Keystroke

A 'MACRO'-MAKING PROGRAM FOR THE ACORN RISC COMPUTER

This is version 4 (2014 April) of the Keystroke Manual, a revision of Manual 3.06 from 2003.

See page 62 for notes about this edition.

QUANTUM SOFTWARE

This is Manual version 4 (2014-April-08) — see page 62. It is based on Manual issue v3.06 (2003-November-14).

Document design: Geoff Stilwell
Keystroke: Alisdair Jørgensen & Stuart Halliday

Policy of Quantum Software

Keystroke is compatible with RISC OS 3.X/4.0/5.0; however, with the large number of software programs available for the RISC OS computer it has been impossible to test Keystroke with them all, therefore Quantum Software can not be held responsible for any work lost whilst using Keystroke, and can not be held responsible for other programs, especially Public Domain or Shareware type programs.

Any upgrades to Keystroke will be made available to users, but a small fee for materials, handling, postage & packing may be made.

Should the Keystroke disc fail to load due to a manufacturing fault of the original disc, then Quantum Software will supply a free replacement on receipt of the disc.

This manual should prove to be adequate in the use of Keystroke; if you still have difficulty then please contact us.

QUANTUM SOFTWARE
35 PINEWOOD PARK
LIVINGSTON
EH54 8NN
SCOTLAND
TEL: (+44) (0)1506 411162.
Hours 9.30pm till 5.30pm Monday - Friday
email : support@quantumsoft.co.uk
Web pages : http://www.quantumsoft.co.uk/

Quantumsoft closed in 2006 and released Keystroke and other applications to the public domain, hence the greyed-out text here. The software and source code is now downloadable from *quantumsoft.riscository.com* See page 62 for notes about this change.

Contents

Chapter 1

Introduction

WELCOME TO KEYSTROKE

Keystroke is a program which has the ability to make other desktop programs perform their functions in a repeatable sequence set up by yourself. You can make these sequences quickly and easily and store them for later use.

This 'macro-maker' type program is unique in the world of Acorn computers and offers you the chance to customise any number of functions into one key press. If you think of some of the repetitive tasks you have had to do in the past, Keystroke will wipe these away forever!

For example, imagine the following scene.

You have a pile of discs and you know a file you want to examine is on one of them.

What do you normally do? You insert the disc into the drive slot and move the mouse pointer to the floppy disc icon and click on the Select mouse button, if it is not the correct disc, again move the mouse pointer to the disc icon and press the Menu mouse button, followed by clicking on the 'Dismount' menu option. What a chore!

Not with Keystroke!

Insert your disc, press <Alt>+F1 and the disc is mounted. Press <Alt>+F2 and it is dismounted!

What has Keystroke done? It has turned a potential chore into a pleasure!

A simple example, but you get the idea. Keystroke improves productivity by saving time and allows you to get on with doing your work without the distraction of looking for those menus and icons to click on.

What are Keystroke's requirements?

Keystroke has been written for any RISC OS computer (running RISC OS 3.x/4/5.

It takes up a minimum of 100K of RAM when on the iconbar and will automatically take extra memory if required. Also running it from a hard drive is not essential.

What can it do?

As mentioned earlier Keystroke can be set up to perform various sequences of mouse Select, Adjust clicks, menu selections, inserting text, moving windows and the mouse pointer.

It can also run external library functions which greatly increase its power over other programs.

Keystroke allows these sequences to be repeated by one of three methods.

(1) By running a small file in a Filer directory which tells Keystroke which sequence to perform. A number of these files could be stuck on the Pinboard or in a provided 'Button bar' type application, so that by clicking on these files with the mouse the sequence will be performed. This allows the exciting possibility of creating your own tool box for any application!

(2) A time delay may be set so that the sequence can be performed every XX minutes. This, for example, allows Keystroke to add an autosave facility to an application which does not have one!

(3) A key combination which when pressed performs the sequence. Virtually any key combination on the keyboard may be set up. You can even include the mouse buttons!

As an alternative way of using Keystroke, it can also play back these sequences at a greatly reduced speed so that you can in fact make a demonstration of another application as a teaching aid!

What's on the disc?

You will find on the disc:

- A unique serial-numbered copy of Keystroke.
- A copy of ArcFS, the read-only compression application, which allows you to read the contents of the Keystrokes archive.
- A specially compressed archive called Keystrokes. This contains many examples of Keystroke sequences already defined. Supplied with the disc are files for Filer, Paint, Draw, DrawPlus, Vector, ChangeFSI, ArtWorks, Ovation and Impression. You can of course add or alter these presets yourself at any time.
- An Extras archive directory containing !Helper, !Buttonbar and !KeysLib.

  !Helper is a small utility which can help the user setup Keystroke by interrogating other programs. (See page 53)

  !Buttonbar is a useful application-launcher-type program which places a small horizontal or vertical window containing lots of icons just above the iconbar. You can drag programs or special action applications to it for instant use. (See page 54)

  !KeysLib is a library full of useful Basic programs which can help in altering text and getting information from other programs while using Keystroke. (See page 55)

- **It is strongly recommended that you copy the Keystroke disc for strictly backup purposes at this stage.**

Installing the software

Insert the Keystroke disc and mount the disc by clicking on the disc drive icon.

You will see a program called !ArcFS, this is an application which allows us to compress the preset files in an archive called Keystrokes. (The files in this archived directory take up approximately 1200Kbytes of disc space when uncompressed. This is too large to fit on a single 800K floppy disc which is the only type manyAcorn owners can read).

Double click on the archive directory called Keystrokes, the !ArcFS application will load and the contents of this archive will appear on screen after a moment.

Select all the files within this directory and copy them onto your hard drive (if you have one) into a directory of your choice. If you do not have a hard drive then copy the files onto floppy discs. Please note the complete contents of this archive will not fit onto one 800Kbyte disc.

Once the archived files have been copied you may quit the ArcFS application from the iconbar, you will not require it again, unless you need to look at the contents of the archive once more.

Create a directory called Keystroke on your disc system and copy the application !Keystroke, !Helper and !KeysLib to this directory. As well as FreeIcons and the text files.

If you have a RiscPC or A7000 machine we strongly recommend that you copy !KeysLib into your !Boot.Resources directory so that it is always available to Keystroke.

Remove the Keystroke disc and store it in a safe place.

Running Keystroke

Double click on the !Keystroke icon on your hard drive or floppy. The Keystroke icon will appear on the right hand side of the iconbar.

It looks like this on the iconbar. 

- Keystroke makes use of the !System application, so !System needs to be ‘seen’ by the Filer before Keystroke will load. If Keystroke can’t find it when it tries to load, a message will pop up and inform you of this. This is not a problem as most programs used on Acorn computers also need !System to be ‘seen’ and you’ve very probably got !System set up correctly anyway!

  But if you don’t have the !System application please refer to your User Guide manual on how to set up the !System application.

  If you own a RiscPC machine you will find that !System is already set up correctly within your !Boot application.

Hard-drive use

Keystroke may be used on any hard-drive system without any system restrictions. But please note the copyright conditions at the front of the manual.

Chapter 2

The Manual

Conventions

This manual is designed to help make Keystroke easy to use with a tutorial feel to it. We have adopted the following conventions.

We have assumed in this manual that you are familiar with the standard Acorn conventions of 'dragging an icon', 'clicking on a icon', etc.

Any text in the Corpus font like this '`an example line of text`' is intended to be typed in exactly as shown.

Control keys such as <Ctrl>, <Shift>, <Alt>, or <Tab> etc. will be shown in angle brackets as here.

Any text starting with the ' ➤ ' character must be carried out by you in the course of the tutorial sections.

Any text indented with a character like this '♦' is additional information for more experienced users of Acorn computers.

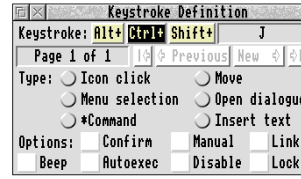
Any text beginning with the '✪' character is **incredibly important** and must be read with great care and attention.

First Steps

Once you've loaded Keystroke onto the iconbar you can bring up its main window which defines what Keystroke does.

➤ Click with the Select mouse button on the !Keystroke icon on the iconbar.

A Keystroke window should appear looking like this.



This **Definition** window as we call it can look confusing at first, but hopefully things will become clearer as you read on.

Here is a brief description of what each option of this window does.

The Title Bar

The title bar at the top of the Keystroke window normally contains the name of the action you want it to do. This name is useful in identifying what each individual action does at a later date and you can alter it to suit your own use. By default it is named 'Keystroke Definition' if no action has been set up, as you can see at the moment. When you start altering the settings within this window it will become '<Untitled>'.

When the mouse pointer is within the confines of the Keystroke window the Title Bar will turn a cream colour. This indicates that Keystroke has 'Input Focus', any keyboard key presses will be intercepted by Keystroke and no other application.

Keypresses

You may choose an appropriate keypress in the following ways:

- Hold any combination of the three control keys, <Alt>,<Ctrl> and <Shift>, and press one other key on the keyboard (not Scroll Lock, Caps Lock or Num Lock)
- Click Select or Adjust to select or deselect any of the cream coloured control key icons at the top of the window

This part of the window shows you which key combination has been chosen to activate one or more actions that you set up. It is shown here as set up for <Ctrl>+A. For historic reasons Keystroke is primary designed around allowing you to choose a key combination for causing an action to start.

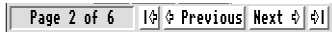
- Mouse clicks can be chosen also in combination with the control keys by clicking one of the mouse buttons while pointing at the area where the keyboard characters appear, to the right of the Shift+ icon.

You will notice that it is impossible to deselect all of the three control key icons. This is to prevent 'silly' Keystrokes being created. The keys not available for use by themselves are: Scroll Lock, Caps Lock, Num Lock, Break, Escape, A to Z, Tab, and Return, but F1 to F12, Insert, Home, Page Up, Page Down, Print and the four cursor keys can be programmed alone.

- This is not the only way in which you can get Keystroke to start an action. You can use a mini-application to do this also, but you still need to select a key combination first. See Chapter 8 for details of **Action** files.

Paging

This small collection of four icons allows you to cycle through the series of actions you may have set up under this particular key combination. It is rather like turning the pages of a book that can only show you one page at a time. These allow the user to add further Keystrokes which can be grouped or linked together.



When using these buttons to move backwards and forwards through a group of defined keystrokes the **New** icon will change to read **Next** if there is more ahead of the displayed keystroke.

Keystroke types

These six radio buttons allow you to choose one of the six actions Keystroke will carry out. Only one of these buttons may be selected at a time.

Icon click

This allows the simulation of a mouse click on virtually any icon in any program using the desktop, including the iconbar. It can be used to mount a hard or floppy disc by simulating a click on the disc icon on the iconbar.

Move

This allows the position, size and scrolling of a window to be controlled. The pointer position can also be set.

Menu selection

This allows a menu selection to be made from any application. It can be used, for example, to select all objects in a filer window or dismount a disc in the floppy drive.

Open dialogue

This allows a dialogue box or menu option to be kept open on screen before performing operations on it. An example of a dialogue box is the Fill Colour box in Draw. **Open Dialogue** is very similar to **Menu selection**.

*Command

This allows a line of text to be executed as a *command, as though it had been typed after pressing F12 to access the command line. It can be used to start applications running or open filer windows.

Insert text

This allows a line of text to be entered into a text editor or word processor or virtually any writable window or icon. It can also simulate the pressing of keys such as <Tab> or the cursor keys. This may be used to pick up text or a value from the caret position and replace it with a new value. Specialised characters or words can be entered with a simple Keystroke.

These Keystroke types will be explained in much more detail later.

Options

These seven buttons allow optional functions to be included with the Keystroke. A tick shows that this option is functioning.

- **Confirm**: When this option is on, then just before the sequence is performed a window will pop up and display the text from the keystroke name with the question mark ('?') at the end. If the 'OK' icon is clicked upon, then the process will continue. If 'Cancel' is selected then the grouped sequence will stop.
- **Manual**: When this option is on it forces Keystroke to perform a 'manual-like' operation on the application you are trying to control. This allows non–Risc OS compliant programs, which may refuse to operate with Keystroke correctly, to work! The side effect of using this option is that the mouse pointer will move as Keystroke executes its preset sequence.

✪ If a keystroke is not working initially it is worth trying it again with the **Manual** option switched on.

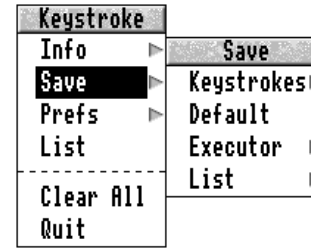
- **Link**: This option allows the displayed keystroke to be 'linked' to the previously defined keystroke sequence. This allows multiple linked keystrokes to be activated as one. If a single keystroke cannot be performed then the next linked keystroke will also not be executed and the sequence will stop. (see Linking Keystrokes, Chapter 6, for details)
- **Beep**: When this option is on after the sequence is performed, a standard beep will sound.
- **Autoexec**: When highlighted the displayed sequence will be automatically performed every XX minutes (XXm) or seconds (XXs). The time period is set from the Prefs › Autoexec option in Keystroke's main menu on the iconbar (or by the use of the 'KeystrokeAuto' command: see page 47 for more details).
- **Disable**: This is a test function of Keystroke. When highlighted the individual displayed keystroke is disabled.
- **Lock**: When this icon is on, the action displayed can not be changed accidentally, but it can be deleted. Normally left off, it allows the user to try out complex sequences with ease.

Chapter 3

Menus

Iconbar menu

A menu click over the Keystroke icon on the iconbar will reveal a standard menu with the following options:



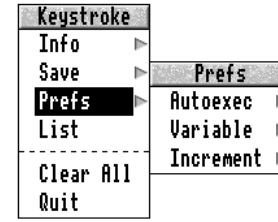
- **Info** Will give you the standard information display, including the serial and version number. Please quote these two items in any correspondence you may have.
- **Save** Leads to a sub menu where each of the save options are listed as follows:

  Keystrokes This leads to a standard Save dialogue box. You may enter a name and drag the icon to a Filer window of your choice or enter a complete pathname and click on the OK icon to save it. Your defined keystrokes are saved as a 'Keydef' data file.

  Default Clicking on default saves your defined keystrokes into the Keystroke directory. These will be loaded into Keystroke by default the next time it is run.

  Executor Leads to a standard Save dialogue box. This will save a copy of Executor, containing all the currently defined keystrokes, to wherever you choose in your filing system. See Chapter 9 for more information on Executor.

  List This leads to a standard Save dialogue box from where a text icon may be dragged to a directory of your choice. The textfile lists all the currently defined keystrokes. Clicking Select on the List menu option will cause the textfile to be loaded directly into a text editor such as Edit.



- **Prefs** This menu allows the settings of the three Keystroke variables to be changed:

  Autoexec Sets the number of minutes or seconds between each auto keystroke action. This can be between 1 second to 99 minutes. The default is 15 minutes.

  Variable shows the current value of the Keystroke system variable called Keystroke%Var. An integer number from -999999 to 9999999 may be entered. See Chapter 8 for more details on this.

Increment shows the current value of the Keystroke system variable called 'Keystroke%Inc'. A integer number from -999999 to 9999999 may be entered. This allows Keystroke%Var to be incremented or decremented by that amount. See Chapter 8 for more details on these options.

- **List** This option will cause a standard window to be displayed containing a list of all the currently defined keystrokes. This window has a menu of its own which is explained on page 17.
- **Clear All** This option will clear all currently defined keystrokes from memory. A confirmation box will appear if you have not saved them first.
- **Quit** This is the standard RISC OS quit option. A confirmation box will appear if you have not saved any modified keystrokes first.

Edit menu

Keystroke's one other menu, called Edit, is available only from the Main Keystroke Definition Window.

- **Name:** leads to a writable field in which the name for the current keystroke, or linked group of keystrokes, may be typed. The name may be up to 32 characters long.

  The text in this title is also used when you perform a List function to help identify the keystroke.

  If the Confirm option is on then it uses this text to display a message with a question mark '?' at the end. (see page 14 for details.)

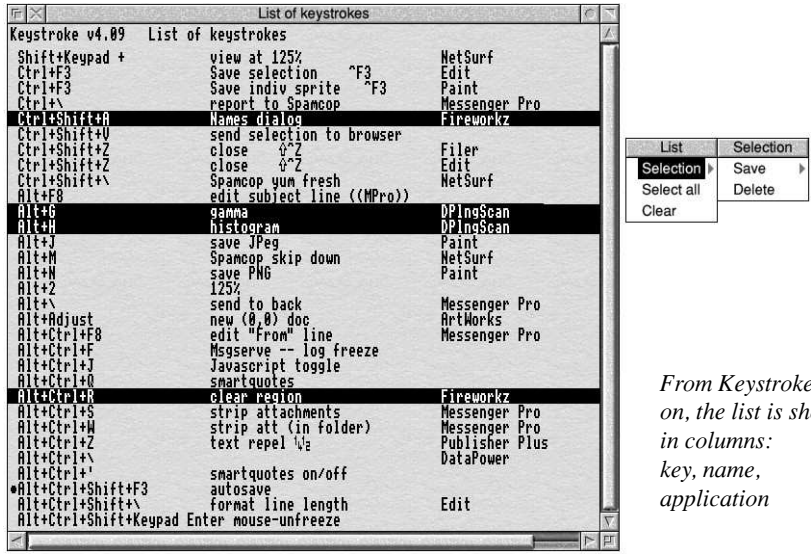
  If the Input variable – Keystroke$Input – is used in a *Command or Insert text, then the text from the keystroke name is used to display a message. (See pages 13–14 and 30 for more details)
- **Copy One:** will copy the currently displayed keystroke onto an internal clipboard leaving the original intact.
- **Copy Group:** allows the user to copy a group, from the currently displayed to the last linked keystroke, onto the internal clipboard leaving the originals intact. Up to ten linked keystrokes can be copied at any one time.
- **Cut One:** will cut the currently displayed keystroke onto the internal clipboard leaving the keystroke undefined.
- **Cut Group:** will cut a group, from the currently displayed to the last linked keystroke, onto the internal clipboard leaving the keystroke undefined. Up to ten linked keystrokes can be cut at any one time.
- **Paste:** will overwrite the currently displayed keystroke (if any) with those stored in the clipboard (the original is lost).
- **Insert:** will insert the keystroke(s) stored in the clipboard before the currently displayed one.

- **Action:** leads to a standard Save dialogue box, which will allow the user to save the currently displayed keystroke '**Action**' in an Obey file. This file will contain the keystroke Name typed in by the user on the Edit menu and the keys pressed, e.g. `Keystroke Ctrl+A@Select all files in window`

  Double clicking on this **Action** file in a filer window, or on the pinboard backdrop, will initiate the keystroke sequence from the first keystroke containing this exact title. (See Chapter 8 for more details).

The List window

The **List** window is available from Keystroke's iconbar menu.



From Keystroke 4.09 on, the list is shown in columns: key, name, application

The **List** menu option brings up a window that shows a complete list of currently defined keystrokes, in order of keys pressed. The text beside the keys is taken from the name given to the keystroke when it was defined.

Clicking on a line in the window with Select will highlight (or select) that line. Clicking with Adjust will deselect the same line or add other lines to the selection.

Double clicking on a keystroke line will display that keystroke definition in the Main Keystroke Definition Window.

The menus available from the **List** window pertain to a selection of keystrokes and allow that selection to be saved to disc (as a keydef data file) or deleted (when the selected keystrokes are removed from memory).

If a Keystroke Action has the 'Autoexec' option set, then the bullet character '•' will be shown at the far left of the Action line (the third-last line in the screenshot is an example). See Chapter 8 for more information on the Autoexec option.

Loading Keystrokes

When Keystoke is run a keydef file called 'Default', stored inside the !Keystroke directory, is automatically loaded.

Further keydef files may be loaded by dragging them from a filer window onto the Keystroke icon on the iconbar. These key definitions are then **merged** with those already loaded. If you wish to **replace** the key definitions with those you are loading you will first have to use the **Clear All** menu option on Keystroke's iconbar menu.

Keydef files may also be loaded using the Keystroke variable **Keystroke**. This is explained in detail in Chapter 8.

Saving Keystrokes

Although this has already been mentioned under the **Save** menu option it may be wise to elaborate a little.

If your keystroke definitions are those you want to be loaded in each time you work on your computer then choose the **Save › Default** menu option which will save the current key definitions as 'Default' inside the !Keystroke directory. These will then be reloaded automatically each time Keystroke is run.

You may also want to save a set of keystrokes separately, perhaps as a backup. In this case use the **Save** › **Keystrokes** menu option. This leads to a standard **Save** dialogue box where you may replace the default name 'Keydefs' with one of your choice and then drag the icon into an appropriate directory window.

Saving only a selection of your defined keystrokes is achieved from the **List** window mentioned above. Select the keystrokes you wish to save using the Select and Adjust mouse buttons (e.g., all those for a single application). Use the Selection › Save menu option from the **List** window. This also leads to a standard **Save** dialogue box where you may replace the default name 'Selection' with one of your choice and then drag the icon into an appropriate directory window. (See Chapter 8 for ways of automatically reloading these selected keystrokes using the Keystroke command **KeystrokeLoad**.)

Chapter 4

Setting up a keystroke

Choosing a key combination

As already said Keystroke requires a key combination to be selected before you can get it to do anything. So the first thing you decide is to tell Keystroke what key combination you want. You do this by moving the mouse pointer to within this Keystroke window and you should notice the title bar of the window turn a cream colour. This tells you that Keystroke will response to any key presses you may now perform on the keyboard.

➤ Move the pointer to within the window and experiment in pressing different keys, for example <Alt> F1, <Shift> <Print>, etc. You will see the various key combinations you press appear in the top section of the window.

✪ As long as the mouse pointer is within this top section of the Keystroke window, keyboard keypresses will be intercepted by Keystroke.

✪ **(Please do NOT press the <Break> key or <Ctrl>+<Shift>+F12 as this will shut your computer down!)**

You may notice that certain key combinations will not change the settings shown in Keystroke. This is due to Keystroke preventing you from choosing any silly combinations which may hinder you from using your computer. Imagine if we allowed Keystroke to start any action on the pressing of a single alphabet or number key!

The keys that it does allow are any of the function keys (F1 to F12) used on their own or with a combination of the <Alt>,<Ctrl> or <Shift> keys.

The alphabetic, numeric (including the keypad) and cursor keys with one or more of the <Alt>, <Ctrl> or <Shift> keys. Not forgetting the group of six keys above the cursor keys as well.

With RISC OS 3.1 and above it is possible to use the <Alt> & <Shift> keys in combination with several keys on the keyboard to access other characters. These keys are: `` `1234569ryop[]\asdf;'zxcm,./ `` In Appendix A is a picture of the keyboard showing the keys affected and the characters that can be accessed.

It is well to remember that many applications have their own set of shortcut keypresses. The application's actions are carried out as well as the action applied to them by Keystroke. These may occasionally conflict so choose your keypresses with care. Avoid using <Ctrl>+X as this is often used to delete or cut items.

Several keys are not used - the Caps Lock, Scroll Lock and Num Lock are not recognised by Keystroke as 'keys'. The Break key is not used for obvious reasons!

The 'Esc' or Escape key is also not used as Keystroke uses this particular key to terminate whatever actions it may have been set up to perform. It will be quite handy to remember that.

As well as allowing keys, Keystroke also allows the <Alt>,<Ctrl> or <Shift> keys to be used in conjunction with the three mouse buttons as well!

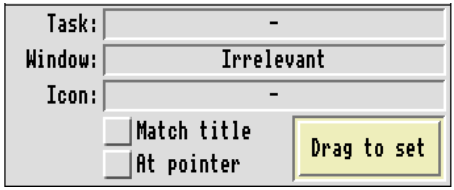
➤ Move the mouse pointer over the 'letter' area (to the right of the 'Shift+' icon) and press one of the mouse buttons. You should see the name of the mouse button appear.

♦ Additionally you can also click with the Select or Adjust mouse buttons on the Alt+, Ctrl+, Shift+ icons to highlight them instead of pressing these keys on the keyboard.

When you have chosen a keystroke combination you need to select one of the radio buttons to choose an appropriate Keystroke action. When a button is selected the Keystroke window expands in different ways depending on the Keystroke type chosen.

We can actually sub divide the six into two different groups. The *Command and Insert text have a similar kind of window and the other four; Icon click, Move, Menu selection and Open dialogue; have another common element in their windows. This common section will be described first.

The 'Task' section



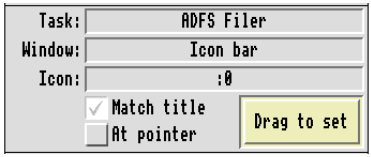
The middle section, shared by these four Keystroke windows, I will term the Task section as it allows the user to specify the task, window and icon that needs to be actioned or referred to.

To apply a keystroke to an application, or any window or icon within that application, Keystroke needs to register that task, window, and icon. This can be done in two ways:

➤ Drag the cream '**Drag to set**' icon into the application window or onto the specific icon you are concerned with.

➤ Point the mouse pointer at the icon or in the window and press both <Shift> keys together. This is most often done where using the mouse will lose the application's window or dialogue box.

✪ Some other applications (notably screen-grab utilities) also use "press both Shifts" as their trigger. You will have to quit them temporarily if you need this key combination while programming a keystroke.



Either of these two methods will cause the '**Task:**', '**Window:**' and '**Icon:**' icons in the Task section to update with the correct names. Keystroke is intuitive in that it will make a calculated guess at what to record here depending on where the '**Drag to set**' icon was dragged or where the mouse pointer was. For instance, if the pointer is pointing into an Edit window the Task name is recorded but the window is 'Irrelevant'. The Icon is most often shown as 'Background'.

If the **Match title** option icon is selected then the **Window:** icon will show the title of the window (that shown on the title bar of the window in question). This option is useful if the keystroke should only apply to the window with that exact title, such as application dialogue boxes. It should, however, be switched off if the window title is likely to change.

✪ Some window titles are constantly changing. Most applications add a * to the title when the contents of that window have been modified. Some have a scaling value in them, e.g. 'Manual at 130%'. These changed titles will cause Keystroke to fail to recognise these windows if the **Match title** option is set.

If the **At pointer** option icon is selected then the keystroke will only apply to the window and/or icon under the mouse pointer. This helps in some instances to make the user's intentions more specific. This is of little use when double clicking to run Keystroke's Action files as the mouse pointer will invariably be inappropriately placed.

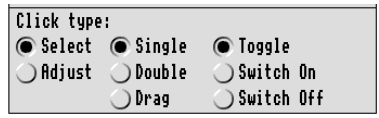
In the next chapter we will go through the individual Keystroke actions one by one using example keystrokes to illustrate each one.

Blank Page

Chapter 5

Keystroke operations

Icon Click



When the **Icon click** button is selected the Keystroke window is expanded. The lowest section gives three groups of radio icons, each group arranged vertically, only one of each group may be selected.

Select or **Adjust**, determine which mouse button will be simulated.

Single, Double or **Drag**, determine the mouse operation. **Single** and **Double** refer to clicks. **Drag** is more complex. It allows you to perform a simulated 'drag' operation from standard Save dialogue boxes to another application window. (See example on page 60.)

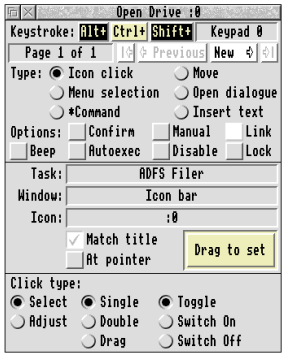
The third group determines the effect on the icon in question:

- **Toggle** Keystroke will always perform an icon click on the icon in question. So if it is already selected, or 'ticked', it will become unselected, effectively toggling the icon on or off.
- **Switch On** Keystroke will only perform an icon click on the icon in question if it is not already selected, effectively forcing it on.
- **Switch Off** Keystroke will only perform an icon click on the icon in question if it is already selected, effectively forcing it off.

♦ Please note that the **Manual** option has no effect for **Double** click or **Drag** operations. (For more about Drag operations, see addendum on page 60.)

We will now define your first Keystroke together using a simulated Icon Click

Example: Mounting a floppy disc in drive :0



First we choose a key combination, say <Alt><Shift>+Keypad 0.

➤ Hold down the <Alt> and <Shift> keys and press the Keypad 0. So the Keystroke window looks like the example shown here.

The first action we are going to get Keystroke to do is one that is very simple to set up and has practical benefits.

We will make Keystroke mount a disc that may be in floppy drive :0. This is the sort of thing we do all the time and it can be a pain at times to have to move the mouse pointer to the :0 drive icon on the iconbar just to investigate the contents of a disc.

To understand how you decide to tell Keystroke what you want it to do, it is important to realise that, in most cases, Keystroke does exactly the same as you would do!

To mount a disc in drive :0, you would perform an icon click on the :0 icon. Keystroke will have to be told to do the same.

You've noticed the six radio button icons just below the keystroke settings, these are the main options that tell Keystroke what you want it to do. One of these options is called Icon click, and this is just what we want Keystroke to do.

➤ Please click on the **Icon click** option so that it is selected.

When you do this the Keystroke window will expand to show more options relating to performing an Icon click as shown above.

The middle section of this expanded window is where you tell Keystroke which icon to 'click' on and the lower section tells Keystroke what type of Icon click you want to do.

When you mount a disc, you normally perform a single Select mouse click on the **:0** icon. You'll notice that the bottom section of the Keystroke window has the Select and Single option already highlighted. (Ignore the Toggle option for now). This means that Keystroke will do a single Select type mouse click on an icon. But you've not told Keystroke on which icon to do this click yet!

To tell Keystroke which icon you want to 'click' on you have to drag the large, cream **'Drag to set'** icon in the Keystroke window to the icon in question.

➤ Drag the **Drag to set** icon to the **:0** icon on the iconbar.

Once you have done this you will notice that the middle section of the Keystroke window has updated to include details of the selected icon.

The **Task:** icon should now reads **ADFS Filer**, this means that Keystroke knows which program or task to perform on. (If for some reason the name in this icon or the Icon name is not correct, then you have dragged the **Drag to set** icon to the wrong icon, please try again).

The **Window** icon tells you which window has been selected to receive Keystroke's attentions. In this case it reads **iconbar**.

Lastly the **Icon:** name icon tells you which icon is being targeted. In this case **:0**

These icons mentioned are purely to tell you what icon you have 'targeted' and so at a glance you can see that Keystroke now knows to do a single, Select type Icon click on the **:0** icon on the iconbar window!

When Keystroke knows all these various details and is initiated it looks at all the windows that may be on the desktop and tries to find a matching window, then it tries to find a matching icon within that window, if it finds one it fools the icon in question into believing that the user has actually clicked on the icon with the mouse, whereas in fact the pointer is nowhere near the **:0** icon! This is the way in which Keystroke operates.

Finally you should give this Keystroke Action a name.

➤ Click the mouse menu button over the Keystroke window. The first item on the resulting menu is '**Name**'. Move the mouse pointer to the right following the little arrow. In the title window that opens type your name for this action; say "`Mount Drive :0`". Click Select over the window or press <Return> and the title will appear on the title bar of the Keystroke window.

Keystroke is now fully set to mount the disc:

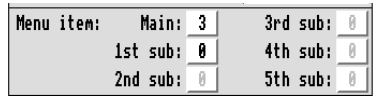
➤ Place a disc in the :0 drive slot and press <Alt>+<Shift>+Keypad 0. You should see the disc contents displayed on screen.

♦ If you're feeling brave you could try setting this action up onto a different key combination, perhaps <Alt>+<Tab>? Or <Alt>+` (key above Tab on keyboard)

♦ If you want you could try dragging the **Drag to set** icon onto other icons on the iconbar to see them working with your key combination.

Menu selection

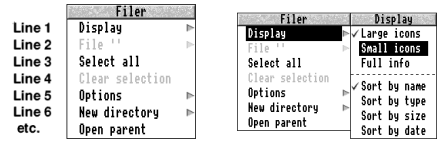
When this window is expanded the lower section looks like this.



The five white icons here are labelled **Main**, **1st sub**, **2nd sub**, etc. and refer to the menus and submenus available within an application. The numbers on the white icons are incremented with the Select button and decremented with Adjust.

To understand how Keystroke simulates a menu selection we need to look at a standard menu:

All menus contain a number of text lines. Keystroke gives these lines numbers as shown. Dotted lines included in some menus are cosmetic and are ignored by Keystroke.



To instruct Keystroke to perform 'Select all', click the **Main** icon until it shows 3.

To perform a 'Display › Small icons' menu selection, click the **Main** icon until it shows `1` and the **1st Sub** icon until it shows `2`.

A number `0` indicates that no more sub menus are set so it is important to leave the last icon set to zero.

Example: Dismounting a floppy disc

The second action we will set up will be to dismount a floppy disc from drive :0. Normally for you to do this, you have to perform a menu selection on the **:0** icon on the iconbar and choose the second line down of the menu.

First select a suitable key combination, say <Alt>+<Shift>+Keypad . (the decimal point next to 0 on the keypad).

➤ Place the mouse pointer within the upper part of the Keystroke definition window so that its title bar turns a cream colour. Hold the <Alt> and <Shift> keys and press the keypad . (decimal point).

We now need to get Keystroke to perform a Menu selection on the :0 icon on the iconbar.

➤ Click on the radio icon **Menu selection** so that it becomes selected.

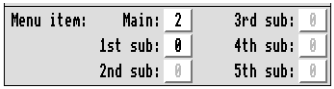
Keystroke now needs to know which task, window or icon the menu in question is to come from. In this case it is the :0 icon on the iconbar.

➤ Drag the **Drag to set** icon from the Keystroke window and drop it onto the **:0** icon.

Now Keystroke needs to know which menu item to choose.

Dismount is on the second line down of the floppy disc menu. (see filer menu above).

➤ Set the number in the **Main**: box by clicking with the mouse until the number '**2**' shows.



♦ A number zero in these boxes tells Keystroke not click on any line of a menu.

You have now set Keystroke to dismount a floppy disc from drive :0!

If you now press <Alt>+<Shift>+Keypad. you can dismount a floppy disc very quickly.

♦ If you want you could choose a different key combination to do this action. Perhaps <Alt>+= or <Alt>+<Tab>.

Open Dialogue

This Keystroke action is very similar to the **Menu selection** action. The expanded window looks exactly the same with **Main:** and **sub** menu icons as before. The only difference between this and the **Menu selection** action is whether the end product, either a writable menu icon or a dialogue box, needs to remain on screen.

Example: Pop up Draw's 'Fill colour' dialogue box

Let's say we want to define a keypress which will open the Fill dialogue box in Draw. Let's choose <Ctrl>+<Shift>+F as our keys (you may choose your own if you prefer).

➤ First make sure that the Draw application is loaded into memory and its Icon is showing on the iconbar

➤ Point at the top half of the Keystroke window with the mouse pointer. Hold down <Ctrl> and <Shift> together and press F. These keys will be registered at the top of the window.

➤ Click on the **Open dialogue** button just below. The rest of the Keystroke window will open looking identical to the **Menu selection** window.

➤ Drag the **Drag to set** icon into the Draw window (makw sure the Draw window has the input focus first). The middle section of the Keystroke window will record the window's details.

To achieve our object we must be in a drawing mode or have a selected object in the Draw window. To open the Fill dialogue box we need to choose the third item from the main menu (Style) and the third item on the sub menu (Fill colour).

➤ Click with Select until the number in the **Main:** icon reads **3**. Do the same with the number next to **1st sub:**.

Give your new keystroke a name, say Open Fill Dialogue.

➤ Click Menu over the Keystroke window. Slide across the arrow next to the first menu item. Type '`Open Fill Dialogue`' into the writable icon and press <Return.>

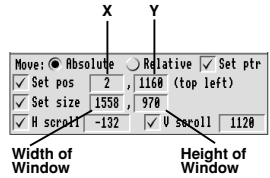
The <Ctrl>+<Shift>+F Keystroke will now cause the Fill dialogue box to open allowing you to choose the fill colour for an object.

♦ Use a different keypress to open the Line colour dialogue box in Draw.

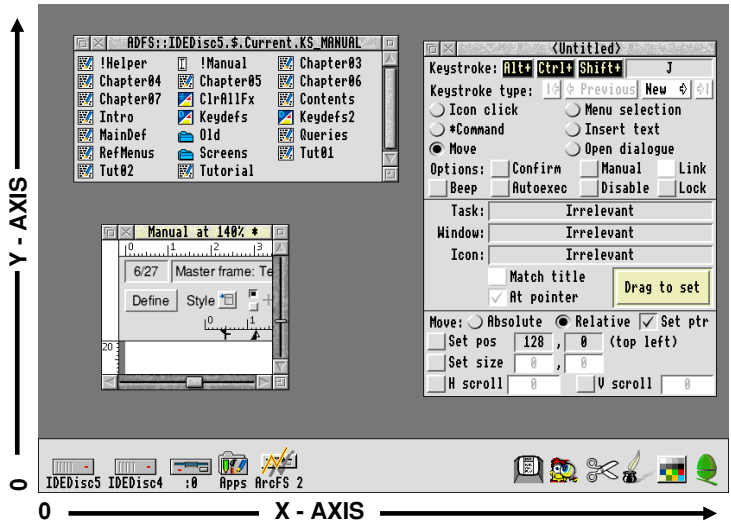
Move

This operation allows any movable window to be placed and scrolled anywhere on the desktop. It also allows the mouse pointer to be moved to a set position.

The lowest section of the Move extended window looks like this. The absolute **X** and **Y** coordinates refer to a position on the desktop (see below).



Absolute: This is the default setting of this action. When the **Drag to set** icon is dropped onto a window (or the two <Shift> keys pressed when the pointer is in position) Keystroke will automatically read in the absolute position of the window (relative to the bottom left of the desktop), its size and its scroll bars if the appropriate option is highlighted (Set pos, Set size, and H or V scroll options).



Relative: When the **Relative** button is selected a position or size can be set by incrementing or decrementing the values in the **X** and **Y** coordinates icons using the Select and Adjust mouse buttons respectively. This will move the window or pointer relative to its current position.

✪ The values in the white **X** and **Y** coordinates icons may be incremented by clicking with Select or decremented with Adjust. The values increment or decrement by 2 for each click. Holding the <Shift> key at the same time accelerates the change to a jump of 64 for each click.

Set ptr: This option, if selected, allows the mouse pointer to be moved by an absolute or relative amount as determined by the X and Y coordinates of the **Set pos** option

When ONLY the **Set ptr** option is used the items in the **Task** section of the Keystroke window all become Irrelevant as the pointer is moved relative to the desktop not the task or window.

Set pos: This option, if selected, allows the user to set the position of the window. The window may be assigned to any absolute position on the desktop. The numbers in the white coordinates boxes correspond to the X & Y position of the top left of the window in question.

If the **Relative** button is selected instead of **Absolute** then the window's position must be controlled by modifying the amounts in the **X** & **Y** coordinates icons. Increment with Select, decrement with Adjust.

Set size: This option, if selected, allows the user to change the width and height of a window.

If the **Relative** button is selected instead of **Absolute** then the window's width and height will be controlled by modifying the amounts in the **X** & **Y** coordinates icons.

H/V Scroll: This option, if selected, allows the horizontal and vertical scroll bars of a window (if it has them) to be set.

Example: Size an Edit window

Let's say that we now want to set up an Edit window to take up the top half of the screen. We'll use the keypress <Alt>+<Ctrl>+E.

- ➤ Make sure that the Edit application is loaded and its Icon is showing on the iconbar.
- ➤ With the pointer in the Keystroke window hold <Alt> and <Ctrl> and press E. The icons at the top will reflect your choice.
- ➤ Click the **Move** radio button. A new bottom section to the Keystroke window will appear.
- ➤ Open an Edit window and place it in the position and at the size you want it.
- ➤ Click the **Set pos** and **Set size** icons in this bottom section and then drag the **Drag to set** icon into the Edit window. The display icons will update to show the top left position of the Edit window and its size.
- ➤ Give the Keystroke a name (e.g. '`Edit standard size`') in the usual way.

Pressing <Alt>+<Ctrl>+E now over an Edit window will cause the window to adopt the same size and position as that you just set.

- ♦ You might like to link this keystroke with a sequence that runs Edit and brings up a window on screen.

Example: Using the Relative Move

For the purposes of this example we will move the pointer 128 screen units to the right. We'll use the keypress <Alt><Ctrl><Shift>+J

➤ Move the pointer into the Keystroke window. Hold <Alt><Ctrl><Shift> and press J

➤ Click on the **Move** button

➤ Click on the **Relative** button and select the **Set ptr** option.

➤ Hold the <Shift> key and click twice on the first **X** coordinate icon (to the right of **Set pos**). The number 128 should show. (If not use Select to increment and Adjust to decrement, with and without the <Shift> key to make this icon show 128.)

➤ Give the keystroke a name (e.g., '`Jump pointer`') in the usual way.

This keystroke is now set. There is no need to use **Drag to set** as the task is irrelevant. To see the keystroke working place the pointer on the left side of the screen, hold <Alt><Ctrl><Shift> and press J. You will see the pointer jump to the right.

*Command

Clicking on this radio button gives a window very different from those we've seen so far. The middle **'Task'** section is not necessary for this operation and is not included. Tasks, windows and icons are not important as as all the text typed into this window is sent to the command line. Many of the elements of the ***Command** window are also shared by the **Insert text** window and have similar effects although some are more appropriate to one or the other.

The bottom section of the ***Command** and **Text Insert** window has a writable **Text**: icon and a number of buttons below which have the effect of entering various words or variables into the **Text:** icon.

Text: Click inside the icon to place the caret and type any single normal *command for executing when you initiate the keystroke, e.g. `*CAT`. Finish by pressing <Return>

✪ Pressing <Return> is very important when inserting text, because Keystroke cannot store the text otherwise.

System variables:

Keystroke allows any system variable to be entered into the text icon. A system variable is denoted by a '<' at the start and '>' at the end of the variable (e.g. <Sys$Date>). These will be translated into their current values when the sequence is activated.

This allows the use of some useful variables like Sys$Time, Sys$Date, Sys$Year, Filer$Dir, Caret$Text, Pointer$Text, Keystroke%Var, etc.

(See the text for the **Time**, **Date**, **Year** and the **Filer window** icons for more details.)

Keystroke supplies the variables mentioned above as icons so all you have to do is click on them and their names will be entered into the text icon.

♦ A system variable is set before anything else on the test line is performed.

Shift keys:

With the caret in the text icon and the mouse pointer over any filer window, pressing both <Shift> keys will cause the window title to be entered in the text icon. This enables long filer pathnames to be entered at a stroke. If the pointer is over an icon then that icon's name or its contents will be entered into the text icon.

✪ Some other applications (notably screen-grab utilities) also use "press both Shifts" as their trigger. You will have to quit them temporarily if you need this key combination while programming a keystroke.

Time: When clicked will insert <Sys$Time> into the text icon. When activated this will translate to the current time, e.g. `17:07:16`.

Date: When clicked will insert <Sys$Date> into the text icon. When activated this will translate to the current date, e.g. `Sat,05 Aug`

Year: When clicked will insert <Sys$Year> into the text icon. When activated this will translate to the current year, e.g. `1995`.

Pointer Text: When this icon is clicked it will insert the Keystroke variable, <Pointer$Text> into the text icon. When activated it allows any icon or window title under the mouse pointer to be read and used in the *command. If Keystroke cannot read the text of an icon then the word 'unknown' will be used.

Caret Text: When this icon is clicked it will insert the Keystroke variable, <Caret$Text> into the text icon. When activated it allows any text present at the caret of a writable icon to be stored in this variable.

Cursor keys: (The ***Command** window has the cursor-control keys (the arrow keys, left, right, down, up) greyed out as these are not appropriate here. They will be explained in more detail in the **Insert text** section.)

Run: If this icon is clicked the word '`Run`' is added to the text in the **Text** icon. If, however, a filer object, such as an application's icon, is dragged to this icon the full pathname of the object will be entered. This allows the quick running of applications using a single keypress.

♦ If you have Keystroke running in RISC OS 3 it will use the command '`Filer_Run`'. this will allow the loading of files into their respective applications without reloading the application.
If you try to run a Basic program using the Filer_Run command under RISC OS 3 then it does not work reliably. In this case Keystroke will use the '`Run`' command automatically.

Open dir: If this icon is clicked the word '`Filer_OpenDir`' is added to the text in the Text icon. If, however, a filer object, such as a filer directory, is dragged to this icon the full pathname of the object will be entered. This allows directories to be opened with a single keypress.

If a file is dragged to this icon then its parent pathname is entered instead. Also if an ArcFS type archive is dragged, then the command will be changed to '`Run`', opening it. (The use of '`Filer_CloseDir`' closes windows).

Library: If this icon is clicked it opens Keystroke's internal Library directory. This simplifies the dragging of the programs in the Library to the Keystroke window for inclusion in keystroke definitions. These programs and their uses are described in more detail in Chapter 9.

Input: When this icon is clicked it will insert the Keystroke variable, <Keystroke$Input>, into the text icon. When activated a small writable text icon will pop up allowing the user to enter text or numbers. When the user presses <Return> the Keystroke sequence will continue and the text or values will be subsequently used. The default title of this input text box is 'Enter text'. It will also take the title from the Name given to the keystroke using the Edit › Name menu option.

Variable: When this icon is clicked it will insert the Keystroke variable, <Keystroke%Var> into the text icon. When activated the value of this variable will be used in the command. The variable is incremented automatically by the Increment variable <Keystroke%Inc> which is available from the **Prefs** option in Keystroke's iconbar menu.

Filer window: When this icon is clicked it will insert the Keystroke variable, <Filer$Dir> into the text icon. When activated it will translate to the full pathname of any filer window the mouse pointer is over. If the pointer is not over a filer window Keystroke will do nothing.

By using the *command: '`Dir <Filer$Dir>`' the window under the pointer would become the currently selected directory (CSD).

Example: Running an application

The most efficient way to Run an application is using Keystroke's ***Command** option. This enables you to run the application without having to find it hidden deep in your directory structure.

Let's say you want to run Edit. This is to be found in the Apps directory on the iconbar of your computer.

First choose a keypress. It is wise to keep them as memorable as possible, say <Alt>+E.

- ➤ Place the mouse pointer within the upper part of the Keystroke definition window so that its title bar turns a cream colour. Hold the <Alt> key and press E.
- ➤ Click on the ***Command** radio button
- ➤ Now open up the Apps directory where Edit is located
- ➤ Drag the Edit application icon over the yellow **Run** icon near the bottom of the Keystroke window. You will notice that the details of Edit's filepath (where it is on your computer) has been entered into the writable area labelled **Text:**.
- ➤ Give this Keystroke Action a name, e.g. '`Run Edit`' in the usual way.
- ♦ Of course Edit is quite easy to find on the desktop. You may have one of your own more commonly used programs buried deep within several layers of sub-directories. Using the same procedure as for Edit you may load any application using whatever keypresses you choose.

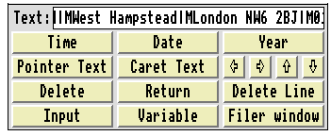
- You may also open directories in the same way. Follow the same procedure as above except just drag the directory you wish to open to the yellow Open Dir icon in the bottom half of the Keystroke window.

Text Insert

This option is a sophisticated text insertion and manipulation tool. At its simplest level it can be used to enter text at the cursor position in most word processors or text editors.

As already mentioned, the **Insert text** window shares many elements with the ***Command** window. The differences will be elaborated here.

As can be seen there are only a few changes to the window (shown here). The icons that are the same as those in the ***Command** window have the same function except the resulting values of the variables are inserted at the caret position.



Text: The maximum length of text you may enter into the **Text** icon is 115 characters. You may also drag a Text or Obey file to the **Text** icon and the first 115 characters of that file will be used.

Time:
Date:
Year:
} These three icons work exactly the same as already described except that they will insert their respective values at the caret position in a document or in a writable icon in a dialogue box.

Pointer Text: When this icon is clicked it will insert the Keystroke variable, <Pointer$Text> into the **Text** icon. When activated it allows any icon or window title under the mouse pointer to be read and inserted at the caret position. If Keystroke cannot read the text of an icon then the word 'unknown' will be used.

This is an invaluable way of reading and inserting window titles or filer pathnames into documents or writable icons.

Caret Text: When this icon is clicked it will insert the Keystroke variable, <Caret$Text> into the **Text** icon. When activated it allows any text present at the caret of a writable icon to be stored in this variable.

Caret$Text is unusual as it will immediately re-enter this information back into the writable icon. It was designed with this specifically in mind as it will allow extra text to be entered, before or after the original text found in the icon, without the use of another keystroke. It does however require the use of the **Delete line** command (`|U`) first (see below) if you do not want to repeat the text in the icon.

An example of the use of **Caret Text** is explained in Chapter 7 'Using Keystroke Variables'.

⇦ ⇨ ⇧ ⇩ Clicking on any of these arrow icons will insert '\⇩\' or '\⇨\' etc.in the **Text** icon. When the keystroke is activated these simulate the pressing of the cursor keys. If the arrow icons are clicked more than once Keystroke enters \⇧⇧\ for you. This Keystroke function is very useful for moving around a text document or between writable icons in a dialogue box, if the application supports this.

- If you wish to enter any of the ⇦ ⇨ ⇧ ⇩ characters into the Text icon to be used as they are, then you must delete any '\' (backslash) characters from around them manually.

Delete: This icon is provided as an aid. When it is clicked it will enter '|?' which represents the standard **Delete** control command. It has the effect of deleting the first character to the left of the character when the keystroke is activated.

Return: This icon is provided as an aid. When it is clicked it will enter '|M' which represents the standard **Return** control command and simulates the pressing of the <Return> key. It has the effect of introducing a newline or carriage return in a document or for activating a dialogue box **OK** or **Go** action if the application supports this.

Delete Line: This icon is provided as an aid. When it is clicked it will enter '|U' which represents the standard **Delete Line** control command. It has the effect of deleting the line of text in a writable icon. Useful for erasing the text or value at the caret before you replace it with a modified value using **Caret$Text**.

Input: This has exactly the same function as for ***Command**. It will give the user the opportunity to input text or a value for the keystroke to use.

Variable: When this icon is clicked it will insert the Keystroke variable, <Keystroke%Var> into the text icon. When activated the value of this variable will be used in the keystroke. The variable is incremented automatically by the **Increment** variable <Keystroke%Inc> which is available from the **Prefs** option in Keystroke's iconbar menu. (See Chapter 8 for more details).

Filer window: When this icon is clicked it will insert the Keystroke variable, <Filer$Dir> into the text icon. When activated it will translate to the full pathname of any filer window the mouse pointer is over. If the pointer is not over a filer window Keystroke will do nothing.

Control characters and keys:

- Control codes may be entered into the **Text** icon by entering the '|' character followed by a single letter (the '|' character represents the <Ctrl> key).
- Characters which are not available from the keyboard may be entered by enclosing their ASCII code in angle brackets, e.g. <160>. They can also be entered by holding down the <Alt> key and tapping the ASCII code on the numeric keypad. When <Alt> is released the character will appear.

\

- Because the '<', '>' and '|' characters are used for defining other control characters, if you want to include only one of these characters in the text string you must either prefix them with the '|' (for example, '|<', '|>', '||') or use their ASCII codes in angle brackets, e.g. '<'=<60>, '>'= <62> or '|'= <124>. If you want to use the '\' character by itself in a **Insert text** command you must double it up like '\\' or Keystroke will interpret it as a start of a command and think you've forgotten to close it and issue a non serious error message informing you of this.
- Leading spaces are automatically deleted by the Operating System, so to place one or more leading spaces in the box you must enclose the entire line with double quotes, e.g. `" hello there".` To use a single double quote character use two together like this `""` or type in <34> instead.
- The ability to enter control codes and keys is very useful as it allows Keystroke to make use of most of the control key shortcuts the target application may already use.
- Keystroke can also emulate the pressing of certain main keys of the keyboard, such as <Escape>, <Tab>, <Copy>, <Insert>, <Print>, <Page Up> or <Page Down>, as well as the Function keys by themselves or in combination with the <Ctrl> and <Shift> keys.
- Here is how they must be entered into the **Text** icon:

  `\ESCAPE\` or `||`, `\COPY\`, `\DELETE\` or `\DEL\`, `\INSERT\`, `\PAGE UP\`, `\PAGE DOWN\`, `\UP\` or `\⇧\`, `\DOWN\`, or `\⇩\`, `\LEFT\` or `\⇦\`, `\RIGHT\` or `\⇨\`, `\F1\` to `\F12\` and `\PRINT\`
- The keys above also allow the use of the <Ctrl> and/or <Shift> keys. This is achieved by placing either '`S-`', '`C-`' or '`CS-`', before the key to be used: e.g. `\S-PRINT\`, `\C-COPY\`, `\C-⇧\` or `\CS-F1\`.

✪ Remember, if you need to enter <Ctrl>+letter commands (<Ctrl>+A, etc.), use the '`|A`' form instead. Sorry, but it is impossible for Keystroke to issue a <Ctrl><Shift> +letter type command.

✪ Remember also to press <Return> when you have finished entering text in the **Text** icon. Keystroke will not be able to store the line until you do.

Example: Insert address at caret

Let's start with simply entering our name and address as we might at the top of a letter. We'll use <Alt>+F11 as our keypress this time.

➤ Point at the Keystroke window. Hold <Alt> and press F11. The keys will be registered at the top of the window.

➤ Click on the **Insert text** button. The window will expand as usual but this time it has no middle section as the receiving application is irrelevant, the text will just be inserted at the caret position.

➤ Type in your address. Click on the **Return** icon when you want to insert a new line in the text. The '`|M`' characters will be inserted which translate to a newline in the word processor etc. (e.g. '`Quantum Software|M35 Pinewood Park|M Livingston|M`' etc.)

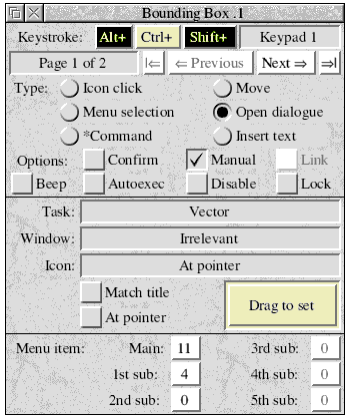
➤ Give the new keystroke a name (e.g., '`Insert address`') in the usual way.

Placing the caret into any text editor, such as Edit, or word processor, such as Impression Style and pressing <Alt>+F11 will insert the address at the caret position, e.g.:

Quantum Software
35 Pinewood Park
Livingston
EH54 8NN

♦ Try inserting other examples of text into a text editor.

The Manual button



The **Manual** button in Keystroke's main Definition Window has been mentioned briefly but it deserves a more elaborate explanation.

Many applications do not fully comply with Acorn's Risc OS Guidelines and they consequently may not behave in the way Keystroke expects. Computer Concepts' applications, Impression Style, Publisher, Publisher Plus, Artworks, etc, are prime examples of this tendency.

By switching on the **Manual** option button Keystroke attemps to perform the operation manually. that is, as if you had performed the action yourself.This has the side effect of causing the mouse pointer to move across the screen as it actually forces the popping up of the Menus, etc.

A general rule would be to try the Keystroke operation but if it didn't work then try it again with **Manual** switched on.

Switching on **Manual** has no effect on the ***Command** and **Insert Text** Keystroke types.

Another use for the **Manual** option is to set up demos or tutorials. This is explained in greater detail under **KeystrokeDemo** in Chapter 8.

Chapter 6

Linking keystrokes

Up to now we have only dealt with single-page keystrokes but the real power of Keystroke is when several of these are grouped or linked together to produce a complete sequence of actions.

The **Link** Option button is used to group together two or more pages of keystrokes. These may then be treated as a group and is recognised and can be saved as such from the **List** window. When the key combination is pressed then the keystroke is activated and each Keystroke page is performed in sequence. If for any reason a single page's action cannot be carried out (e.g. if the application it refers to is not loaded) then the keystroke will end there and the following pages will not be activated.

Keystrokes may have several pages that are not linked together. If the **Link** option is not selected then the keystroke will not be interrupted if one page cannot perform its action. This is mainly referring to a keystroke activated using a keypress combination. Keystrokes activated using the saved **Action** files or the **Keystroke** variable are explained in more detail in Chapter 8.

Example: Running several applications including !Printers and a template document

Many people have a task that they perform at the computer more regularly than others. It may be they have a regular correspondence session or developing their Club's newsletter. Whatever it is, there is a need to load up a number of applications with perhaps a template document ready to drag in the textfiles from a directory and with the printer driver loaded.

This sequence uses four keystroke pages linked together.

1st Page:

- With the pointer in the Keystroke window hold down <Alt> and press N. The icons will reflect the keys chosen.
- Click on the ***Command** button.
- Drag the Impression Style icon (or your preferred DTP or word processing application) from its filer window onto the yellow **Run** icon below. The pathname will be entered into the **Text:** icon.
- Give the Keystroke a name (e.g. '`Newsletter startup`') in the usual way.
- Click on the **New** icon near the top of the window. A new Keystroke window will open.

2nd Page:

- Click on the ***Command** button again.
- Now drag your !Printers application icon from its filer window onto the **Run** icon. Again the pathname will be entered.
- Now click on the **Link** Options button in the centre of the Keystroke window. A tick will appear. This will link this page with the 1st.
- Click on **New** once again. Another blank Keystroke window will open.

3rd Page:

- Click on the ***Command** button again.
- Drag the directory (e.g. Newsletter) from its filer window to the yellow **Open dir** icon. The pathname will be entered.
- Click on the **Link** Options button again. This page will be linked with the others.
- Click on **New** one more time. Another window will open.

4th Page:

- Click on the ***Command** button again.
- Drag your template document (e.g. Newsform) from its filer window to the **Run** icon. The command '`Filer_Run`' and the full pathname will be entered.
- Click on the **Link** Options button again. This page will be linked with the others.

We have now linked four Keystroke pages together which will all be initiated by the <Alt>+N keypress. Do that now:

- Hold <Alt> and press N.

Impression Style will load onto the iconbar followed by the printer driver. The Newsletter directory will open and the NewsForm template file will be loaded into Impression ready for you to start work on your Newsletter.

More complex multiple keystrokes

Example: Removing newlines from an Edit textfile

When transferring text from a text editor, such as Edit, into a word processor, such as Impression Style, the text should be continuous. If you typed the text in Edit with the Wordwrap option switched on then each line of text ends with a newline character (ASCII 10) which leaves the lines the same length when imported into the word processor.

One way to overcome this is to use the Find and replace function in Edit before the text is imported. This method does amount to rather a lot of keypresses, menu selections and icon clicks not to mention typing in the Find and replace strings into the dialogue box.

Keystroke can automate this whole process with one keypress although there are six keystrokes linked together.

As the standard key for bringing up a Find and replace dialogue box is F4 it seems reasonable to choose <Alt>+F4 as the keys to initiate this keystroke.

Before you define this keystroke you will need to have some wordwrapped text in an Edit window on screen.

1st Page:

- With the pointer in the Keystroke window hold down <Alt> and press F4. The icons will reflect the keys chosen.
- Click on the **Open dialogue** button.
- Drag the **Drag to set** icon into the Edit window. Make sure that match title is not selected as you want this to work with any Edit document.
- Increment the **Main:** menu item icon to **4** using the Select mouse button. Increment the **1st Sub:** icon to 1. This will select the first item from the Edit sub menu which will cause Edit's Find dialogue box to appear when <Alt>+F4 is pressed.
- Name the Keystroke (e.g., '`Remove newlines from text`') using Keystroke's Edit › Name menu option.
- Click on the **New** icon. Another window will open.

2nd Page:

- Click on the **Icon click** button.
- Click in the Edit window to ensure that it has Input Focus (Edit is paying attention) and press F4. This will bring up the Find dialogue box.

The intention here is to simulate a click on the Wildcarded expressions icon. If we try to drag the **Drag to set** icon onto the dialogue box it will disappear. Keystroke has another method of registering windows and icons using both <Shift> keys.

- Move the mouse pointer so that it is pointing at the Wildcarded expressions icon. Now press both <Shift> keys together. You will notice that the **Task** the **Window** and **Icon** section of the Keystroke window will be updated with the correct names (ie Edit, Find text, Wildcarded). The **Match title** icon will also be automatically selected which is correct in this case.

The final thing to do in this window is to make sure that Wildcarded expressions is not switched off if it had already been turned on previously. The default **Toggle** option will cause this to happen but the **Switch on** option will have the desired effect.

- Click on the **Switch on** radio button in the **Click type** section of the Keystroke window.
- Click on the **Link** icon.
- Click on the **New** icon opening a new Keystroke window.

3rd Page:

- Click on the **Insert text** radio button.

This is where we will insert the Find and Replace strings into the two fields in the dialogue box.

One method of removing newline characters from the ends of lines of text but leaving behind the double newlines or paragraph breaks is as follows:

Find: ~$$~$

Replace: ?0 ?1

Which means: Find a character which is not a newline followed by a newline then another character which is not a newline. Replace this with the first character found (not a newline) then a space and then the last character found (also not a newline).

Now we have to instruct Keystroke to insert these strings into the dialogue box.

➤ Click in the writable **Text**: icon.

➤ First click on the yellow **Delete Line** icon below which will enter '|U' into the **Text** icon. This will ensure that any previously entered text is deleted first.

➤ Now type ~$$~$

➤ Click on the **down arrow** icon below which inserts \↓\ This will move the caret into Edit's Replace field when this keystroke is implemented.

➤ Click on **Delete Line** again to clear the text in this field.

➤ Finally type '?0 ?1' and press <Return>.

✪ Pressing <Return> is very important when inserting text as Keystroke cannot store the text otherwise.

➤ Click the **Link** icon.

➤ Click the **New** icon.

4th Page:

Here we need to set the search going by clicking on the Go icon in Edit's Find text window.

➤ Click on the **Icon click** button in Keystrokes window.

➤ Hold <Alt> and press F4 to set the Keystroke in action. The three you have already completed and linked will be carried out in turn and you will be left with the Find text dialogue box on screen with the Find and Replace strings in place.

➤ Move the pointer over the GO button on the Find dialogue box and press both <Shift> keys together. The **Task**: **Window**: and **Icon**: fields in the Keystroke window will update accordingly and the **Match title** option is ticked. This time you can leave the **Click type** on **Toggle**.

➤ Click on the **Link** button.

➤ Click on the **New** icon.

5th Page:

The next thing that will happen when the keystroke is initiated is Edit's Text found dialogue box will pop up (if your search string is found). To automate the whole process we need to simulate a click on the End of file icon.

➤ Click on the **Icon click** radio button in Keystrokes window.

- ➤ Click at the start of your text in the Edit window and initiate the keystroke by pressing <Alt>+F4. You will be left with the Text found box on screen.
- ➤ Move the pointer over the End of file button and press both <Shift> keys together. The keystroke fields will update appropriately.
- ➤ Click on the **Link** button.
- ➤ Click on the **New** icon.

6th Page:

To finish off the whole keystroke neatly we need to simulate a click on the **Stop** button to get rid of the Text found box.

- ➤ Click on the **Icon click** radio button in Keystrokes window.
- ➤ Click at the start of your text in the Edit window and initiate the keystroke by pressing <Alt>+F4. You will be left with the Text found box on screen after the find and replace operation has been completed.
- ➤ Move the pointer over the **Stop** button and press both <Shift> keys together. The Keystroke fields will update appropriately.
- ➤ Click on the **Link** button.

Your keystroke is now complete. Place the caret at the start of any text where you want the newlines removed. Hold <Alt> and press F4. You will be able to see the process happening on the screen. After a few seconds (depending on the length of your text) control will be returned to you and you will have a textfile with continuous lines of text suitable for importing into another word processor.

- ♦ The keystroke name can be added at any time during the definition of a keystroke, it will even work without a name. The name, however, should be included in the first keystroke of a group. Click on the ⏮ button (to the left of the ◁ **Previous** button) to jump back to the first keystroke from where you can enter a name using Keystroke's Edit menu.
- ♦ Experiment with different Find strings so that you avoid joining lines beginning with a space or Tab character. Try and produce a similar operation with your favourite text editor (e.g. Desk Edit, Zap or StrongEd).
- ♦ If when you are trying out various things in Keystroke when **Insert text** is highlighted you may come across the error message "You must close commands with a backslash '\' ". This is because Keystroke uses pairs of backslash characters to signify special commands and so it expects to find even numbers of these in the **Text:** line. So if you are trying to insert, for example, '\n' into Edit use '\\n' instead as Keystroke will interpret this as a single backslash. See page 35 for more details.

Blank Page

Chapter 7

Using Keystroke variables

We have already mentioned some of the variables available in Keystroke in ***Command** and **Insert Text** (see Chapter 5). It is also possible to create your own variables which may be used alongside the system variables within Keystroke to pass values back and forth between applications.

The use and manipulation of variables in RISC OS is an enormous subject and not appropriate for inclusion in this little manual. More details of the use of variables in the command line can be found in RISC OS User Guides and elsewhere.

We will use examples to show how these variables can be used with Keystroke.

Pointer$Text

This example keystroke will be used to enter the filer pathname of a collection of programs on several floppy discs into a database field.

- Set up a single keystroke with your own choice of keypress (e.g. <Ctrl><Shift> with the right-arrow key)
- Click on the ***Command** icon.
- Click on the **Pointer Text** icon which will insert '`<Pointer$Text>`' into the **Text** icon.
- Press <Return> to store the text.
- With your database loaded and ready to accept the data, with the caret in the correct field, and the mouse pointer over the window containing the application, hold <Shift> and <Ctrl> and press the right-arrow key. This will read the window title at the pointer and place it in the database field icon effectively entering the pathname of the application for you.

Caret$Text

Example: Pluralising text in a database

Picking up text from the caret and being able to manipulate it is extremely useful.

A simple example would be to add the letter 's' to a word in a dialogue box field.

- ➤ Set up a single keystroke with your own choice of keypress (e.g., <Ctrl><Shift> and the letter S). The Keystroke icons will reflect your choice.
- ➤ Click on the **Insert text** icon. The window will expand.
- ➤ Click on the yellow **Delete Line** icon which enters '`|U`' into the **Text** icon.
- ➤ Click on the yellow **Caret Text** icon which enters '`<Caret$Text>`' into the **Text** icon.
- ➤ Now type the letter '`s`'. The **Text** icon will read: '`|U<Caret$Text>s`' (without the '').
- ➤ Press <Return> to store the text.
- ➤ Name the keystroke in the usual way (e.g., '`Pluralise words`').

For the purposes of this simple example the writable icons containing the caret will contain words like Dog, Cat, Horse, etc. When you activate the keystroke the text in the field will be read into Caret$Text first. Then the text will be deleted. Caret$Text will then be replaced in the field with the letter 's' appended. Dogs, Cats, Horses, etc. More practical uses of the **Caret$Text** variable are explained below and in the next Chapter.

EVAL

With the ability to take text from a writable icon by using **Caret$Text**, it makes sense to allow a way to manipulate the text before replacing it. Keystroke uses the **Insert text** command **EVAL.** This command reads text from any system string variable (e.g. **Caret$Text**) and extracts any numeric value it may have. In a mixed, alphanumeric, string Keystroke will extract the first numeric value it finds. Therefore '`size is 6.5lxyz45`' will be evaluated to `6.51`. When using the EVAL you must place the command and any operation between two backslash characters,
e.g. `\EVAL <Caret$Text>*0.175\`

- ✔ If you wish to use the **EVAL** command to extract a numerical value from a variable – whether it be **Caret$Text**, or any other system variable – without needing to use any mathematical operation on the value it is still necessary to do so, if only to add 0 (zero). For example, to extract and use the value from the text string in the example above: '`size is 6.5lxyz45`' use the Keystroke command: '`\EVAL <Caret$Text>+0\`'

Example: Increase a numerical value at the caret

This next example is a little more useful and also uses the **EVAL** command. The writable icon or field you are targeting will contain a number to which you want to add 64.

- ➤ Choose an appropriate keypress to define the keystroke
- ➤ Click on the **Insert text** icon.
- ➤ Using the yellow icons below or by typing it in by hand, enter the following into the **Text** icon: `|U\EVAL <Caret$Text>+64\.` Finish by pressing <Return>.

With the caret in a writable icon containing a number the keystroke when initiated will store the text at the caret in the <Caret$Text> variable and delete the text already there. Then it will add 64 to the numeric value found in the variable and replaces that value in the writable icon. So if the writable icon held '`100`' this keystroke will replace it with '`164`'.

Creating your own variables

Let's suppose that you want to pick up some text from the caret in a writable icon using **<Caret$Text>** and then use it in another application's dialogue box or transfer it to a document. Here we cannot use **<Caret$Text>** to place the text elsewhere because **<Caret$Text>** will be redefined in the process. We need a new variable.
Creating variables is easy using the RISC OS command ***Set**.
We actually need to define two separate keystrokes to achieve this the first to 'pick up' the text, or whatever, and store it in a variable we define, and the second is to place whatever is stored back wherever we want it.

1st keystroke

- Set up a single keystroke with your own choice of keypress (e.g., <Alt> and keypad - (minus symbol)). The Keystroke icons will reflect your choice.
- Click on the ***Command** radio button.
- Type into the **Text** icon '`Set TextVar <Caret$Text>`' This will store the contents of **<Caret$Text>** in the variable **TextVar**.
- Name your keystroke in the usual way (e.g., '`Pick up Icon text`').

2nd keystroke

- Set up another keystroke with another keypress (e.g., <Alt> and keypad + (plus symbol)). The Keystroke icons will reflect your choice.
- Click on the **Insert text** radio button.
- Type into the **Text** icon '`<TextVar>`'.
- Name your keystroke in the usual way (e.g., '`Place variable contents`').

This keystroke can now be used to place the contents of the variable <TextVar> at the caret in any other application. It may be used over and over again and will not change until you redefine <TextVar> by using the first keystroke again. Your variables may also be manipulated using **EVAL** in the same way as described above.

Hints and tips

An Obey file called ‘startup’ is held within the Keystroke and Executor directories. You simply add any Keystroke commands you wish to be run just after Keystroke or Executor starts up.

The most obvious example is to remove the icon from the iconbar at start up.
*KeystrokeIcon Off
Or if you want Keystroke to perform an action straight away.
In fact you can add any valid CLI command, not just Keystroke commands.

New feature in version 4.02 onwards

Version 4.02 of Keystroke introduces a new feature which a customer requested.

Because Keystroke works from RISC OS 3.11 to RISC OS 3.7+ it has been a problem that certain programs such as !Draw and !Paint (for example) have altered somewhat in the way they work across the different versions of RISC OS. To help users use Keystroke across these different operating systems we’ve slightly altered the way the Default file is loaded.

If you place within !Keystroke (or !Executor) a KeyDef file called Default370 and then run Keystroke on a RISC OS 3.7 machine, Default370 will be loaded instead of the ‘Default’ file.

Here is a list of each different version of RISC OS that Keystroke can handle:

RISC OS 3.11 needs a Default311 file.
RISC OS 3.50 needs a Default350 file.
RISC OS 3.60 needs a Default360 file.
RISC OS 3.7+ needs a Default370 file.

Of course if Keystroke can’t find its particular OS file then it simply loads the normal ‘Default’ file.

Note: This modification applies only to the *loading* of the Default file; the *saving* remains unaltered.

Chapter 8

More Keystroke variables

Keystroke makes use of eight other system variables which can be manipulated by the user to great effect. They can be set from within and sometimes outside of Keystroke. This is achieved using the *commands ***Set** and ***SetEval**.

Keystroke%Var <number>

This variable is normally used within Keystroke in a ***Command** or **Insert text** keystroke. It allows the user to enter an integer numeric value with text. **Keystroke%Var** is updated after each use by another variable **Keystroke%Inc** (see below). The range of numbers available is -9999999 to 999999. It can be set using '`SetEval Keystroke%Var 22`', from the command line or in an Obey file, or by using the Main iconbar menu **Keystroke › Prefs › Variable** option. The default is 1.

Keystroke%Inc <number>

This variable is also normally used within Keystroke in a ***Command** or **Insert text** keystroke. It will automatically update <Keystroke%Var> each time it is used by incrementing or decrementing <Keystroke%Var> by the amount set in <Keystroke%Inc>. It can be set using '`SetEval Keystroke%Inc 2`', from the command line or in an Obey file, or by using the Main **Keystroke › Prefs › Increment** menu. The default is 1.

KeystrokeAuto <number(s)>

This command allows the Autoexec number to be changed at any time to anything from 1 to 99 minutes, or by using the extra optional **s** parameter the value refers to seconds. (e.g. 240s is 240 seconds). It can be set using '`KeystrokeAuto 20`', from the command line or in an Obey file, or by using the iconbar menu: Prefs › Autoexec. The default is 15 (minutes). If it is set to 0 then it is automatically reset to 15. If a Keystroke Action has the 'Autoexec' option set then the bullet character '•' will be shown in the List window on the left of the Action line (see page 17).

✪ In previous versions of Keystroke (before version 4.00) this was known as a variable Keystroke%Auto. This has been changed along with some other variables (Keystroke$Icon, Keystroke$Demo, Keystroke$Do) to achieve a greater efficiency and speed. The previous variables have been retained for backward compatibility, but users are advised to use the new commands, because the others will be phased out in future versions of Keystroke.

Auto-saving

The variables and command described above may be used together to define a keystroke that will automatically save a document. This is very useful for applications that don't have a built-in Autosave facility.

There are different ways we could achieve this.

1. Subsequent saves overwrite the original – Most existing autosaving applications on Acorn machines already adopt this method, but any major changes or mistakes are not recoverable once the file has been saved.
2. Subsequent saves have a different name to the original – Simply adding an incremented suffix number to the name overcomes the disadvantages of the first method, although it will obviously take up more disc space.

We'll try the first method first.

Example: Auto-saving 1

As many applications use the F3 keypress to initiate a save we will use a keystroke that will apply to any application with Icon Focus, the one we're working on. We'll also use the same keypress <Alt><Shift><Ctrl>+F3.

Before we start the autosave process we should save our file at least once. From then on it will be handled automatically by Keystroke.

- ➤ Move the mouse pointer into the Keystroke window, hold <Alt>, <Shift> and <Ctrl> and press F3
- ➤ Click the **Insert text** button
- ➤ Select the **Autoexec** Options icon
- ➤ Type '`\F3\`' into the **Text** icon
- ➤ Click on the yellow **Return** icon or type '`|M`' and press <Return>. The **Text** icon will now read '`\F3\|M`'
- ➤ Give this keystroke a name (e.g., `Autosave`) in the usual way
- ➤ Now set the time limit for your autosave: from the iconbar, enter a value into the **Keystroke › Prefs › Autoexec** writable menu field, or leave it at 15 minutes. Click on the **Autoexec** line in the iconbar menu window if there is not a tick beside it. The tick means Autoexec is activated and counting down.

Now every fifteen minutes a save box will pop up for a while and automatically save your work.

Example: Auto-saving 2 – Incremented filenames

Because we are now going to add numbers to the end of a specific filename we need to adopt a different method to make our autosave specific to a Task, in this example we'll use Draw. We should still save our file once first.

1st Page

- ➤ Click the **Open dialogue** button

✪ Make sure the Draw window has the input focus before you do the following as Keystroke will remember that the window it is to operate on is a Draw window with or without the input focus depending on the state of the window at the time you dragged the Drag to set icon to it.

This is sometimes the root cause of commands appearing to fail as Keystroke has been set up to act upon a window which did not have the input focus at the time and at a later date you might be trying to get it to operate on a window which has the input focus and probably wondering why Keystroke appears to no longer work!

➤ Drag the **Drag to set** icon into the Draw window. The **Task** section of the window will update with: **Task:** Draw, **Window:** Topmost, and **Icon:** Background.

➤ Click on the **Menu item** icons so that they show the following: **Main:** 2, **1st Sub:** 1

➤ Click on the **New** icon to open a new Keystroke window.

2nd Page

➤ Click on the **Insert text** icon again.

➤ By clicking on the cream **Delete** icon twice or typing enter '`|?|?`'

➤ Next click on the cream **Variable** and **Return** icons to enter: '`<Keystroke%Var>|M`' into the **Text** icon. e.g. The finished line should read: '`|?|?<Keystroke%Var>|M`'. Press <Return> to finish.

➤ Click on the **Link** icon

➤ Make sure that the **Autoexec** main menu line is ticked and your preferred countdown time is set.

Now wait for the time to elapse and watch your work saved with an incremented suffix.

e.g. '`Drawfil1`', '`Drawfil2`'... '`Drawfil13`'... '`Drawfil99`', etc.

The two delete commands we used makes sure that there is room for 1 to 99 backup files to be used if the filename you've used happens to be exactly ten characters long.

✪ With older versions of RiscOS, remember with this method that your base filename should be eight characters or less to allow for a two-digit number to be placed at the end of the filename, making a maximum of ten characters — assuming that you will not want to save the same file more than 99 times. (Newer versions of RiscOS no longer have the 10-character limit on filenames.)

Keystroke <Keys@Action name> — Action files

This command allows the user to perform keystroke sequences without pressing a single key! When this command is used, for example as part of a !Boot file or an **Action** file saved from Keystroke's Edit menu, Keystroke will look for a corresponding exact match in its preloaded **Action** name keystroke definitions. When it finds a match it will attempt to activate the keystroke.

- *Keys* is the key combination used to define the keystroke, e.g. `Ctrl+A`. (This part of the command is optional but allows Keystroke to find the name faster).
- *Action name* is the name given to the keystroke when it was defined, e.g. '`Select all files in window`'.

Examples:
Keystroke Alt+Shift+Keypad 0@Open Drive :0
Keystroke Alt+F4@Remove newlines from text
Keystroke Alt+Z@Run Zap

Keystroke is the command used in the Obey **Action** file saved from the Keystroke Edit menu. Double click on this file to activate the corresponding keystroke. **Action** files are useful where a keypress would not normally work with an application.

Making a Mini-App

- If you drag a sprite (smaller than 100x50) with an internal name beginning with '!' and drop it over a Keystroke definition window with a name, the internal name of the sprite will be used to build a miniature application. A standard Save box will pop up allowing you to drag the application into an appropriate Filer directory window. This application consists of the application directory represented by the icon drawn from the sprite you dragged. Inside is the sprite saved as !sprites and the Keystroke **Action** file renamed as !Run. Double clicking on this application in a filer window or on the pinboard will activate the keystroke with the matching name. These mini applications are also an easy way to use the **!Buttonbar** application supplied with Keystroke (see Chapter 9).
- Where two separate keystroke actions have been defined using the same keypress combination, using the **Keystroke** command has a different effect to using the keypress. The **Keystroke** command, when activated, will look for the matching named keystroke despite the fact there may be two or more using the same keypress. The single keystroke or group of linked keystrokes with the matching name will be activated only. This means that separate keystrokes may be defined using the same keypress and activated separately using the **Keystroke** command in a !Boot file or a saved Action Obey file. If you want the Action to operate in the same way as if you had pressed the keys you must manually edit out the ***Name*** part of the Keystroke$Do command.

✪ **Keystroke** commands or **Action** files will not work without the corresponding Keystroke definitions loaded into Keystroke or Executor.

KeystrokeLoad <Filepath.name>

This command allows the user to load and merge additional sets of pre-defined key definition files. This means an application's own set of Keystroke key definitions may be loaded by inserting the KeystrokeLoad command in the application's !Run file.

e.g. '`KeystrokeLoad <Vector$Dir>.VectorKeys`' where the keydef file is stored in the Vector application directory (`<Obey$Dir>` could be used instead of `<Vector$Dir>`).

or '`KeystrokeLoad <KeystrokeKey$Dir>.VectorKeys`' where the keys for the different applications are all stored in a single directory with the pathname `<KeystrokeKey$Dir>` set up by Keystroke.

- If an application is run, closed and run again in a single session this will cause the Keystroke$Load line in the !Run file to load the set of keystrokes a second time. this will cause problems when you come to use them. the following example will avoid this.

  ```
  If "<Keystroke$Vector>" = "" Then KeystrokeLoad
  <Vector$Dir>.VectorKeys
  Set Keystroke$Vector <Vector$Dir>.VectorKeys
  ```

  These lines are part of Vector's !Run file. The first line checks for the variable <Keystroke$Vector> if this has not been defined the Keystroke key definition file 'VectorKeys' is loaded from Vector's directory. The second line then sets the Keystroke$Vector variable to ensure that the keystrokes are not loaded again.

- See page 47 on how to make Keystroke or Executor execute a KeystrokeLoad command on startup.

KeystrokeIcon *On/Off/Toggle*

This command allows the user to decide whether they want the Keystroke icon to appear on the iconbar. It is set to ***On*** by default. This variable has three parameters *On, Off* or *Toggle*. (These parameters are also acceptable entered as upper case: *ON*/*OFF*/*TOGGLE*, or lower case: *on*/*off*/*toggle*). *On* and *off* have obvious results, making the Keystroke icon visible on the iconbar, or not. the third parameter, *Toggle,* makes the icon visible if it is not already and vice versa.
The user would define a ***Command** type keystroke –
e.g. '`KeystrokeIcon Toggle`'.

KeystrokeDemo <*0–99*>

This command allows Keystroke sequences that use the **Manual** Option to be slowed down so that the pointer movement gradually moves across the desktop. **0** is the default 'full–speed' setting. Manually changing this to a greater value will slow down Keystroke. Recommended value 10.

This is a powerful and very useful option if users wish to create demonstrations of other software packages, or simply see Keystroke performing more slowly. An Obey file could be created consisting of lots of Keystroke-type lines which perform one after the other. We have provided a (small) demo of !Paint (look in the Keystrokes.Paint directory on your Keystroke disc).

Chapter 9

Additional programs

!Executor

Executor is a playback only version of Keystroke. It has no edit window and will only perform predefined keystrokes and **Action** files.

✪ Keydef files loaded into Executor using the **KeystrokeLoad** command or dragged to the Executor icon on the iconbar will **REPLACE** any previously loaded. they are NOT **merged** as in Keystroke.

To save a version of Executor containing your preferred key definitions use the iconbar Save › Executor menu option and drag the !Executor icon into a directory window. (!Executor actually lives inside the !Keystroke directory but **leave this alone**. Only use the menu save option described above).

Executor retains a short iconbar menu. This will enable the user to change the values of the three variables **KeystrokeAuto**, **Keystroke%Var** and **Keystroke%Inc**, and obtain a list of the loaded key definitions. There are no **Save** options from the iconbar menu or the **List** window.

Executor could be useful for providing demonstrations on a machine where you do not want the key definitions to be tampered with.

♦ If you wish to switch off !Executor's iconbar icon then manually edit Executor's !Run file and add the line 'KeystrokeIcon Off' at the second line from the end.

!Helper

Helper is a simple, free, application which can help the user define keystrokes by providing information about the Tasks, Windows and Icons under the pointer. Helper gives advice interactively about different aspects of the keystroke definition (e.g. the **Click Type** required for different icons or how far you have dragged a window when defining a **Relative Move**).

A !Help file is contained inside the !Helper directory which provides more detailed information on its uses. Interactive Help is also available with Helper by loading the !Help application from the Apps directory on your computer.

!ButtonBar

ButtonBar is a separate application which is designed to work alongside and with Keystroke. Buttonbar does not install itself on the iconbar. It starts as a scrollable window containing up to 100 blank buttons. Dragging a standard (34x34 or 34x17 sized) sprite in 16 or 256 colours to one of the boxes will display the sprite icon in the button. Dragging a Keystroke **Action** file will apply that action to that button. A single click on a button will activate the keystroke with the matching name as in the **Action** file.

Any settings you create with ButtonBar by dragging sprites and **Action** files to the buttons will be automatically saved when you close the ButtonBar window. If, however, you subsequently move any of the sprites or **Action** files ButtonBar will no longer be able to find them and will not work with those settings.

- **An effective way of using ButtonBar is to define a set of keystrokes for a specific application.**
- Collect or design an appropriate set of sprites to accompany the keystrokes.
- Give the sprites internal names beginning with a '!' (e.g., '`!sizepage`').
- Open the Apps directory found inside the ButtonBar application directory.
- Drag each sprite onto the Keystroke definition window corresponding to its accompanying keystroke and save the resulting mini application into the Apps directory window.
- Now drag each mini application onto a button on the ButtonBar.

ButtonBar will now work with the keystrokes applied to it acting on the specific application for which they were defined. As long as the sprites, action files and/or mini applications are saved in ButtonBar's Apps directory ButtonBar will still work if you move it to a different location on your computer assuming that you also have the corresponding key definitions loaded into Keystroke or Executor.

To remove an **Action** from a button click with Adjust while holding down <Shift>.

- ButtonBar may be configured to display any number of buttons between 2 and 100 vertically or horizontally on the screen. To change the number of buttons you must edit the value applied to the **BB%Icons** variable in ButtonBar's !Run file. The bar's orientation is configured by altering the value of the **BB$Orientate** variable also in the !Run file. Further instructions are in the !Run file itself.

!Impressive

!Impressive is a ready-made Button Bar and Executor application for the Impression series of word-processor/DTP applications. It allows the user to nudge frames, to apply or remove borders, to apply or remove background colours and many other useful functions all with the click of a button.

!Impressive is available separately: download it from *quantumsoft.riscository.com*

!Blinds

Blinds is another separate application originally supplied by Quantum Software and now downloadable from *quantumsoft.riscository.com*

Blinds is a professional pinboard-type program designed to make it easier for you to organise your files and applications. With Blinds you can quickly launch applications and files, organise all of your files into groups of windows for easier recall. Blinds and Keystroke are designed to work co-operatively with each other forming a partnership of utilities that you won't know how you did without.

!KeysLib

Keys Library is a separate application directory which holds a large number of Basic programs. Previous versions of Keystroke had the Library directory situated beside the !Keystroke application directory. These Basic programs interact with Keystroke or Executor and enable the user to achieve a number of useful operations.

A list of the programs is stored in the Library directory in a textfile called '`List`' and gives full directions for their use. I will use a few of the more useful library programs here as examples of their general use.

Example: Changing text to sentence case

This program will take the text in a writable icon and transform it so that the first letter is capitalised and the rest are lower case, e.g. `DRAWFILE` or `drawfile` will be changed to `Drawfile`.

Page 1:

- Make sure that the filing system of the computer has seen the !KeysLib application. Double-click on the application if necessary, this will open a directory window with all the Basic programs inside. The Library directory window can also be opened with a click on the yellow **Library** icon in the Keystroke definition window.
- Open up a Keystroke definition window and choose an appropriate keypress combination, say <Ctrl>+<Alt>+S
- Click on the ***Command** button.
- Find the Basic file '`ABC-Abcl`' in the Library directory window and drag it on to the **Run** icon. This inserts the command: '`Run <Keystroke$Lib>.ABC-Abcl`'. As you can see Keystroke has automatically inserted the variable `<Keystroke$Lib>` which sets the pathname for the Basic files.
- Type a space in the **Text** icon and click on the **Caret Text** icon which inserts the `<Caret$Text>` variable. Finally press <Return>.

The **Text** icon should read '`Run <Keystroke$Lib>.ABC-Abcl <Caret$Text>`' When this is performed, Caret$text will contain the contents of the writable icon containing the caret. This information is passed to the Basic program which in turn sets a system variable called KS$Line to the new text now converted to Sentence case.

Now we need to replace the text in the writable icon with the new text.

Page 2:

- Click on the **New** page icon.
- Click on the **Insert Text** button.
- Click on the **Delete Line** icon or type '|U' into the **Text** icon.
- Type '<KS$Line>' and press <Return>.
- Click on the **Link** option button.
- Name the keystroke in the usual way.

This keystroke will now take the text it finds in a writable icon from a menu or dialogue box and converts it into Sentence case. Two other Basic programs in the **Library** convert text to lower case or UPPER case. These are imaginatively titled 'lower' and 'UPPER' and work in exactly the same way as the example above.

Example: Bring a window to the front

Often the window we want is hidden behind several others on screen. We can see a small corner of the bottom left of the window but not the title bar so we can't bring it to the front. In this situation we would have to send all the other windows to the back until we could see the title bar of the one we wanted. It is possible to simulate an icon click on the title bar of a window with a normal keystroke but this will not work with some applications.

KeysLib has a program called **forceWfrnt** that will do the job. It forces the Window at the mouse Pointer to the front.

- Set up a ***Command** keystroke with an appropriate keypress combination.
- Drag the forceWfrnt file from the **Library** directory to the **Run** icon. This will enter: '<Keystroke$Lib>.forceWfrnt' into the **Text** icon. Now press <Return>.

Example: Centre a window

This is another simple example of a program that centres a window on the desktop.

- Set up a ***Command** keystroke with an appropriate keypress combination.
- Drag the **centrewin** file from the **Library** directory to the **Run** icon. This will enter: '<Keystroke$Lib>.centrewin' into the **Text** icon. Now press <Return>.

The program **centrewin** does have an optional parameter which can be added to alter the effect. the parameter can be a number from 0 to 4:

0. The window at the mouse pointer is centred. If it is not present no window is moved.

Note: `Run <Keystroke$Lib>.centrewin` is the same as
`Run <Keystroke$Lib>.centrewin 0`

1. The window at the mouse pointer is centred and forced to the top.
2. Only a window with the input focus will be centred. If it is not present NO window is moved.
3. Only a window with the input focus will be centred and forced to the top. If it is not present NO window is moved.
4. Only a window with the input focus will be centred and forced to the top. If it is not present NO window is moved. Also the mouse pointer will move to the window as well.

Example: Batch processing

This next example is not for the faint-hearted but is a reasonably simple and fairly useful set of keystrokes that use two more of the programs in the Library. The first program, **FileList**, takes a pathname and sets a system variable to each file object or application (not directories) found within the directory pointed to by the pathname. The pathname can be supplied explicitly or within a variable such as <Pointer$Text>.

This list of files in a directory enables the user to work through the files one by one achieving a similar object in each case.

The second program in this example, **/xxxSTRIP** takes a single parameter which is either a file name or a filepath and returns a Keystroke system variable with the PC DOS three-character suffix removed. So '`CONFIG/SYS`' would be transformed into '`CONFIG`'. Furthermore, if the suffix is one of 54 recognised at present by the program, it will give that file a corresponding RISC OS Filetype number, e.g. the DOS suffix for a JPEG file, JPG would be filetyped C85, and a DOS TXT file would be given the FFF filetype. Adding the letter 'a' to the end of the filepath parameter will allow the file type to be automatically translated.

With the advent of the Risc PC and Apple Mac conversion software, more Acorn users are transferring files between platforms. Envisage a directory full of files dragged out of your 'Drive_C' DOS directory. With one keypress Keystroke can be set to process each file, stripping off the suffix and applying Acorn filetypes.

We will need to set up two separate keystrokes. The first will set up the list of file pathname variables from the directory, and then call the second. The second will process each file in turn and call itself recursively until the batch is finished.

First keystroke

Page 1:

- Open the Keystroke definitions window. Setup an appropriate key combination, say <Ctrl>+<Alt>+B.
- Click the ***Command** button
- Name this Keystroke in the usual way, say, '`File List`'.
- Open the Library directory by clicking on the **Library** icon.
- Drag the **FileList** Basic file from the directory onto the **Run** icon. This will enter '`Run <Keystroke$Lib>.FileList`'
- Type a space and then click on the **Pointer Text** icon to enter '`<Pointer$Text>`' into the **Text** icon. Finally press <Return>.
- Click on the **New** page icon.

Page 2:

- Click on the ***Command** button.
- Type '`Keystroke Start StripPC`' and press <Return>. This instruction will call the second keystroke with the name '`Start StripPC`'.
- To finish this keystroke definition click on the **Link** icon.

When activated this pair of keystrokes will feed the pathname of the directory the mouse pointer is over to the Basic program **FileList**. This program develops a list of all the files in that directory, storing its pathname in a system variable called '`Ks$dir#`', the # refers to a number between 1 and the total number of files in the directory. The program also sets up another system variable called '`Ksno`' which holds the total number of files found.

Second keystroke

Page 1:

- Set up a new keystroke with an obscure keypress, say <Shift>+<Ctrl>+<Alt>+'~' (the tilde character). This keystroke is called by the '`File List`' keystroke and cannot work without the list of files this generates, so you don't want to accidentally press keys that cause this keystroke to run on its own.
- Click on the ***Command** button.
- Type '`seteval ksfile Ks$dir<ksno>`' into the **Text** icon and press <Return>. This defines a new variable '`ksfile`' derived from the name of the '`ks$dir`' variable and the number of the total number of files counted in the directory, held in the variable '`<ksno>`'.
- Name this keystroke '`Start StripPC`' in the usual way.
- Click on the **New** page icon to open a new keystroke page.

Page 2:

- Click on the ***Command** button.
- If the Library directory window is not already open click on the **Library** icon.
- Drag the '`/xxxStrip`' file onto the **Run** icon. This will enter: '`Run <Keystroke$Lib>./xxxSTRIP`' into the **Text** icon.
- Type a space and '`<ksfile>a`' into the **Text** icon. Press <Return>. The letter '`a`' after the variable ensures that the program will also filetype the file appropriately.

This will run the Basic program which will strip the PC suffix from the file whose pathname is held in the '`<ksfile>`' variable. The letter '`a`' after the variable ensures that the program will also filetype the file appropriately.

- Click on the **Link** button to group the keystrokes. This is important as the group is called by another keystroke and not a key press.
- Click on the **New** page icon to open a new keystroke page.

Page 3:

- Click on the ***Command** button.
- Type '`seteval ksno ksno - 1`'. This command decrements the variable '`ksno`' by one.
- Click on the **Link** button to group the keystrokes.
- Click on the **New** page icon to open a new keystroke page.

Page 4:

- Click on the ***Command** button.
- Type '`Keystroke Start StripPC`' and press <Return>.
- Click on the **Link** button to group the keystrokes.

This last keystroke calls the group again with the decremented variable '`ksno`'. The group will run the '`/xxxSTRIP`' program again with a new filepath held in the '`<ksfile>`' variable. This will cycle through all the files held in the original directory pointed at until '`ksno`' reaches '`0`' when an '`Unknown Operand`' error will occur. Hold down the <Escape> key to clear the screen.

- More experienced users may want to program the keystroke so that this error will not occur but I will leave that to you.

Addendum: more about the Drag option

The Drag option in the Icon Click window (page 23) lets you do a simulated 'drag' operation from standard Save-type dialogue boxes to another application's window.
To set this up you must first pop up the Save dialogue box in question using a Open dialogue command or, as the example below shows, using a F3 simulated keypress.
Then perform a Move Mouse command to set the mouse pointer at the window or icon where you wish the drag operation to finish (assuming of course that the destination window is already open).
Finally, perform the Drag icon-click operation on the Save icon in the Save dialogue box.

Example: Simulate dragging a Save dialogue to another application

Load !Edit and bring up two blank Edit windows by clicking twice (with the Select button on the mouse) on the Edit icon on the iconbar.
Reduce the size of these Edit windows but keep them side by side and visible. Type some text into the left-hand Edit window.

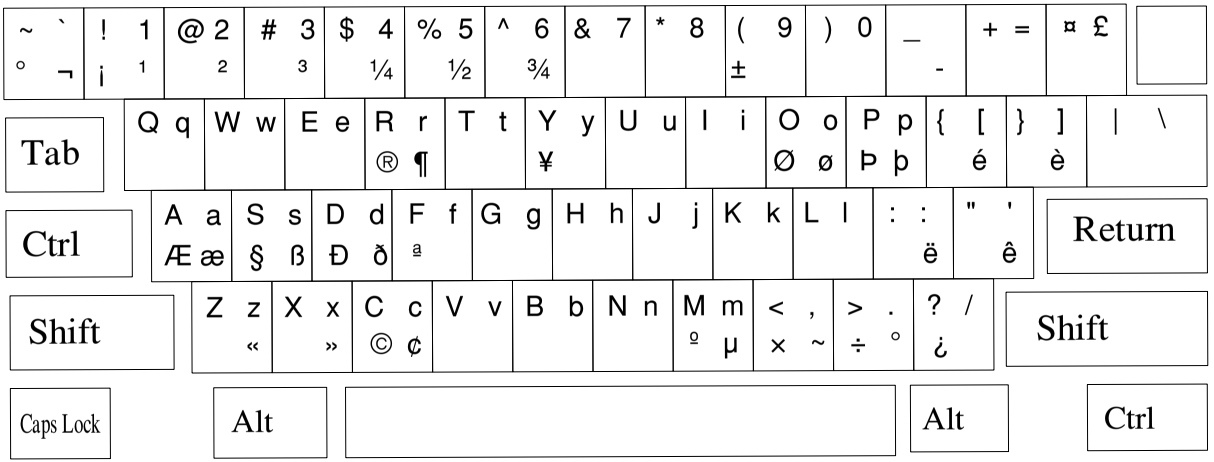
- Choose a Keystroke key combination, e.g. F1.
- Highlight the Insert Text option and type: `\F3\` (this will bring up the 'Save as' dialogue box, as Edit normally uses F3 to do this.)
- Click on the New icon and highlight the Move option.
- Click the 'Set ptr' option and drag the 'Drag to set' icon to the right-hand Edit window.
- Set the Link option on. (This will move the mouse pointer to where the saved file is to be placed.)
- Click on the New icon and highlight the 'Icon click' option.
- Make sure the left-hand Edit window has the input focus by clicking inside the window with the Select button and press F1. (This will cause the 'Save as' box to appear and the mouse pointer will move over the right-hand Edit window.)
- Now position the mouse pointer over the Text sprite icon inside the 'Save as' box and press the two Shift keys at the same time. Keystroke will now contain the technical details of the 'Save as' box.
- Finally highlight the Link and Drag option.

The Keystroke sequence is now set.
To try it out, first make sure the left-hand Edit window has the input focus by clicking inside the window with the Select button and press F1.
On pressing F1 the contents of the left-hand Edit window will be automatically 'dragged' into the right-hand window!

✪ If you want to drag a filer file onto an application then due to RISC OS limitations this can be done only via the Filer Copy operation. This pops up a 'Copy as' dialogue box, which can then be 'dragged' to the destination of your choice as described above.

Keyboard layout

Keyboard with OS version 3.10 or later.
The top line shows the normal characters with and without shift.
The bottom line shows the corresponding characters obtained by holding <Alt>+<Shift>+key and <Alt>+key. To produce letters with accents (é, â, û, etc.) hold <Alt> and type accent key (shown below as é, è, ë, ê) release <Alt> and then type the letter under the accent.



Appendix B

About this edition of the manual

This comprehensive manual for Keystroke was originally produced many years ago and was distributed alongside !Keystroke as Manual306 since 2003. The 3.06 was a version number for the *manual*, nothing to do with the Keystroke version number; indeed that 2003 edition included a note (now page 46) about a feature new in Keystroke 4.02.

Keystroke 4.09 came out in April 2014, containing Martin Avison's improvements to polling (Keystroke now makes fewer demands on the processor) and to the List display. See Martin's readme file within the application for further improvements.

I took this opportunity to make some comparatively small updates to the 2003 manual, and am calling the result Manual4. The biggest change is to page-numbering.

• This revision incorporates (on page 60) material from a 2001 "Manual Update" file that was distributed alongside the 2003 edition and never seems to have found its way into it.

• I changed some fonts to standard ones. Arrow-key symbols ⇦ ⇨ ⇧ ⇩ were in a font called Acorn (by Richard Hallas, supplied by Quantum Software); I changed these to embedded graphics — all with the aim of making things display correctly in the RISC OS PDF readers.

• In some localized sections of text, I tidied up some tangles in Impression DDF tags before making PDF from the document.

• **Page-numbering:** To make it easier to jump to a specified page when using either Impression or a PDF reader, page numbers now run simply from 1 to the end. The previous edition started again at 1 for the main text (after the six introductory pages). In other words, printed page numbers are now the same as internal page numbers. Accordingly, I adjusted any page references in the body of the text and regenerated the Contents and Index.

—Jim Nagel, 2014 April 08

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y