PRM-in-XML style gallery

Introduction

This document is intended to show some examples of the different styles of the Acorn manuals and some presentations using the PRM-in-XML formatted content. The content has been taken from scanned PDFs, for the original manuals, and the HTML and PDF generated by experimental versions of the PRM-in-XML. That is to say, it's not perfect, but it demonstrates some of the flexibility.

Example pages

To provide examples of the formatting of content, 3 sample pages have been selected from the manuals:

- The contents page
- The start of the introdution to RISC OS chapter
- The OS_Claim SWI definition.

These pages demonstrate many of the features of the manual. They should be easy to compare between the different versions.

Acorn manuals

The Acorn manuals that are being examined here cover a few years of development, during which time Acorn refined the style of the manuals considerably. The manuals which will be shown are:

- RISC OS 2 reference manual
- C release 4 reference manaul
- RISC OS 3 reference manual
- RISC OS 3 reference manual volume 5a

Other manuals exist within the timeline, with varying features, but these are most relevant to the intended use of the PRM-in-XML system.

PRM-in-XML formats

PRM-in-XML is flexible in how it can generate content, but the examples here will concentrate solely on the HTML 5/CSS format. This will vary only the CSS used within the content. Much greater flexibility is afforded by being able to configure the CSS as required but here only limited canned variants of the standard CSS template are being shown.

In addition to the HTML, the same content is passed to *PrinceXML* for conversion to a PDF. This is done without modification to the intermediate files. Other conversion solutions exist and could be used with the paged media CSS.

Some of the example content is incomplete - the images have some bad lines - and on some pages the contents and images have not been styled properly. These are artifacts of incomplete stylesheets, which can be addressed in time.

The PRM-in-XML tool has a configuration which allows for layering of CSS snippets on top of a base stylesheet. This configuration is used to change the presentation of the content. The variants which are available at the present time are:

Variant	Meaning
prm	Closer to the RISC OS 3 PRMs for paged media and screen rendering.
acornfs	Acorn Functional Specification style.
prm-ro2	Closer to the RISC OS 2 PRMs for paged media. Not complete for screen rendering.
numbered-sections	Apply numbers to the sections on the page.
body-novarese	Change body font to ITC Novarese (requires local installation of this commercial font).
body-fraunces	Change body font to Fraunces (requires local installation of this freely available Google font).
webfont-fraunces	Download the Fraunces font as required. Use in conjunction with 'body-fraunces'.
heading-raleway	Change heading font to Raleway (requires local installation of this freely available Google font).
webfont-raleway	Download the Raleway font as required. Use in conjunction with 'heading-raleway'.
large-bullets	Apply larger bullets to lists. This is closer in style to the reference manuals.
drop-character	Apply a drop character to the first letter of the first paragaph.
no-edge-index	Remove the grey region from the right pages.

For reference, this document was generated with the standard settings, but an extra CSS file was added to give the images a rounded border.

```
riscos-prminxml -p css-file=extra.css -f html5 gallery.xml
```

Acorn: RISC OS 2 manuals

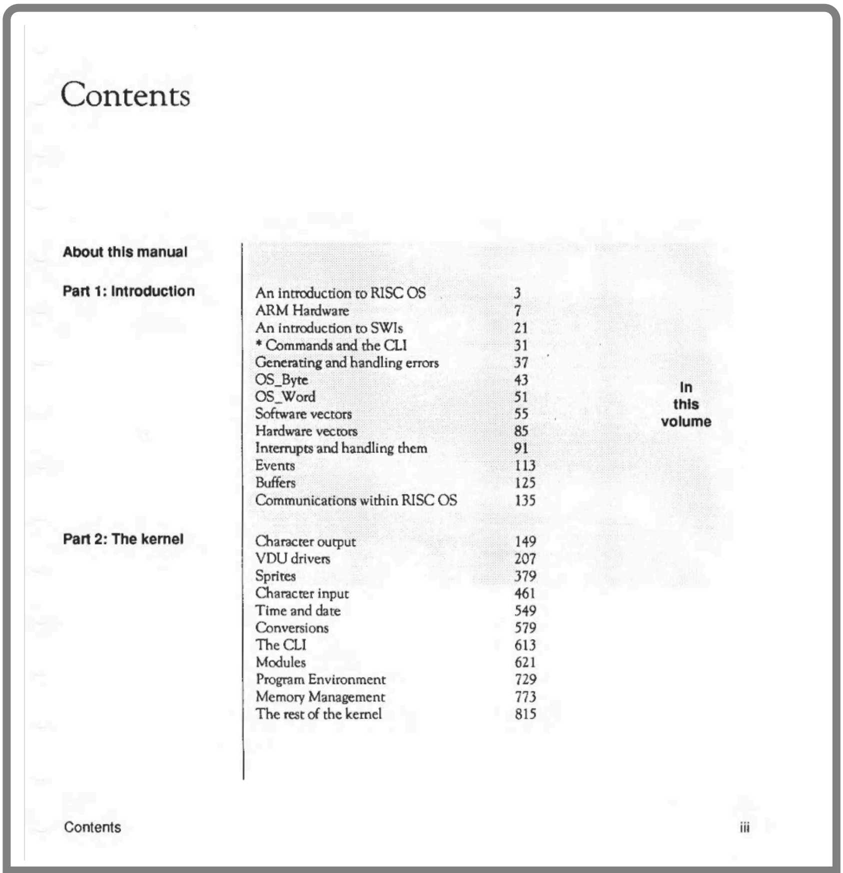
The RISC OS 2 manuals had some distinctive features which make it stand out from the later manuals. It is notable that these manuals use the Novarese font which was retained for later publications.

- Headings are restricted to the left of the page. Content is on the right.
- The whole manual uses a vertical dividing line to sepatate headings from the content.
- Chapter and sections are shown in the footers, together with the page number.
- Page numbers in the contents page line up vertically.
- An edge index is not used.

Compare this to the *SunOS manuals* of the same period.

Because of this separation of the content, there is a lot of space wasted on many pages. However, finding sections in the API definition pages is a lot easier. In the later versions of the manuals this left indent is still present (although not as large).

Example pages

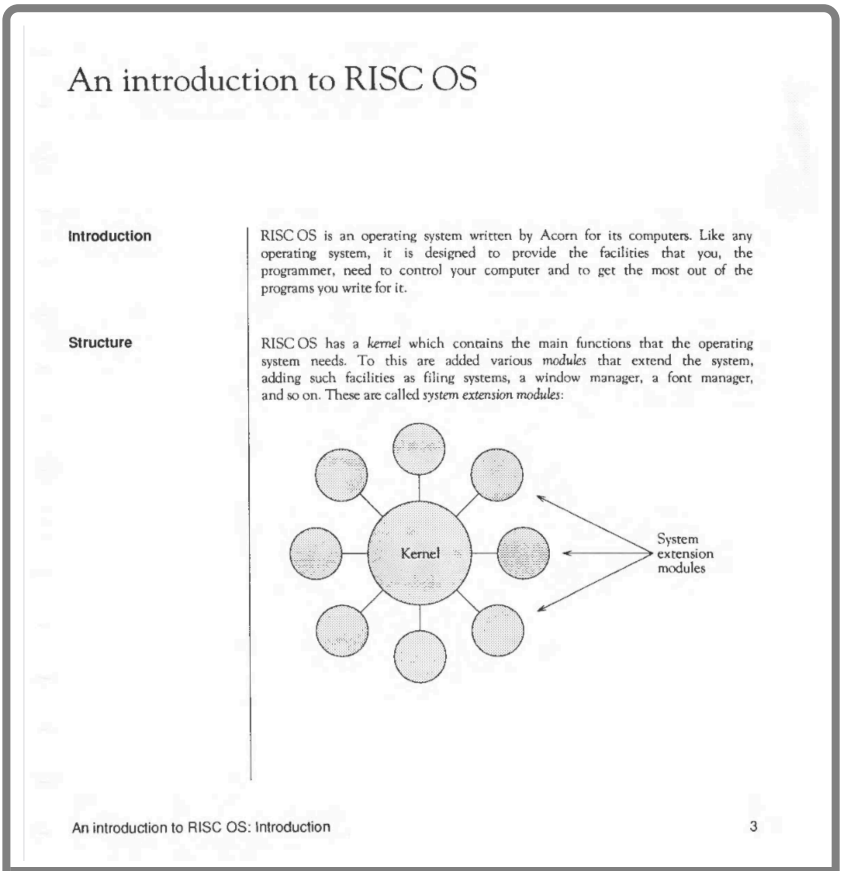


Contents

In this volume

Contents iii

RISC OS 2 contents page



An introduction to RISC OS

Introduction

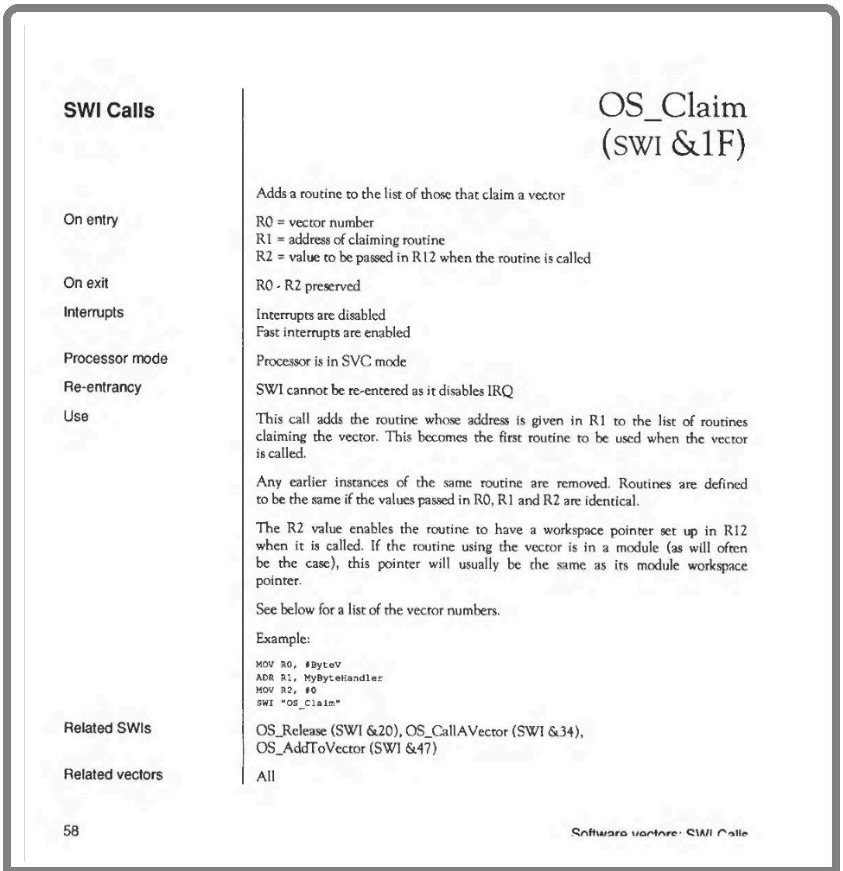
RISC OS is an operating system written by Acorn for its computers. Like any operating system, it is designed to provide the facilities that you, the programmer, need to control your computer and to get the most out of the programs you write for it.

Structure

RISC OS has a *kernel* which contains the main functions that the operating system needs. To this are added various *modules* that extend the system, adding such facilities as filing systems, a window manager, a font manager, and so on. These are called *system extension modules*:

An introduction to RISC OS: Introduction 3

RISC OS 2 intro chapter



SWI Calls

OS_Claim (SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number
R1 = address of claiming routine
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI cannot be re-entered as it disables IRQ

Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

Any earlier instances of the same routine are removed. Routines are defined to be the same if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

See below for a list of the vector numbers.

Example:

```
MOV R0, #ByteV
ADR R1, MyByteHandler
MOV R2, #0
SWI "OS_Claim"
```

Related SWIs

OS_Release (SWI &20), OS_CallAVector (SWI &34),
OS_AddToVector (SWI &47)

Related vectors

All

58 Software vectors: SWI Calls

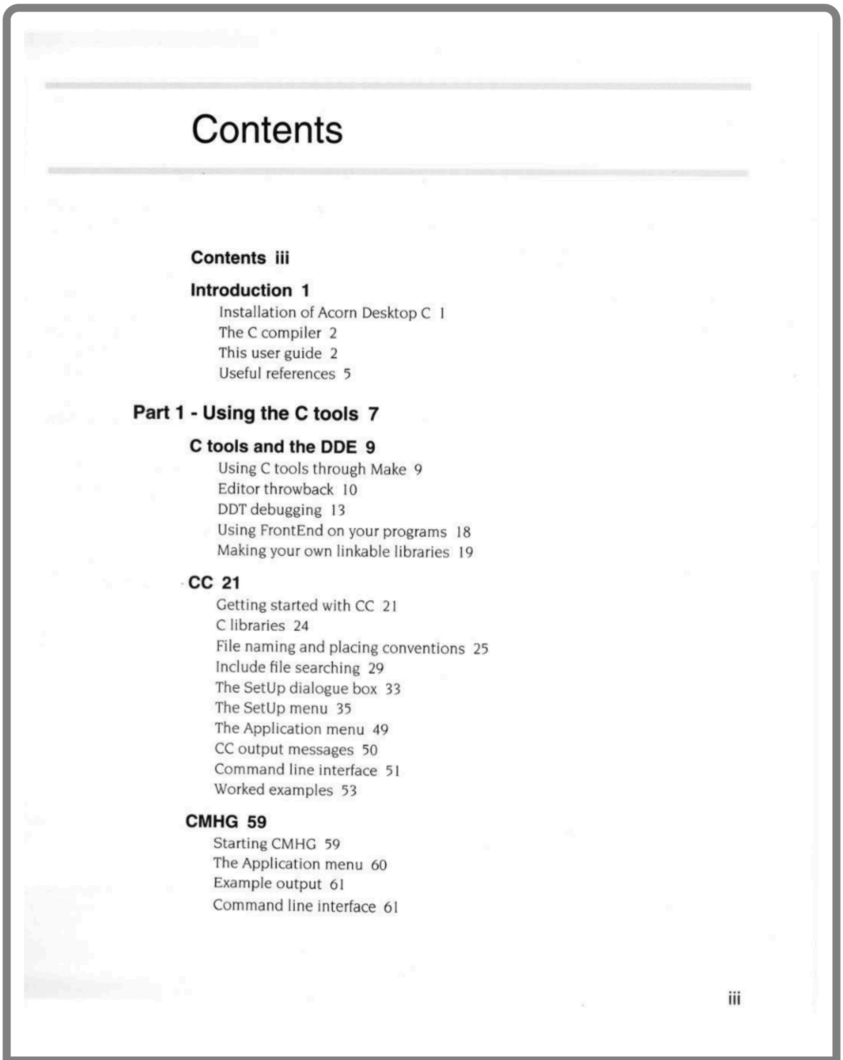
RISC OS 2 SWI definition

Acorn: Acorn C Release 4

The Acorn C Release 4 manual is an updated style from that of the RISC OS 2 PRMs, and has many of the features of the later publications.

- The contents page uses grey horizontal bars, but only on some of the headings.
- The contents page has page numbers alongside the sections, which isn't as clear.
- The contents page not only references the chapter name, but also sections within the chapter.
- The first paragraph of each chapter has a drop initial applied to the first character.

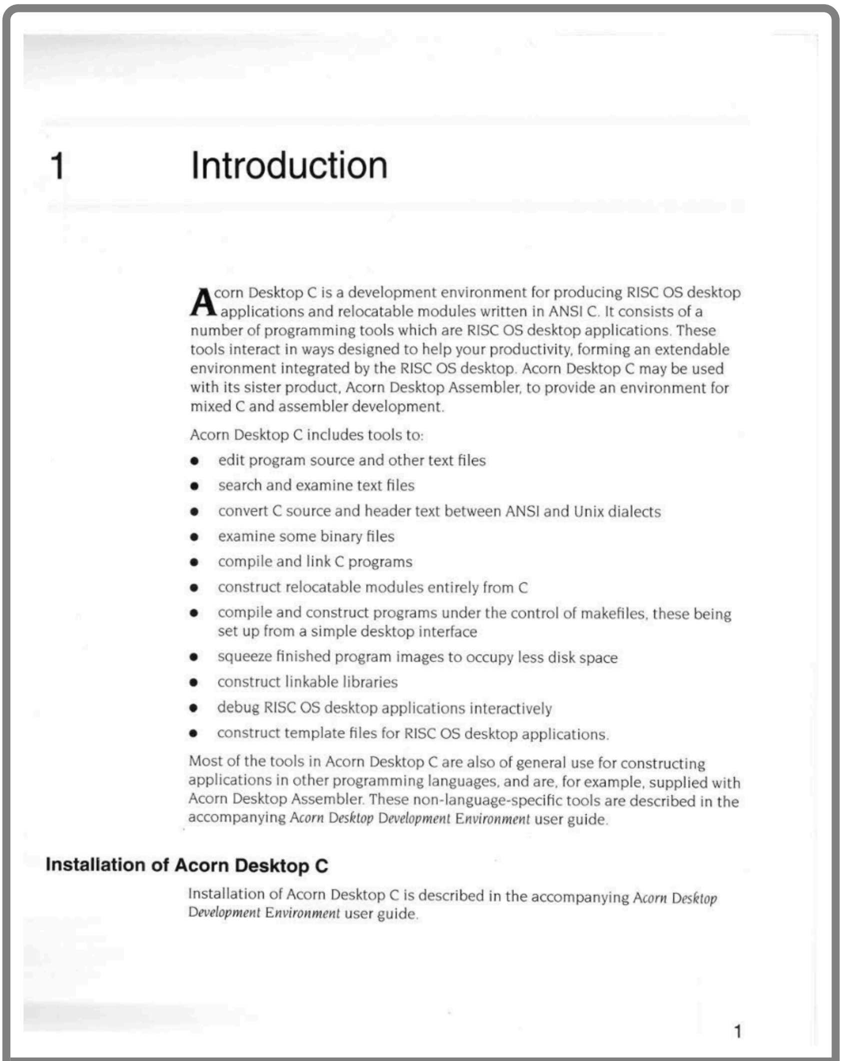
Example pages



Contents

iii

Acorn C Release 4 contents page



1 Introduction

Acorn Desktop C is a development environment for producing RISC OS desktop applications and relocatable modules written in ANSI C. It consists of a number of programming tools which are RISC OS desktop applications. These tools interact in ways designed to help your productivity, forming an extendable environment integrated by the RISC OS desktop. Acorn Desktop C may be used with its sister product, Acorn Desktop Assembler, to provide an environment for mixed C and assembler development.

Acorn Desktop C includes tools to:

- edit program source and other text files
- search and examine text files
- convert C source and header text between ANSI and Unix dialects
- examine some binary files
- compile and link C programs
- construct relocatable modules entirely from C
- compile and construct programs under the control of makefiles, these being set up from a simple desktop interface
- squeeze finished program images to occupy less disk space
- construct linkable libraries
- debug RISC OS desktop applications interactively
- construct template files for RISC OS desktop applications.

Most of the tools in Acorn Desktop C are also of general use for constructing applications in other programming languages, and are, for example, supplied with Acorn Desktop Assembler. These non-language-specific tools are described in the accompanying *Acorn Desktop Development Environment* user guide.

Installation of Acorn Desktop C

Installation of Acorn Desktop C is described in the accompanying *Acorn Desktop Development Environment* user guide.

Acorn C Release 4 intro chapter

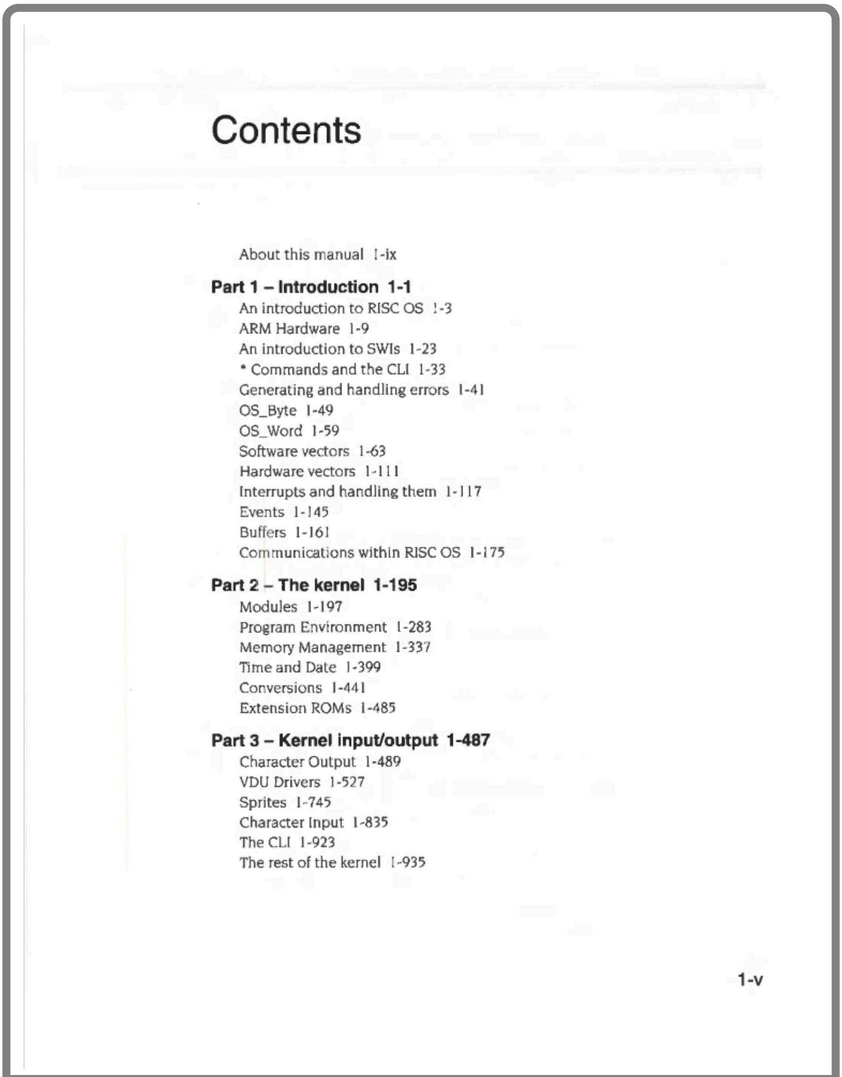
Acorn: RISC OS 3 manuals

The RISC OS 3 manuals were are probably what most people will remember.

- Whilst the earlier manuals appear to be square in their presentation, the RISC OS 3 manuals appear to use a slightly rectangular portrait layout.
- The contents page has dropped chapter sections, but now separates the manual into logical 'parts'.
- Page numbers now include a volume number, which is distinct from the 'part' of the manual.
- Parts of the manual are named and use the edge index to locate them.
- The drop initial used in the Acorn C Release 4 manual has been dropped.
- The style of the API definitions is basically unchanged from the RISC OS 2 manuals, save the headings now taking vertical space, instead of being in the margin.
- Page headers now alternate between the chapter name and the section name.
- Page footers only include the page number.

The vertical space used by the headings on the API definitions is arguably a poorer use of space than in the RISC OS 2 manuals. However, the style is familiar and therefore this usage is largely expected.

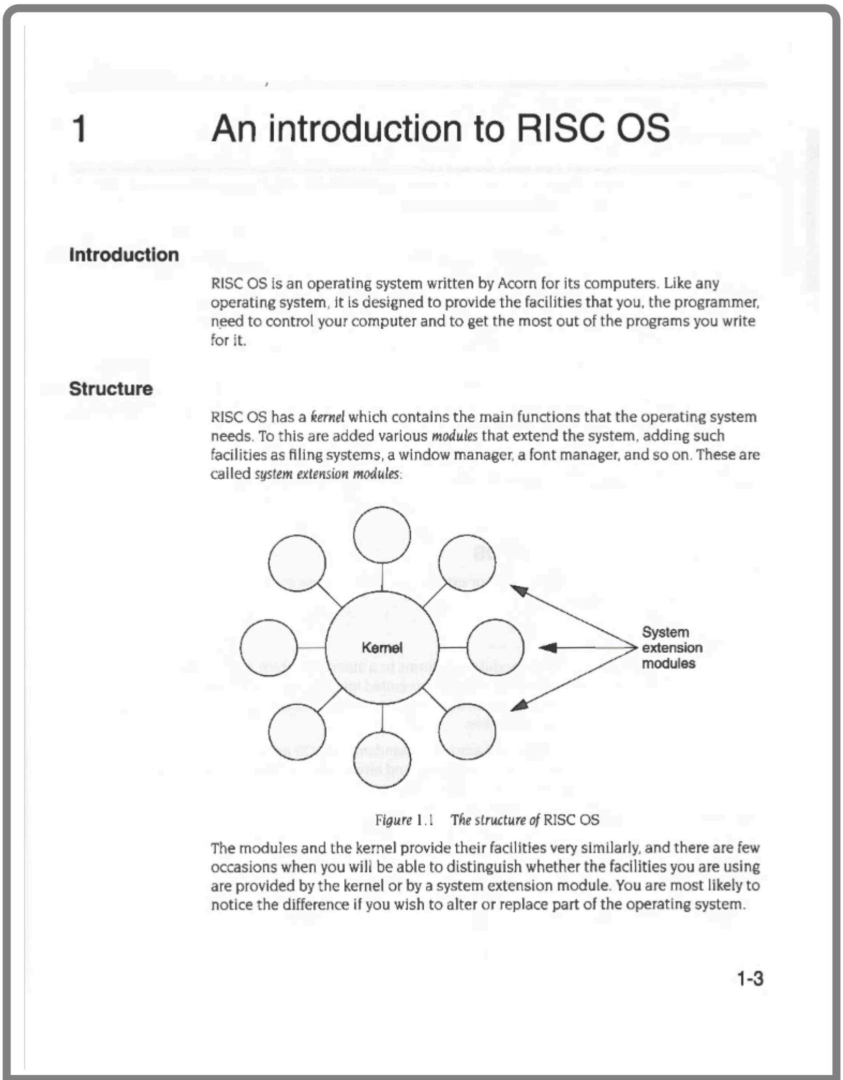
Example pages



Contents

1-v

RISC OS 3 contents page



1 An introduction to RISC OS

Introduction

RISC OS is an operating system written by Acorn for its computers. Like any operating system, it is designed to provide the facilities that you, the programmer, need to control your computer and to get the most out of the programs you write for it.

Structure

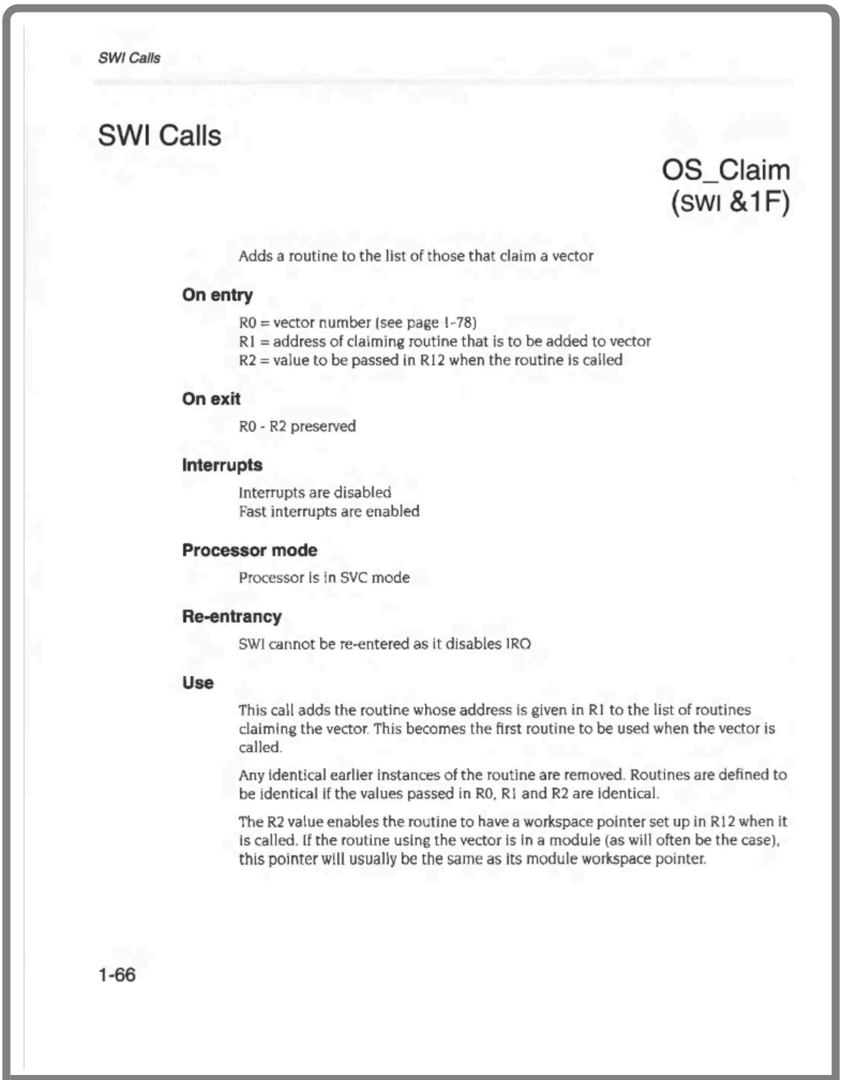
RISC OS has a *kernel* which contains the main functions that the operating system needs. To this are added various *modules* that extend the system, adding such facilities as filing systems, a window manager, a font manager, and so on. These are called *system extension modules*:

Figure 1.1 *The structure of* RISC OS

The modules and the kernel provide their facilities very similarly, and there are few occasions when you will be able to distinguish whether the facilities you are using are provided by the kernel or by a system extension module. You are most likely to notice the difference if you wish to alter or replace part of the operating system.

1-3

RISC OS 3 intro chapter



SWI Calls

SWI Calls

OS_Claim (SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number (see page 1-78)
R1 = address of claiming routine that is to be added to vector
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI cannot be re-entered as it disables IRQ

Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

Any identical earlier instances of the routine are removed. Routines are defined to be identical if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

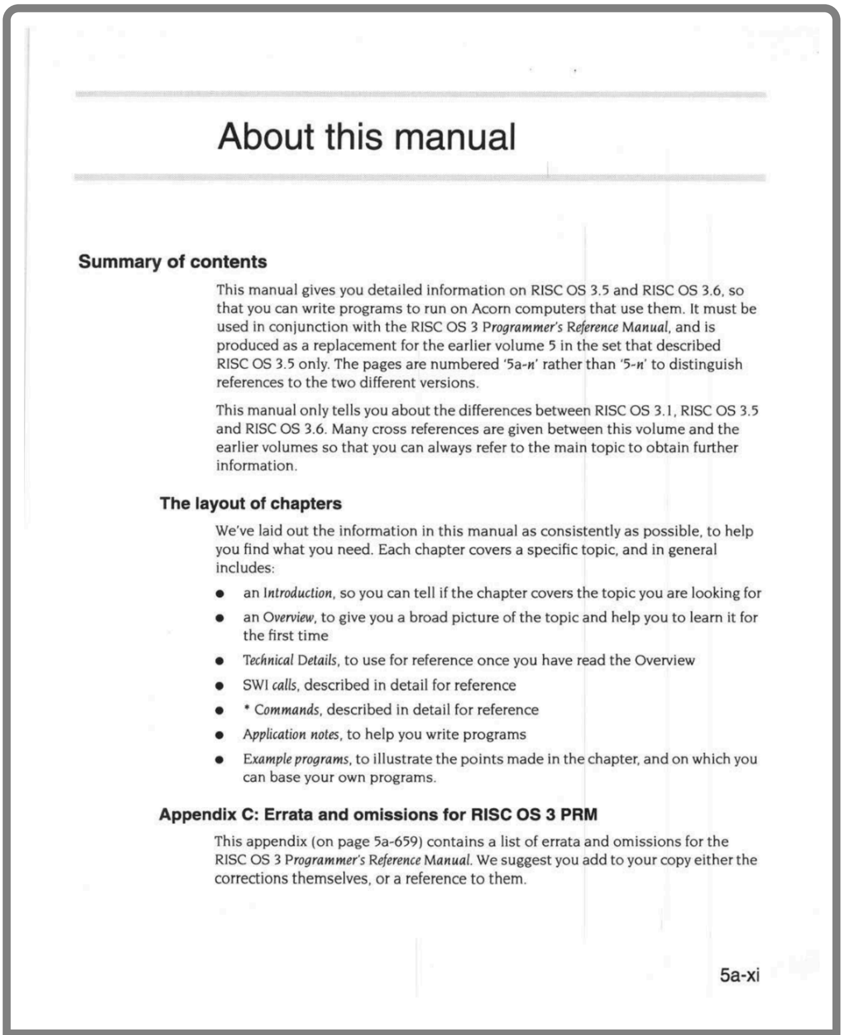
1-66

RISC OS 3 SWI definition

Acorn: RISC OS 3 manual, volume 5a

Volume 5a was largely unchanged in style from the RISC OS 3 manuals, although some elements have been resized slightly.

Example pages



About this manual

Summary of contents

This manual gives you detailed information on RISC OS 3.5 and RISC OS 3.6, so that you can write programs to run on Acorn computers that use them. It must be used in conjunction with the RISC OS 3 *Programmer's Reference Manual*, and is produced as a replacement for the earlier volume 5 in the set that described RISC OS 3.5 only. The pages are numbered '5a-*n*' rather than '5-*n*' to distinguish references to the two different versions.

This manual only tells you about the differences between RISC OS 3.1, RISC OS 3.5 and RISC OS 3.6. Many cross references are given between this volume and the earlier volumes so that you can always refer to the main topic to obtain further information.

The layout of chapters

We've laid out the information in this manual as consistently as possible, to help you find what you need. Each chapter covers a specific topic, and in general includes:

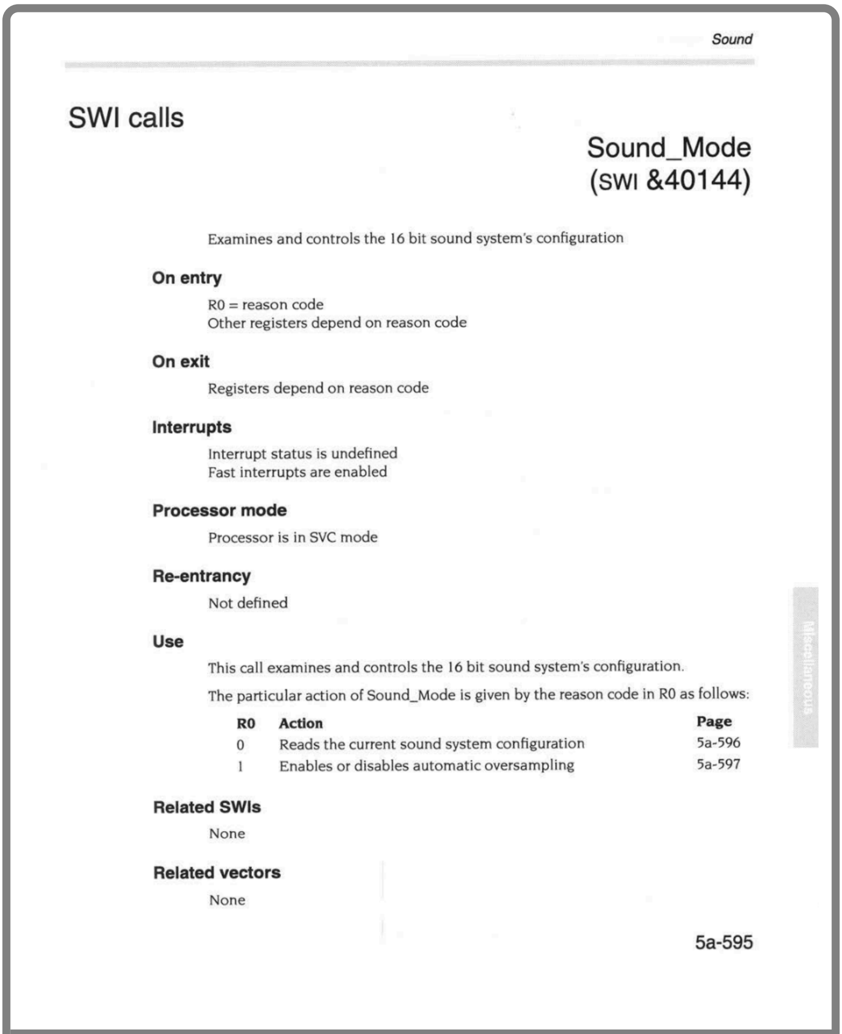
- an *Introduction*, so you can tell if the chapter covers the topic you are looking for
- an *Overview*, to give you a broad picture of the topic and help you to learn it for the first time
- *Technical Details*, to use for reference once you have read the Overview
- SWI *calls*, described in detail for reference
- * *Commands*, described in detail for reference
- *Application notes*, to help you write programs
- *Example programs*, to illustrate the points made in the chapter, and on which you can base your own programs.

Appendix C: Errata and omissions for RISC OS 3 PRM

This appendix (on page 5a-659) contains a list of errata and omissions for the RISC OS 3 *Programmer's Reference Manual*. We suggest you add to your copy either the corrections themselves, or a reference to them.

5a-xi

Volume 5a intro chapter



Sound

Miscellaneous

SWI calls

Sound_Mode (SWI &40144)

Examines and controls the 16 bit sound system's configuration

On entry

R0 = reason code
Other registers depend on reason code

On exit

Registers depend on reason code

Interrupts

Interrupt status is undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Not defined

Use

This call examines and controls the 16 bit sound system's configuration.

The particular action of Sound_Mode is given by the reason code in R0 as follows:

R0	Action	Page
0	Reads the current sound system configuration	5a-596
1	Enables or disables automatic oversampling	5a-597

Related SWIs

None

Related vectors

None

5a-595

Volume 5a SWI definition

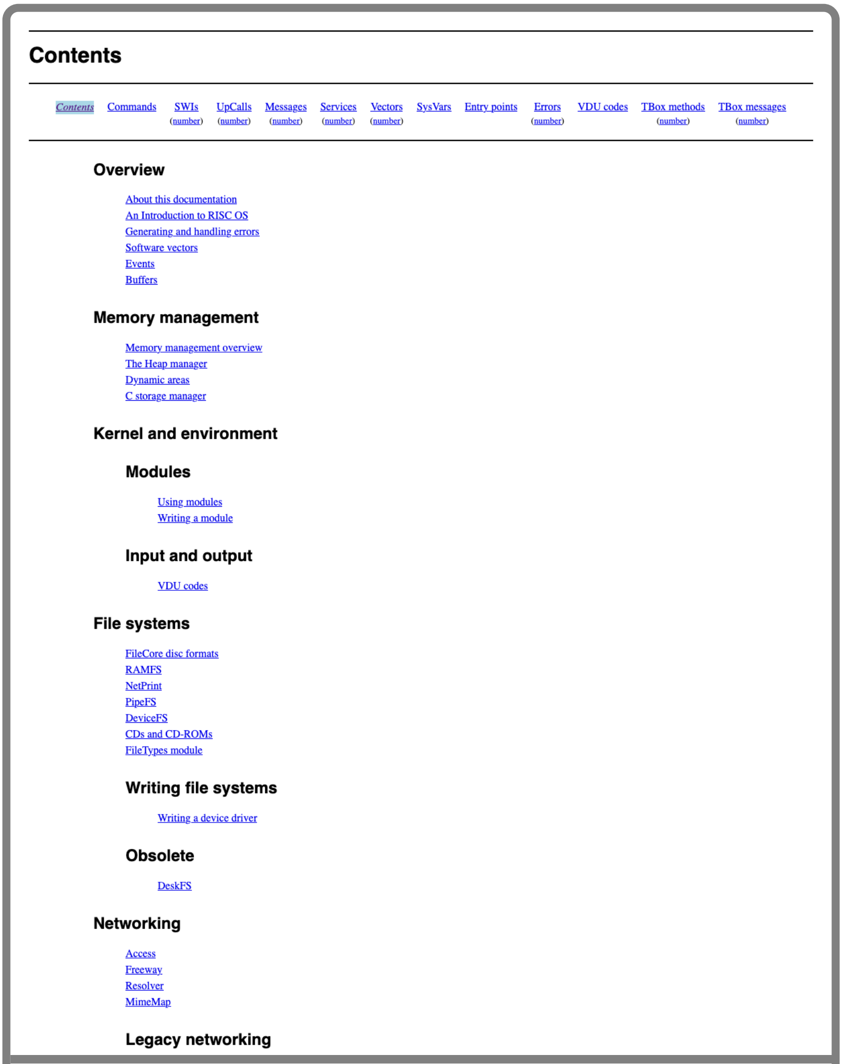
PRM-in-XML: Default configuration

The default configuration of PRM-in-XML is intended to take on the style of the original RISC OS 3 manuals, whilst being able to be used on a variety of desktop sizes. It is suitable for printing, but has not been tailored specifically for any given device size.

- The contents page is a similar style to that of the RISC OS 3 contents pages.
- Navigation bars are included on the contents to take you to index pages for each of the API definition types. Bars are included both at the top and bottom of the contents page.
- Documentation is organised into named sections, which may be nested arbitrarily.
- Horizontal bars divide sections within the chapters, in addition to the heading being left aligned.
- Chapters open with a navigation block which links to the sections present within the chapters.
- Bullets use the standard browser indentation, not the highly condensed form of the PRM.
- Links are just regular HTML links, which take you to the relevant section. No page numbers are used.
- The PDF generation uses page breaks to split the chapter content at section boundaries.
- The PDF has page numbers beside the links on the contents page, in italic to make them stand out.
- Within the chapter the PDF shows links together with the page number which contains the content.

Content mistakes here are easy to see, and will be present on all the PRM-in-XML examples. The SWI examples have excessive whitespace - this is an authoring error. The image on the intro chapter has a rogue line on the left for some reason.

Example pages (HTML)



Contents

Contents	Commands	SWIs	UpCalls	Messages	Services	Vectors	SysVars	Entry points	Errors	VDU codes	TBox methods	TBox messages
		(number)	(number)	(number)	(number)	(number)			(number)		(number)	(number)

Overview

Memory management

Kernel and environment

Modules

Input and output

File systems

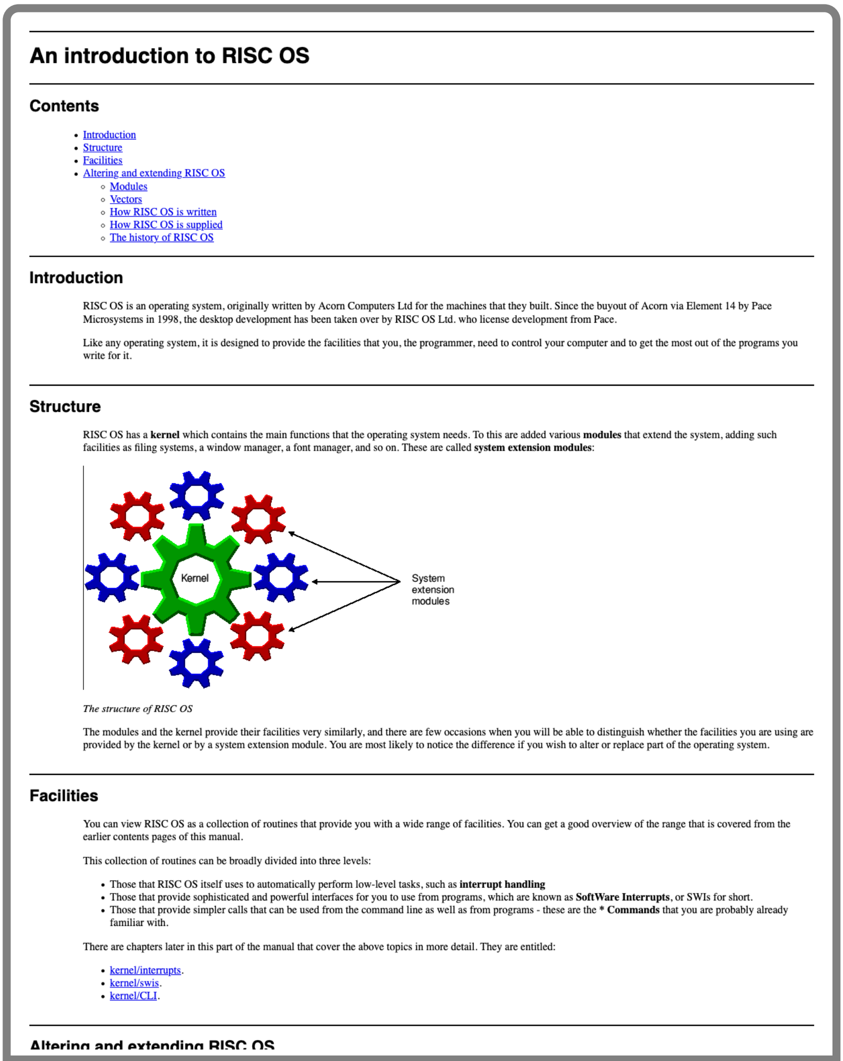
Writing file systems

Obsolete

Networking

Legacy networking

PRM-in-XML default: contents page



An introduction to RISC OS

Contents

Introduction

RISC OS is an operating system, originally written by Acorn Computers Ltd for the machines that they built. Since the buyout of Acorn via Element 14 by Pace Microsystems in 1998, the desktop development has been taken over by RISC OS Ltd. who license development from Pace.

Like any operating system, it is designed to provide the facilities that you, the programmer, need to control your computer and to get the most out of the programs you write for it.

Structure

RISC OS has a **kernel** which contains the main functions that the operating system needs. To this are added various **modules** that extend the system, adding such facilities as filing systems, a window manager, a font manager, and so on. These are called **system extension modules**:

The structure of RISC OS

The modules and the kernel provide their facilities very similarly, and there are few occasions when you will be able to distinguish whether the facilities you are using are provided by the kernel or by a system extension module. You are most likely to notice the difference if you wish to alter or replace part of the operating system.

Facilities

You can view RISC OS as a collection of routines that provide you with a wide range of facilities. You can get a good overview of the range that is covered from the earlier contents pages of this manual.

This collection of routines can be broadly divided into three levels:

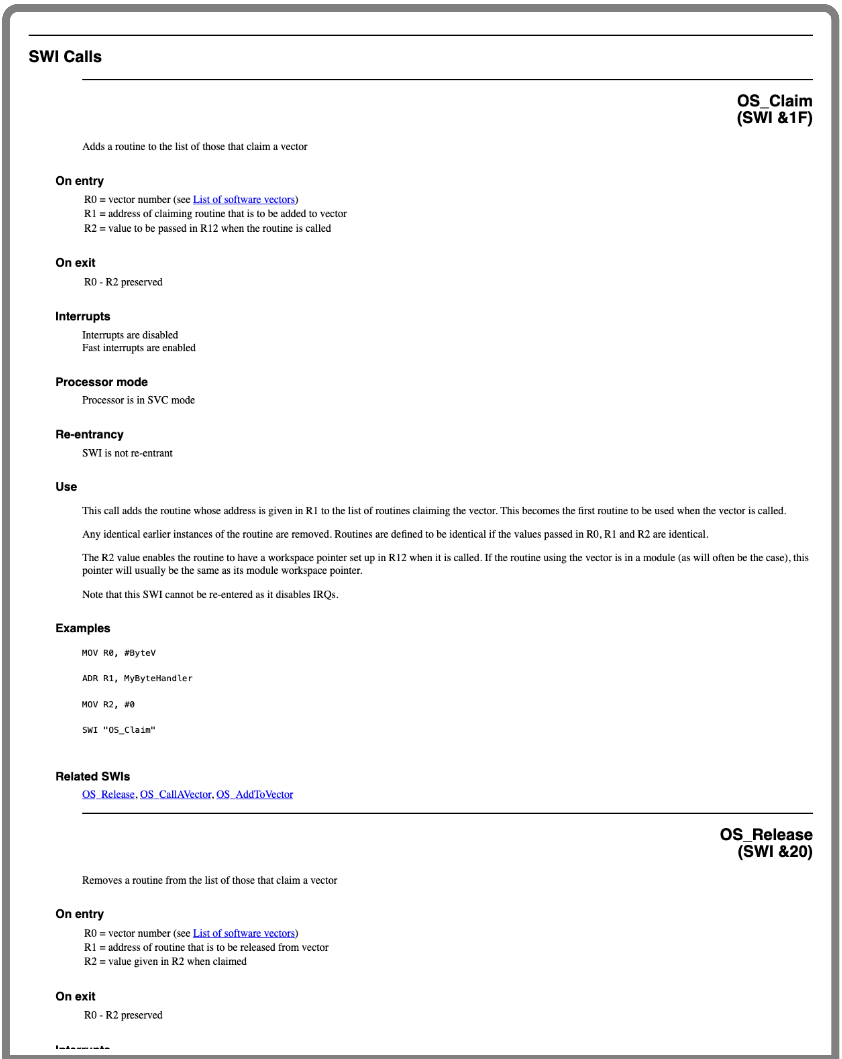
- Those that RISC OS itself uses to automatically perform low-level tasks, such as **interrupt handling**
- Those that provide sophisticated and powerful interfaces for you to use from programs, which are known as **SoftWare Interrupts**, or SWIs for short.
- Those that provide simpler calls that can be used from the command line as well as from programs - these are the *** Commands** that you are probably already familiar with.

There are chapters later in this part of the manual that cover the above topics in more detail. They are entitled:

- kernel/interrupts.
- kernel/swis.
- kernel/CLI.

Altering and extending RISC OS

PRM-in-XML default: intro chapter



SWI Calls

OS_Claim (SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number (see List of software vectors)
R1 = address of claiming routine that is to be added to vector
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

Any identical earlier instances of the routine are removed. Routines are defined to be identical if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

Note that this SWI cannot be re-entered as it disables IRQs.

Examples

```
MOV R0, #ByteV

ADR R1, MyByteHandler

MOV R2, #0

SWI "OS_Claim"
```

Related SWIs

OS_Release, OS_CallAVector, OS_AddToVector

OS_Release (SWI &20)

Removes a routine from the list of those that claim a vector

On entry

R0 = vector number (see List of software vectors)
R1 = address of routine that is to be released from vector
R2 = value given in R2 when claimed

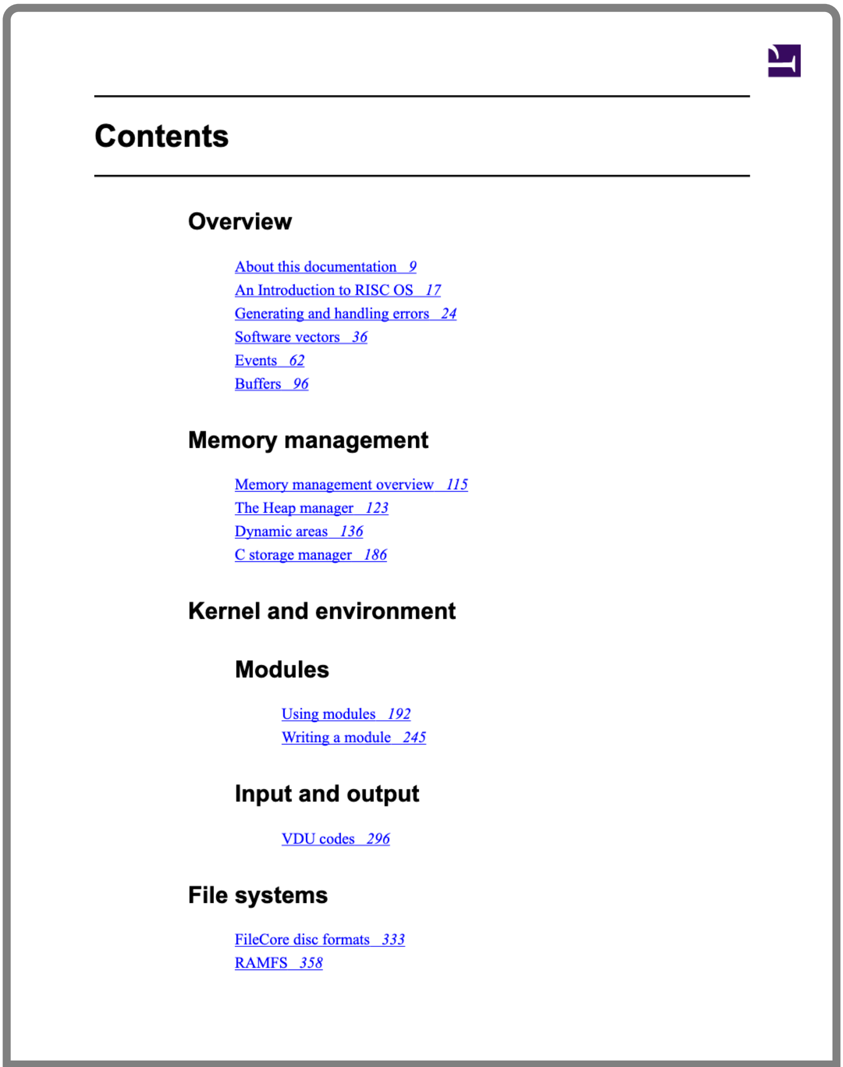
On exit

R0 - R2 preserved

Interrupts

PRM-in-XML default: SWI definition

Example pages (PDF)



Contents

Overview

Memory management

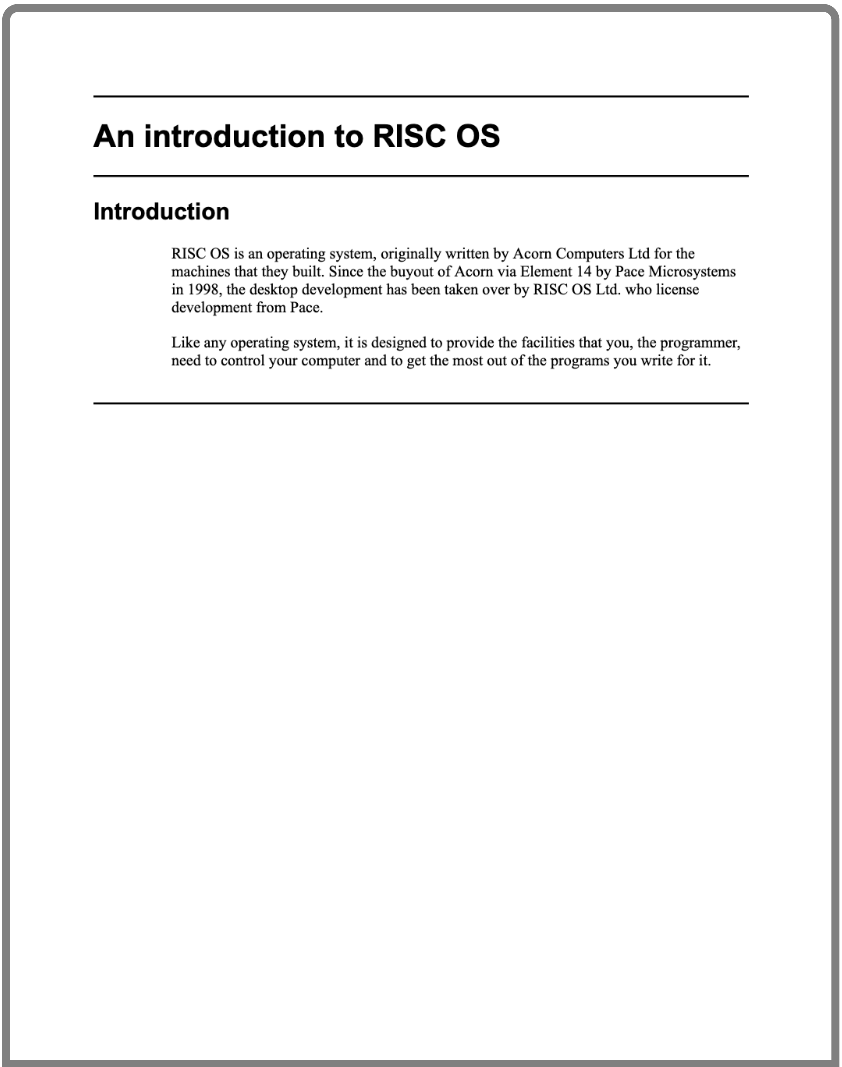
Kernel and environment

Modules

Input and output

File systems

PRM-in-XML default (PDF): contents page



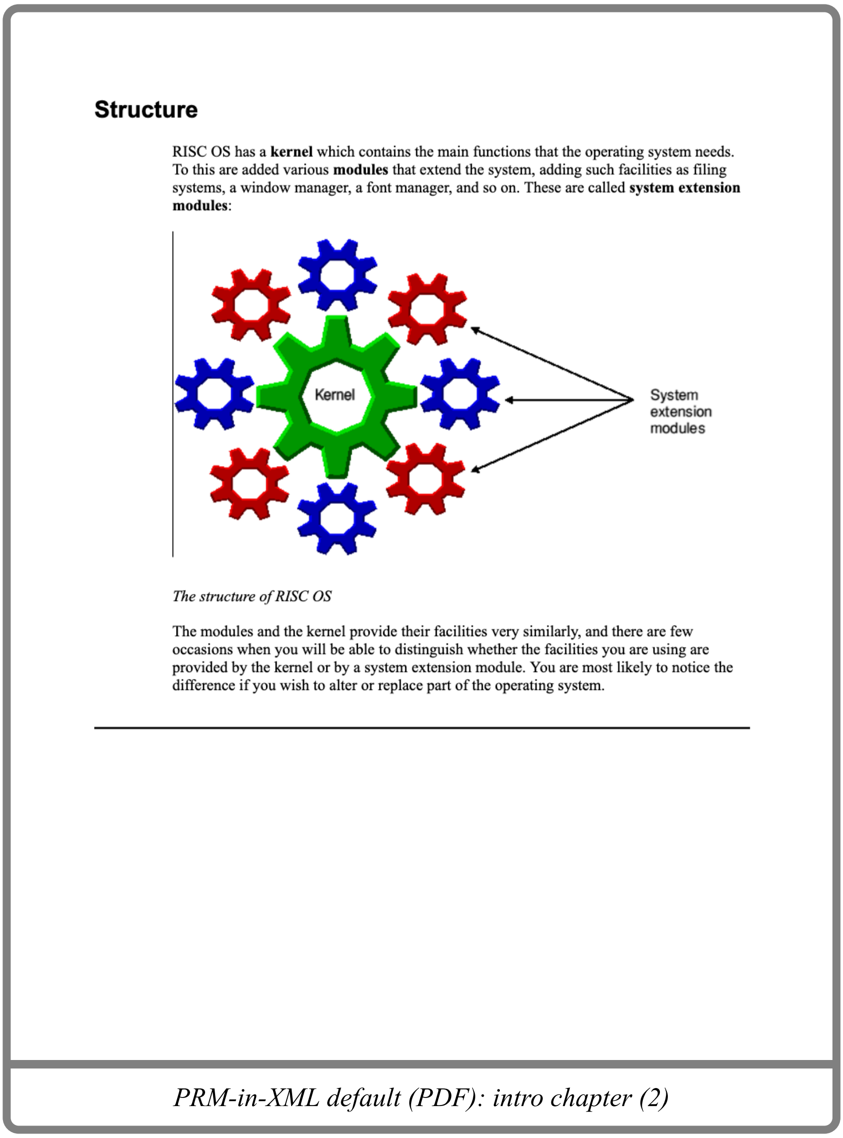
An introduction to RISC OS

Introduction

RISC OS is an operating system, originally written by Acorn Computers Ltd for the machines that they built. Since the buyout of Acorn via Element 14 by Pace Microsystems in 1998, the desktop development has been taken over by RISC OS Ltd. who license development from Pace.

Like any operating system, it is designed to provide the facilities that you, the programmer, need to control your computer and to get the most out of the programs you write for it.

PRM-in-XML default (PDF): intro chapter (1)



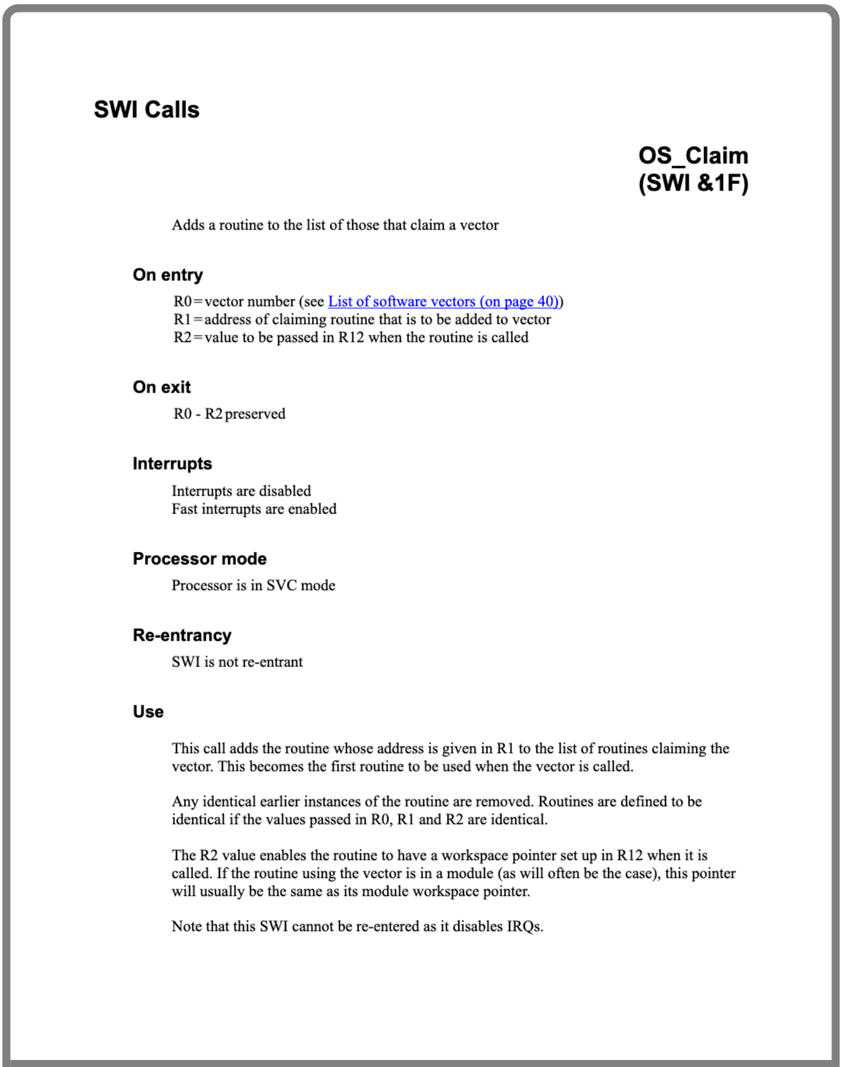
Structure

RISC OS has a **kernel** which contains the main functions that the operating system needs. To this are added various **modules** that extend the system, adding such facilities as filing systems, a window manager, a font manager, and so on. These are called **system extension modules**:

The structure of RISC OS

The modules and the kernel provide their facilities very similarly, and there are few occasions when you will be able to distinguish whether the facilities you are using are provided by the kernel or by a system extension module. You are most likely to notice the difference if you wish to alter or replace part of the operating system.

PRM-in-XML default (PDF): intro chapter (2)



SWI Calls

OS_Claim (SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number (see List of software vectors (on page 40))
R1 = address of claiming routine that is to be added to vector
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

Any identical earlier instances of the routine are removed. Routines are defined to be identical if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

Note that this SWI cannot be re-entered as it disables IRQs.

PRM-in-XML default (PDF): SWI definition

PRM-in-XML: 'prm' configuration

The 'prm' configuration of PRM-in-XML tries to mimic the printed form of the reference manuals much more closely. Whilst the default style is intended for general use the 'prm' style is intended for cases where the look of the RISC OS 3 PRMs is desired.

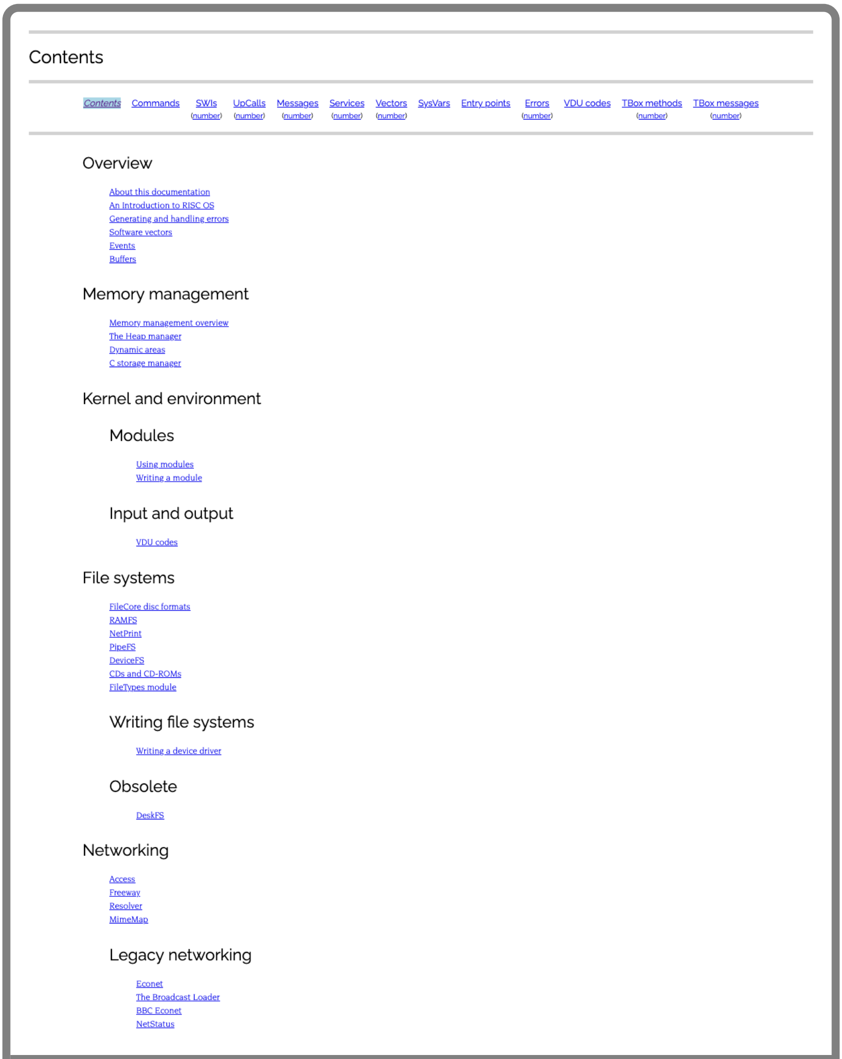
The variant setting used in this configuration was:

- 'prm': PRM style
- 'body-novarese': Use ITC Novarese font for the body.
- 'heading-raleway': Use Raleway as a reasonable approximation.
- 'large-bullets': Use the larger bullets.

Features of this configuration:

- Font is slightly smaller than the default.
- Horizontal grey bars used to divide chapters and sections.
- Alignment of headings is closer to the original style.
- Relative sizes of headings are closer to the original style.
- Bullets sit closer to the left edge, and are themselves larger, closer to the original.
- The PDF generated pages are much closer to the original style.
- Within the PDF, the chapter heading is indented to match the text, leaving space for a chapter number (which is not currently implementated).
- Within the PDF, he edge index is present, and included the name of the document group configured within the chapter.
- When printed, the API definitions describe each related SWI on a separate line to make it easier to see the page numbers.
- In the PDF, the page headers include the chapter and section names, and footers include the page number.

Example pages (HTML)



Contents

Contents	Commands	SWIs	UpCalls	Messages	Services	Vectors	SysVars	Entry points	Errors	VDU codes	TBox methods	TBox messages
		(number)	(number)	(number)	(number)	(number)			(number)		(number)	(number)

Overview

About this documentation
An Introduction to RISC OS
Generating and handling errors
Software vectors
Events
Buffers

Memory management

Memory management overview
The Heap manager
Dynamic areas
C storage manager

Kernel and environment

Modules

Using modules
Writing a module

Input and output

VDU codes

File systems

FileCore disc formats
RAMFS
NetPrint
PipeFS
DeviceFS
CDs and CD-ROMs
FileTypes module

Writing file systems

Writing a device driver

Obsolete

DeskFS

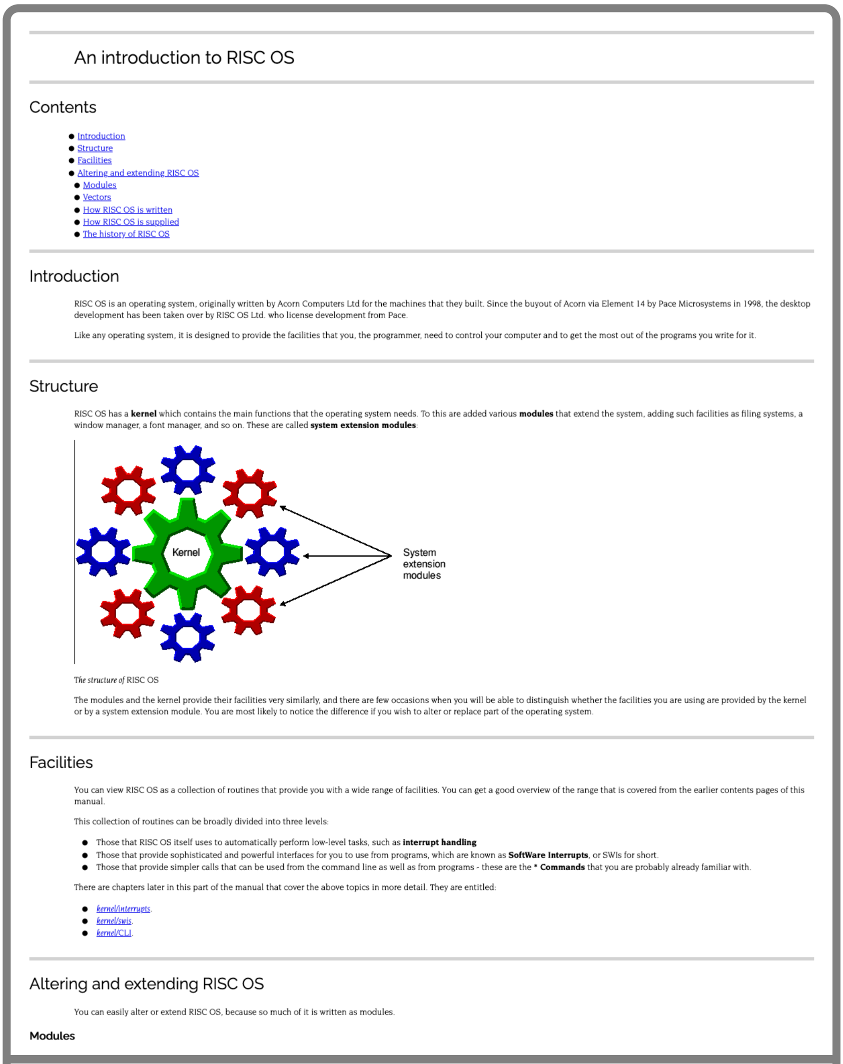
Networking

Access
Freeway
Resolver
MimeMap

Legacy networking

Econet
The Broadcast Loader
BBC Econet
NetStatus

PRM-in-XML 'prm': contents page



An introduction to RISC OS

Contents

Introduction

RISC OS is an operating system, originally written by Acorn Computers Ltd for the machines that they built. Since the buyout of Acorn via Element 14 by Pace Microsystems in 1998, the desktop development has been taken over by RISC OS Ltd. who license development from Pace.

Like any operating system, it is designed to provide the facilities that you, the programmer, need to control your computer and to get the most out of the programs you write for it.

Structure

RISC OS has a **kernel** which contains the main functions that the operating system needs. To this are added various **modules** that extend the system, adding such facilities as filing systems, a window manager, a font manager, and so on. These are called **system extension modules**:

The structure of RISC OS

The modules and the kernel provide their facilities very similarly, and there are few occasions when you will be able to distinguish whether the facilities you are using are provided by the kernel or by a system extension module. You are most likely to notice the difference if you wish to alter or replace part of the operating system.

Facilities

You can view RISC OS as a collection of routines that provide you with a wide range of facilities. You can get a good overview of the range that is covered from the earlier contents pages of this manual.

This collection of routines can be broadly divided into three levels:

- Those that RISC OS itself uses to automatically perform low-level tasks, such as **interrupt handling**
- Those that provide sophisticated and powerful interfaces for you to use from programs, which are known as **SoftWare Interrupts**, or SWIs for short.
- Those that provide simpler calls that can be used from the command line as well as from programs - these are the * **Commands** that you are probably already familiar with.

There are chapters later in this part of the manual that cover the above topics in more detail. They are entitled:

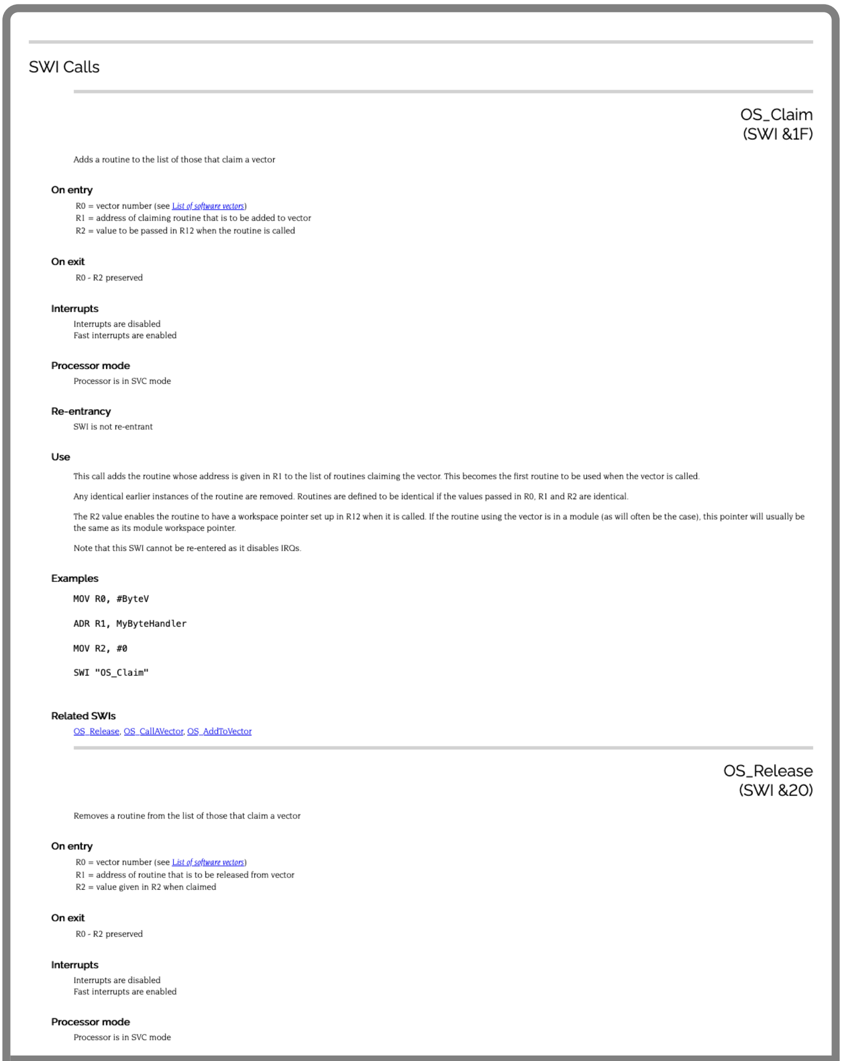
- *kernel/interrupts*.
- *kernel/swis*.
- *kernel/CLI*.

Altering and extending RISC OS

You can easily alter or extend RISC OS, because so much of it is written as modules.

Modules

PRM-in-XML 'prm': intro chapter



SWI Calls

OS_Claim (SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number (see List of software vectors)
R1 = address of claiming routine that is to be added to vector
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

Any identical earlier instances of the routine are removed. Routines are defined to be identical if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

Note that this SWI cannot be re-entered as it disables IRQs.

Examples

```
MOV R0, #ByteV

ADR R1, MyByteHandler

MOV R2, #0

SWI "OS_Claim"
```

Related SWIs

OS_Release, OS_CallAVector, OS_AddToVector

OS_Release (SWI &20)

Removes a routine from the list of those that claim a vector

On entry

R0 = vector number (see List of software vectors)
R1 = address of routine that is to be released from vector
R2 = value given in R2 when claimed

On exit

R0 - R2 preserved

Interrupts

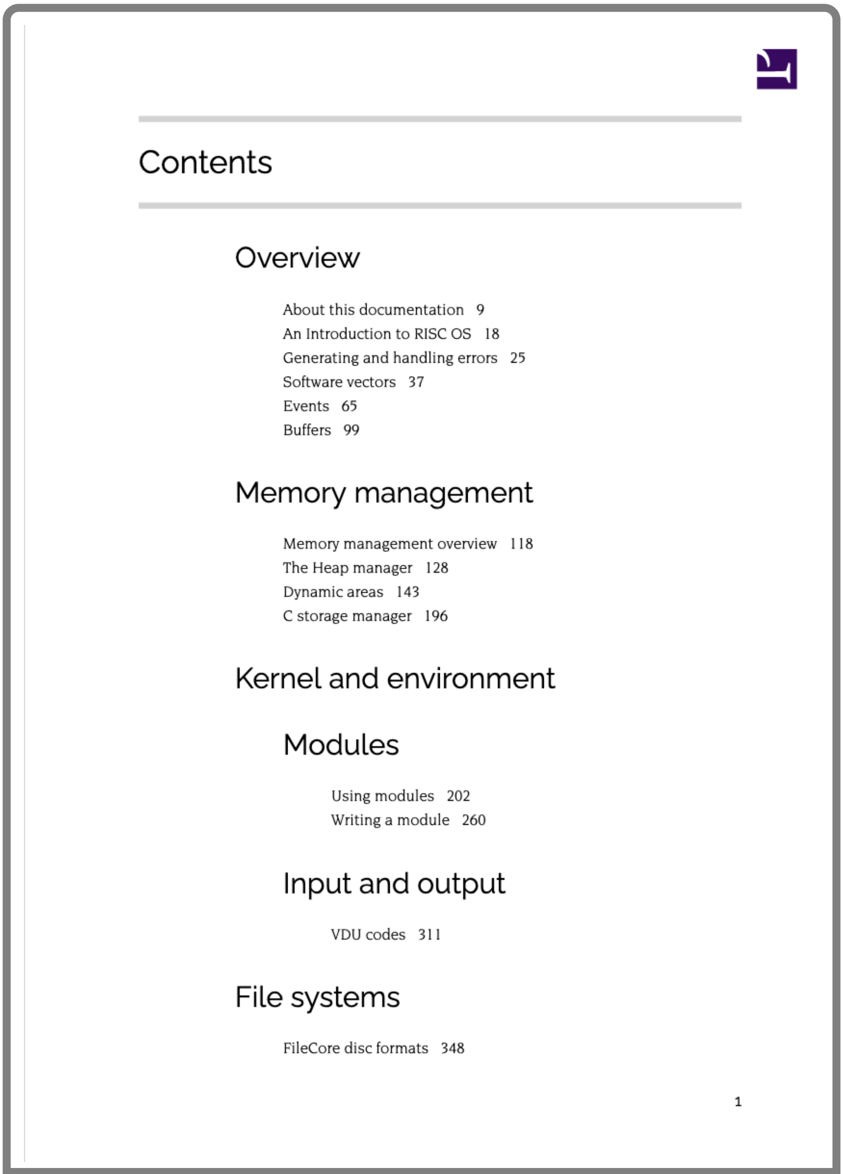
Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

PRM-in-XML 'prm': SWI definition

Example pages (PDF)



Contents

Overview

Memory management

Kernel and environment

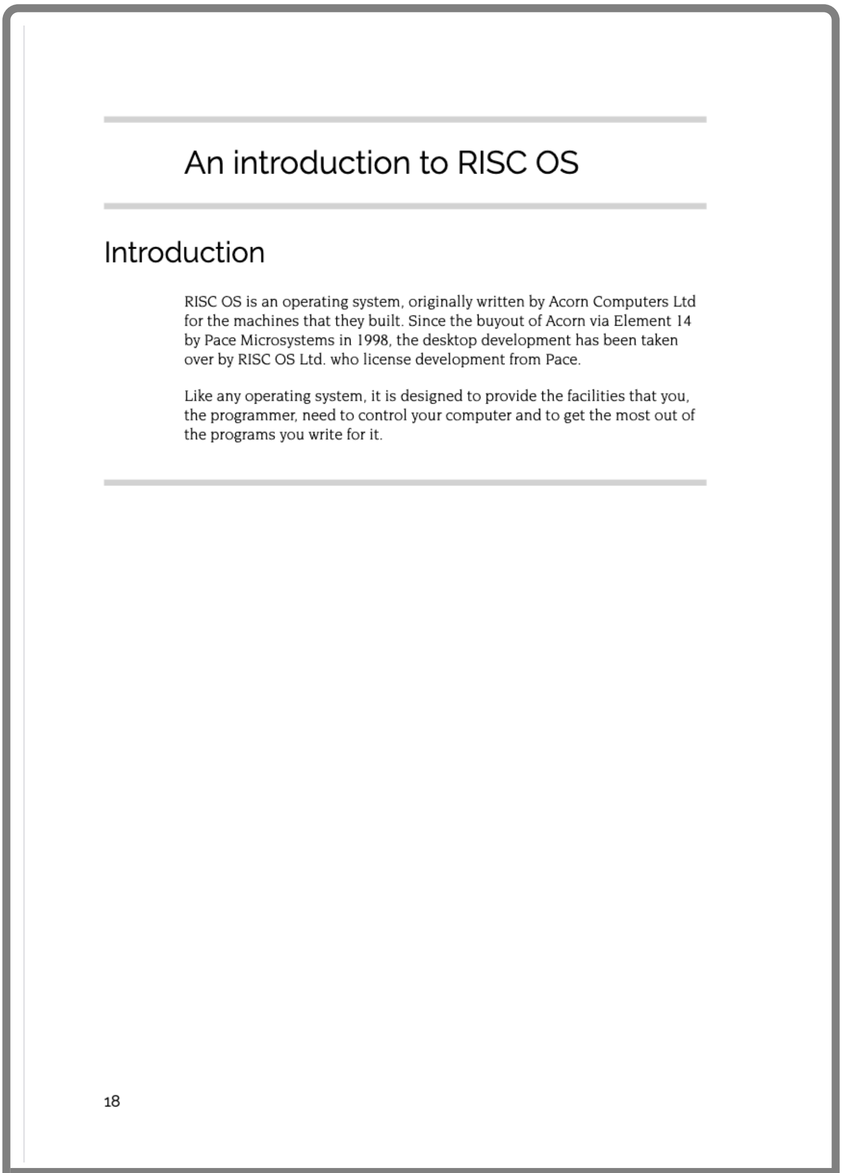
Modules

Input and output

File systems

1

PRM-in-XML 'prm' (PDF): contents page



An introduction to RISC OS

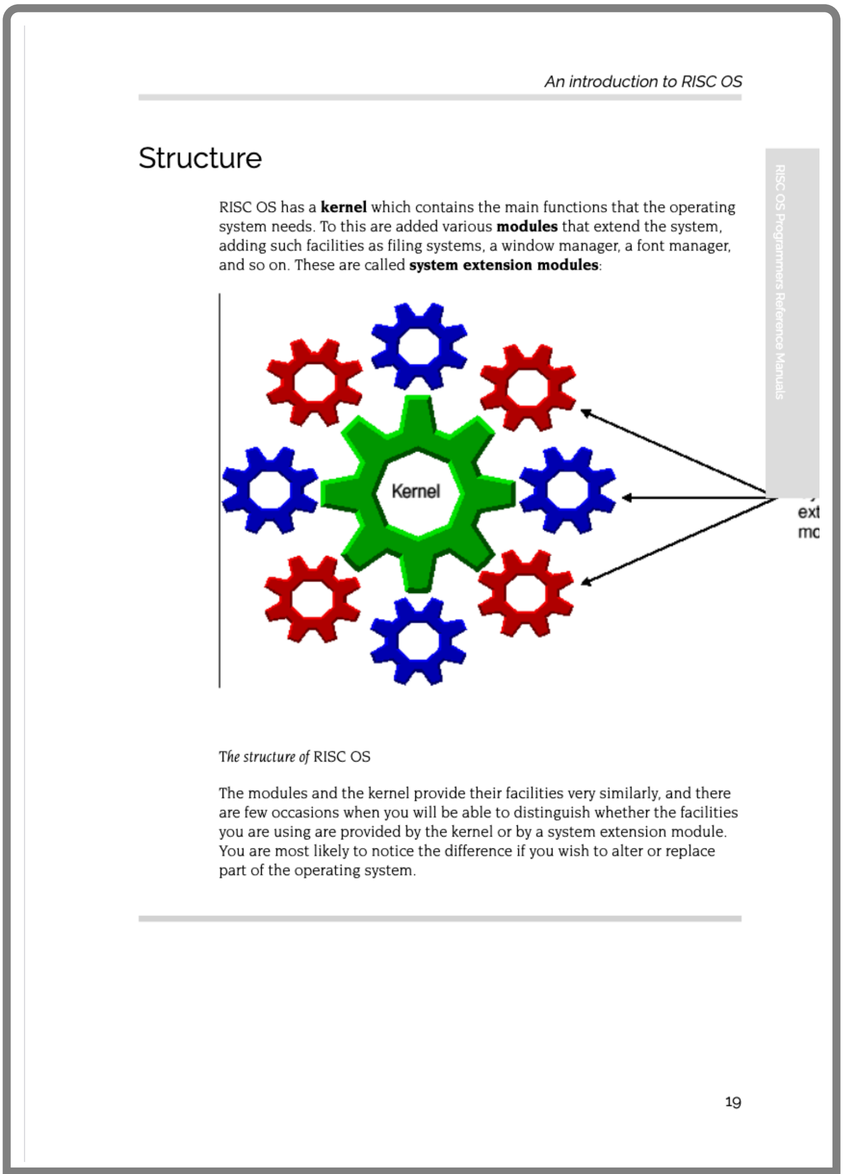
Introduction

RISC OS is an operating system, originally written by Acorn Computers Ltd for the machines that they built. Since the buyout of Acorn via Element 14 by Pace Microsystems in 1998, the desktop development has been taken over by RISC OS Ltd. who license development from Pace.

Like any operating system, it is designed to provide the facilities that you, the programmer, need to control your computer and to get the most out of the programs you write for it.

18

PRM-in-XML 'prm' (PDF): intro chapter (1)



An introduction to RISC OS

RISC OS Programmers Reference Manuals

Structure

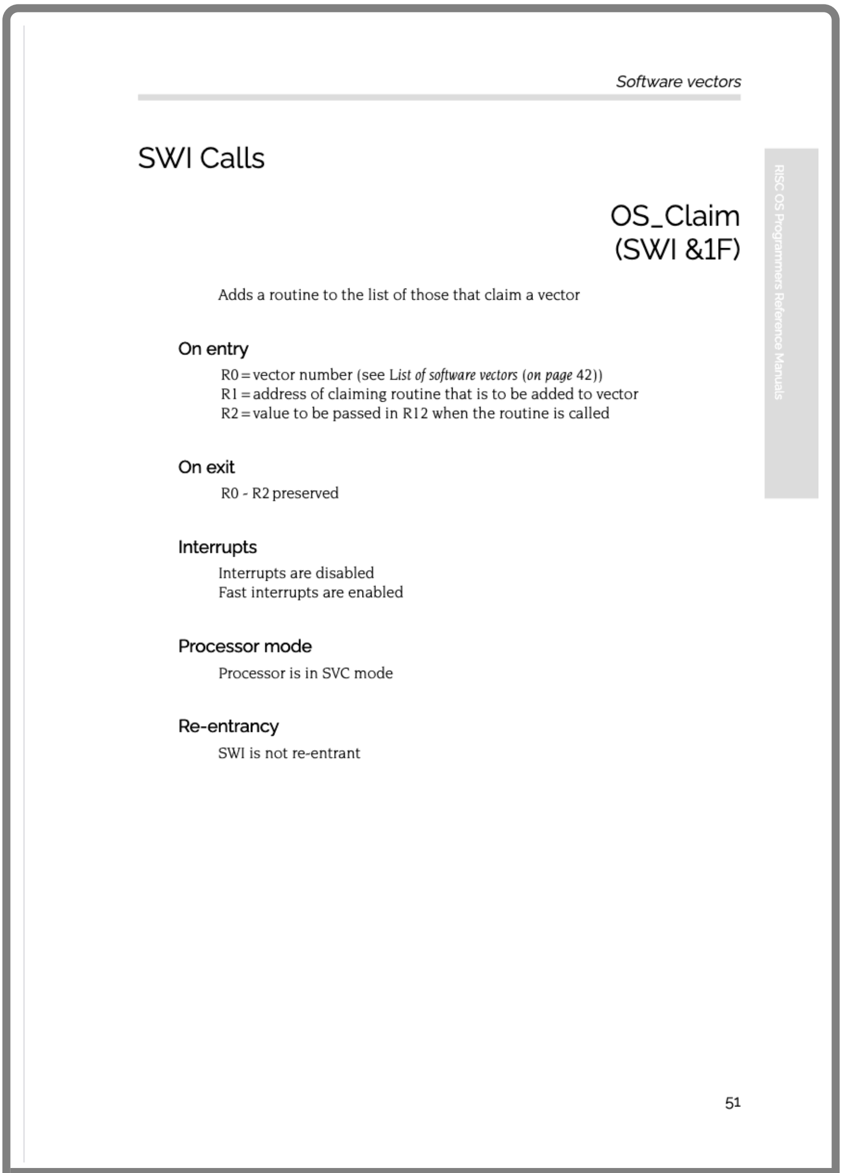
RISC OS has a **kernel** which contains the main functions that the operating system needs. To this are added various **modules** that extend the system, adding such facilities as filing systems, a window manager, a font manager, and so on. These are called **system extension modules**:

The structure of RISC OS

The modules and the kernel provide their facilities very similarly, and there are few occasions when you will be able to distinguish whether the facilities you are using are provided by the kernel or by a system extension module. You are most likely to notice the difference if you wish to alter or replace part of the operating system.

19

PRM-in-XML 'prm' (PDF): intro chapter (2)



Software vectors

RISC OS Programmers Reference Manuals

SWI Calls

OS_Claim (SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number (see *List of software vectors (on page* 42))
R1 = address of claiming routine that is to be added to vector
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

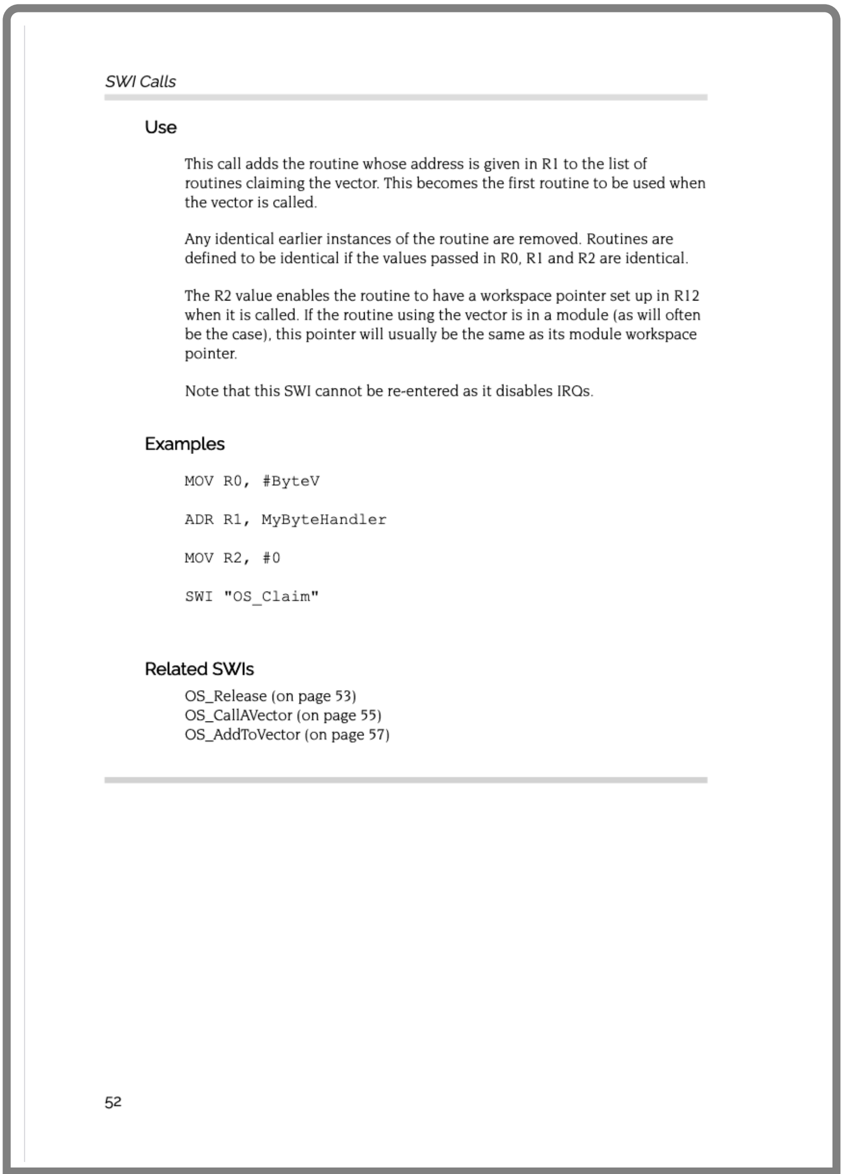
Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

51

PRM-in-XML 'prm' (PDF): SWI definition (1)



SWI Calls

Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

Any identical earlier instances of the routine are removed. Routines are defined to be identical if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

Note that this SWI cannot be re-entered as it disables IRQs.

Examples

```
MOV R0, #ByteV

ADR R1, MyByteHandler

MOV R2, #0

SWI "OS_Claim"
```

Related SWIs

OS_Release (on page 53)
OS_CallAVector (on page 55)
OS_AddToVector (on page 57)

52

PRM-in-XML 'prm' (PDF): SWI definition (2)

PRM-in-XML: 'prm-ro2' configuration

The 'prm-ro2' configuration of PRM-in-XML tries to mimic the RISC OS 2 PRMs. It is not a complete configuration, but it is highly effective at present..

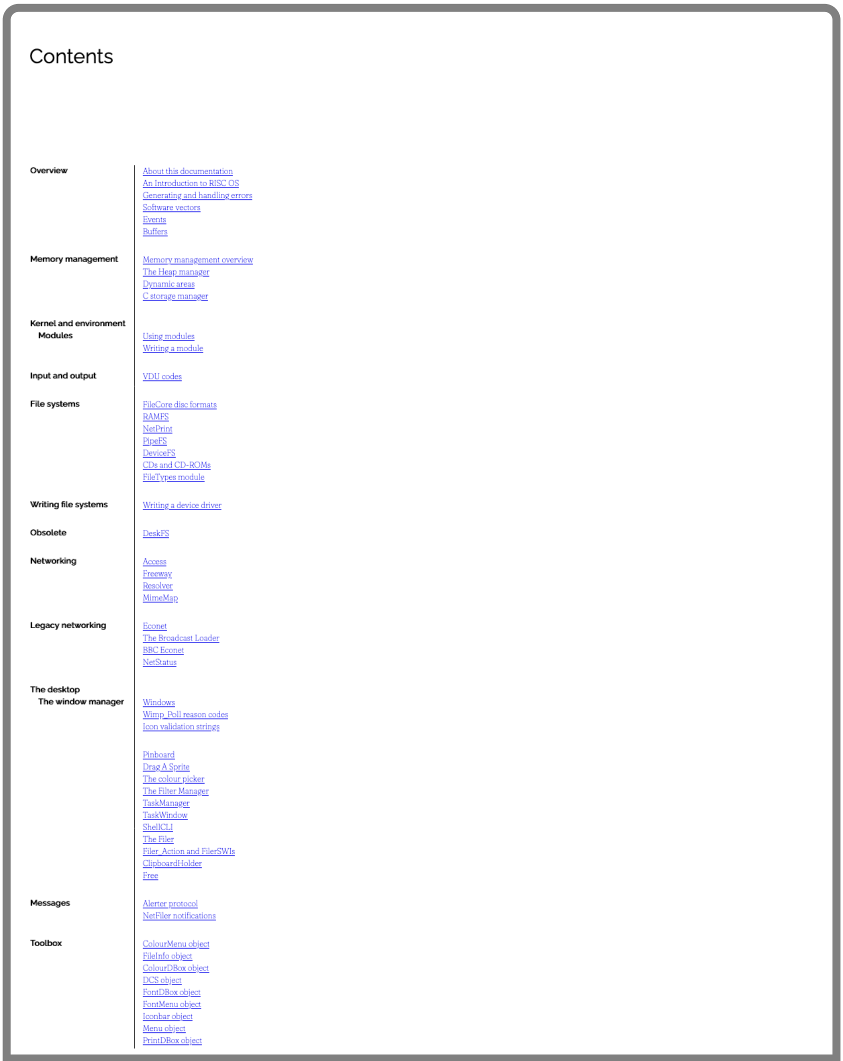
The variant setting used in this configuration was:

- 'prm': PRM style
- 'prm-ro2': RISC OS 2 style (layers on top of the base PRM style)
- 'body-fraunces': Use Fraunces font for the body.
- 'heading-raleway': Use Raleway as a reasonable approximation.
- 'large-bullets': Use the larger bullets.

Features of this configuration:

- Not really suitable for use on the desktop at the current time - really only for PDF.
- Separated headings and content style, like the RISC OS 2 PRMs is reproduced.
- Style is retained in both the contents and the chapter pages.
- Sometimes the layout of the headings on the left overlap when the sections are small. Some of this is avoided but it's not perfect.
- The HTML form has contents section that looks unsightly.
- PDF version lays out well.
- Page numbers are positioned appropriately in a vertical line away in the contents.
- API page looks very close to the original.

Example pages (HTML)



Contents

Overview

- About this documentation
- An Introduction to RISC OS
- Generating and handling errors
- Software vectors
- Events
- Buffers

Memory management

- Memory management overview
- The Heap manager
- Dynamic areas
- C storage manager

Kernel and environment

Modules

- Using modules
- Writing a module

Input and output

- VDU codes

File systems

- FileCore disc formats
- RAMFS
- NetPrint
- PipeFS
- DeviceFS
- CDs and CD-ROMs
- FileTypes module

Writing file systems

- Writing a device driver

Obsolete

- DeskFS

Networking

- Access
- Freeway
- Resolver
- MimeMap

Legacy networking

- Econet
- The Broadcast Loader
- BBC Econet
- NetStatus

The desktop

The window manager

- Windows
- Wimp_Poll reason codes
- Icon validation strings

- Pinboard
- Drag A Sprite
- The colour picker
- The Filter Manager
- TaskManager
- TaskWindow
- ShellCLI
- The Filer
- Filer_Action and FilerSWIs
- ClipboardHolder
- Free

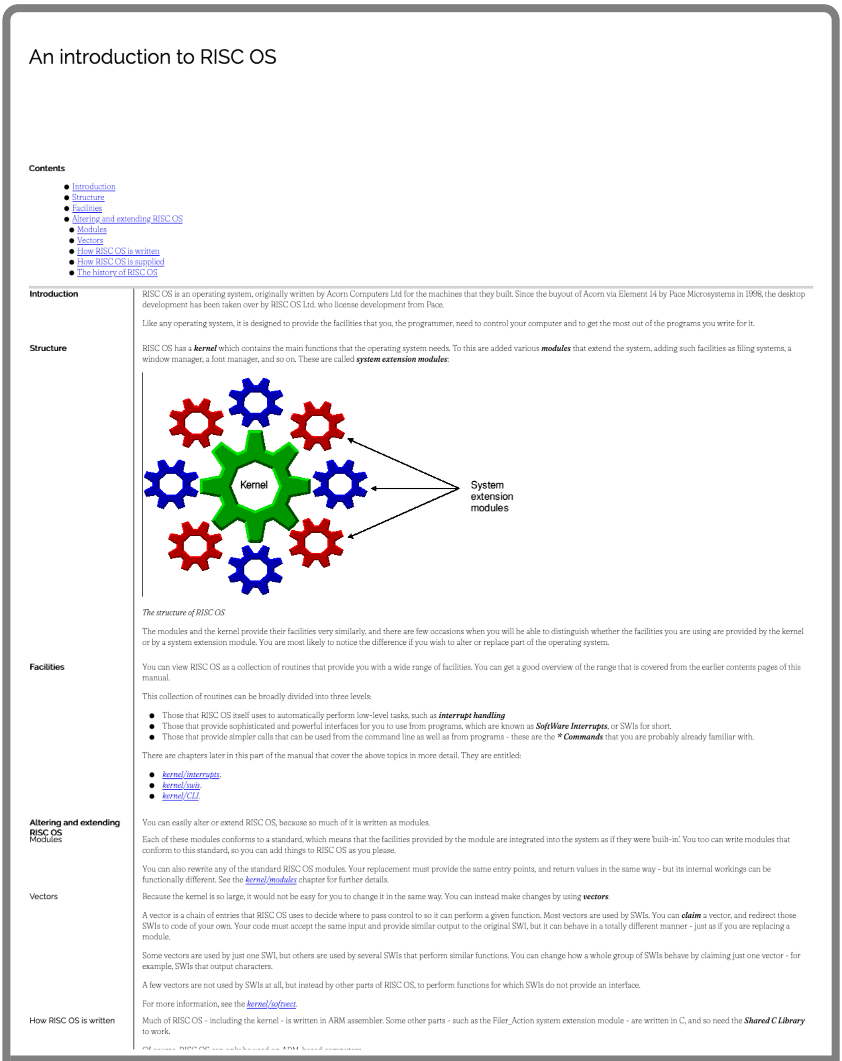
Messages

- Alerter protocol
- NetFiler notifications

Toolbox

- ColourMenu object
- FileInfo object
- ColourDBox object
- DCS object
- FontDBox object
- FontMenu object
- Iconbar object
- Menu object
- PrintDBox object

PRM-in-XML 'prmro2': contents page



An introduction to RISC OS

Contents

Introduction

RISC OS is an operating system, originally written by Acorn Computers Ltd for the machines that they built. Since the buyout of Acorn via Element 14 by Pace Microsystems in 1998, the desktop development has been taken over by RISC OS Ltd. who license development from Pace.

Like any operating system, it is designed to provide the facilities that you, the programmer, need to control your computer and to get the most out of the programs you write for it.

Structure

RISC OS has a ***kernel*** which contains the main functions that the operating system needs. To this are added various ***modules*** that extend the system, adding such facilities as filing systems, a window manager, a font manager, and so on. These are called ***system extension modules***:

The structure of RISC OS

The modules and the kernel provide their facilities very similarly, and there are few occasions when you will be able to distinguish whether the facilities you are using are provided by the kernel or by a system extension module. You are most likely to notice the difference if you wish to alter or replace part of the operating system.

Facilities

You can view RISC OS as a collection of routines that provide you with a wide range of facilities. You can get a good overview of the range that is covered from the earlier contents pages of this manual.

This collection of routines can be broadly divided into three levels:

- Those that RISC OS itself uses to automatically perform low-level tasks, such as ***interrupt handling***
- Those that provide sophisticated and powerful interfaces for you to use from programs, which are known as ***SoftWare Interrupts***, or SWIs for short.
- Those that provide simpler calls that can be used from the command line as well as from programs - these are the **** Commands*** that you are probably already familiar with.

There are chapters later in this part of the manual that cover the above topics in more detail. They are entitled:

- *kernel/interrupts*.
- *kernel/swis*.
- *kernel/CLI*.

Altering and extending RISC OS

Modules

You can easily alter or extend RISC OS, because so much of it is written as modules.

Each of these modules conforms to a standard, which means that the facilities provided by the module are integrated into the system as if they were 'built-in'. You too can write modules that conform to this standard, so you can add things to RISC OS as you please.

You can also rewrite any of the standard RISC OS modules. Your replacement must provide the same entry points, and return values in the same way - but its internal workings can be functionally different. See the *kernel/modules* chapter for further details.

Vectors

Because the kernel is so large, it would not be easy for you to change it in the same way. You can instead make changes by using ***vectors***.

A vector is a chain of entries that RISC OS uses to decide where to pass control to so it can perform a given function. Most vectors are used by SWIs. You can ***claim*** a vector, and redirect those SWIs to code of your own. Your code must accept the same input and provide similar output to the original SWI, but it can behave in a totally different manner - just as if you are replacing a module.

Some vectors are used by just one SWI, but others are used by several SWIs that perform similar functions. You can change how a whole group of SWIs behave by claiming just one vector - for example, SWIs that output characters.

A few vectors are not used by SWIs at all, but instead by other parts of RISC OS, to perform functions for which SWIs do not provide an interface.

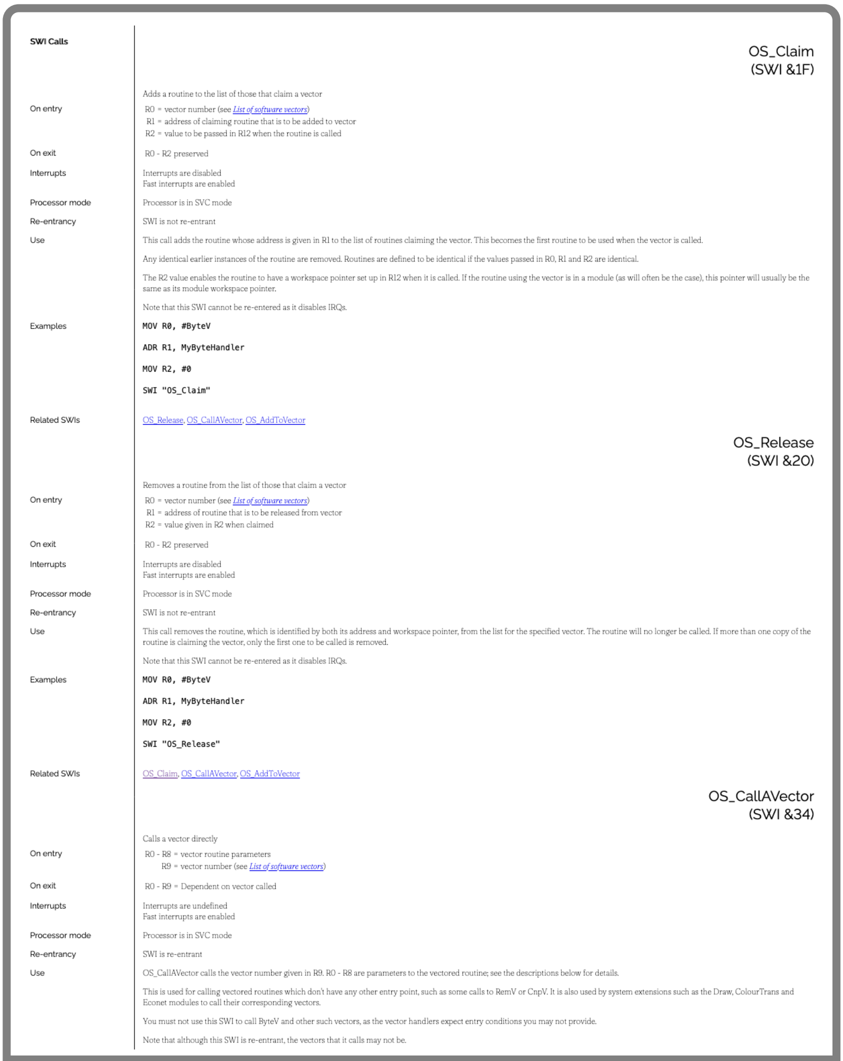
For more information, see the *kernel/softvect*.

How RISC OS is written

Much of RISC OS - including the kernel - is written in ARM assembler. Some other parts - such as the Filer_Action system extension module - are written in C, and so need the ***Shared C Library*** to work.

Of course, RISC OS can only be used on ARM-based computers

PRM-in-XML 'prmro2': intro chapter



SWI Calls

OS_Claim
(SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number (see *List of software vectors*)
R1 = address of claiming routine that is to be added to vector
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

Any identical earlier instances of the routine are removed. Routines are defined to be identical if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

Note that this SWI cannot be re-entered as it disables IRQs.

Examples

```
MOV R0, #ByteV

ADR R1, MyByteHandler

MOV R2, #0

SWI "OS_Claim"
```

Related SWIs

OS_Release, OS_CallAVector, OS_AddToVector

OS_Release
(SWI &20)

Removes a routine from the list of those that claim a vector

On entry

R0 = vector number (see *List of software vectors*)
R1 = address of routine that is to be released from vector
R2 = value given in R2 when claimed

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This call removes the routine, which is identified by both its address and workspace pointer, from the list for the specified vector. The routine will no longer be called. If more than one copy of the routine is claiming the vector, only the first one to be called is removed.

Note that this SWI cannot be re-entered as it disables IRQs.

Examples

```
MOV R0, #ByteV

ADR R1, MyByteHandler

MOV R2, #0

SWI "OS_Release"
```

Related SWIs

OS_Claim, OS_CallAVector, OS_AddToVector

OS_CallAVector
(SWI &34)

Calls a vector directly

On entry

R0 - R8 = vector routine parameters
R9 = vector number (see *List of software vectors*)

On exit

R0 - R9 = Dependent on vector called

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is re-entrant

Use

OS_CallAVector calls the vector number given in R9. R0 - R8 are parameters to the vectored routine; see the descriptions below for details.

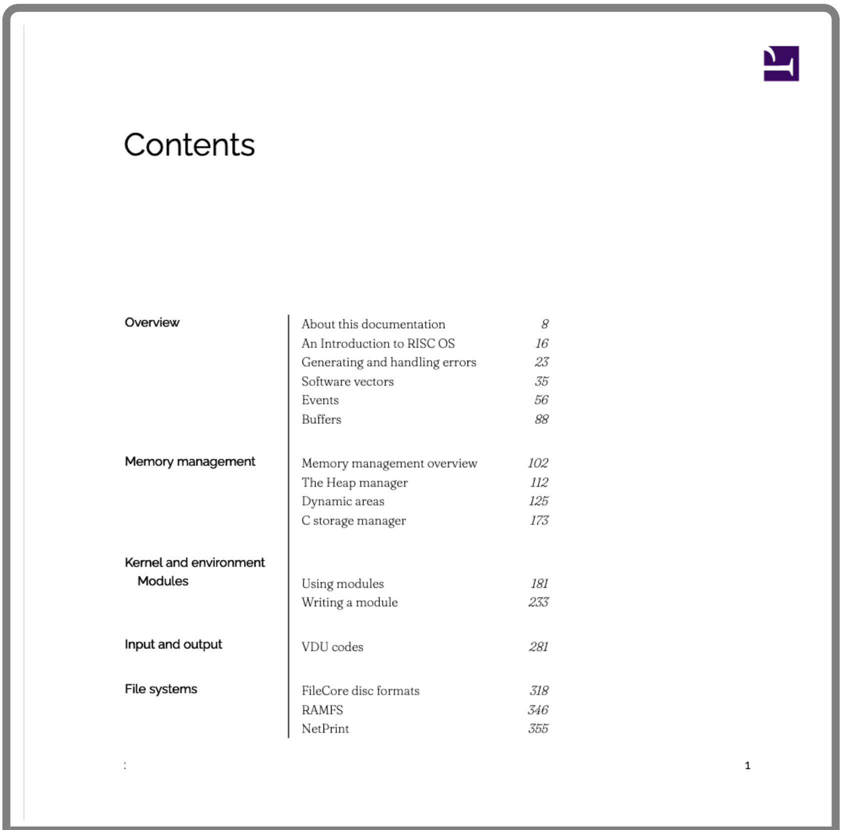
This is used for calling vectored routines which don't have any other entry point, such as some calls to RemV or CnpV. It is also used by system extensions such as the Draw, ColourTrans and Econet modules to call their corresponding vectors.

You must not use this SWI to call ByteV and other such vectors, as the vector handlers expect entry conditions you may not provide.

Note that although this SWI is re-entrant, the vectors that it calls may not be.

PRM-in-XML 'prmro2': SWI definition

Example pages (PDF)

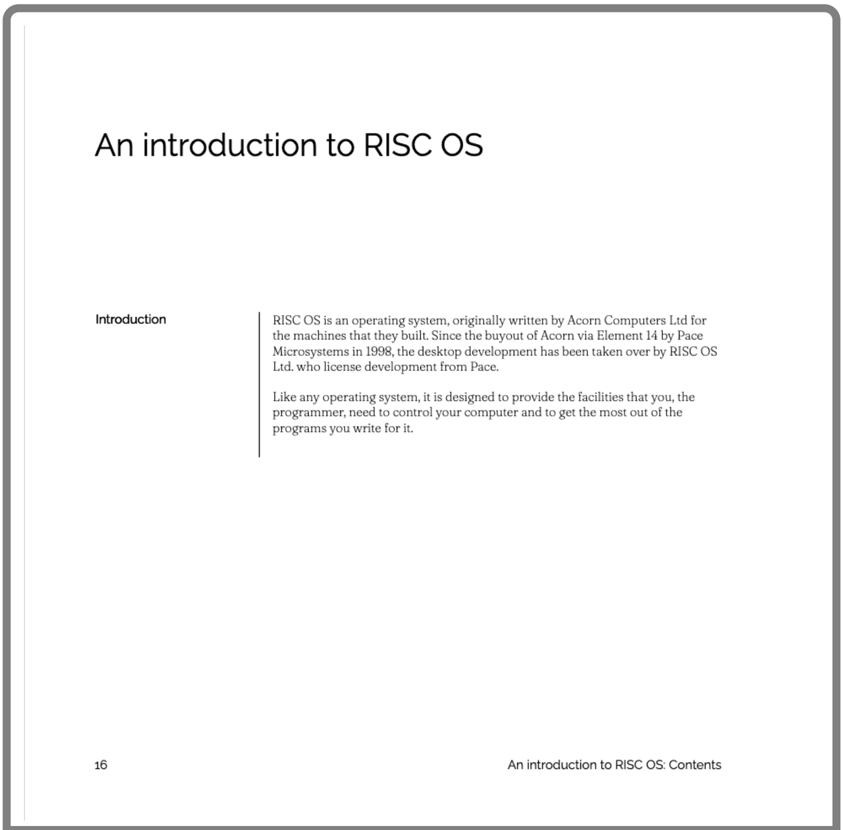


Contents

Overview	About this documentation	*8*
	An Introduction to RISC OS	*16*
	Generating and handling errors	*23*
	Software vectors	*35*
	Events	*56*
	Buffers	*88*
Memory management	Memory management overview	*102*
	The Heap manager	*112*
	Dynamic areas	*125*
	C storage manager	*173*
Kernel and environment		
Modules	Using modules	*181*
	Writing a module	*233*
Input and output	VDU codes	*281*
File systems	FileCore disc formats	*318*
	RAMFS	*346*
	NetPrint	*355*

: 1

PRM-in-XML 'prmro2' (PDF): contents page



An introduction to RISC OS

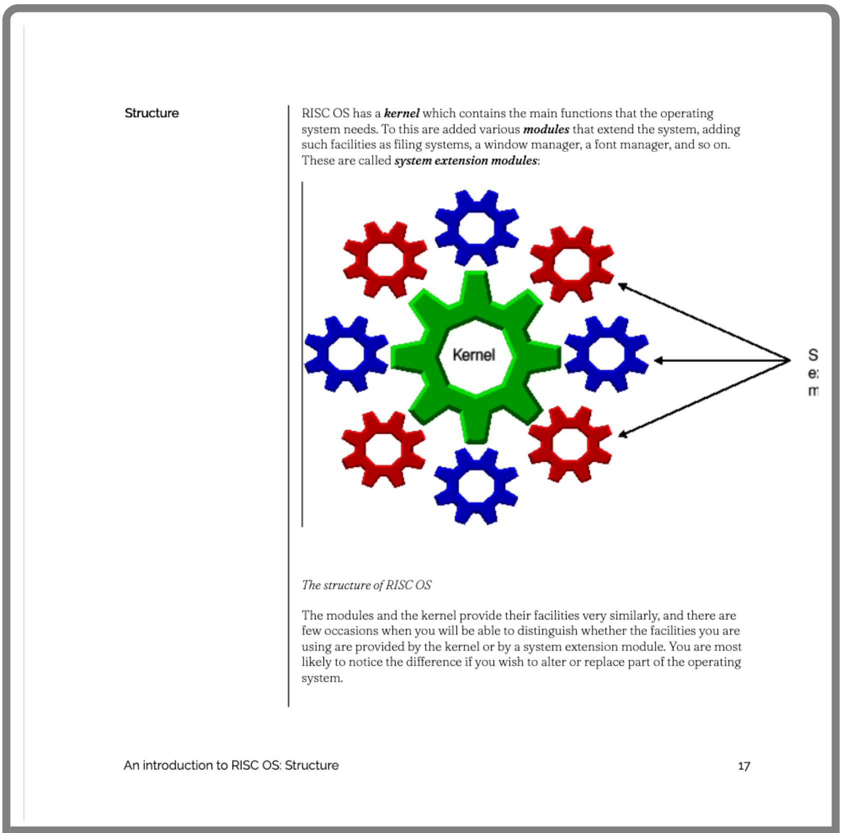
Introduction

RISC OS is an operating system, originally written by Acorn Computers Ltd for the machines that they built. Since the buyout of Acorn via Element 14 by Pace Microsystems in 1998, the desktop development has been taken over by RISC OS Ltd. who license development from Pace.

Like any operating system, it is designed to provide the facilities that you, the programmer, need to control your computer and to get the most out of the programs you write for it.

16 An introduction to RISC OS: Contents

PRM-in-XML 'prmro2' (PDF): intro chapter (1)



Structure

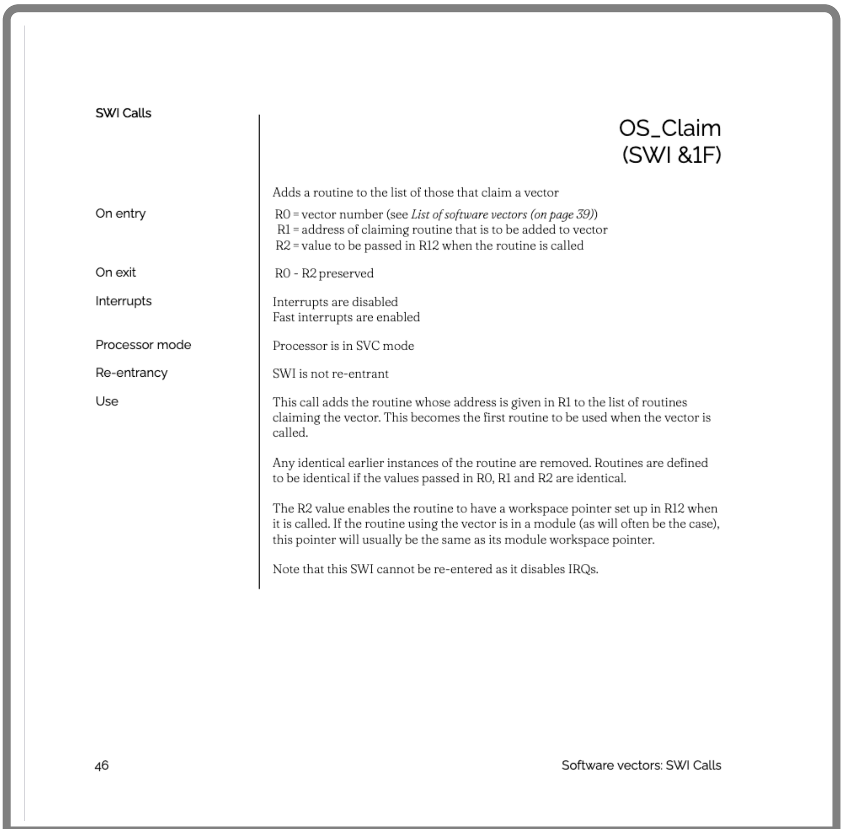
RISC OS has a ***kernel*** which contains the main functions that the operating system needs. To this are added various ***modules*** that extend the system, adding such facilities as filing systems, a window manager, a font manager, and so on. These are called ***system extension modules***:

The structure of RISC OS

The modules and the kernel provide their facilities very similarly, and there are few occasions when you will be able to distinguish whether the facilities you are using are provided by the kernel or by a system extension module. You are most likely to notice the difference if you wish to alter or replace part of the operating system.

An introduction to RISC OS: Structure 17

PRM-in-XML 'prmro2' (PDF): intro chapter (2)



SWI Calls

OS_Claim (SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number (see *List of software vectors (on page 39)*)
R1 = address of claiming routine that is to be added to vector
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

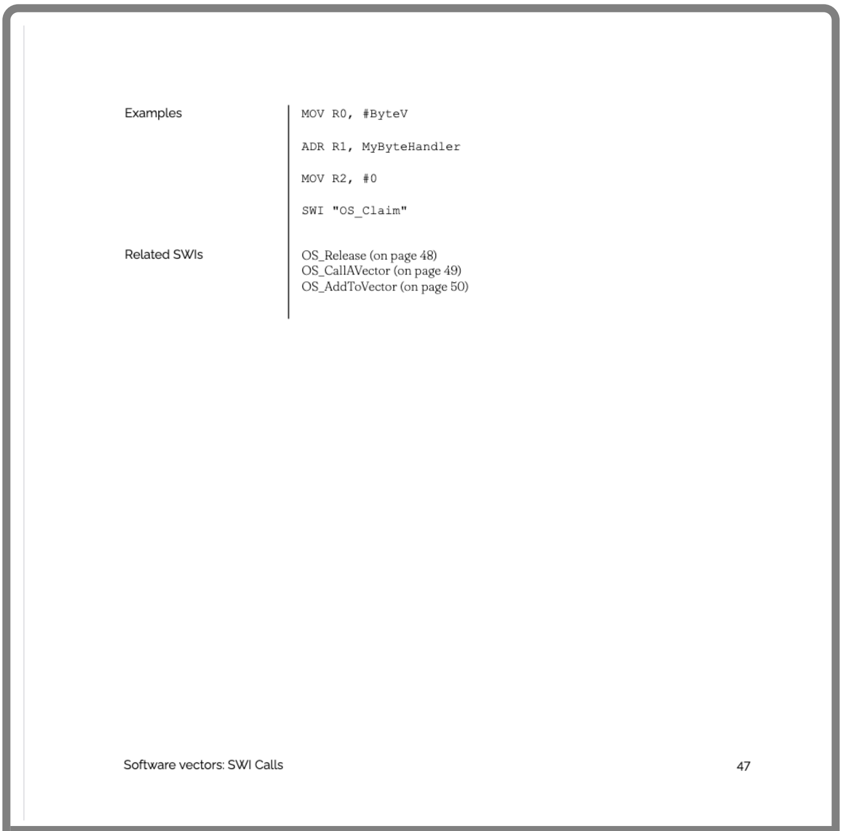
Any identical earlier instances of the routine are removed. Routines are defined to be identical if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

Note that this SWI cannot be re-entered as it disables IRQs.

46 Software vectors: SWI Calls

PRM-in-XML 'prmro2' (PDF): SWI definition (1)



Examples

```
MOV R0, #ByteV

ADR R1, MyByteHandler

MOV R2, #0

SWI "OS_Claim"
```

Related SWIs

OS_Release (on page 48)
OS_CallAVector (on page 49)
OS_AddToVector (on page 50)

Software vectors: SWI Calls 47

PRM-in-XML 'prmro2' (PDF): SWI definition (2)

PRM-in-XML: 'c release 4' configuration

The 'C release 4' configuration of PRM-in-XML adds a few small things that match that manual. It is not a complete configuration, but it demonstrates the ability to vary the layout.

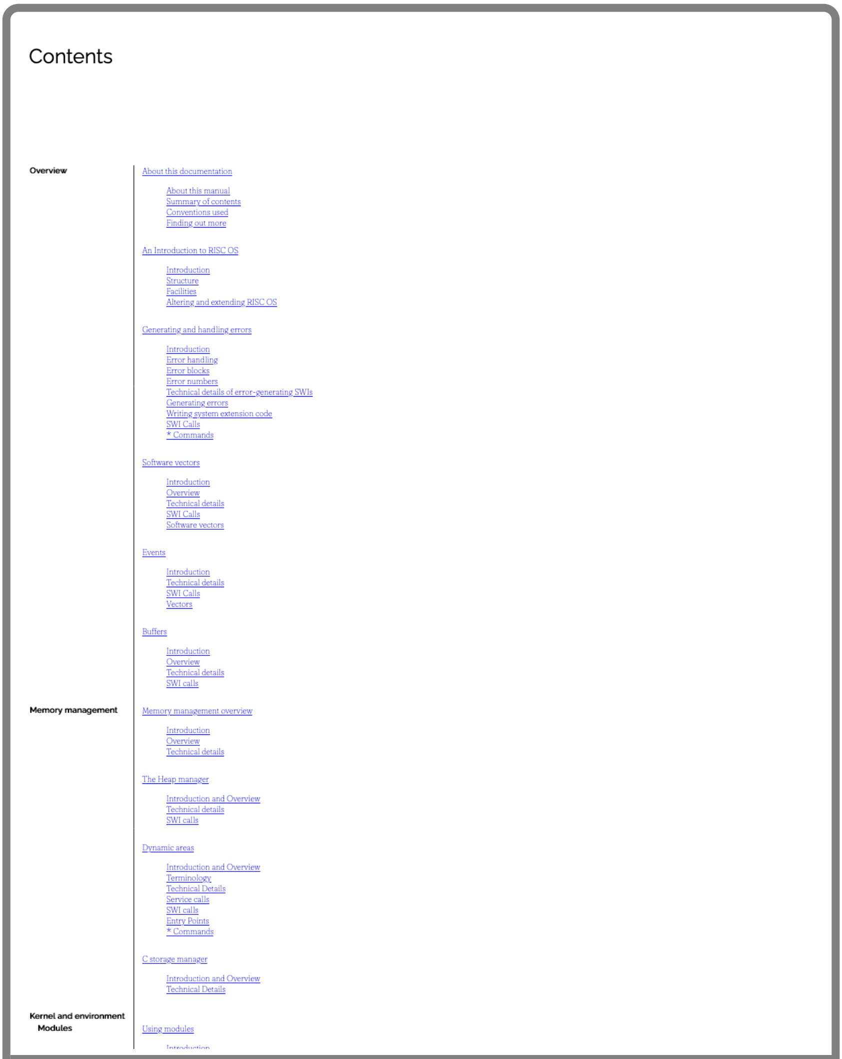
The variant setting used in this configuration was:

- 'prm': PRM style
- 'prm-ro2': RISC OS 2 style (layers on top of the base PRM style)
- 'body-fraunces': Use Fraunces font for the body.
- 'heading-raleway': Use Raleway as a reasonable approximation.
- 'large-bullets': Use the larger bullets.
- 'drop-character': Initial drop character on the first character of the first paragraph.
- Additionally the setting to include the sections in the contents was enabled in the contents generation, for a depth of 1 level.

Features of this configuration:

- Exhibits the same flaws as the PRM-ro2 version; there's only a few changes.
- Sections are expanded and linked in the contents page.
- Drop characters are present on the chapter pages.
- In the PDF, the links on the contents page are indented further for the sections.

Example pages (HTML)



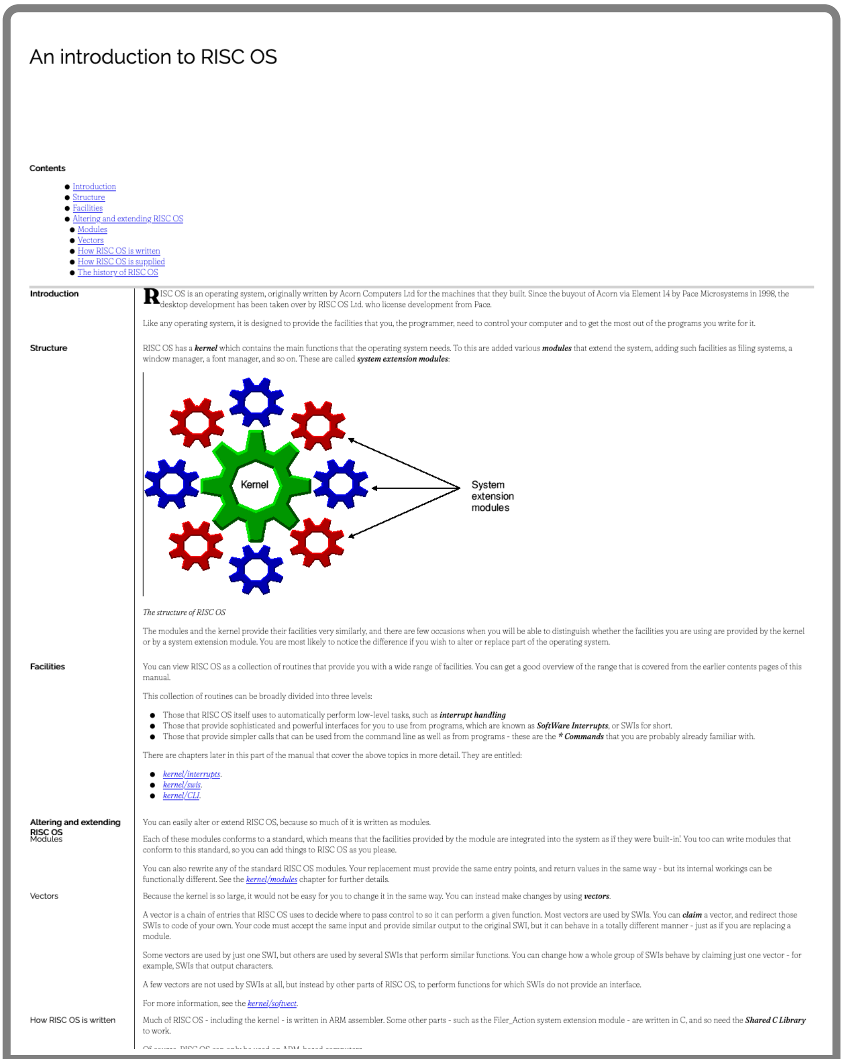
Contents

Overview

Memory management

Kernel and environment
Modules

PRM-in-XML 'C release 4': contents page



An introduction to RISC OS

Contents

Introduction

RISC OS is an operating system, originally written by Acorn Computers Ltd for the machines that they built. Since the buyout of Acorn via Element 14 by Pace Microsystems in 1998, the desktop development has been taken over by RISC OS Ltd. who license development from Pace.

Like any operating system, it is designed to provide the facilities that you, the programmer, need to control your computer and to get the most out of the programs you write for it.

Structure

RISC OS has a ***kernel*** which contains the main functions that the operating system needs. To this are added various ***modules*** that extend the system, adding such facilities as filing systems, a window manager, a font manager, and so on. These are called ***system extension modules***:

The structure of RISC OS

The modules and the kernel provide their facilities very similarly, and there are few occasions when you will be able to distinguish whether the facilities you are using are provided by the kernel or by a system extension module. You are most likely to notice the difference if you wish to alter or replace part of the operating system.

Facilities

You can view RISC OS as a collection of routines that provide you with a wide range of facilities. You can get a good overview of the range that is covered from the earlier contents pages of this manual.

This collection of routines can be broadly divided into three levels:

- Those that RISC OS itself uses to automatically perform low-level tasks, such as ***interrupt handling***
- Those that provide sophisticated and powerful interfaces for you to use from programs, which are known as ***SoftWare Interrupts***, or SWIs for short.
- Those that provide simpler calls that can be used from the command line as well as from programs - these are the **** Commands*** that you are probably already familiar with.

There are chapters later in this part of the manual that cover the above topics in more detail. They are entitled:

- *kernel/interrupts*.
- *kernel/swis*.
- *kernel/CLI*.

Altering and extending RISC OS

You can easily alter or extend RISC OS, because so much of it is written as modules.

Modules

Each of these modules conforms to a standard, which means that the facilities provided by the module are integrated into the system as if they were 'built-in'. You too can write modules that conform to this standard, so you can add things to RISC OS as you please.

You can also rewrite any of the standard RISC OS modules. Your replacement must provide the same entry points, and return values in the same way - but its internal workings can be functionally different. See the *kernel/modules* chapter for further details.

Vectors

Because the kernel is so large, it would not be easy for you to change it in the same way. You can instead make changes by using ***vectors***.

A vector is a chain of entries that RISC OS uses to decide where to pass control to so it can perform a given function. Most vectors are used by SWIs. You can ***claim*** a vector, and redirect those SWIs to code of your own. Your code must accept the same input and provide similar output to the original SWI, but it can behave in a totally different manner - just as if you are replacing a module.

Some vectors are used by just one SWI, but others are used by several SWIs that perform similar functions. You can change how a whole group of SWIs behave by claiming just one vector - for example, SWIs that output characters.

A few vectors are not used by SWIs at all, but instead by other parts of RISC OS, to perform functions for which SWIs do not provide an interface.

For more information, see the *kernel/softvect*.

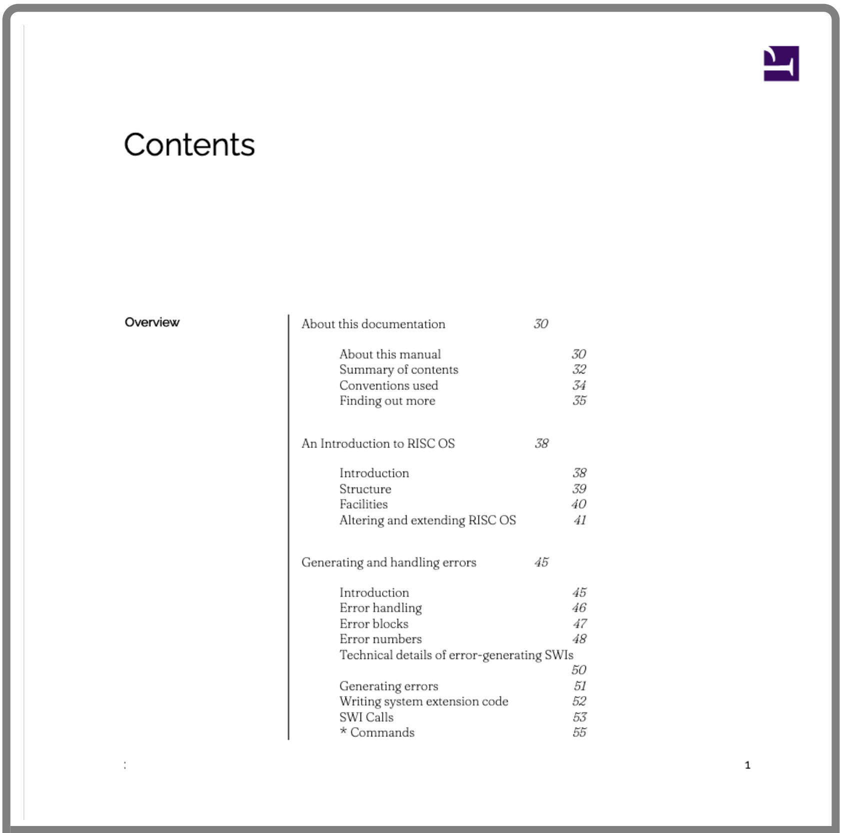
How RISC OS is written

Much of RISC OS - including the kernel - is written in ARM assembler. Some other parts - such as the Filer_Action system extension module - are written in C, and so need the ***Shared C Library*** to work.

Of course, RISC OS can only be used on ARM-based computers

PRM-in-XML 'C release 4': intro chapter

Example pages (PDF)

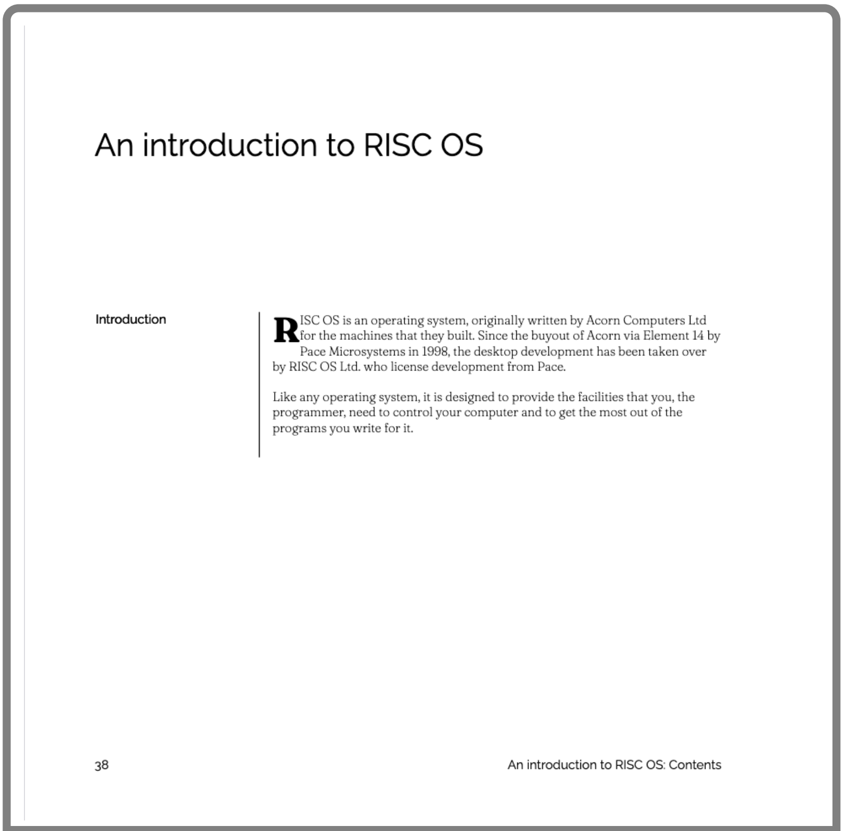


Contents

Overview

: 1

PRM-in-XML 'C release 4' (PDF): contents page



An introduction to RISC OS

Introduction

RISC OS is an operating system, originally written by Acorn Computers Ltd for the machines that they built. Since the buyout of Acorn via Element 14 by Pace Microsystems in 1998, the desktop development has been taken over by RISC OS Ltd. who license development from Pace.

Like any operating system, it is designed to provide the facilities that you, the programmer, need to control your computer and to get the most out of the programs you write for it.

38 An introduction to RISC OS: Contents

PRM-in-XML 'C release 4' (PDF): intro chapter

PRM-in-XML: 'acornfs' configuration

The 'Acorn functional spec' configuration of PRM-in-XML tries to mimic the style of the functional specifications that Acorn produced in the later years It is not a complete configuration, but it demonstrates the ability to vary the layout.

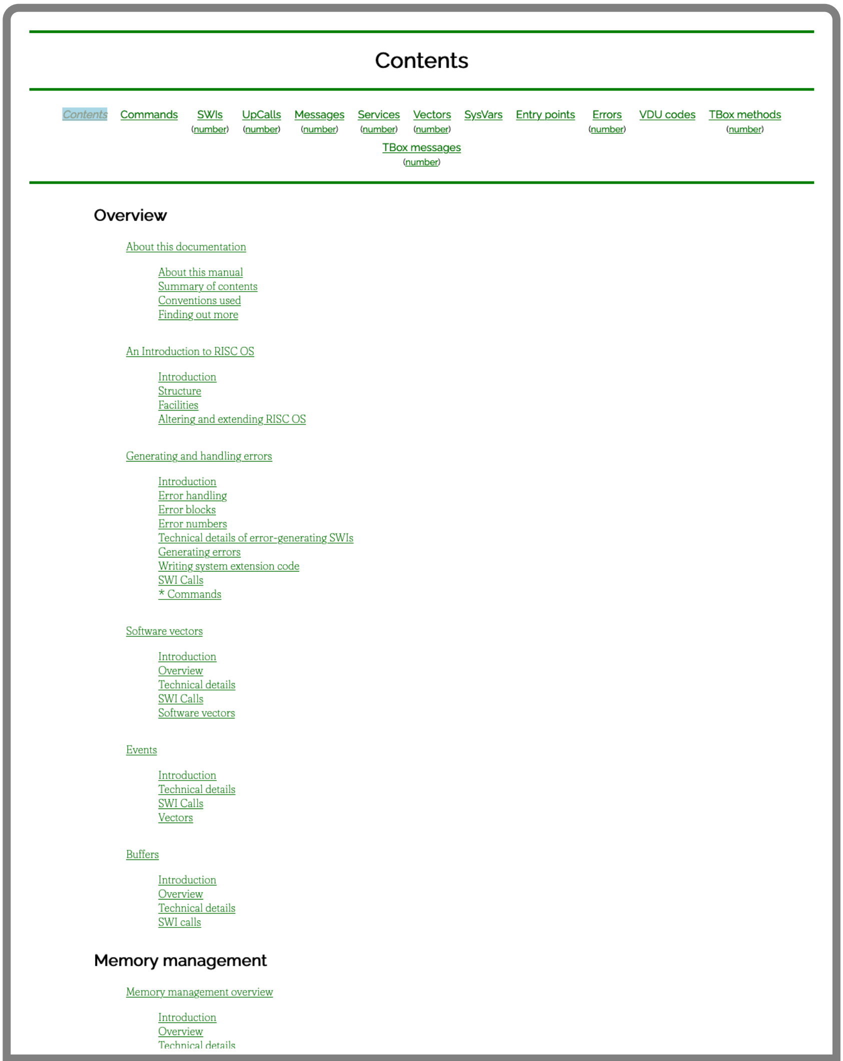
The variant setting used in this configuration was:

- 'acornfs': Acorn Functional Specification variant
- 'body-fraunces': Use Fraunces font for the body.
- 'heading-raleway': Use Raleway as a reasonable approximation.
- 'large-bullets': Use the larger bullets.

Features of this configuration:

- Green dividing lines instead of grey.
- Green link text.
- Chapter and section headings are centred.
- All text is left aligned, with no indentation.
- Subsection, subsubsection, category are left aligned, with small indentations to show nesting.
- API page has a similar style to the PRMs, but is less indented.
- The API name is given a grey border.
- API description is right aligned and italic.

Example pages (HTML)



Contents

Contents Commands SWIs (number) UpCalls (number) Messages (number) Services (number) Vectors (number) SysVars Entry points Errors (number) VDU codes TBox methods (number) TBox messages (number)

Overview

About this documentation

- About this manual
- Summary of contents
- Conventions used
- Finding out more

An Introduction to RISC OS

- Introduction
- Structure
- Facilities
- Altering and extending RISC OS

Generating and handling errors

- Introduction
- Error handling
- Error blocks
- Error numbers
- Technical details of error-generating SWIs
- Generating errors
- Writing system extension code
- SWI Calls
- * Commands

Software vectors

- Introduction
- Overview
- Technical details
- SWI Calls
- Software vectors

Events

- Introduction
- Technical details
- SWI Calls
- Vectors

Buffers

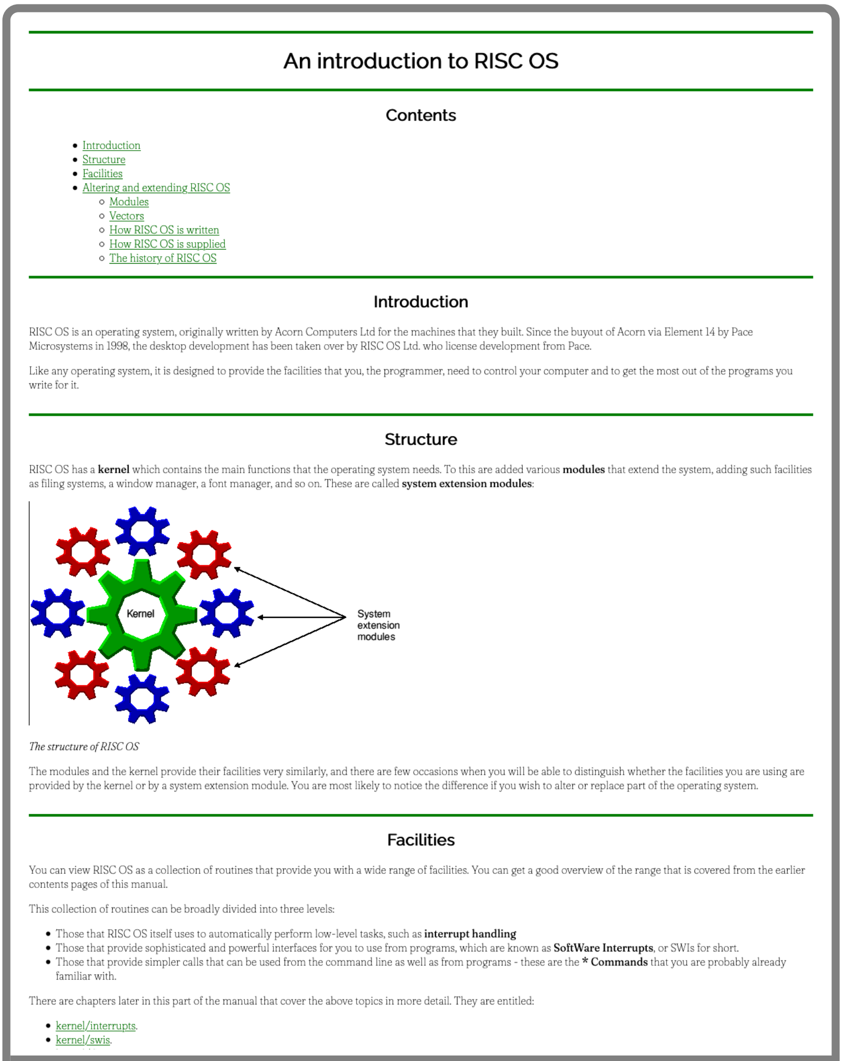
- Introduction
- Overview
- Technical details
- SWI calls

Memory management

Memory management overview

- Introduction
- Overview
- Technical details

PRM-in-XML 'acornfs': contents page



An introduction to RISC OS

Contents

Introduction

RISC OS is an operating system, originally written by Acorn Computers Ltd for the machines that they built. Since the buyout of Acorn via Element 14 by Pace Microsystems in 1998, the desktop development has been taken over by RISC OS Ltd. who license development from Pace.

Like any operating system, it is designed to provide the facilities that you, the programmer, need to control your computer and to get the most out of the programs you write for it.

Structure

RISC OS has a **kernel** which contains the main functions that the operating system needs. To this are added various **modules** that extend the system, adding such facilities as filing systems, a window manager, a font manager, and so on. These are called **system extension modules**:

The structure of RISC OS

The modules and the kernel provide their facilities very similarly, and there are few occasions when you will be able to distinguish whether the facilities you are using are provided by the kernel or by a system extension module. You are most likely to notice the difference if you wish to alter or replace part of the operating system.

Facilities

You can view RISC OS as a collection of routines that provide you with a wide range of facilities. You can get a good overview of the range that is covered from the earlier contents pages of this manual.

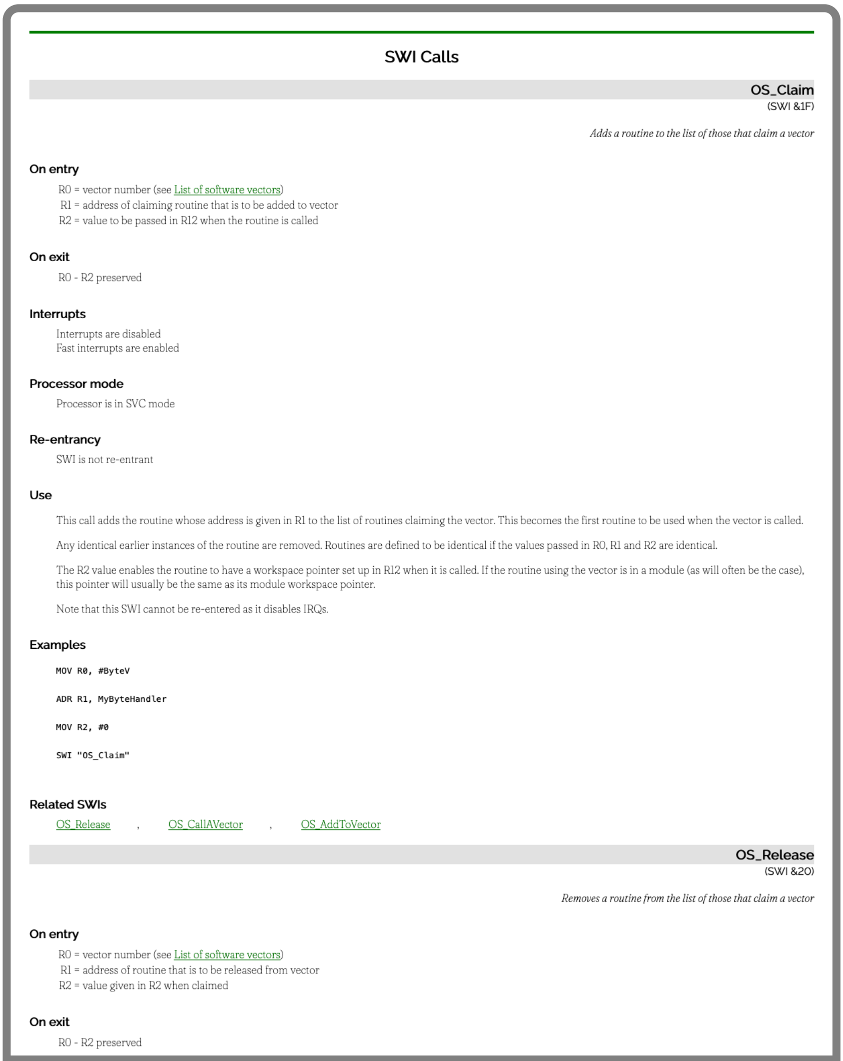
This collection of routines can be broadly divided into three levels:

- Those that RISC OS itself uses to automatically perform low-level tasks, such as **interrupt handling**
- Those that provide sophisticated and powerful interfaces for you to use from programs, which are known as **SoftWare Interrupts**, or SWIs for short.
- Those that provide simpler calls that can be used from the command line as well as from programs - these are the *** Commands** that you are probably already familiar with.

There are chapters later in this part of the manual that cover the above topics in more detail. They are entitled:

- kernel/interrupts.
- kernel/swis.

PRM-in-XML 'acornfs': intro chapter



SWI Calls

OS_Claim
(SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number (see List of software vectors)
R1 = address of claiming routine that is to be added to vector
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

Any identical earlier instances of the routine are removed. Routines are defined to be identical if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

Note that this SWI cannot be re-entered as it disables IRQs.

Examples

```
MOV R0, #ByteV

ADR R1, MyByteHandler

MOV R2, #0

SWI "OS_Claim"
```

Related SWIs

OS_Release , OS_CallAVector , OS_AddToVector

OS_Release
(SWI &20)

Removes a routine from the list of those that claim a vector

On entry

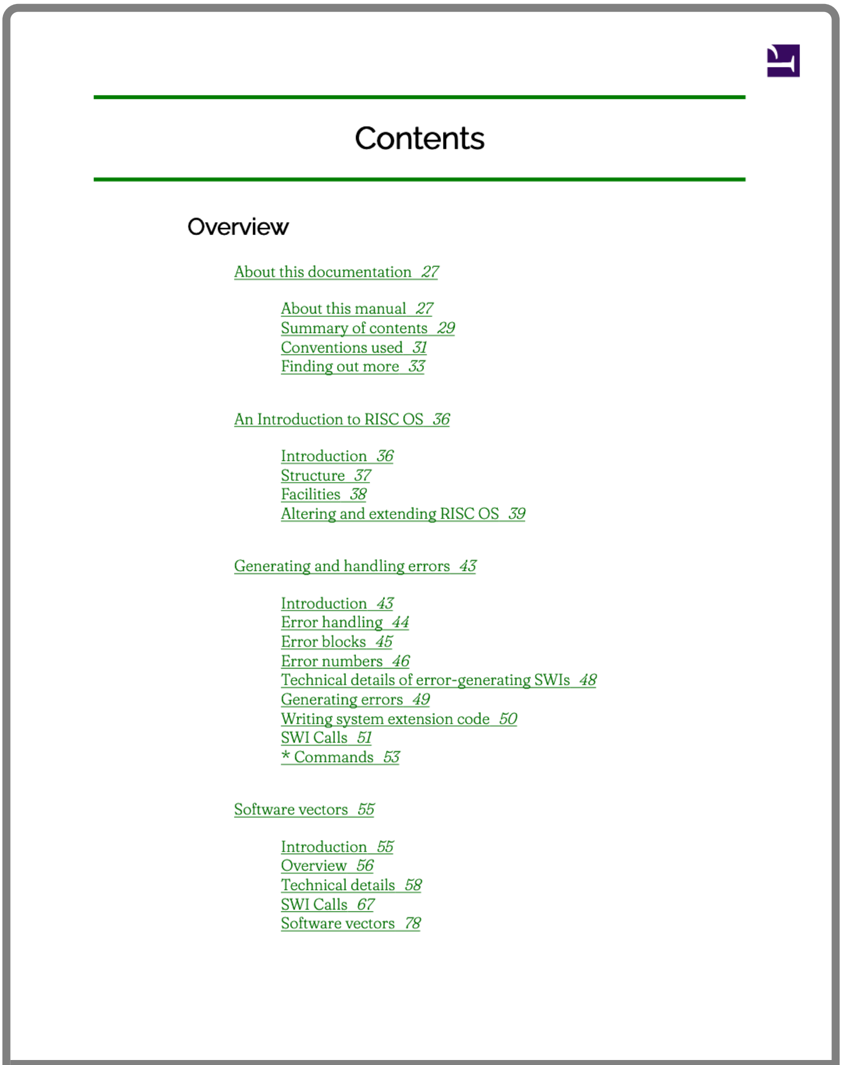
R0 = vector number (see List of software vectors)
R1 = address of routine that is to be released from vector
R2 = value given in R2 when claimed

On exit

R0 - R2 preserved

PRM-in-XML 'acornfs': SWI definition

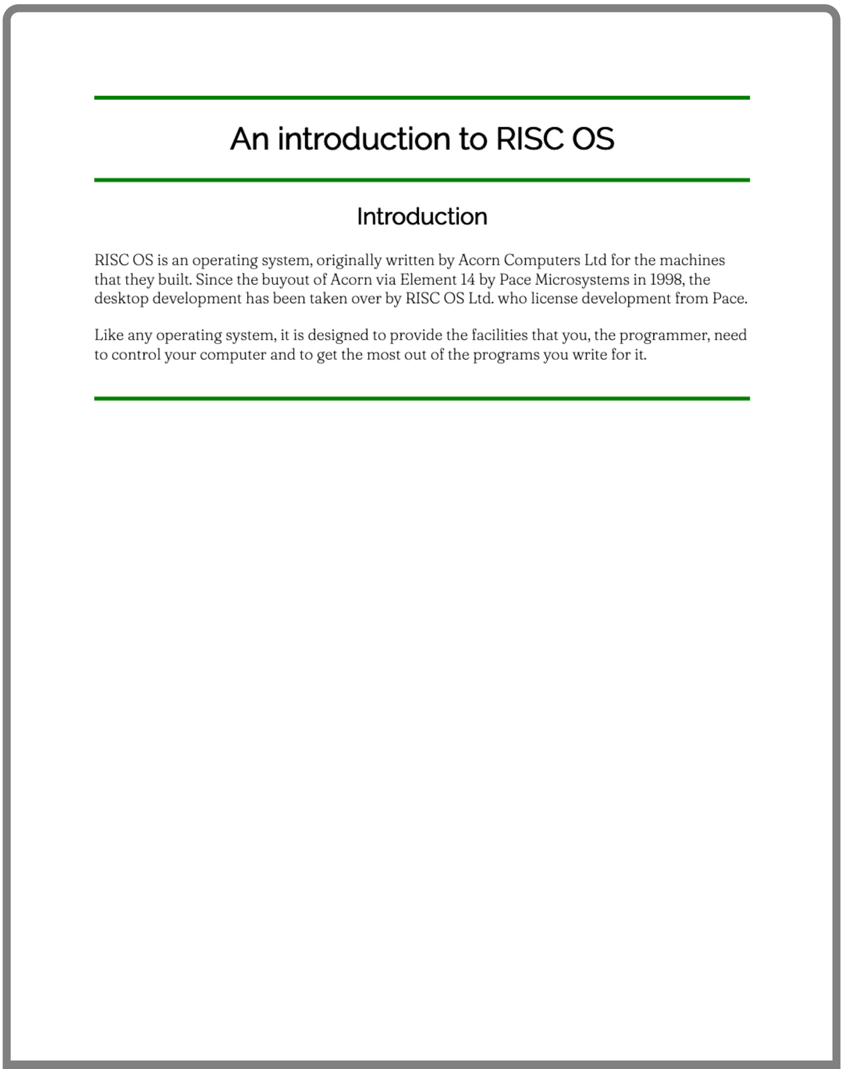
Example pages (PDF)



Contents

Overview

PRM-in-XML 'acornfs' (PDF): contents page



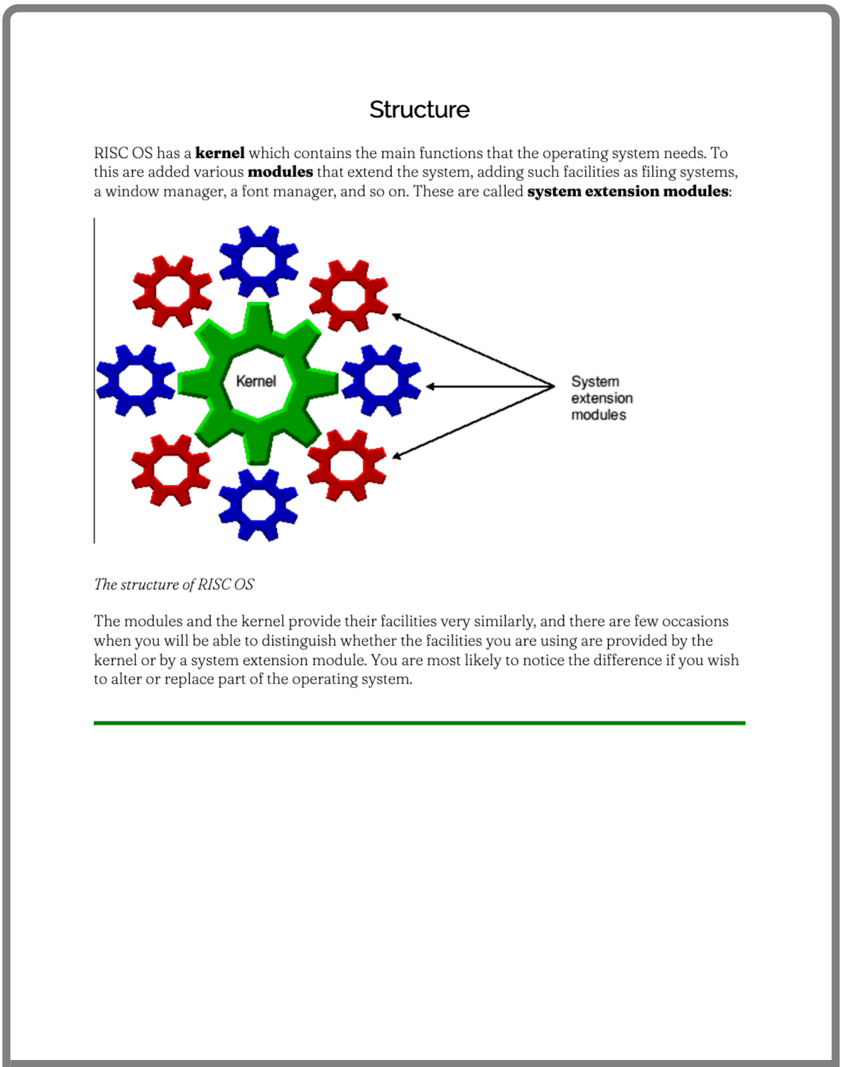
An introduction to RISC OS

Introduction

RISC OS is an operating system, originally written by Acorn Computers Ltd for the machines that they built. Since the buyout of Acorn via Element 14 by Pace Microsystems in 1998, the desktop development has been taken over by RISC OS Ltd. who license development from Pace.

Like any operating system, it is designed to provide the facilities that you, the programmer, need to control your computer and to get the most out of the programs you write for it.

PRM-in-XML 'acornfs' (PDF): intro chapter (1)



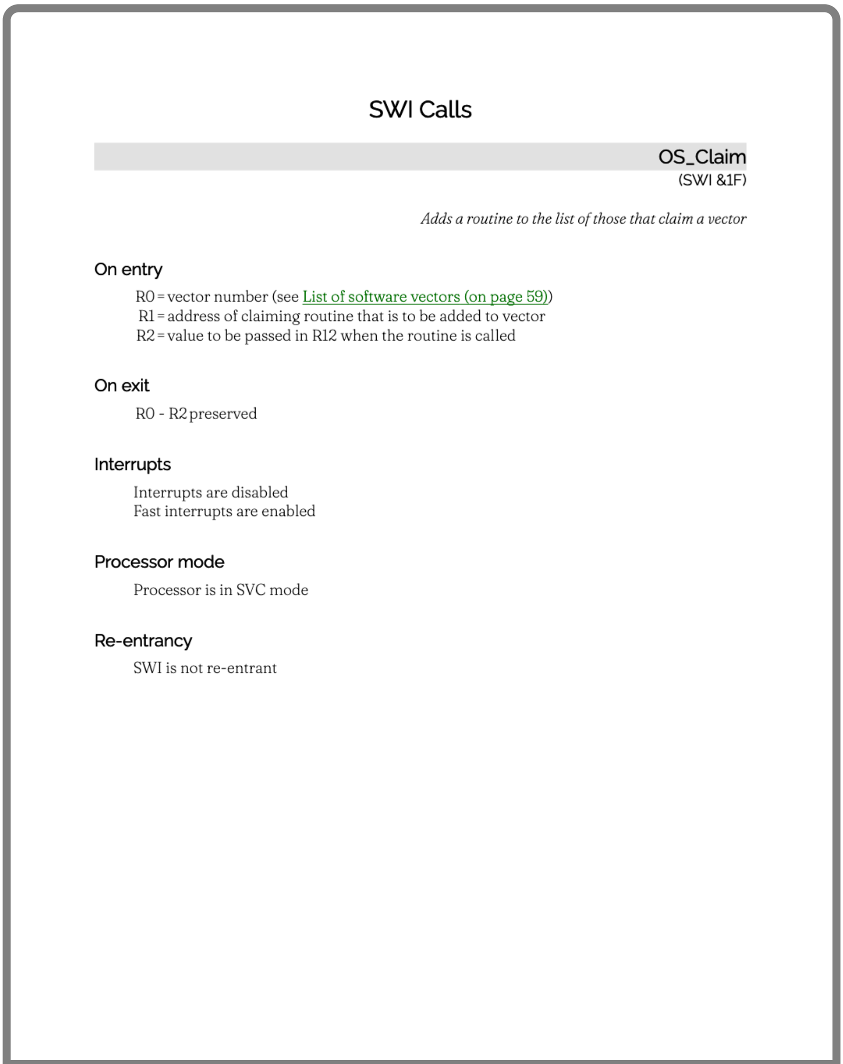
Structure

RISC OS has a **kernel** which contains the main functions that the operating system needs. To this are added various **modules** that extend the system, adding such facilities as filing systems, a window manager, a font manager, and so on. These are called **system extension modules**:

The structure of RISC OS

The modules and the kernel provide their facilities very similarly, and there are few occasions when you will be able to distinguish whether the facilities you are using are provided by the kernel or by a system extension module. You are most likely to notice the difference if you wish to alter or replace part of the operating system.

PRM-in-XML 'acornfs' (PDF): intro chapter (2)



SWI Calls

OS_Claim

(SWI &1F)

Adds a routine to the list of those that claim a vector

On entry

R0 = vector number (see List of software vectors (on page 59))
R1 = address of claiming routine that is to be added to vector
R2 = value to be passed in R12 when the routine is called

On exit

R0 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

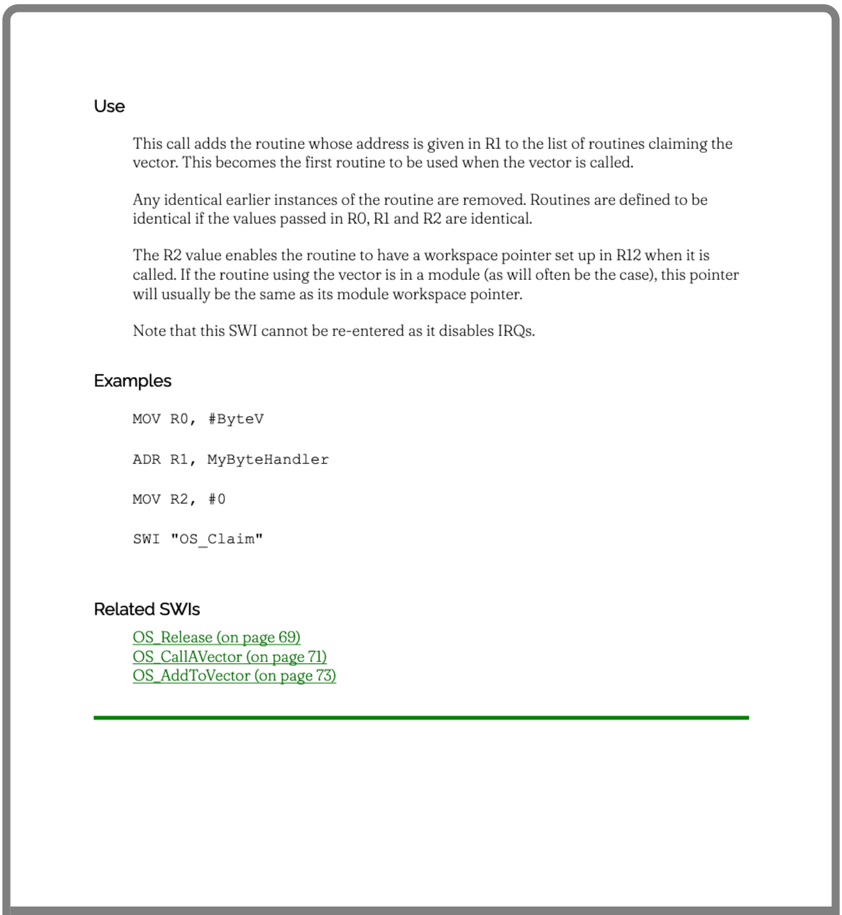
Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

PRM-in-XML 'acornfs' (PDF): SWI definition (1)



Use

This call adds the routine whose address is given in R1 to the list of routines claiming the vector. This becomes the first routine to be used when the vector is called.

Any identical earlier instances of the routine are removed. Routines are defined to be identical if the values passed in R0, R1 and R2 are identical.

The R2 value enables the routine to have a workspace pointer set up in R12 when it is called. If the routine using the vector is in a module (as will often be the case), this pointer will usually be the same as its module workspace pointer.

Note that this SWI cannot be re-entered as it disables IRQs.

Examples

```
MOV R0, #ByteV

ADR R1, MyByteHandler

MOV R2, #0

SWI "OS_Claim"
```

Related SWIs

OS_Release (on page 69)
OS_CallAVector (on page 71)
OS_AddToVector (on page 73)

PRM-in-XML 'acornfs' (PDF): SWI definition (2)

Document information

Maintainer(s): Gerph <gerph@gerph.org>

History:

Revision	Date	Author	Changes
1	31 Aug 2021	Gerph	**Initial version** • Created the collection of Acorn examples from PDFs. • Created a few examples from the existing content as HTML and PDFs and then described them.

Related: RISC OS 2 PRM PDF
C Release 4 PDF
RISC OS 3 PRM PDF
Volume 5a PRM PDF

Disclaimer: © Gerph, 2021