid using spaces to align text strings – this won't work with a proportionally spaced font.

The examples in this chapter show the 3D icons you will need to use in your applications; the icons are available from Acorn. All new applications must use these new standard icons from the Wimp sprite pool, and the 3D look and feel; any that use their own, non-standard icons will look odd.

Types of dialogue box

There are two types of dialogue box:

- Persistent dialogue boxes
- Transient dialogue boxes.

Persistent dialogue boxes

A *persistent* dialogue box appears when the user clicks on a menu item that is followed by an ellipsis (...) or performs an equivalent action (such as using a keyboard shortcut). It usually suspends its parent application until it is filled in; this is not an essential feature of persistent dialogue boxes, but it is easier to implement.

A persistent dialogue box has at least one action button (such as **Save** or **Cancel**). It must not have a Close icon, but has a **Cancel** action button instead; it is not clear to a user whether any settings chosen will be implemented if he or she clicks on a Close icon. A persistent dialogue box appears after a user clicks on its parent item in the menu tree, which must have an ellipsis after it – so Style... is an example.

A persistent dialogue box is not removed from the screen if a user clicks outside the dialogue box.

Transient dialogue boxes

A transient dialogue box appears as a submenu, and functions in the same way – the Save dialogue box is an example. It has at least one action button (such as **Save** or **Cancel**) but no Close icon. It is typically small, to make it easy to browse through the functions an application offers. It is removed from the screen if the user clicks a mouse button or moves the pointer back over the menu tree.

A click outside a transient dialogue box removes the dialogue box without taking any action.

Both types of dialogue box should be characterised by delayed action: the user has to make choices and then click on **OK** (or some other appropriately named action button) before the choices take effect. This contrasts with the instant effect of, for instance, dragging a scroll bar. Toolboxes, such as that used by Paint, have an instant effect. Toolboxes are described below in the section entitled *Toolboxes* on page 64. Some of the dialogue boxes used by the Wimp (in Configure, for example) don't conform to the delayed effect rule, combining instant-effect sliders and delayed effect settings with the result that they are confusing for users.

Deciding which type of dialogue box to use

Whenever possible, use small transient dialogue boxes rather than persistent dialogue boxes.

Sometimes you may need to use a persistent dialogue box because of technical restrictions – for example, you can't use a transient dialogue box if you want the dialogue box

● to have menus of its own

● to have panes

● to have icons dragged onto it, or to remain on screen when any other mouse input is required.

It is also better to use a persistent dialogue box if, displayed from the menu, the dialogue box would be large enough to obscure the menu that called it up.

Dialogue boxes and keyboard shortcuts

A dialogue box must work in exactly the same way whether it is opened from a menu or using a keyboard shortcut.

For full details of using keyboard shortcuts, see the chapter entitled *Handling keyboard input* on page 67.

Default actions

If a dialogue box has a default action it should be clear what this is. The default action should do what the user probably intended as long as this is safe. For example, if a user has edited a file and tries to close it without saving it, the default action should be to save the file before closing it. The file is closed – as intended – but the user doesn't lose the new data.

The default action should be performed if the user presses Return. Escape must perform the same function as clicking on **Cancel**. A dialogue box must take the input focus when opened and whenever the user clicks on its window so that Return and Escape work.

The dialogue box must remain on screen if the user clicks on an action button using Adjust.

Standard components in dialogue boxes

There are various standard components that you may need to use in dialogue boxes:

- Writable fields
- Display fields
- Action buttons
- Option buttons
- Radio buttons
- Adjuster arrows
- Sliders
- Scrollable lists
- Pop-up menus
- Standard selectors (for colour, font and view scale).

The sections below explain how and when to use each of these; the chapter entitled *Implementing the design* on page 91 explains how to implement each of them.

Writable fields

Writable fields are used when the user has to type text to give a value or name. Use either validation strings or your own filtering code to make sure the field accepts only legal strings. A writable field looks like this:

Default screen mode   27

The user may want to use any of the following keystrokes within a dialogue box with writable fields, and your application should support all of them:

Key	Function
←	Move the caret to the left one character position.
→	Move the caret to the right one character position.
↓ or Tab	Set the value in the current field and move the caret to the end of the next field (cycling from last to first if necessary).
↑ or Shift-Tab	Set the value in the current field and move the caret to the end of the previous field (cycling from first to last if necessary).
Return	Implement the current settings and remove the dialogue box from the screen. The action button that corresponds to the Return key should have a thicker border than the others.
Escape	Cancel the operation and remove the dialogue box. Data must not be lost, and the environment must revert to the state it was in before the dialogue box was opened. Each dialogue box must have a **Cancel** button that is equivalent to pressing Escape.

When the user moves to a new writable field, your application should place the caret at the end of any text already in the field.

Display fields

Display fields are used to show information the user can't change by typing in the field, so the caret doesn't appear in the field. You can use it to show settings that can be altered using other elements in the dialogue box, settings that can't be changed from the dialogue box or settings that are updated automatically.

Author| © Acorn Computers Ltd, 1993

Action buttons

An action button is a 'button' users click on to cause an action to occur – usually the user will have made some settings in the dialogue box that relate to the action. An example is the **Save** button in a Save dialogue box; the user will have chosen a pathname for the file (which may be its existing name) and clicks on **Save** to save the file.

Try to use simple active verbs to label action buttons – for example, **Save** or **Print**. Don't use **Yes** and **No**. It must be clear from the label and the context within the dialogue box what the result of clicking on the button will be. Make sure the label is never ambiguous.

Most dialogue boxes will have at least two action buttons, one of which will be **Cancel**. The other(s) will offer different actions.
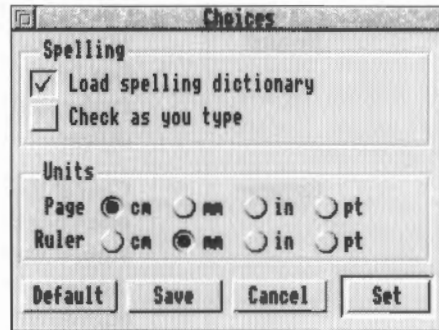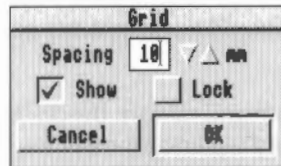
Always put the default action button in the bottom right-hand corner of the dialogue box, and use the thick border to make it prominent. It is usual to line up the other action buttons along the bottom of the dialogue box, but in some cases you may want to align them vertically down the right-hand side of the dialogue box or use some other appropriate arrangement. The action buttons should be evenly spaced along the edge they occupy.

Clicking on any action button with Select removes the dialogue box from the screen and implements the chosen action. Clicking on an action button with Adjust leaves the dialogue box on screen and implements the chosen action.

Option buttons

An option button is a 'switch', and can either be on or off. Option buttons look like this:

☑ Horizontal scroll
☐ Vertical scroll
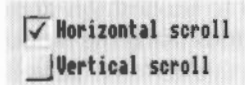
Any associated text must be to the right of an option button. Pressing either Select or Adjust over an option button or its text must toggle its state.

Only use an option button if changing the state of the button won't affect any other settings. If the current settings make an option meaningless or unavailable, its option button must be greyed out.

You should use option buttons when the user may pick more than one of the options; if the options are mutually exclusive, so that only one can be used at a time, use radio buttons.

Radio buttons

A radio button is one of a group of mutually exclusive buttons: only one may be selected at once, and clicking on one turns off the currently set button. Radio buttons look like this:

◉ Quiet beep   ○ Loud beep
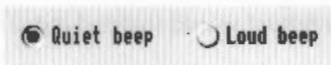
Any text associated with a button must be to the right of the radio button. Pressing either Select or Adjust over a radio button or its text must select it, and deselect any other radio button in the group that was previously selected. If there is an option to turn off all the buttons, give this as a radio button labelled None, as one button must always be on.

If there are only two settings available (such as Hide graphics/Show graphics) and this is the only option set from the dialogue box, a ticked or unticked menu item is simpler for the user than radio buttons in a dialogue box. Ticked menu items are described in the section entitled *Toggling menu items* on page 39.

Adjuster arrows

An adjuster arrow is used to increase or decrease a numeric value; it is used for setting a **Zoom** value in Draw or Paint, for example. It may be used in conjunction with a slider (described below). The up and down adjuster arrows look like this:

Font cache 1024 /△ KBytes
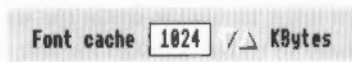
The user must be able to click with Select on an up arrow to increase a value and on a down arrow to decrease a value. It must also be possible to reverse the action of the buttons by clicking with Adjust. This means that clicking on an up arrow with Adjust decreases the value and clicking on a down arrow with Adjust increases the value. It is important to support this apparently superfluous option as some users have physical difficulties using the mouse, or may be using an alternative input device that assumes this action is possible.

Sliders

A slider is another method of altering a numeric value. It is particularly useful where a wide range of values is possible, or where the user is unlikely to know the exact number required. The proportions of red, green and blue in a colour would be a typical example.

A simple slider looks like this:

You may want to use a more complex type of slider; you can add a knob or handle to a slider that may be dragged.

Pressing Select must move the slider in one direction and pressing Adjust must move it the opposite way. So if pressing Select on a left button moves a slider to the left, pressing Adjust would instead move the slider to the right.

Standard selectors

Some settings are so commonly made in applications that it is helpful to users to have a standard method of selection. There is a standard selector for colour; this is described in the section entitled *Selecting colour* on page 58.

Scrollable lists and pop-up menus

Sometimes you may wish to include a list of alternative choices within a dialogue box. There are two ways you can do this: scrollable lists, and pop-up menus.

Scrollable lists

A scrollable list shows several of the available choices in a scrolling pane with one or more of the mechanisms for scrolling: scroll arrows, scroll bar and slider. These work in exactly the same way as in an ordinary window (see the section entitled *Scrolling a window* on page 30). The selected choice in the list is highlighted; the current selection must be visible when the window is first displayed. Users can drag the scroll bars to move through the list, find the choice they want, and then

click Select to make a choice. If it is possible to choose more than one item at the same time, the user must be able to click with Adjust to add extra items to the selection. Clicking with Adjust on a selected item deselects it.

Pop-up menus

A pop-up menu takes up less space within the dialogue box than a scrollable list.

A pop-up menu is indicated by a button beside the field showing the current selection:

Voice | **WaveSynth-Beep** ⊡

Clicking on the menu button with either Select or Menu brings up the pop-up menu, which then works in the same way as an ordinary menu.

There is more about pop-up menus in the section entitled *Pop-up menus* on page 45; the section entitled *Pop-up menus* on page 96 explains how to position a pop-up menu.

Standard dialogue boxes

Some options are offered by many applications and we can achieve a greater degree of consistency on the desktop if all developers offering these options use the same dialogue box to make settings.

Use the dialogue boxes or guidelines described below if you need to support these functions:

- Print
- Save
- View scale
- Find/replace
- Colour selection
- Font selection
- Closing a window which may contain unsaved information.

Print

Printing may be a simple or a complex operation depending on the type of application. If you are offering a simple screen-dump type print, you will need to show the printer driver loaded and allow the user to set the number of copies to

print. If at all possible, include a scale option as well; typically, this will let the user set the print size by giving a percentage of full size. Use a dialogue box like this, showing the name of the configured printer in the Title bar:
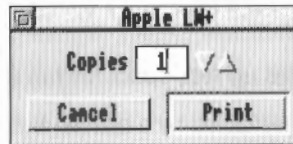
If printing is very important to your application, you will want to let the user set many more options. Here is an example of a more complex dialogue box for **Print**:
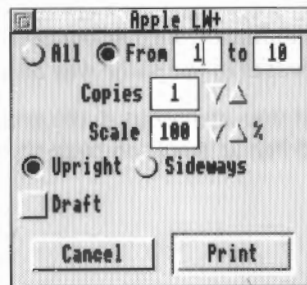
Make it easy for non-expert users to make settings. For example, use the terms 'upright' and 'sideways' instead of 'portrait' and 'landscape' for page orientation, and consider illustrating the options with an icon. (For some products and some markets, this type of simplification may be inappropriate.)

Save

A **Save** dialogue box looks like this:

If there is a selection in effect when the user calls up this dialogue box, and it is possible to save the selection, include a Selection button so that the user may save just the selection.
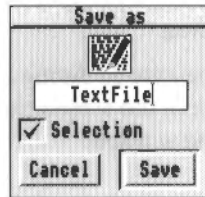
When there is a selection, change the default pathname to Selection to prevent users accidentally overwriting the whole file with a selection.

The button can be greyed out when there is nothing selected; the button is always turned off when the dialogue box appears.

The file icon shown must be the right icon for the filetype being used. If you offer saving as different filetypes, use the appropriate icon for the type the user has chosen. The section entitled *File menu* on page 40 explains how to offer different filetypes for saving.

If the file has already been saved using the filetype displayed in the icon, the full pathname of the file must be displayed in the writable field. This enables the user to click on **Save** or press Return to save the file with the same name, or edit the name in the field to save it with a different name. If the file has not been saved previously, the name in the field should be the same as the default name for the file shown in the Title bar. Providing a default name allows the user to drag the file icon to a directory display immediately without generating an error message. When the file has been saved, the name in the Title bar is updated to the new filename and the * removed until further changes are made to the document.

The writable field in a Save dialogue box 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. The field must not accept a pathname longer than 255 characters.

Scale view

If the user can scale the view, use a dialogue box like this:
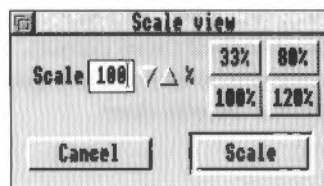
You may offer different scales on the action buttons, and a different number of standard scales, depending upon the requirements of your application. Clicking on one of the buttons offering a value should enter this value in the writable field but **not** add an automatic Return; the user must still click on an action button or press Return to initiate rescaling.

It is useful to users to be able to define a box on screen by dragging with the pointer to show the area of the screen they would like to look at in detail. This area is then rescaled to fit the window.

Find/Replace

The options you want to offer as part of a find/replace facility will depend on your application and how you expect users to use it. The following illustration is a guideline only.
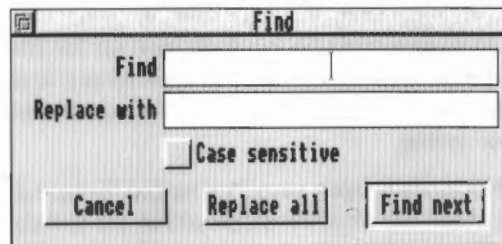
It is a good idea to allow a search to be case sensitive, and to allow users to restrict it to a selection or some other sensible subset of the whole file. However, searches should be case insensitive by default, with case sensitive as an option.

Selecting colour

There are four common ways of defining colours:

- RGB, or red-green-blue
- HSV, or hue-saturation-value
- CMYK, or cyan-magenta-yellow-key (black)
- Instant selection of desktop colours.

Computer monitors and other cathode ray tube (or CRT) displays make colour by mixing light beams of red, green and blue (RGB). The colour we see is the result of adding proportions of each of these colours. To define colours using this method, we need to specify the proportions of red, green and blue displayed on the screen; this is what the RISC OS palette does.

Most users find the RGB colour selection method easy to use and this is the system's own method of defining colours. Using it in your applications increases the consistency of the desktop. If your application needs HSV or CMYK, you can of course use it. If possible, though, offer RGB as an option for users who are not familiar with other methods of colour definition and give full documentation of how to use the others. Remember that the standard 16-colour screen modes give poor approximations of cyan and magenta.

An RGB colour selector looks like this:
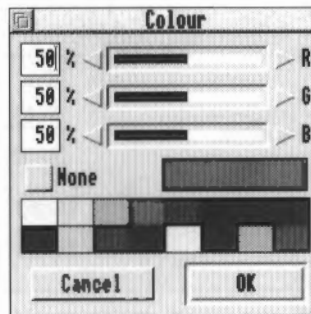
It offers users three methods of adjusting the value of each colour component:

* Sliders
* Writable fields
* Adjuster arrows.

The values for each colour are shown as percentages, not 256ths. There is no decimal point value. Although this limits the range of colours that can be defined, it is much easier for users. It is ideal where simple colour definition is required; you may prefer to offer HSV and/or CMYK if colour selection is particularly important (as it may be in a graphics application, for example) or if it may be relevant for output (if the user may be preparing colour separations, perhaps).

Selecting fonts

Font selection is sometimes of great importance in an application; often it is just a fairly basic setting that is not crucial to the function of the application. Font selection can be intimidating for users, and it is best to provide emboldening and italicisation separately from font selection. For example, a user may decide to set some text in bold. Choosing Homerton.Bold from a long menu listing all the fonts available in the Fonts directory is an unnecessarily technical procedure that may discourage inexperienced users from using different effects. It also doesn't allow the user to set the whole of a piece of text that uses more than one font to be emboldened in a single action. While it is important to retain a mechanism for full

59

control over fonts for experienced users, there is a need to provide a simple way of applying bold and italic effects to help inexpert users make the most of applications.

There is a suggested menu structure for font selection in the chapter entitled *Menus* and you should use this for font selection through the menu tree. This is suitable for minor changes the user may want to make while typing. Sometimes, though, you may want to offer font selection from a dialogue box. This is more suitable when the user is likely to be making several settings, as when defining styles. In this case, use this standard selector:
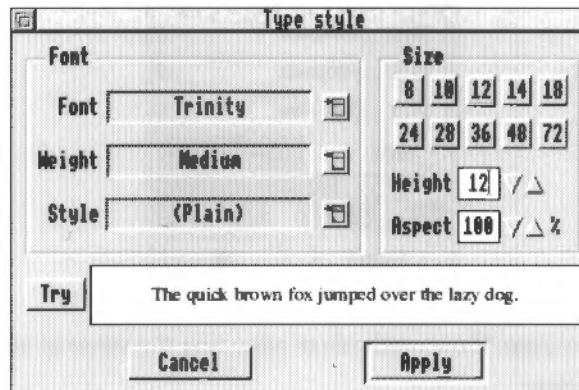
This is quite easy for users to use as bold and italic effects are set after the font has been chosen. A selection of standard sizes is offered and you may like to include a writable field for users to give a different size. Don't offer height and width settings, but use aspect ratio instead to adjust the proportions of text. This uses the fixed height of the chosen point size, but alters the width to give the proportion specified. For example, an aspect ratio of 50% will give characters that are half their normal width.

The **Try** button allows users to check that they have chosen what they really want before applying it to their text. It must show text in the chosen font, effects (if any) and aspect ratio.

When it isn't obvious that the dialogue box is a better means of font selection, use the menu structure suggested.

Closing windows

If your application is an editor of any type, it is possible that a user may click on the Close icon of a window that contains unsaved information. If this happens, your application must not close the window and discard the user's work without

warning, but must display a dialogue box like this, giving the user the chance to save the file, discard the work done in the window, or cancel the operation leaving the window open.
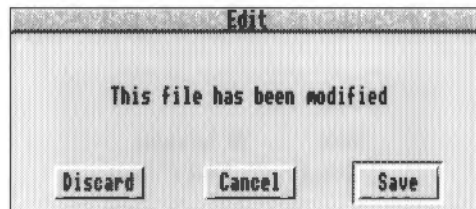
There is more about the wording used in this dialogue box in the section entitled *Wording of dialogue boxes* on page 62.

Appearance of dialogue boxes

The chapter entitled *Implementing the design* on page 91 explains how to create and position the elements of a dialogue box.

Size of dialogue boxes

First and foremost, strive to make your dialogue boxes small, simple and comprehensible for first time users. A whole screenful densely packed with controls is likely to discourage and intimidate the user. Think through which operations are easiest to understand and most commonly used and hide the others away or remove them altogether. Don't provide options which, in practice, no one will use.

If you find that a dialogue box has to be very large to include everything that you need to include, you will need to consider dividing it up in some way. The options are:

- Split up the task the dialogue box performs so that you can use more than one dialogue box
- Use action buttons to lead to further dialogue boxes
- Use a set of radio buttons to switch between different parts of a dialogue box
- Use pop-up menus to replace scrolling panes or radio buttons where appropriate.

If you can't avoid splitting up a dialogue box, action buttons are usually the best method of providing extra options. Make sure that the most frequently needed options or those most likely to be used or understood by a beginner, are at the top level.

Grouping items

In a large dialogue box, you may like to group together all the items that relate to a particular setting or subject. You can do this using a group box. This is a box that encloses the related items, and has a label overlaying the top of the box:


Don't over-use group boxes, so that everything in your dialogue box is grouped, and don't put just a single item in a group box. Don't nest groups. If an entire group is rarely used, consider making it a separate dialogue box.

Wording of dialogue boxes

Remember that the point of a dialogue box is to enable users to make choices, perform actions they want to perform and receive any feedback from the computer about what they have done. Using clear, plain English will help users and will make your application easy and pleasant to use.

●   Don't use ambiguous phraseology or wording on buttons, and make sure the wording on or by any buttons is a valid answer to the question as you have phrased it.
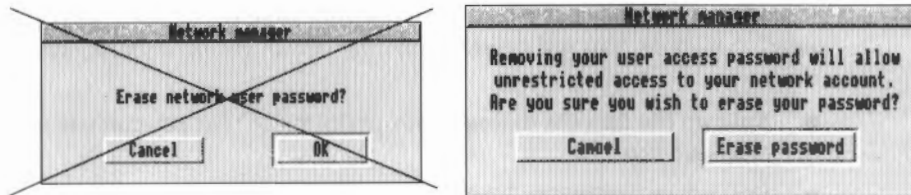
● Use wording the user will understand; avoid jargon unless you know your users will understand it.
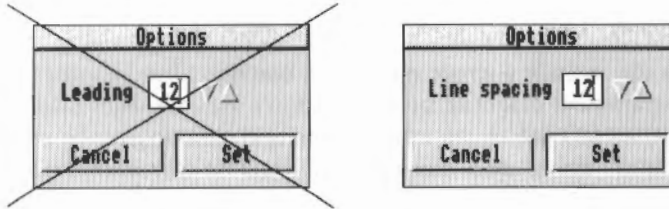
● Label the action buttons with an accurate description of their function – don't use **Yes** and **No**. Aim to use active verbs, such as **Print** or **Save**. If a dialogue box has allowed a user to make many settings and a single active verb is not appropriate, use **OK** rather than, for example, **Yes** or **Go**.
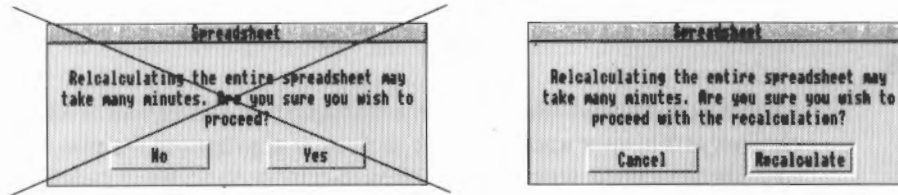
See the section entitled *Standard dialogue boxes* on page 55 for details of the dialogue box you should use when a user tries to close a document that has not been saved.

There is more information on action buttons in the section entitled *Action buttons* on page 52.

Error messages

One type of dialogue box in which wording is particularly important is error boxes. Your application will need to display an error message if the user attempts an action that isn't allowed, or if the application goes wrong or can't find some resource it needs.

All error messages should use simple, plain English with no jargon. Don't include diagnostic messages of help only to programmers, but tell the user simply what has gone wrong and say what the user can do (if anything) to correct the situation. If your application is reporting a potentially serious problem, make that clear, and don't obscure it with polite phrasing. Examples of suitable messages are:

```
Not enough memory to open a new window
```

```
No printer driver loaded: load a printer driver before
printing
```

```
Width must be less than 2.5cm
```

```
Please fill in the Value field
```

You can include a single number in brackets after an error message to help programmers diagnose problems; don't use messages such as 'Address exception at &0004A23' which are intimidating and meaningless to users. Avoid words such as 'fatal', 'abort', and 'corrupt', which can be very worrying for users. If your application has crashed irretrievably, use a polite message that all users can understand, such as:

```
Sorry, Appl has suffered an internal error and must close
down immediately.
```

A short apology is acceptable (as in the last example); put it at the start of the message, not the end. Don't over-use apologies.
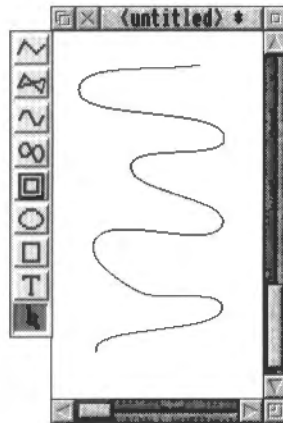
Toolboxes

A toolbox is a useful alternative to a dialogue box as a means of letting users make choices within your application. A toolbox is appropriate if a user is likely to want to make selections repeatedly, as they may from a toolset or palette. Toolboxes have been under-used in the past, but offer a valuable, flexible and easy method for users to make commonly-needed choices.

The Colours and Tools windows in Paint and the toolbox in Draw are typical examples of toolboxes. A toolbox may be associated with a particular window – for example Paint's colours – or be shared by several windows, as Paint's tools may be. Toolboxes may be free-standing windows, as Paint's Colours and Tools windows are, or attached to another window, as Draw's toolbox is. There are other alternatives; a toolpane may be included as a row or panel of buttons within a window, for example.

Toolboxes remain on screen until explicitly removed by the user or until the application or window is closed down.

A toolbox in a pane can be turned on or off from a menu item; the menu item is ticked when the pane is displayed. The pane disappears when a user closes the window it is attached to.

A tool box in an independent window has a Close icon, Back icon and Title bar like any other window and may have scroll arrows, a slider and scroll bar, and an Adjust size icon. A toolbox in its own window must be implemented as a standard RISC OS window. It is displayed when a user calls it up from a menu and closed when the user clicks on its Close icon or when the object it is associated with is closed.
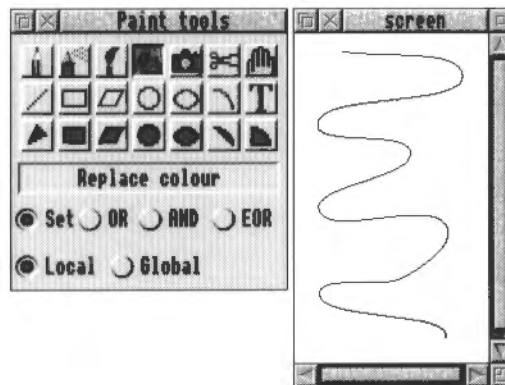
The buttons in a toolbox must be 3D, as in the new Paint and Draw toolboxes.

10   Handling keyboard input

Introduction

Many applications will support text input in their windows; even those that don't will often need text input to dialogue boxes. This chapter explains how to handle input from the keyboard, including keyboard shortcuts.

Gaining the caret

The caret is a single, red, I-shaped bar which shows where input from the keyboard will appear. The window containing the caret is said to have the *input focus*. When it first gets the input focus, sometimes called *gaining the caret*, the caret moves from a different window to the new one. Your application may only gain the caret if:

- a user clicks with Select or Adjust inside your window
- a user calls up a menu or dialogue box
- a user clicks on the application's icon on the icon bar to open a new document.

If the user clicks in an application window, the window has the input focus and receives text input until the user clicks in a different window or calls up a dialogue box or menu. The Title bar of a window changes colour when the window gains the input focus. This is handled by the Wimp under the control of the window template.

If the user calls up a dialogue box, the first field in the dialogue box will have the input focus. In dialogue boxes that do not take text input, input is usually from the mouse. However, it may still be from the keyboard, as users can use *Return* to choose the default action icon and possibly other keys to choose other action icons. Your application must surrender the caret when the menu or dialogue box is closed. Normally RISC OS automatically reassigns the input focus. It is often your application's main document window that will regain the caret in these circumstances.

A window should not automatically come to the front of the desktop when it gains the caret. Similarly, if your application brings a window to the front of the desktop, it should not automatically gain the caret. A window should not gain the caret just because the pointer passes over it, but only when the user takes some definite action, such as clicking in the window.

If your application loses the input focus, it doesn't need to remember the position of the caret or the selection. When the application regains the input focus, the action the user takes (such as clicking in the window) may also place the caret. If the user's action doesn't place the caret, the application should claim the input focus but not position the caret in the document.

Unknown keystrokes

If your application receives a keystroke that it doesn't recognise or can't use, it should pass it on to other applications using Wimp_ProcessKey rather than claiming it. This allows other windows to provide hot key operations that work anywhere; it also allows the Wimp to interpret function key presses if necessary. If your application doesn't pass on unknown keystrokes, F12 won't work while your application has the input focus.

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 access operations they use frequently. To avoid confusion, common commands should have consistent shortcuts across different applications.

- There is a small set of common function key shortcuts that you should use if your application supports the functions. These are listed in the section entitled *Function keys* on page 70.

- There are some common shortcuts that are alternative methods of choosing menu items offered in many applications.

- Some keys on the keyboard have specific functions (such as the Print key); your application should support their normal functions.

- There are some standard keys and key combinations to help users move around files.

The tables over the following pages show the keyboard shortcuts you should use. The left columns show the abbreviation for the shortcut – use this in your menus – and the right columns give a description of what each shortcut does.

Whenever you offer a keyboard shortcut for a menu option, you need to show the shortcut to the right of the item in the menu. Function keys must be referred to as F1, F2, etc. following the labels on the key-caps. Unless you are offering only single key-press shortcuts using the function keys, you will need to use these symbols for *Control* and *Shift* in your menus:

- ^ means hold down the *Control* key while pressing the other key(s) indicated, eg ^X is *Control-X*.

- ⇑ means hold down the *Shift* key while pressing the other key(s) indicated, eg ⇑F3 is *Shift*-F3.

- ^⇑ means hold down the Shift and Control keys while pressing the other key(s) indicated, eg ^⇑F3 is *Control-Shift*-F3.

The character code for ^ is &5E and the code for ⇑ is &8B in the system font.

Typical menu entries would look like this:

```
┌─────────────────────────┐
│░░░░░░░░░░Edit░░░░░░░░░░░░│
├─────────────────────────┤
│ Cut             ^X      │
│ Copy            ^C      │
│ Paste           ~V      │
│ Delete          ^K      │
├─────────────────────────┤
│ Alter pages...  ◊^A     │
│ Alter graphic...^F11    │
├─────────────────────────┤
│ Select all      ^A      │
│ Clear           ^Z      │
└─────────────────────────┘
```

Control and Shift

The *Shift* key is the natural modifier for any function key, so use this to provide similar functionality to the unshifted key, but with some subtle modification. Use the *Ctrl* key to provide different functionality. So for a function key F*n*:

Key presses	Action
F*n*	some function
⇑F*n*	a modified form of F*n*
^F*n*	some other function (probably unrelated to F*n*)
^⇑F*n*	a modified form of ^F*n*

Alt

The *Alt* key is used by RISC OS as a shifting key to generate international characters and in the future is likely to be used to support a revised system of keyboard shortcuts more compatible with other non-Acorn systems. Because of this, you must **not** use the *Alt* key for keyboard shortcuts.

69

Function keys

The function keys are often used to call up editing and filing operations within applications. The table below shows the function keys you should use for some operations that are common to many applications. Where a function corresponds to one your application provides, you must use the shortcut below rather than any other. If you don't provide one of the functions below, you can use its shortcut for some other function – but don't allocate a different function to F12, *Shift*-F12, *Ctrl*-F12 or *Shift*-*Ctrl*-F12 as these are all used by the operating system.

Abbreviation	Action
F1	Help
F2	Load named document
⇑F2	Insert named document
^F2	Close window
F3	Save document
F4	Find/Search and replace
F5	Go to…
F6	Sort
F8	Undo
F9	Redo
F12	Give access to * Commands using the command line interface – do not use this key for anything else
⇑F12	Bring the icon bar to the front of the desktop
^F12	Open a task window
^⇑F12	Shutdown

Menu shortcuts

There are several functions that many applications provide from their menus. To build up consistency between applications and so help users find their way around new programs quickly, all applications must use the same shortcuts for the same functions. Some of these will be the function key shortcuts listed above, but others use the ordinary keys with the *Control* key. You must use the following shortcuts if you provide a shortcut for the functions listed in the table.

Key combination	Action
^U	Delete line
^Z	Clear selection
^C	Copy selection to clipboard
^X	Cut selection to clipboard
^V	Paste clipboard contents at cursor position

Key combination	Action
^D	Insert date
^T	Insert time
^A	Select all
^B	Change selected text to bold
^I	Change selected text to italic

The final two shortcuts must correspond to the font selection method described in the section entitled *Effect menu* on page 38. There is some advice on keyboard shortcuts and international support in the section entitled *Language* on page 89. Where you need to refer to an arrow key in a keyboard shortcut, you will need to give the name of the direction in full (for example ^Left for *Ctrl-Left arrow*). This is because neither the system fonts nor the fonts used in later versions of the Wimp have characters for the arrow keys.

Named keys

Several keys have their functions shown on the key-caps. It is confusing to users if these keys do not do what they claim to do. Your applications must support the following key-presses:

Key	Action
Esc	Cancel operation
Return	Begin a new line of text in an editor window. In a dialogue box, perform the default action.
Print	Print document
Tab	Move to the next tab position in a text editor window. In a dialogue box, set the value in the current field and move the caret to the end of the next field (cycling from last to first if necessary)
Shift-Tab	In a dialogue box, set the value in the current field and move the caret to the end of the previous field (cycling from first to last if necessary)
Insert	Paste in the current contents of the clipboard at the position of the caret
Backspace	Delete the character to the left if there is a caret; cut selection to the clipboard if there is no caret (as ^X)
Delete	Delete the character to the left if there is a caret (as Backspace); cut selection to the clipboard if there is no caret (as ^X)

Copy	Delete the character to the right if there is a caret, or copy selection to the clipboard if there is no caret (as ^C)
Home	Move to start of document (as ^↑)
PageUp	Scroll window up (as clicking top of the scroll bar)
PageDown	Scroll window down (as clicking bottom of scroll bar)

The table above describes the current functions of the keys. In the future, non-proprietary PC keyboards may be supported, in which case the *Copy* key will be named *End* and the functions of this, *Backspace* and *Delete* will be as follows:

Backspace	Delete the character to the left if there is a caret; delete selection if there is no caret (as ^X)
Delete	Delete the character to the right if there is a caret; delete selection if there is no caret (as ^X)
End	Move to the end of the document (as ^↓)

Moving around a document

Besides moving around a document using the scrolls bars, users must be able to move around using some standard keypresses. Your application must support those shown in the table below.

Key		Action
←	→	Move left/right by a character
↑		Move up a line. In a dialogue box, set the value in the current field and move the caret to the end of the previous field (cycling from first to last if necessary)
↓		Move down a line. In a dialogue box, set the value in the current field and move the caret to the end of the next field (cycling from last to first if necessary)
⇑←	⇑→	Move left/right by a word
⇑↑	⇑↓	Move up/down by a page (like clicking on the scroll bar background)
^←	^→	Move to start/end of line
^↑ or *Home*	^↓	Move to start/end of document

Applications ported from other systems

If you are porting an existing application from another operating system (or are writing an emulation of one) you may feel there is a strong case for not changing the keystrokes it uses, so that existing users of the package do not need to learn new keystrokes. However, there will be more new users who are already familiar with RISC OS than there will be existing users moving to RISC OS, so use the shortcuts described above rather than the originals. If you wish to supply a compatibility mode, offer it as an option from the menu or allow the user to choose from the Choices dialogue box.

Special needs support

Remember that not all users have full mobility, vision and hearing and may need to use input devices which are not supplied as standard. Any RISC OS compliant application will benefit from supporting input from devices such as concept keyboard, switches, trackerball and touchscreen technology. If you would like advice on special needs support, contact Acorn.

11   Handling selection

Introduction

Many applications allow the user to make selections of text or other objects and then do something with the selection. This chapter explains how to handle this.

Selecting text

If your application supports text selection, use this method supporting the following options:

- Clicking Select to set the caret position.
- Dragging Select in any direction to select a range of text.
- Clicking or dragging Adjust to adjust the extent of the selection, either forwards or backwards.

Using the following conventions will make your application more powerful and consistent:

- The caret **cannot** appear at the same time as a selection.
- If the user types when there is a selection, the selected text is deleted and replaced with the new text.
- A double-click when setting/dragging a selection should select words.

  At the simplest level, a word should consist of a sequence of alphanumeric characters between spaces. It may also include any following non-alphanumeric characters such as punctuation that come before the next space, but not a newline character.

  You may use a more complex model if you wish to provide 'intelligent' delete, copy and move functions to preserve spaces between words, and to retain correct punctuation.

Selecting objects

There are several established rules governing the selection of objects. Your application should follow these rules:

Simple selection

● Clicking Select over an object deselects all other objects, and selects that one. If there are several objects beneath the pointer, the 'front' object is selected (if your application recognises this concept).

● If the user clicks Adjust instead of Select, the state of the object clicked on is toggled between selected and deselected.

Box selection

● Clicking Select outside an object and then dragging in any direction creates a rectangular *select box*. One corner is given by the position in which Select was clicked, the other by the pointer's current position:
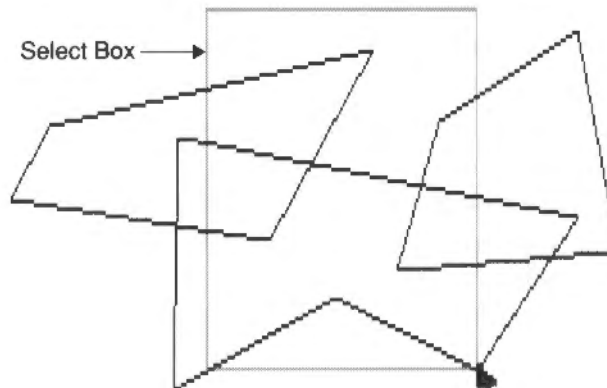
Any object that is partly or wholly within the select box is selected. It is also useful to offer an option which allows only objects wholly within the box to be selected. Draw does this if you hold down Shift while dragging. If the user presses Adjust instead of Select, the state of the object clicked on is toggled between selected and deselected.

Selecting from stacked objects

The method for selecting one object from a stack of objects varies between applications. The following describes how Draw handles selection from a stack. This method is appropriate if only graphic objects are involved, but if it is possible for text and graphics to be stacked in your application, double-clicking to select an object hidden behind text won't work.

• When a user double-clicks Select over a stack of objects, the topmost object that is already selected is deselected, and the next one down the stack is selected in its place. This wraps around from the bottom of the stack to the top, so if the lowest object is already selected, the top object becomes selected instead.

Cut and paste

The cut and paste method of copying and moving objects should be adopted for new applications as this is taking over from alternative methods. Eventually, cut and paste will be augmented by drag and drop, which is described briefly below.

The cut and paste method requires an application to keep a clipboard on which cut or copied items are stored until the user pastes them back in or closes down the application. Instead of a single operation, two stages are needed to cut or copy and paste a selection.

Copying a selection

To copy a selection, the user has to copy the selection to the clipboard and then paste it back into the same document or another document.

• To copy selected text or objects, the user must copy the selection to a clipboard. It will overwrite anything currently stored on the clipboard. The selection must also remain in its original place in the document.

  The existing contents of the clipboard are lost when another selection is cut or copied to it.

• The user needs to position the caret and use **Paste** to insert the copy into the same document or a different document.

• Text should be pasted in immediately after the position of the caret. Objects of other types should be pasted in so that the top left-hand corner of the object is at the position of the pointer.

Moving a selection

To move a selection, the user has to cut the selection from one part of the file and paste in to another part (or another file).

• To move a selection, the user removes it from its original position and stores it temporarily on the clipboard. This operation is called **Cut**.

• The user needs to position the caret or pointer and use **Paste** to insert the cut selection into the same document or a different document.

• Text should be pasted in immediately after the position of the caret. Objects of other types should be pasted in so that the top left-hand corner of the object is at the position of the pointer.

A clipboard may be shared between applications, allowing a user to copy or cut text or objects from a document using one application and paste it into a document opened with another application.

These guidelines only apply to moving a selection using the cut and paste method. You can also move a selection within a document by dragging; see the section entitled *Dragging objects that are within a window* on page 32

Drag and drop

Although cut and paste is the current method for copying and moving selections, it is likely that this will eventually evolve into 'drag and drop'. This involves selecting the text or objects to be moved or copied and then dragging the selection to its destination, whether within the same document or in a different document. The likely procedure is as follows:

● The user makes a selection in the usual way, and may adjust the selection using Adjust.

● If the user clicks with Select over the selection, it is deselected.

● The user drags the selection using Select from its original position to its destination. The destination may be in the same document or a different document.

● Dragging *within* a window moves the selection by default, or copies it if *Shift* is held down. Dragging *between* windows copies the selection by default, but moves it if *Shift* is held down.

● During dragging, a ghost caret may appear in the destination window to allow the user to position the dragged selection precisely. For text selections, the ghost caret will typically resemble the normal caret. For non-text selections, it may take another shape, such as the bounding box of the dragged selection scaled to the destination window's view scale.

● If the user holds the pointer near the edge of any window while dragging the selection, the window should scroll to show the previously hidden area of the document.

● If the user drags the pointer across the boundary of a window, copying between windows becomes possible.

● When the user releases the mouse button, the selection is dropped into its new position.

An application note explains how to implement drag and drop; contact Acorn for a copy.

12   Colour and sound

Introduction

Colour and sound are valuable ways of adding meaning to the information used by an application. Used well, colour or sound can add significantly to the value of an application.

Colours and the palette

Users may choose to set their own palette for a number of reasons, ranging from personal preference to impaired vision. Applications can't therefore rely on the default palette being used, but must read and use the current palette and handle changes of palette on the fly. Rather than the old GCOL mechanism of setting colours, use one of the following methods to avoid problems for the application and the user:

- Use the standard desktop 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. This method doesn't restrict the application to the limitations of today's hardware.

Even if your program doesn't use many colours, you must check it works correctly in all modes. Take particular care to check that operations like EOR (exclusive OR) work correctly, because the results will differ depending on whether the mode in use has 1, 2, 4, 8, 16 or 24 bits per pixel. Similarly, check two-colour modes carefully; these use ECF patterns (stippling) for different shades of grey, and again using EOR may give unexpected results.

An application note on colour selection using the standard colour selector is available from Acorn. The standard colour selector is described in the section entitled *Standard selectors* on page 54.

Guidelines for using colour

Colour increases the amount of information your application can convey to the user. It can make things stand out, attract attention, and highlight differences and similarities between things. However, it can also cause confusion, obscuring

important elements of the screen display if it is used without thought. Remember, too, that some users have only monochrome or greyscale screens; the Acorn A4 portable has a greyscale screen.

The following guidelines will help you use colour successfully:

Test your design in monochrome

Use form and text as the main means of communicating with a user; use colour to *add* meaning, not to be of central importance. For example, you shouldn't use colour as the only difference between two icons. There are several good reasons for this:

- Not all users have colour displays
- Good desktop colour printing is not yet widely available
- A significant proportion of users can't distinguish between some colours.

The best test is that your design should work on a monochrome or a greyscale display as well as it does on a colour display. The only time you can reasonably justify using colour alone to give meaning is where you are asking the user to select or define a colour. Remember that although it is becoming relatively rare to use a simple monochrome display, the use of high-quality greyscale monitors is increasing in some fields. Make sure your applications look good and work well in greyscale.

Use colour with restraint

A large number of gaudy colours on the screen looks a mess, distracts users, and so devalues your application. Although the system can display up to 256 colours on the screen at the same time, research has shown that when colours are being used to convey information, people can only work effectively with a maximum of half a dozen different colours.

Limit the number of colours you use to take account of this, and make them significantly different from each other – though they don't have to be bright. Let users change the colours if they wish; this is particularly helpful for users with impaired colour vision.

Make colours stand out

Any colours you use will stand out best over white or grey backgrounds, rather than over other colours. Blues are easily overlooked by the eye, so you should not use them to convey important information. You can turn this into a benefit, however, if you use blue for gridlines or other guides.

You must always make pointers stand out from their background. You can do this best by using contrasting colours for the background and pointer.

You must highlight text by reversing it out of its background, as you would on a monochrome screen. The text must adopt the colour of the background, and vice versa.

Avoid coloured text

A high contrast between text and its background is necessary to make text legible. You should use black for text, and a colour with a high brightness for the background – such as white, grey or yellow. Reversed-out text – white or grey text on a dark background – has good legibility, but is not a good match for the style of the RISC OS desktop.

Where to use colour, and where not to

Resist the temptation to over-use colour to make your application 'pretty'. You should only use colour within your application's work area(s), to show its data.

Other parts of your application's displays (such as window borders, menus and dialogue boxes) must use the standard colours defined elsewhere in this Guide, to be consistent with other applications. The main exceptions to this are

● where you need to present colours for the user to select one

● where you need to display a file icon

● in the application's icon bar icon.

You may, if you wish, use non-standard colours in the **Info** dialogue box displayed by your application.

Sound

Like colour, sound provides an additional channel of communication between the computer and the user. Like colour, too, it can add to the meaning of an application, but, if handled badly, can equally well detract from its value.

Sound can be used in two main ways:

● As a warning, to alert the user that something has happened (that mail has been received, for example) or that action needs to be taken – an error message acknowledged, or a choice made in a dialogue box.

● As data within an application that processes sound.

Much of what is said below applies to the first case, but not to the second.

Guidelines for using sound

Sound should be used in much the same way as colour – with restraint and to add meaning.

The design should work with sound turned off

Use sound to **add** meaning to an event, not as the only way of marking an event. There are good reasons for this:

- some users may be away from their machines, or distracted by a phone ringing when the event occurs

- a significant proportion of users have impaired hearing.

If you use sound as a warning, back it up with a visual indication.

Use sound with restraint

A jolly jingle in an application or game might be delightful to the ear the first time it is heard, but can soon become very irritating. Don't overdo the sound, and allow users to turn it off if they wish.

Melodious but not too subtle

You should avoid harsh and raucous sounds – they can be annoying and even frightening to users, and will set a tone for your application which you might not have intended. You may like to give users the option of replacing sounds with alternative modules.

At the same time, if you are giving meaning to your sounds, you should use different voices rather than different notes in order to distinguish between sounds.

- Very few users have perfect pitch and so can remember precisely how a particular note sounds.

- Many users cannot distinguish between similar notes, unless they hear them back to back, but most users will be able to distinguish (say) a 'crash' from a 'beep'.

Controlling volume

You should give users the option of turning off sound altogether. You must instruct your users to use the Configure application to set overall volume levels (and, of course, your application must respond to those settings), rather than providing your own mechanism. Don't include any instructions in your application that turn off sound globally (such as *Speaker Off); always allow users control over the sound in their system, and make settings only for your own application.

13   Configurations and user choices

Introduction

It is important that your applications support all possible combinations of hardware and software that a user may wish to use, and that they respect as far as possible the user's choice about configuring the machine. This is becoming increasingly difficult as more types of peripheral become available, but it is essential if your applications are to be accessible to as large a market as possible. Your applications must support

●   the hardware a user has bought and must use

●   the software preferences a user has chosen to use (such as screen mode).

If it is appropriate, your application should allow users to set up and save choices about how your application behaves and appears.

Hardware configuration

You will need to bear in mind that a user may have any combination of RISC OS computer, monitor and printer and may have only 1MB of RAM.

Monitors

Most importantly, the user may have a colour, greyscale or monochrome monitor which may be an ordinary RGB monitor, or multi-sync, VGA, SVGA, LCD display, or even a TV screen. Don't make assumptions about the type of monitor a user may have and therefore about screen size and the screen modes that may be in use or available.

Users will have chosen a screen mode that is suitable for the type of monitor they need or can afford, and will have decided on the resolution they want. Don't try to override that choice by changing the screen mode; make sure your applications can start up in any mode. Make the application read the current screen mode when it is loaded (and any associated information such as resolution and aspect ratio). If it is impossible to support some screen modes, the application should display an error message if it can't operate in the current mode. Make sure your application supports modes 12 and 27. Depending on the nature of your application, you may want to make sure Mode 23 works for users with big monochrome screens, but this is really only necessary for specialist applications likely to be used with these monitors.

Screen modes

As the range of monitor types and screen modes increases, it is getting more important to make sure that applications are independent of screen mode and can work in any screen mode that the window system can use. In particular, make sure your applications work with the square pixel modes used on multi-sync monitors as well as the rectangular pixel modes; VGA-only monitors can't display modes 12 and 15 correctly.

Don't forget the requirements of LCD screens used with the Acorn A4 that display 15 levels of greyscale, and the need to support the special needs mode 22. Also, check that applications work in modes 13, 15, 16, 35 and 36. Mode 16 is highly non-square – 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 correct the aspect ratio.

Remember that users may change screen mode while your application is running, so it must be able to handle changes of screen mode on the fly. It also means you can easily move your application to new and better screens and modes when they become available. New modes using 16 and 24 bits per pixel will be introduced in the future.

Screen size

Because users may have different size screens, you can't rely on screen size, so work in OS graphic units thinking 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.

Printers

If your application produces output for printing, it must produce output in a standard format that can be handled by the RISC OS printer drivers. Do not assume that users have a particular type of printer, and don't make assumptions that will prevent users adding new types of printer as they become available.

Software configuration

Users will have made choices using Configure and the Palette about the way they want their computer to behave. This may include default screen modes, sound volume and voice, mouse speed, colours used in the palette, instant effect window drags and so on. They may have based their choices on their own priorities regarding processing speed and use of the computer's resources, on whimsical preference or on some more pressing consideration such as impaired vision or

restricted hand movement. You should not make changes to the configuration, but should instead issue a dialogue box asking the user to make changes if the current settings are not suitable for your application.

Other applications

Remember that users will want to use the multi-tasking facility of their computer and may run your application alongside others. Make sure that your application works with as many others as possible, and particularly with very popular applications, and check that there are no conflicts in resource names and so on. Always register the names of your applications with Acorn, apply for filetypes and any other resources you need. Make sure your environment strings are prefixed with the registered application name (eg Application$String). These precautions avoid conflicts with names used in applications produced by other developers.

User choices

If it is possible for a user to set choices that will be used each time your application is run, you should offer this as an item called **Choices** in the icon bar menu. This should display a dialogue box allowing the user to implement and save choices.
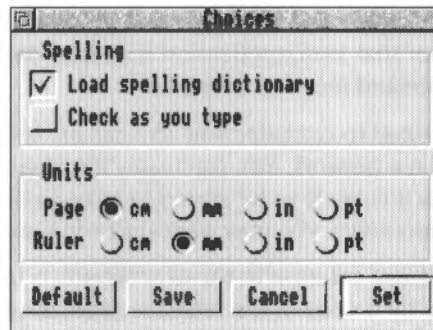
Any choices which are not currently available should be greyed out. The four action buttons are:

● **Default** resets the default values set within the application.

● **Save** saves the choices in the Choices file to be reused in subsequent sessions and implements the choices immediately.

● **Cancel** resets the choices in use before the dialogue box was displayed

● **Set** implements the choices for the current session.

In some applications, a single dialogue box for setting all choices may not be appropriate. If there are several mechanisms for setting different groups of choices, your application may instead have a menu item **Save choices** in the icon bar menu that saves all the choices set within the application. If the user doesn't use **Save choices**, any choices set are used for the current session only. The RISC OS 3 Printers application uses this method for setting and saving choices.

The choices a user sets should apply to all documents used within your application for which they are relevant. They will typically include default modes of operation and features of the appearance of documents. There is information on storing the choices a user has made in the section entitled *The Choices file* on page 107.

Network considerations

If you write your applications to be independent of the filing system in use, they should run on a network without difficulty. However, there are a few points you need to bear in mind to make your applications easy to use over a network.

- Avoid using a large number of resource files as this makes applications slow to start up over a network.

- If your application must write to its own files, make these separate from the application itself so that they can be stored locally.

- Remember that your application may read data (including choices) from a write-protected location or medium.

You will also need to consider how choices are to be saved if the application is to be installed on a network. You can't save them into a Choices file within the application as it is likely to be stored in a read-only location and is in any case shared between users who may want individual settings.

A general-purpose solution to these problems is under discussion and you should contact Acorn for a copy of the relevant application note.

14 Writing applications for CD-ROM

If your application is to be distributed and run from CD-ROM, remember that the access time for CD-ROM is slower than for disc and that performance will be affected by the CPU speed. The following guidelines will help you to write software that works well from CD-ROM:

- Don't use System, Font or Scrap directories on the CD-ROM as this may cause problems after the CD-ROM has been dismounted. If you need to include any modules or other shared resources, put them in a SetUp directory.

- Don't include any applications other than your main application in the root directory of the CD-ROM; other directories beginning with the character ! in the root directory increase the application start up time and use up memory.

- Include no more than 40 files in a directory to achieve optimum performance and directory search times.

- Make sure disc names are not longer than twelve characters, and filenames not longer than ten characters (or eight characters if you want PC-compatibility). Only use upper case alphabetic and numeric characters, underscore (_) and ! to comply with ISO 9660 restrictions on names. Always use system variables to refer to the disc or directories on it.

- Don't assume there is a hard disc connected to the system.

- If your CD-ROM includes Replay movies, group them together in a directory Replay in the root, at the beginning of the disc. Follow the instructions in the ProgIf file in the ARMovie directory. (You need a licence from Acorn to ship ARMovie.)

- Keep application-specific resources together in a directory called Data in the root directory.

- Don't allow your application to generate large data files when it starts up.

- Read data in large chunks as reading data involves considerable delays before data start to flow.

- Try to support computers with only 2MB of RAM; you can optimise your application for high resolution monitors, though, as most systems with a CD-ROM drive have a VGA or multisync monitor attached.

- Check all data and applications for viruses before pressing the CD-ROM.

- Press a gold disc to trial the software before pressing the CD-ROM in large numbers.

- Instruct users to set CDFS buffers to a non-zero value; 16K is usually acceptable, but 32K or 64K may be more appropriate if a lot of disc accessing will be needed.

- You can assume users with CD-ROM drives are using RISC OS 3.1 or a later version.

- As CD-ROM is a read-only medium, don't include anything that would appear to let users write to the CD-ROM; open files on the CD-ROM for read only.

- It is best to include all data and applications needed by the CD-ROM on the CD-ROM itself. You may also like to include extra material, such as curriculum materials, on the disc.

15 International support

Introduction

It is very important to support users whose first language is not English. RISC OS computers are sold in many non-English-speaking countries and RISC OS documentation is provided in some other languages; more languages are likely to be added in the future. Every step you make towards helping users understand programs in their native language helps your sales in the international market.

RISC OS already provides multiple alphabets/keyboards to support international use. It is also possible to translate ROM messages to another language; if you need translations for a particular language, ask Acorn as we may already have or know of an implementation. Other international support facilities are planned for the future.

The following guidelines will help you to accommodate non-English use of your applications.

Language

Remember that English is not the first language of all users.

- Consider translating any mnemonic shortcuts (such as ^D for 'down') which require the user to know the word for the required action, using the first letter of the translated word.
- Avoid using culture-specific icons, or icons that are 'puns' on a name.
- Use pictorial icons rather than text/picture combinations.

Character sets

International users may have their computers set to a different territory setting and may want to use accented and other characters that are not part of the standard British character set. The following guidelines will help you to write applications that support non-British character sets.

- Don't trap or use the Alt key in your applications.

  Different forms of international keyboards have standardised the use of *Alt* for entering accented characters; allow RISC OS to interpret its use. If you don't, users may be unable to type some of the accented characters they need.

- Don't forbid the use of top-bit-set characters in your program.
  Again, this may prevent users using accented characters.

- Don't assume that Latin1 is the current character set.

- Don't assume the user has a standard British/American keyboard layout.

- Use the operating system facilities for alphabetic sorting, lower/upper case testing and conversion as they handle accented as well as unaccented characters.

Information formats

Some information, such as dates and decimal numbers, is represented differently in countries other than Britain.

- Use system facilities for date and time string conversions. This allows the user to choose the format required (month-day-year, for example). If you use your own mechanism for setting information like this, non-British users may not be able to set the format they are familiar with.

- Include an option to recognise comma (,) as a decimal point in values used for calculations and in determining the positioning of text with a decimal tab.

The part on Internationalisation in the RISC OS 3 *Programmer's Reference Manual* deals with internationalisation issues.

16 Implementing the design

Introduction

Once you have designed a new application, you will begin to write code for it. This Guide does not offer any advice on structuring or writing your program; you will find all the detailed advice you need about writing the program in the *Programmer's Reference Manual*.

This chapter and the next give advice on

- precisely how to position and create the elements of the user interface described in earlier chapters of the Guide
- building the application directory for your application.

It is assumed that this chapter and the next will be used by programmers and so more technical language has been used in some places than in the other chapters of the Guide. Technical terms are included in the glossary.

Choice of programming language

We strongly urge you to program using C. We believe that you'll find large applications easier to maintain if they're written in C rather than (say) BASIC or Assembler.

You may feel that applications developed using C will be larger and slower than those developed using the ARM assembler. However, there need not be a significant difference if you're careful to write efficient code. You can incorporate chunks of assembly language in C programs for parts where speed is critical. Even so, BASIC or assembly language may be more suitable for a few tasks, particularly if speed or code size is very important. It is up to you to balance the benefits of speed and code size on the one hand against development time, ease of maintenance and ease of porting your code between environments.

If you wish to write your applications in C, you will find the manual supplied with Acorn's Desktop C compiler useful. The chapter entitled *How to use the template editor* is especially relevant; you can use the template editor to design windows for the desktop interactively. The chapter entitled *How to write desktop applications in* C is also useful, as you may wish to use the RISC OS library that it describes.

If you have ANSI C Release 3 or earlier we strongly recommended you upgrade to the current version, as this contains far more extensive support for writing RISC OS applications.

If you wish to write BASIC programs on your RISC OS computer you need the BBC BASIC *Reference Manual*.

If you wish to write your applications using the desktop assembler, you will find the manual supplied with Acorn's Desktop Assembler useful.

Using legal operations

To make sure your applications will work well on computers released in the future, only use legal methods in your programs

● do not bypass operating system interfaces or access hardware devices directly

● do not read and write page zero locations (the hardware vectors, etc) or kernel workspace

● do not use illegal interface operations.

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, so it is sensible to build your software to last.

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 redrawing are worth striving for as they make it easier for a user to make effective and productive use of your application.

Redrawing speed

All applications 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 application from the Applications Suite.)

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.

Units of measurement

Sizes and positions are given in this chapter in OS units where possible; you should work in OS units rather than pixels whenever you can so that you don't restrict your applications to particular monitor types and screen modes. Where dimensions are given in pixels, they are assumed to be 4 OS units high by 2 OS units wide.

Sprites

Design sprites to use as little valuable system resources as possible. Sprites for icons are normally defined in mode 12 – but if you can use a mode with fewer colours or lower resolution (such as mode 9) then do so. Don't forget to include sprites for high resolution screen modes in !Sprites22. Rectangular sprites do not need a transparency mask; those with irregular outlines do.

Check the appearance of your sprites in one, two, four, eight, 16 and 24 bit screen modes; the Wimp will do its best to translate from mode 12 colours to those available.

You will need to provide versions of the sprites your application uses, including file sprites and corresponding ic_icons, for standard resolution, high resolution and high resolution monochrome screen modes (if appropriate).

Size of sprites

Sprites that can appear in a directory display will need large and small versions in each of high and standard (low) resolutions. If you don't define a small sprite, RISC OS will display the large sprite at half size, but this is unlikely to look as good as a specially designed small sprite.

Large icons

A large icon must be 68 OS units high. Try to use a square sprite for file icons, or a square bounding box for application icons (which have an irregular outline). The square will obviously be 68 OS units wide. For mode 12 this is 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 small icon to have the same proportions

● as wide as 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 borders one pixel high.

Small icons

A small icon must be half the size of a large icon – that is, 34 OS units high. Again, it should preferably be square if it is a file icon (ie 34 OS units wide), or have a square bounding box if it has an irregular outline. In mode 12 this corresponds to 17 pixels wide by 9 pixels high (rounding up halves). If you **have** to, you can make a small icon up to 50 OS units wide but avoid using non-standard sizes as it make the directory display untidy.

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.

There is information on how icons are 'made known' to RISC OS in the section entitled *Application resource files* on page 103.

Positioning icons on the icon bar

When you place an icon on the icon bar, put icons relating to physical devices and resources such as filing systems on the left, and others on the right. RISC OS uses the icon's width to position it horizontally, but 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 shows you how to position icons vertically on the icon bar:
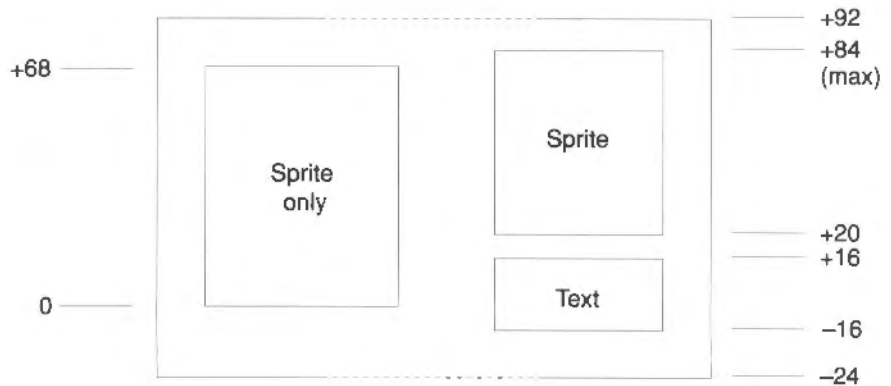
In the diagram, y coordinates are given in terms of the icon bar work area origin; lower coordinates are inclusive, and upper co-ordinates are exclusive.

Your application must position icons with text underneath them 16 OS units below the icon bar's work area origin, and those without text level with it.

Sprites for iconised windows

The sprites used for iconised windows should be the same size as large icons. You can look at the iconised window icons in the ROM if you want to see an example.

Windows

The first window your application opens must be horizontally and vertically centred on the screen, whatever the current screen mode. It should occupy no more than a quarter of the screen, to emphasise that your application does not replace the existing desktop world, but is merely added to it. Open any subsequent new windows at an offset of 48 OS units moving down the screen, unless there is a good reason not to do so. The initial size and position of windows may be user-configurable and saved as a preference.

Colours

Standard colours you must use for the application window are

- black (Wimp colour 7) on a grey background (Wimp colour 2) for the title when it is not highlighted (that is, when the application doesn't have the input focus)
- black (7) on a cream (12) background for the title when it is highlighted (when the application does 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.

Menus

Each menu item must be 44 OS units high. Try to keep the width of submenus as small as possible; this reduces the amount of mouse movement users need to reach an item, so making it faster and easier for them to use the menu.

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

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

```
┌──────────────┐
│   Example    │
│ File       ▷ │
│ Edit       ▷ │
│ Effect     ▷ │
│ Style      ▷ │
│ Utilities  ▷ │
└──────────────┘
```
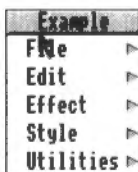
Icon bar menus

For icon bar menus, the base of the menu must be 96 OS units from the bottom of the screen. This stops the menu obscuring the icon bar sprites.

```
┌──────────┐
│   Draw   │
│  Info  ▷ │
│  Quit    │
└──────────┘
```

Pop-up menus

When a pop-up menu appears, it must appear immediately to the right of the button the user clicked on to display it.

```
                               ┌──────────────────────┐
                               │     Paper sizes      │
Paper │   A4 (Generic DP)   │◫ │ A2 (Generic DP)      │
                               │ A3 (Generic DP)      │
                               │✓A4 (Generic DP)      │
                               │ Fanfold (Generic DP) │
                               └──────────────────────┘
```
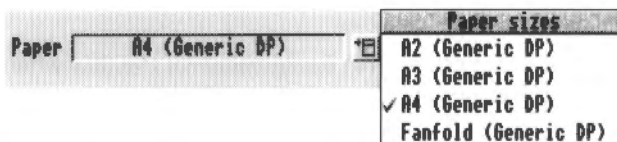
Menu colours

The standard colours you must use for a menu are

- black (Wimp colour 7) on a grey background (Wimp colour 2) 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.

Dialogue boxes

You can use FormEd to prepare dialogue boxes. It is important to edit templates in a TV-resolution screen mode (such as mode 12) and check them in high resolution modes afterwards to make sure they work in both.

Size of dialogue boxes

The size of a dialogue box will depend on what it has to include; there is some advice on considering size when designing a dialogue box in the section entitled *Size of dialogue boxes* on page 61. However, it should not be larger than 800 by 600 OS units. The proportions of an A4 page (1:1.414) can be used to give a shape that is pleasing to the eye. When working out how large to make your dialogue box, you will need to bear in mind the sizes of the standard components set out in the table below.

Component	Vertical size	Horizontal size
Action button	52 OS units	text +5 pixels each side
Default action button	68 OS units	text +9 pixels each side
Radio button	44 OS units	as needed
Option button	44 OS units	as needed
Writable field	68 OS units	as needed
Display field	52 OS units	as needed
Slider	40 OS units	as needed
Adjuster arrow	32 OS units	32 OS units
Text label	40 OS units max per line	as needed

Leave 8 OS units clear space between components.

Creating elements of dialogue boxes

To create the controls and fields you may need to include in a dialogue box, use the following instructions.

Default action button

This is a text icon of click button type, with black foreground (Wimp colour 7) and grey background (Wimp colour 1). It is vertically and horizontally centred and has a verify string 'R6,3'.

A default action button is 17 pixels tall; its width is large enough to hold the text, plus three pixels clear each side within the button, plus a further six pixels each side for the remainder of the icon.

Where possible, use a single word for the text label, preferably an imperative verb such as **Print** or **Save**. You can use **OK** if there is no sensible alternative; don't use **Yes** or **Go**.

There must be only one default action button on each dialogue box.

Action button

This is a text icon of click button type, with black foreground (7) and grey background (1). It is vertically and horizontally centred and has a verify string 'R5,3'.

An action button is 13 pixels tall, with three pixels below the baseline of the text and 10 above. Its width is large enough to hold the text, plus three pixels clear each side within the button, plus a further two pixels each side for the remainder of the icon. All action buttons in a set in a dialogue box should be the same width.

Where possible, use a single word for the text label, preferably an imperative verb such as **Cancel**. If the action button leads to a further dialogue box, the label must end with an ellipsis (...).

Display field

This is a text icon, with black foreground (7) and grey background (1). It has a validation string 'R2'.

A display field is 13 pixels tall, with three pixels below the baseline of the text and 10 above. Its width is large enough to hold the longest likely text, plus three pixels each side.

A display field can't be directly edited by the user, but the value it shows may change as a result of making other settings.

Writable field

This is a text icon of writable button type, with black foreground (7) and white background (0). It has a validation string 'Ktar;Pptr_write'.

A writable field is 13 pixels tall, with three pixels below the baseline of the text and 10 above. Its width is large enough to hold the likely text, plus three pixels clear each side within the button, plus a further six pixels each side for the remainder of the icon.

Pressing Return after giving input to a writable field should activate the default action button, not move the caret to the next field. The section entitled *Writable fields* on page 50 describes the keystrokes that can be used in a dialogue box.

Option button

This icon has text and a sprite; it is button type Radio. It has a black foreground (7); the background is not filled; the border is turned off. It has a validation string 'soptoff, opton'.

You will need to create the icon and fill in the text. The option button must be vertically centred, but not horizontally centred.

Radio button

This icon has text and a sprite; it is button type Action. It has a black foreground (7); the background is not filled; the border is turned off. It has a validation string 'sradiooff, radioon'.

You will need to create the icon and fill in the text. The radio button must be vertically centred, but not horizontally centred.

Set the ESG (Exclusive Selection Group) value to a non-zero figure.

Adjuster arrows

These are always presented in pairs. Each icon consists of text and a sprite and has button type auto-repeat. The background is not filled and the border is not set. The down arrow has a validation string 'R5;sdown,pdown'; the up arrow has a validation string 'R5;sup,pup'.

Each sprite is 32 OS units square; each icon is just large enough to hold the sprite.

The two arrows are usually positioned side by side. When they are, there is no space between them, and the down arrow is always to the left of the up arrow. The adjuster arrows should be to the right of the item they control, with eight OS units space between it and the down arrow.

If the adjuster arrows increment a value in a display or writable field, they must be aligned horizontally with the field, with their lower edge one pixel below the baseline of the text in the field. If the field has an associated text label showing the units or a % symbol, the adjuster arrows should be right next to the field, with the label eight OS units to the right of the arrows.

If the adjuster arrows are separated by a line or graphic (as they may be if you are using them to adjust the dimensions of a square, for example), leave 8 OS units between the line and each arrow.

You may occasionally want to use a left/right pair of adjust arrows; they are created in much the same way as up/down arrows.

Slider

A slider comprises three icons: the well, the background and the value. These must be numbered in the order given so that they stack correctly. The instructions below are for a simple horizontal slider.

The well is an icon with no text. It has a validation string 'R2'. The background is unfilled. It is nine pixels tall and as long as the slider.

The background is an icon with no text. It has a white background (0); the border is not set. It is three pixels tall and ten pixels shorter than the well. It is centred inside the well.

The value is an icon with no text. It has a grey background (5); the border is not set. It is three pixels tall; its width is sufficient to display the value of the slider. It is vertically aligned with the background, and the left end is coincident with the left end of the background.

For a vertical slider, the well is 18 pixels wide, the background and value are eight pixels wide and the background is six pixels shorter than the well.

You may design more complex sliders if appropriate.

Pop-up menu icon

This is an icon made up from text and sprite, with button type click and with a validation string 'R5;sgright,pgright'. The border is not set and the background not filled.

The icon is 44 OS units square, the size of the sprite.

The pop-up menu icon is centred vertically and set to the right of the value it relates to.

If an item has both adjuster arrows and a pop-up menu, the pop-up menu icon is set to the right of the adjuster arrows. However, this combination can look cluttered and is best avoided if possible.

Text label

This is a text icon, with black foreground (7) and the background unfilled. It is vertically centred; if it abuts an item on the right, it is right-justified.

A text label is 9 pixels tall, with two pixels below the baseline of the text and seven above; leave one pixel between lines of text. If the line contains any other items, such as a writable field, the height of the line will be dictated by the tallest element in it. The width of a text label is sufficient to hold the text.

Don't terminate a label with a colon (:). The text associated with a radio or option button is created as part of that button and not as a separate text label.

Group box

This comprises two icons: the box itself, and a label. The box must have a lower icon number than the label and any items inside the box.

The box is a large icon surrounding all the contents of the box. It has an indirection string 'R4'. It has no text; the background is not filled.

The label is the same as a text label described above. It is positioned to overlay the top face of the box at the left, leaving 16 pixels of the top face visible at the left. It has a black foreground (7) and grey background (1).

Leave eight pixels horizontally, or four vertically, for clearance inside and outside the box. For very large boxes, increase the clearance slightly.

Scrolling pane

Use a real pane window for a scrolling pane. Don't include a Title bar, but label it with an ordinary text label. Don't put a group box around the pane.

Set the size appropriately, but remember that a very short scroll bar makes the pane look cluttered. Allow some variation in the size of the scroll bar.

Dialogue box colours

Use these standard colours for a dialogue box:

- Black (Wimp colour 7) on a grey background (Wimp colour 2) for the title.
- Black (7) on a grey (1) background for the body.
- The title bar changes to cream (12) when the window has the input focus.

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 writable fields then use colour 1 rather than 0 as the window background. Large expanses of white background can make writable fields harder to see.

Positions of dialogue boxes

Open a dialogue box called from a menu so that it is centred on the mouse pointer (subject to screen boundary constraints).

All error boxes must be centrally positioned on the screen.

17 Application directories

Introduction

You must place your RISC OS applications in a directory whose name begins with '!', such as !Draw. When you refer to the application in documentation or help text, however, you should leave the '!' off the name. The Filer modules provide various mechanisms to help such applications. For example, the Filer will run its boot file, load its sprites and make its help information available.

There is also provision for handling shared resources – ones that may be of use to other applications. This is explained in the section entitled *Shared resources* on page 107.

Application resource files

You can hold any form of resource within an application directory. There are several standard ones; for those your application uses, it must use the filename(s) given below. An application may not need all of these resources.

!Boot	*Run by the Filer when it first displays the application directory
!Sprites|*nn*|	Passed to *IconSprites by the !Boot file, or the Filer, the files !Sprites, !Sprites22 and !Sprites 23 (if provided) contain the application's sprites for different screen resolutions
!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
!Help	Information about the application; it is run by the Filer when the user chooses Help from the Filer menu
Choices	User choices

In addition, many applications will have an accompanying ReadMe file to give release notes. This should not be held within the application directory.

Most of these resources are discussed in more detail below.

The !Boot file

A file called !Boot inside your application directory will be executed when the application directory is first 'seen' by the Filer. It is usually an Obey file – a list of commands to be passed to the command line interpreter. (The *Obey command is documented in the RISC OS *User Guide* and the RISC OS 3 *Programmer's Reference Manual*.)

You will probably use a !Boot file to set up the icons, filetypes and corresponding system variables that RISC OS needs so that it can show your application in a directory display and run it when you double-click on its icon. If your application is called Appl, this might involve:

● setting Alias$@RunType_*ttt*, Alias$@PrintType_*ttt* and File$Type_*ttt* variables

● loading !*appl*, sm!*appl*, file_*ttt* and small_*ttt* sprites from the !*Appl*.!Sprites file (see below).

However, an application should only grab filetypes on start-up if filetypes are not currently set. This means that instructions such as SETFile$Type, Alias$@RunType and Alias$@PrintType should only be executed if File$Type, RunType or PrintType are not set at all (null) when the user starts that application.

The Filer only runs the !Boot file if an application with this full pathname has not been 'seen' before. 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 !Sprites file

Your application directory must contain sprite files called !Sprites (eg !Appl.!Sprites), !Sprites22 and !Sprites23 (if provided). These must provide sprites for the Filer to use to represent your application's directory; sprites for standard (low resolution) screen modes are held in !Sprites, sprites for high resolution screen modes are held in !Sprites22. Each sprites file needs to have both large and small versions of the application's sprite and the sprite ic_sprite which will be used to represent iconised windows from the application on the pinboard. For an application !Appl the large and small sprites must be named !appl and sm!appl respectively. (The names of the sprites must be in lower case.) The !appl sprite is also used when the application is installed on the icon bar. There is more about the design and size of these sprites in the section entitled *Sprites* on page 93.

!Sprites (and the other sprite files) can also provide sprites for data files that your application 'owns'. Again, you will need sprites 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. For example, the sprites used for a Maestro file are called file_af1 and small_af1.

All the sprites in !Sprites are merged into the Wimp's shared sprite pool using *IconSprites. If your application uses any private sprites, you must load them in the Sprites resource file inside your application, and your application must load them into a private sprite area. If there is a standard sprite available from the Wimp's sprite pool, use this as users will already be familiar with it. For example, icons for standard filetypes are available from the sprite pool (see below). Your application must not redefine sprites in the pool automatically.

Standard icons provided

If your application creates or uses one of the following standard filetypes, you must not provide a file_*ttt* icon for it but use the standard icons. Many of these are provided in the Wimp sprite ROM area, for example:

Sprite	Type
file_ae9	Alarm
file_aff	DrawFile
file_fc6	PrntDefn
file_fc8	DOSDisc
file_fcc	Device
file_fca	Squash
file_fd6	TaskExec
file_fd7	TaskObey
file_fe4	DOS
file_fea	Desktop
file_feb	Obey
file_fec	Template
file_fed	Palette
file_ff2	Config
file_ff4	Printout
file_ff5	PoScript
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

There are also two sprites named application and small_app, which are used for applications which don't have a sprite called *!appl*.

There is more detail in the RISC OS 3 *Programmer's Reference Manual*. The list changes frequently; you can obtain a copy of the current list from Acorn.

The !Run file

The !Run file is *Run when a user double-clicks on the application directory. It is usually an Obey file. It is common to duplicate much of the !Run file within the !Boot file to make sure Boot actions are taken even if the application is run using a command (perhaps as part of a desktop boot file, for example). Don' t execute !Boot from within !Run.

Although the presence of more than one application with the same name should be thought of as an unusual case, it should not cause anything to crash. Your application should issue an explanatory error message and should not crash if it can no longer find its resources after program startup.

The Messages file

A text file called Messages must be used to store all an application's textual messages, including menus, help text, etc. It is easy to replace your application's messages with a set in a different language if you decide to supply your application on the international market, simply by switching the Messages file.

Try to make your application read in every textual message when it starts up. It must not read them only as they are needed, as this forces a user of a floppy disc-based system to have your application disc permanently in the drive. Make sure all error messages are read in when the application starts up, so that an error message can be displayed immediately when required without the need first to display a request for the disc holding the messages.

The !Help file

The !Help file is used to store plain text that provides brief help about your application and its function. If this file is present, the Filer adds a Help entry to its menu so a user can display the help text.

The Choices file

A file called Choices is used to store user-settable choices so they are preserved from one use of the application to the next. These choices will of course apply to all users of a particular copy of your application. Consequently this scheme will not always be the most appropriate to use; for example, a copy of your application on a network may well have many users, each of whom wishes to save a different set of options. You may prefer to use an alternative scheme, such as reading an environment string. There is more about supporting choices in the section entitled *User choices* on page 85.

Environment strings

An environment string is a string of characters representing the setting of user choices. It should be stored in a system variable called *Appl*$Options where *Appl* is the name of your application. In general, this method makes it more difficult for users to set such choices as they need to understand *commands in order to do so. You should use it only for rare or unlikely settings.

If you do choose to use environment strings, use only the application name to preface the environment string. Remember to register the application name with Acorn to prevent clashes with other applications.

Shared resources

Some resources are of general interest to more than one program. Typical examples include fonts, patches to RISC OS, and modules that provide general facilities.

Your application will be slightly harder to install if you use shared resources. Make sure that your application checks that the resources are available and gives helpful error messages if it can't find them.

The System and Merge applications

The System application is used to hold shared resources provided by Acorn and other system-wide resources which may be shared between applications. (Its !Boot file sets a system variable named System$Path giving its full pathname.) An application called Merge (or SysMerge) is used to manage the updating of System to a newer version; contact Acorn for further details.

The resources in the System application can be shared amongst many users, and are typically only needed when a program is loading. Consequently:

● On a network, only a single copy of the System application is needed

- On a single-floppy based computer, only a single master copy of the System application is needed. The user may have to insert this when starting an application, but should not subsequently have to.

If your application requires a more recent version of a shared Acorn resource than most users are likely to have, you must distribute it within the current version of System, together with the Merge application and instructions on their use. You can distribute RISC OS 2 resources without a licence. If your application uses some system-wide resources, these, too, may now be placed within System and be distributed, with Merge. Before you can add any system-wide resources to those distributed for RISC OS 3, you must contact Acorn for approval.

The Scrap application

The Scrap application is used as a location to store temporary files. You may freely distribute this if your application needs to store temporary files. (Its !Boot file sets a system variable named Wimp$ScrapDir.) You should encourage users of single-floppy based computers to have a copy of the Scrap application on every floppy disc, and to double-click on it when first viewing a new disc.

An application may create its own directory to hold temporary files. The directory must be called <WimpSScrapDir>.*Appl*. Your application must create this only when it is needed, and not on start up.

Providing your own shared resources

You must not place your own shared resources in the System or Scrap application but instead provide your own shared resource application directory. You must also register its name with Acorn, to avoid unnecessary duplication. If you plan to distribute a system-wide resource, this may be eligible for inclusion in System; contact Acorn for advice if you are in any doubt.

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.

On the whole, though, it is better to aim to make an application compact and precise in its functionality. Always bear in mind that some users only have a 1MB machine and may want to run your application alongside others. Indeed, they may

have to run it alongside some (such as a printer driver). If your application can't be used with a 1MB machine, state this clearly, preferably on the packaging so that users don't buy it if they can't use it. You will be able to sell to more users if you can keep within the requirements of a 1MB machine, of course. Even users with more than 1MB will prefer to use the extra memory for multi-tasking than running a single memory-hungry application.

Appendix A: Significant changes

The major changes since the last issue of this Guide are as follows.

● The desktop has changed to offer a 3D option, with new versions of the standard icons and window components.

● Cut and paste has become the standard method of cutting, copying and moving selections.

● Drag and drop is introduced as a new method for cutting, copying and moving selections, and will eventually exist alongside cut and paste.

● Outline menu structures are suggested to help you structure the menu tree within an application.

● A new font selection mechanism has been introduced to make it easier for users to make changes such as applying bold or italic to text in more than one font.

● The save dialogue box now has a selection button so that users may save a selection if there is one.

● RGB colour selection has changed from fractional 255ths to using percentages as a means of colour definition.

● The introduction of the pinboard in RISC OS 3 means that windows can be iconised; a sprite ic_sprite must be supplied for applications to use to represent iconised windows.

● !Sprites must be accompanied by !Sprites22 to support high resolution colour screen modes; !Sprites23 can be included to support high resolution monochrome screen modes.

● There are changes in the way some keys work within dialogue boxes; more changes will be necessary if PC keyboards are used in the future with some systems.

18   Glossary

action button

A 'button' in a *dialogue box* on which the user can click in order to cause some event to occur.

ADFS

Advanced Disc Filing System – the standard RISC OS disc-based filing system.

Adjust

The righthand button of the mouse.

Adjust size icon

An *icon* at the bottom right corner of a *window*, which the user can drag to adjust the size of the window.

Adjuster arrow

An *icon* used in a *dialogue box* to increase or decrease an associated value, often shown in an adjacent *writable field*.

application

A set of programs and accompanying resources having a specific purpose, and represented by a single *icon*.

application directory

A directory holding the programs and resources that form an *application*.

Applications Suite

A set of *applications* supplied with every RISC OS-based computer.

ARM

The name of the processor used to run RISC OS. It is now developed by Advanced RISC Machines Limited.

Back icon

An *icon* at the top left corner of a *window*, which the user can click to send the window to the back of the *desktop*.

caret

A single red I-shaped bar which shows where input from the keyboard will appear.

CLI

The 'Command Line Interpreter', which gives users control of the computer using a traditional command line.

Close icon

An *icon* at the top left of a *window*, which the user can click to close the window.

default action

The action taken if a user presses the Return key when a *dialogue box* is displayed. This should do what the user originally intended, in as 'safe' a way as possible.

desktop

The GUI supplied as a part of RISC OS.

dialogue box

A *window* used for a dialogue with an *application* or the *desktop*.

directory display

A *window* showing the contents of a directory.

document

A data file that an *editor* can load, edit, save and print.

editor

An *application* that presents files of a particular format as abstract objects which a user can load, edit, save and print.

editor window

A window used to display a *document* that is being edited.

environment string

A string used to store environment settings: these might typically be start-up options for an *application*.

error box

A special type of *dialogue box* that gives information to the user, and requires acknowledgement that it's been read.

Filer

The part of RISC OS that provides facilities for the user to control filing systems from within the *desktop*.

filetype

A value associated with every file, that specifies the type of data that it contains.

gaining the caret

The time when a window first has the *input focus*, and hence contains the *caret*.

GUI

A 'Graphical User Interface' such as the RISC OS *desktop*.

hourglass

A *sprite* displayed to show that an *application* running under the *desktop* has temporarily taken over the computer to the exclusion of other applications.

icon

A small graphic object (usually a *sprite*) used symbolically by the *desktop*. Amongst the things an icon might typically represent are: an option or action within a *dialogue box*, a file, an *application*, or a physical device.

icon bar

The bar at the bottom of the screen used by the *desktop* to hold *icons*. These usually represent *applications* or physical devices.

icon bar menu

A *menu* produced as a result of the user clicking Menu over an *icon* on the *icon bar*.

115

input focus

What the *window* containing the *caret* is said to have, shown by changing the border colour of the window

kernel

The main part of RISC OS.

leafname

The last part of a *pathname*.

Menu

The middle button of the mouse.

menu

A set of options from which the user can choose, typically having a tree structure.

menu item

One available option or choice on a menu.

modified flag

A flag used by an *editor* to record, for each *document* currently being edited, whether it has been modified.

multi-document editor

An *editor* that can edit several *documents* of the same type concurrently. The opposite is a *single-document editor*.

multi-tasking

The ability to run multiple tasks or *applications* at the same time. RISC OS is a multi-tasking operating system.

NetFS

Network Filing System – a RISC OS filing system that uses Acorn's proprietary Econet network.

Obey file

A file of commands for execution by RISC OS.

option button

A 'button' representing a switch, that can either be on or off.

OS graphic unit

A unit used for defining graphics under RISC OS, so that they are independent of the current *screen mode*. There are nominally 180 OS graphic units (or just 'OS units') to the inch.

palette

A file or data that maps between the colours that are to be displayed on the screen and the much larger number of potential colours.

pane

A *dialogue box* that is attached to a particular *window*.

parent

The precursor of an object: so for a file its parent is the directory that holds it, and for a *window* its parent is the window from which it was opened.

pathname

A complete specification of where a file is stored, including the filing system, all *parent* directories, and the file's own name (or *leafname*).

persistent dialogue box

A *dialogue box* that appears when the user chooses a menu item followed by an ellipsis. It remains on screen when the parent menu has been closed, and may suspend its *parent application* until it is filled in.

pointer

An *icon* on the *desktop* the movement of which is linked to the mouse.

pop-up menu

A *menu* within a *dialogue box* that normally just shows the currently selected option, but that the user can make 'pop up' to choose an alternative option.

printer driver

A RISC OS *application* used to print *documents*: several are supplied as part of the *Applications Suite*.

radio button

One of a group of 'buttons', only one of which may be selected at once.

RISC

Reduced Instruction Set Computer: a design philosophy used in the ARM which implements only the most frequently used processor instructions, and concentrates on making them execute at great speed.

RISC OS

Acorn's operating system and GUI, supplied in ROM on all its current range of computers (except for the Master series). It is pronounced as 'RISC-OH-ESS'.

RISC_OSLib

A library supplied with Acorn's ANSI C compiler, designed to help program *applications* to run under the *desktop*.

screen mode

A number that defines the appearance of the display: its resolution, and the number of available colours.

scroll arrow

An *icon* on the right-hand side of a *window* and/or the bottom, used to scroll the contents of the window by a small amount.

scroll bar

An area on the right-hand side of a *window* and/or the bottom, used to scroll the contents of the window, by approximately the height/width of the window.

scrollable list

A *window* within a *dialogue box* that shows a set of available options, and has *icons* with which the user can scroll through the options before choosing one.

SCSIFS

SCSI Filing System – a RISC OS filing system that uses a SCSI (Small Computer Systems Interface) expansion card to communicate with external peripherals.

Select

The lefthand button of the mouse.

select box

A rectangular box used to outline an area within which any objects will be selected.

selection

A portion of a *document* selected by a user, and on which operations may be performed.

single-document editor

An *editor* that can edit only one *document* at a time. The opposite is a *multi-document editor*.

slider

A bar on the right-hand side of a *window* and/or the bottom, used to scroll the contents of the window.

sprite

A graphic object that is pixel-based (ie one that is defined as a bit-map).

sprite pool

An area of memory used and maintained by RISC OS for storing *sprites*.

style

Indicates a stylistic variation in the letters of a font (for example Italic, Oblique or Shadow). See also *typeface* and *weight*.

submenu

A *menu* reached from another menu (its *parent*).

Task Manager

An *application* that is a standard part of the RISC OS *desktop*, with which the user can control and monitor *applications* and the use of the computer's memory.

template editor

An Acorn *application* used to interactively design and create *windows* and *dialogue boxes* for use within an *application*.

Title bar

A bar across the top of a *window*, used to display its title and (sometimes) *status words*.

Toggle size icon

An *icon* at the top right corner of a *window*, which the user can click to toggle the size of the window between a 'standard' size and a 'maximum' size.

toolbox

Window or pane of tool icons from which a user may select a tool to use in an application. A toolbox may be free-standing or attached to another window.

transient dialogue box

A *dialogue box* that appears as a *submenu*, and functions in the same way, disappearing when the parent menu is closed.

transparency mask

An optional part of a *sprite* that defines which pixels of that sprite are transparent.

typeface

A name used for all similar looking fonts (eg Homerton). This may also include a component specifying a variation on the standard font (eg HomNarrow). See also *style* and *weight*.

validation string

A string associated with a *writable field* that specifies what characters may be legally typed.

weight

Indicates the density of the letters of a font (for example Medium or Bold). See also *style* and *typeface*.

Wimp

The part of RISC OS that manages *windows* within the *desktop*, incidentally providing much of its functionality.

window

A rectangular area of the desktop devoted to a particular function, such as a *dialogue box, directory display, editor window* or *error box.*

Window Manager

The formal name for the *Wimp.*

writable field

A field in a dialogue box or displayed from a menu item within which the user can type text.

Index

T

U

V

W

Z

Reader's Comment Form

RISC OS 3 *Style Guide*

We would greatly appreciate your comments about this Manual, 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
Acorn House
Vision Park
Histon
Cambridge CB4 4AE

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

Acorn

AKJ21