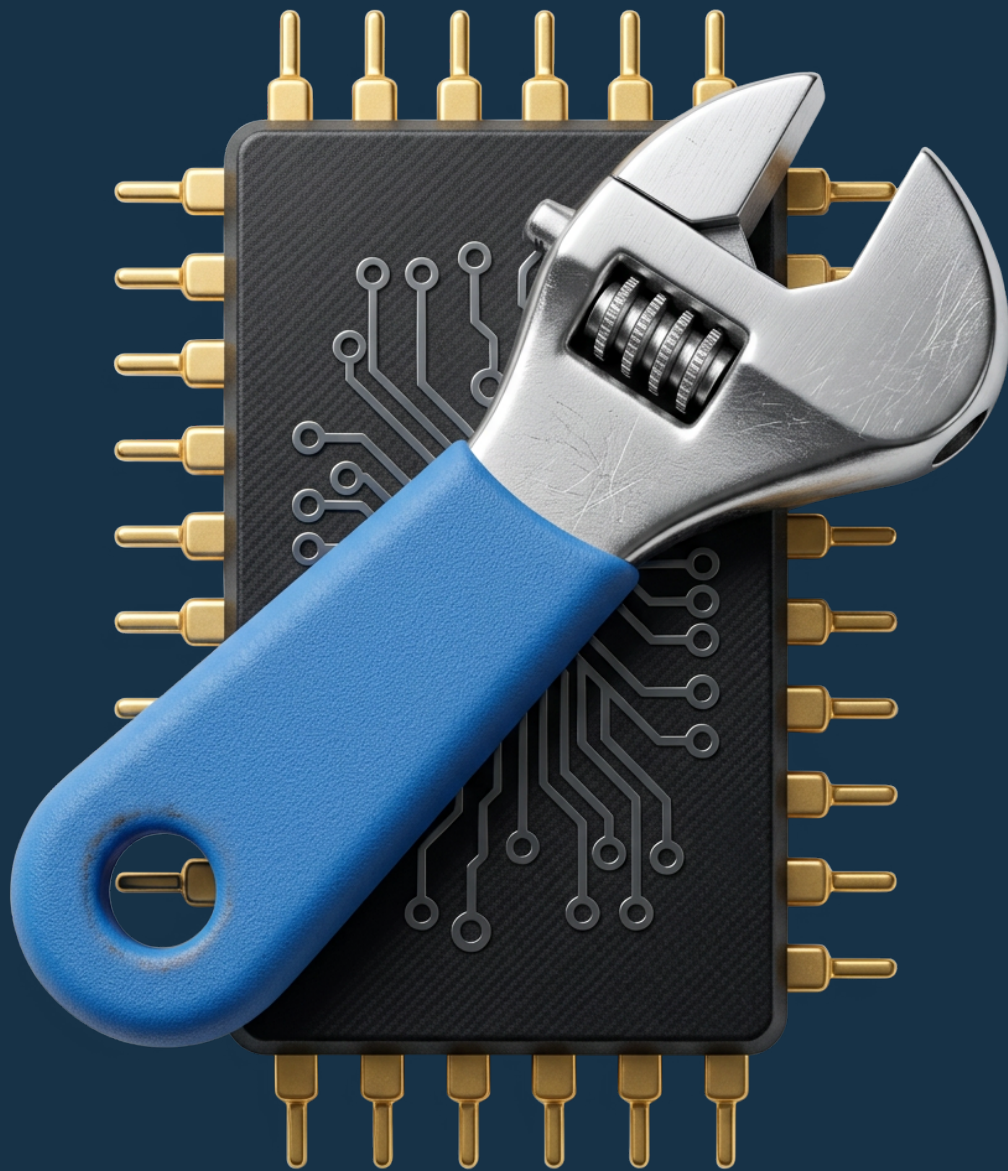




RISC OS

Programmer's Reference Manual

READER EDITION 2026

2,538 source entries · 24 local supplements

Based on the RISC OS Open documentation index · 18 September 2026

Using this manual

This reader edition organizes the public RISC OS Open Programmer's Reference Manual wiki into individual SWI, service, command, message, and topic entries. Each call begins with what it does, then shows inputs and outputs in separate blocks. Reason codes have their own entries when the source provides them. The link under each entry opens the corresponding source page.

What the register labels mean

"Before calling" lists values the caller supplies. "On return" lists values the call returns or changes. A "preserved" register is left unchanged; "corrupted" means the caller must not rely on its former value. For reason-coded SWIs, read both the parent call and the specific reason entry.

Scope and verification

This edition contains 2,538 substantive wiki entries and 24 local supplement manuals. The ROOL index says that its RISC OS 5 PRMs are still being made available online. Some linked pages are only indices or empty stubs and are not presented as complete API contracts. Confirm behavior against the specific OS/module version in use, especially for hardware and newer local interfaces.

Source index: <https://www.riscosopen.org/wiki/documentation/show/Programmer%27s%20Reference%20Manuals>

New layout and explanatory arrangement: Copyright (c) RISC OS Technologies 2026. All rights reserved. Apache License 2.0. Source documentation and the cover illustration retain their original owners' rights.

Current development practice

Use the named SWI rather than a numeric literal where practical; names make the interface easier to read and maintain. Handle X-form errors explicitly when the caller needs to recover. These conventions are described in the ROOL Introduction to SWIs.

For newer ARM targets, check 32-bit and ARMv6/ARMv7 compatibility of old code, especially assembly and unaligned memory accesses. Synchronise generated or modified executable code with OS_SynchroniseCodeAreas as described by its contract. Keep user-visible messages in MessageTrans catalogues when internationalisation is needed.

References: <https://www.riscosopen.org/wiki/documentation/show/Introduction%20To%20SWIs> ;
<https://www.riscosopen.org/wiki/documentation/show/ARMv7%20compatibility%20primer> ;
https://www.riscosopen.org/wiki/documentation/show/OS_SynchroniseCodeAreas ;
<https://www.riscosopen.org/wiki/documentation/show/MessageTrans>

SWI calls

Software interrupts, including specific reason-code pages. Read the general call and its reason entry together when both are listed.

AcornSSL_Close

SWI &50F84 · AcornSSL

Analogous to `Socket_Close`.

Before calling

R0 ssl handle

On return

R0 0 for success

In practice

Close a connection.

[RISC OS Open source page](#)

AcornSSL_Connect

SWI &50F82 · AcornSSL

Analogous to `Socket_Connect`.

Before calling

R0 ssl handle

R1 pointer to address structure

R2 size of address structure

On return

R0 0 for success

In practice

Connects the secure session to a remote host.

[RISC OS Open source page](#)

AcornSSL_Creat

SWI &50F80 · AcornSSL

Analogous to `Socket_Creat`, this SWI initialises a new secure session, returning a handle to use with the other SWIs. Note that the handle is not a socket handle and must be treated as opaque.

Before calling

R0 bits 0-7: · domain

bits 8-31: · reserved for future use (zero)

R1 type ·

R2 protocol ·

On return

R0 SSL handle

In practice

If a socket handle (from `Socket_Creat`) has already been opened and needs to be upgraded to a secure session, see `AcornSSL_CreateSession`.

[RISC OS Open source page](#)

AcornSSL_CreateSession

SWI &50F88 · AcornSSL

This SWI performs a similar function to `AcornSSL_Creat`, but allows the caller to hand over a previously opened socket.

Before calling

R0 socket handle from `Socket_Creat`

R1 bits 0-7: · reason code 0, create

bits 8-31: · reserved for future use (zero)

On return

R0 SSL handle

In practice

Initialises a secure session with an existing socket.

[RISC OS Open source page](#)

AcornSSL_Getpeername

SWI &50F89 · AcornSSL

Analogous to `Socket_Getpeername`.

Before calling

R0 SSL handle

R1 pointer to address structure

R2 pointer to size of address block

On return

R0 0 for success

In practice

Get details of the remote host's address.

[RISC OS Open source page](#)

AcornSSL_Getsockname

SWI &50F8A · AcornSSL

Analogous to `Socket_Getsockname`.

Before calling

R0 SSL handle

R1 pointer to address structure

R2 pointer to size of address block

On return

R0 0 for success

In practice

Get details of the local host's address.

[RISC OS Open source page](#)

AcornSSL_Getsockopt

SWI &50F85 · AcornSSL

Analogous to `Socket_Getsockopt`.

Before calling

R0 ssl handle

R1 option level

R2 option

R3 pointer to buffer for option value

R4 size of option value buffer

On return

R0 0 for success

In practice

Get a socket option.

[RISC OS Open source page](#)

AcornSSL Ioctl

SWI &50F81 · AcornSSL

Analogous to `Socket Ioctl`.

Before calling

R0 ssl handle

R1 operation

R2 pointer to argument

On return

R0 0 for success

In practice

Perform miscellaneous socket input/output control.

[RISC OS Open source page](#)

AcornSSL_Read

SWI &50F8E · AcornSSL

Analogous to `Socket_Read`, equivalent to `AcornSSL_Recv` with flags of 0.

Before calling

R0 SSL handle

R1 pointer to data to receive

R2 size of data buffer

On return

R0 amount of data received

In practice

Read data from a secure connection.

See `AcornSSL_Recv`.

[RISC OS Open source page](#)

AcornSSL_Recv

SWI &50F87 · AcornSSL

Analogous to `Socket_Recv`.

Before calling

R0 ssl handle

R1 pointer to data to receive

R2 size of data buffer

R3 option flags, only `MSG_PEEK` is significant

On return

R0 amount of data received

In practice

Read data from a secure connection.

While establishing a secure link, and performing a certificate exchange, potentially requiring user confirmation, the link status may be reported as `ENOTCONN` in order to match the response the Internet module would give prior to a link being up.

[RISC OS Open source page](#)

AcornSSL_Send

SWI &50F8F · AcornSSL

Analogous to `Socket_Send`.

Before calling

R0 SSL handle

R1 pointer to data to send

R2 amount of data to send

R3 option flags, none are significant

On return

R0 amount of data written

In practice

Write data to a secure connection.

While establishing a secure link, and performing a certificate exchange, potentially requiring user confirmation, the link status may be reported as `ENOTCONN` in order to match the response the Internet module would give prior to a link being up.

[RISC OS Open source page](#)

AcornSSL_Setsockopt

SWI &50F8B · AcornSSL

Analogous to `Socket_Setsockopt`.

Before calling

R0 ssl handle

R1 option level

R2 option

R3 pointer to buffer for option value

R4 size of option value buffer

On return

R0 0 for success

In practice

Set a socket option.

[RISC OS Open source page](#)

AcornSSL_Shutdown

SWI &50F83 · AcornSSL

Analogous to `Socket_Shutdown`.

Before calling

R0 ssl handle

R1 direction of shutdown

(see `Socket_Shutdown` for allowable values)

On return

R0 0 for success

In practice

Shut one or both halves of a connection.

[RISC OS Open source page](#)

AcornSSL_Stat

SWI &50F8C · AcornSSL

Analogous to `Socket_Stat`.

Before calling

R0 ssl handle

R1 pointer to stat structure

On return

R0 0 for success

In practice

Read status information for a socket.

[RISC OS Open source page](#)

AcornSSL_Version

SWI &50F8D · AcornSSL

Analogous to `Socket_Version`.

On return

R0 100 * the version

In practice

Read the module's version number.

If extra features are added in the future, the version number can be read to determine whether the loaded version is able to support those features.

[RISC OS Open source page](#)

AcornSSL_Write

SWI &50F86 · AcornSSL

Analogous to `Socket_Write`, equivalent to `AcornSSL_Send` with flags of 0.

Before calling

R0 ssl handle

R1 pointer to data to send

R2 amount of data to send

On return

R0 amount of data written

In practice

Write data to a secure connection.

See AcornSSL_Send.

[RISC OS Open source page](#)

ADFS_ATAPIOp

SWI &40255 · ADFS

The purpose of this call is to perform ATA operations on ADFS attached devices.

Before calling

R0 bit 0 set => reset controller

bit 0 clear => process command; other bits then act as below

bits 12-15 = device ID (prior to ADFS 4.00, only device IDs 0 and 8 were valid, corresponding to master and slave PATA devices)

bits 16-23 = controller/port ID

bits 24-25 = transfer direction:

00 => no transfer

01 => read (ie bit 24 set)

10 => write (ie bit 25 set)

11 reserved

bit 26 set => data transfer uses multiword DMA or UDMA from device's perspective (note that the host controller may actually use bus mastering even for PIO transfers) (since ADFS 4.00)

bit 27 set => device requires DMADIR indicator in Features task file register (determine this using IDENTIFY PACKET DEVICE) (since ADFS 4.00)

R1 length of Control block - must be 12 or 16 (prior to ADFS 4.00, must be 12)

R2 pointer to Control block

R3 pointer to buffer

R4 length to transfer

R5 timeout in centiseconds (0 => use default, currently 10 seconds)

On return

R0 0 if successful, or a disc error number (with V flag clear)

R1 Preserved?

R2 Preserved? The Control block is not updated

R3 Pointer to first byte not transferred

R4 Length not transferred

R5 Corrupted

Control block (12 byte format)

Offset / Description

+0 – Description: Operation code

+1 – Description: Service action

+2 – Description: 32-bit LBA

+6 – Description: 32-bit Length

+10 – Description: Reserved

+11 – Description: Control

Notes

This SWI was first present in ADFS 3.34 (RISC OS 5.02)

[RISC OS Open source page](#)

ADFS_ControllerType

SWI &40248 · ADFS

The purpose of this call is to return the controller type of a given drive.

Before calling

R0 Drive Number

On return

R0 Controller Type

0 = Disc not present

1 = 1772

2 = 710/711

3 = ST506

4 = IDE

Notes

This call is not available for versions earlier than RISC OS 2.xx

[RISC OS Open source page](#)

ADFS_DescribeDisc

SWI &40245 · ADFS

The purpose of this call is to check the disc's format and internal structure. It uses the same entry and exit structure as FileCore_DescribeDisc SWI Call.

Before calling

See FileCore_DescribeDisc

On return

See FileCore_DescribeDisc

Notes

This call utilises the FileCore_DescribeDisc SWI Call after first passing a pointer to the instance of ADFS in R8.

[RISC OS Open source page](#)

ADFS_DiscOp

SWI &40240 · ADFS

The purpose of this call is to provide a number of low-level disc operations. It uses the same entry and exit structure as FileCore_DiscOp SWI Call.

Before calling

See FileCore_DiscOp

On return

See FileCore_DiscOp

Notes

This call utilises the FileCore_DiscOp SWI Call after first passing a pointer to the instance of ADFS in R8.

[RISC OS Open source page](#)

ADFS_DiscOp64

SWI &40254 · ADFS

The purpose of this call is to provide a number of low-level disc operations using the more advanced Disc Operation DiscOp64.

Before calling

See FileCore_DiscOp64

On return

See FileCore_DiscOp64

Notes

This call utilises the FileCore_DiscOp64 SWI Call after first passing a pointer to the instance of ADFS in R8.

[RISC OS Open source page](#)

ADFS_Drives

SWI &40242 · ADFS

The purpose of this call is to return information on the filing system's drives.

Before calling

See FileCore_Drives

On return

See FileCore_Drives

Notes

This call utilises the FileCore_Drives SWI Call after first passing a pointer to the instance of ADFS in R8.

[RISC OS Open source page](#)

ADFS_ECCSAndRetries

SWI &40250 · ADFS

????

Before calling

On return

Notes

This command is reserved for internal use, and must not be used.

[RISC OS Open source page](#)

ADFS_FlpProcessDCB

SWI &40247 · ADFS

????

Before calling

On return

Notes

This command is reserved for internal use, and must not be used.

[RISC OS Open source page](#)

ADFS_FreeSpace

SWI &40243 · ADFS

The purpose of this call is to return the total free space on given disc, and the largest object that can be created on it.

Before calling

See FileCore_FreeSpace

On return

See FileCore_FreeSpace

Notes

This call utilises the FileCore_FreeSpace SWI Call after first passing a pointer to the instance of ADFS in R8.

[RISC OS Open source page](#)

ADFS_FreeSpace64

SWI &40252 · ADFS

The purpose of this call is to return the total free space on given disc, and the largest object that can be created on it.

Before calling

See FileCore_FreeSpace64

On return

See FileCore_FreeSpace64

Notes

This call utilises the FileCore_FreeSpace64 SWI Call after first passing a pointer to the instance of ADFS in R8.

[RISC OS Open source page](#)

ADFS_HDC

SWI &40241 · ADFS

The purpose of this call is to set the address of an alternative ST5056 hard disc controller to be used by the ADFS.

Before calling

R2 Address of alternative disc controller

R3 Address of poll location for IRQ/DRQ

R4 Bits for IRQ/DRQ

R5 Address to enable IRQ/DRQ

R6 Bits to enable IRQ/DRQ

On return

--

Notes

An expansion card can supply an alternative controller to the machines own, and must be HD63463 compatible.

[RISC OS Open source page](#)

ADFS_IDEDeviceInfo

SWI &40253 · ADFS

The purpose of this call is to return information on attached devices.

Before calling

R0 0 - Device info by Legacy ID

R1 bit 0 set => find information for ATADriver device ID 8 rather than 0

(corresponds to slave rather than master device on PATA systems)

bit 1 = find information for ATADriver controller/port ID 1 rather than 0

bits 2-31 reserved, should be 0

R0 1 - Device info by controller/port and device ID

R1 bits 0-3 = device ID

bits 4-11 = controller/port ID

bits 12-31 reserved, should be 0

R0 2 - Device info by drive number

R1 ADFS drive number of device

On return

R0 Preserved

R1 Type of device:

0 - no device

1 - non-packet

2 - packet device (ATAPI)

R2 ADFS drive number of device, otherwise undefined

R3 Pointer to a 512-byte block containing the data returned by an ATA IDENTIFY DEVICE or IDENTIFY PACKET DEVICE, or 0 if no device

R0 Preserved

R1 Type of device:

0 - no device

1 - non-packet

2 - packet device (ATAPI)

R2 ADFS drive number of device, otherwise -1

R3 Pointer to a 512-byte block containing the data returned by an ATA IDENTIFY DEVICE or IDENTIFY PACKET DEVICE, or 0 if no device

R0 Preserved

R1 Type of device:

0 - no device

1 - non-packet

R2 bits 0-3 = device ID

bits 4-11 = controller/port ID

R3 Pointer to a 512-byte block containing the data returned by an ATA IDENTIFY DEVICE or IDENTIFY PACKET DEVICE, or 0 if no device

In practice

Input is dependent on the version of ADFS Module in use.

[RISC OS Open source page](#)

ADFS_IDEUserOp

SWI &4024B · ADFS

The purpose of this call is to access low-level IDE commands.

Before calling

- R0** Flag bits
- R2** Pointer to Parameter block for command and results
- R3** Pointer to buffer
- R4** Length to transfer
- R5** Timeout in centiseconds (if 0 is passed then use the default value)
- R12** Pointer to static workspace

On return

- R0** Command status (0 or a disc error number)
- R2** Preserved
- R3** Preserved
- R4** Updated
- R5** Corrupted

Flag bits

- 0 / 0 = process command
- 1 = reset controller
- 3 – 0 = process command: 1 = parameter block pointed to by R2 is 11 bytes long (new in ADFS 3.34 / RISC OS 5.02)
- 12 – 15 = device ID (since ADFS 4.00)
- 16 – 23 = controller/port ID (since ADFS 3.34 / RISC OS 5.02)
- 24 – 25 / 00 = no transfer
- 01 = read (bit 24 set)
- 10 = write (bit 25 set)
- 11 = reserved
- 26 · 1 = data transfer uses multiword DMA or UDMA from device's perspective (note that the host controller may actually use bus mastering even for PIO transfers) (since ADFS 3.34 / RISC OS 5.02)

Parameter block

CHS/28-bit LBA form (R0 bit 3 clear):

- +1** – Features/Error: Sector Count
- +2** – Features/Error: LBA bits 0-7/Sector No
- +3** – Features/Error: LBA bits 8-15/Cylinder Low
- +4** – Features/Error: LBA bits 16-23/Cylinder High
- +5** – Features/Error: Device:
 - – Features/Error: bits 0-3 – LBA bits 24-27/Head
 - – Features/Error: bit 4 – 0=master, 1=slave
 - – Features/Error: bit 6 – 0=CHS mode, 1=LBA mode

+6 – Features/Error: Command/Status

48-bit LBA form (R0 bit 3 set, since ADFS 3.34 / RISC OS 5.02):

+0 / Features/Error

+1 – Features/Error: 16-bit Sector Count

+3 – Features/Error: 48-bit LBA

+9 – Features/Error: Device:

– – Features/Error: bit 4 – 0=master, 1=slave

– – Features/Error: bit 6 – 0=CHS mode, 1=LBA mode

+10 – Features/Error: Command/Status

Since ADFS 4.00, the use of bit 4 of the Device byte of the parameter block to select between devices on the same controller/port is deprecated (even for PATA controllers; PATADriver will substitute the device number into the Device byte prior to transmission for you). Use bits 12-15 of R0 instead. Also since ADFS 4.00, the parameter block is no longer updated if R0 bit 0 was set on entry.

Notes

This call must not be called in the background.

[RISC OS Open source page](#)

ADFS_LockIDE

SWI &40251 · ADFS

The purpose of this call is to lock or unlock the IDE bus.

Before calling

R0 Lock Flag (as described below)

On return

R0 Preserved

Flag bits

0 · 0 = unlock IDE bus

1 = lock IDE bus

Notes

All non-used bits within the flag must be set to zero. An error is generated if an attempt to lock and already locked bus.

When attempting to lock the bus in a background process, you must not do so within a loop. Instead, reschedule a retry for a later time.

[RISC OS Open source page](#)

ADFS_MiscOp

SWI &4024C · ADFS

The purpose of this call is to provide a number of miscellaneous low-level functions for accessing drives.

Before calling

See FileCore_MiscOp

On return

See FileCore_MiscOp

Notes

This call utilises the FileCore_MiscOp SWI Call after first passing a pointer to the instance of ADFS in R8.

[RISC OS Open source page](#)

ADFS_PowerControl

SWI &40249 · ADFS

The purpose of this call is to control the power-saving features of ADFS.

Before calling

R0 Reason code (as described below)

R1 Drive

R2 Command (as described below)

On return

R2 Drive spin status, if R0 = 0 on entry

Value of 0 = drive is not spinning

Value != 0 = drive is spinning

R3 Previous value for drive autospindown, if R0 = 1 on entry

Reason Codes

0 · Read drive spin status

1 · Set drive autospindown

2 · Control drive spin directly without affecting autospindown

Command

Drive autospindown (if R0 = 1)

0 · Disable autospindown and spinup drive

≠ 0 · Set autospindown to n seconds (where n = value specified in R2 x 5)

Specify spin up or spin down of Drive (if R0 = 2)

0 · Spin down drive immediately

≠ 0 · Spin up drive immediately

Notes

The drives should fully support drive spin control, otherwise unexpected errors may occur.

[RISC OS Open source page](#)

ADFS_Retries

SWI &40244 · ADFS

The purpose of this call is to specify the number of retries used by writing to the retry word. The format of this word is:

Before calling

R0 Mask of bits to change

R1 New values of bits to change

On return

R0 Preserved

R1 R0 AND entry value of R1

R2 Old value of retry word

R3 New value of retry word

Byte / Number of retries for

0 – Number of retries for: Hard disc read/write sector

1 – Number of retries for: Floppy disc read/write sector

2 – Number of retries for: Floppy disc mount (per copy of the disc map)

3 – Number of retries for: Verify after *Format, before sector is considered a defect

In practice

The new value is calculated as follows:

(old value AND NOT R0) EOR (R1 AND R0)

Notes

None.

[RISC OS Open source page](#)

ADFS_SectorDiscOp

SWI &4024D · ADFS

The purpose of this call is to provide a number of low-level disc operations.

Before calling

See FileCore_SectorDiscOp

On return

See FileCore_SectorDiscOp

Notes

This call utilises the FileCore_SectorDiscOp SWI Call after first passing a pointer to the instance of ADFS in R8.

[RISC OS Open source page](#)

ADFS_SetIDEController

SWI &4024A · ADFS

The purpose of this call is to supply the IDE Driver with details of an alternative controller.

Before calling

R2 Pointer to IDE controller

R3 Pointer to interrupt status of controller

R4 AND with status, NE => IRQ

R5 Pointer to interrupt mask

R6 OR into mask enables IRQ

R7 Pointer to data read routine (0 for default)

R8 Pointer to data write routine (0 for default)

R12 Pointer to status workspace

On return

All registers preserved

Notes

None.

[RISC OS Open source page](#)

ADFS_VetFormat

SWI &40246 · ADFS

The purpose of this call is to Vet the given disc format structure for achievability with the available hardware. ADFS updates the disc format structure with values that it can actually achieve, rather than the requested values.

Before calling

R0 Pointer to interrupt status of controller

R1 Parameters previously passed by ADFS in R2 to ImageFS_DiscFormat (ie. Drive Number)

On return

R0 Preserved

R1 Preserved

Notes

If ADFS is unable to sensibly downgrade the parameters given in the disc format structure, it will generate an error.

This call is typically made by FileCore in response to ADFS calling FileCore_DiscFormat, or by ImageFS calling ImageFS_DiscFormat – for example, DOSFS is an Image Filing System, and it calls DOSFS_DiscFormat.

[RISC OS Open source page](#)

ARM_IMB

SWI &F00000 · UtilityModule

This SWI implements an IMB operation, equivalent to calling OS_SynchroniseCodeAreas with R0=0.

On return

- All registers preserved

Notes

This SWI was introduced in RISC OS 5, and is one of a pair of OS-agnostic SWIs which the ARMv5 ARM recommended that operating systems should implement (see ARM DDI 0100E, section 2.7.4). More recent versions of the ARM ARM do not include these recommendations. Unless you are writing software which aims for compatibility with other operating systems which are known to implement these SWIs, it's recommended to use the RISC OS OS_SynchroniseCodeAreas SWI instead, as that will provide compatibility with a wider range of OS versions.

[RISC OS Open source page](#)

ARM_IMBRange

SWI &F00001 · UtilityModule

This SWI implements a ranged IMB operation, equivalent to calling OS_SynchroniseCodeAreas with R0/R1 shuffled up to R1/R2 and R0=1.

Before calling

R0 Start address (inclusive)

R1 End address (exclusive)

On return

- All registers preserved

Notes

This SWI was introduced in RISC OS 5, and is one of a pair of OS-agnostic SWIs which the ARMv5 ARM recommended that operating systems should implement (see ARM DDI 0100E, section 2.7.4). More recent versions of the ARM ARM do not include these recommendations. Unless you are writing software which aims for compatibility with other operating systems which are known to implement these SWIs, it's recommended to use the RISC OS OS_SynchroniseCodeAreas SWI instead, as that will provide compatibility with a wider range of OS versions.

[RISC OS Open source page](#)

ATA_Op

SWI &59783 · SATADriver

The purpose of this call is to perform low-level ATA commands.

Before calling

R0 Flag bits

R1 Length of parameter block, must be 1 (for device control), 7 (for 28-bit LBA), 11 (for 48-bit LBA) or 12 (for wide features)

R2 Pointer to Parameter block for command and results

R3 Pointer to buffer or scatter list

R4 Length to transfer. Must be 0 if no data transfer is specified in the Flag bits

R5 Timeout in centiseconds if bit 29 of the Flag bits is set

R6 Address of callback routine, if bit 26 of the Flag bits is set

R7 Value to pass in R12 to the callback routine, if bit 26 of the Flag bits is set

On return

R0 Pointer to error block, or 0 if successful

R4 Bytes not transferred, or 0 if transfer complete

Flag bits

0 · 1 = don't wait for DRDY before issuing command

2 · 1 = data transfer uses multiword DMA or UDMA from device's perspective (note that the host controller may actually use bus mastering even for PIO transfers)

12 - 15 · device ID

16 - 23 · controller/port ID

24 - 25 / 00 = no transfer

01 = read (bit 24 set)

10 = write (bit 25 set)

26 · 1 = scatter list transfer

27 · 1 = don't poll for escape during foreground transfers

28 · reserved

29 · 1 = background transfer (not supported initially)

Parameter block

Device control form:

+0 / Features/Error

+0 – Features/Error: Device control

28-bit LBA form:

+1 – Features/Error: Sector Count

+2 – Features/Error: LBA bits 0-7/Sector No

+3 – Features/Error: LBA bits 8-15/Cylinder Low

+4 – Features/Error: LBA bits 16-23/Cylinder High

+5 – Features/Error: Device:

– – Features/Error: bits 0-3 - LBA bits 24-27/Head

– – Features/Error: bit 4 - 0=master, 1=slave

– – Features/Error: bit 6 - 0=CHS mode, 1=LBA mode

+6 – Features/Error: Command/Status

48-bit LBA form:

+0 / Features/Error
 +1 – Features/Error: 16-bit Sector Count
 +3 – Features/Error: 48-bit LBA
 +9 – Features/Error: Device:
 – – Features/Error: bit 4 - 0=master, 1=slave
 – – Features/Error: bit 6 - 0=CHS mode, 1=LBA mode

+10 – Features/Error: Command/Status

Wide features form:

+0 / Features bits 0-7/Error
 +1 – Features bits 0-7/Error: Features bits 8-15
 +2 – Features bits 0-7/Error: 16-bit Sector Count
 +4 – Features bits 0-7/Error: 48-bit LBA
 +10 – Features bits 0-7/Error: Device:
 – – Features bits 0-7/Error: bit 6 - 0=CHS mode, 1=LBA mode
 +11 – Features bits 0-7/Error: Command/Status

Notes

If a scatter list is used, it is not updated.

[RISC OS Open source page](#)

ATAPI_GetDrives

SWI &4A740 · CDFSSoftATAPI

The purpose of this call is to return the number of ATAPI drives attached.

Before calling

--

On return

R0 Number of drives found

Notes

No more information about this SWI is provided.

[RISC OS Open source page](#)

BASICTrans_Error

SWI &42C81 · BASICTrans

The purpose of this call is to lookup an error message. The buffer pointer to by R1 must be at least 252 bytes long.

Before calling

R0 Error number

R1 Pointer to buffer

R4-R7 Message parameters

On return

R0-R3 Corrupt

Notes

Prior to BASICTrans 2.14, requesting an unknown error number would result in an error being generated from within BASICTrans. In BASICTrans 2.14 this was fixed so that the error would be returned by the SWI in the usual manner, i.e. with V set and R0 pointing to an error block.

[RISC OS Open source page](#)

BASICTrans_HELP

SWI &42C80 · BASICTrans

The purpose of this call is to lookup and output (via OS_WriteC) a help message for a given BASIC keyword.

Before calling

R0 Pointer to tokenised keyword/help term

R1 Pointer to program name (BASIC or BASIC64)

R2 Pointer to the lexical analyser's tables

On return

R0 Corrupt

R1 Zero: Message was displayed

Nonzero: Message not displayed

R2 Corrupt

[RISC OS Open source page](#)

BASICTrans_Message

SWI &42C82 · BASICTrans

The purpose of this call is to lookup and display (via OS_WriteC) a message.

Before calling

R0 Message number

R1-R3 Message parameters

On return

R0-R1 Corrupt

[RISC OS Open source page](#)

BCMSupport_AllocPropertyBuffer

SWI &591C2 · BCMSupport

This SWI is used to allocate a physically contiguous buffer for use with the mailbox property interface (BCMSupport_SendPropertyBuffer)

Before calling

R0 Required buffer length

On return

R0 Pointer to buffer (16 byte aligned)

Notes

Memory allocated via this SWI is inaccessible in user mode.

Buffers allocated via this SWI must be freed via BCMSupport_FreePropertyBuffer.

[RISC OS Open source page](#)

BCMSupport_FreePropertyBuffer

SWI &591C3 · BCMSupport

This SWI is used to free a buffer that was previously allocated by BCMSupport_AllocPropertyBuffer.

Before calling

R0 Pointer to buffer

On return

All registers preserved

Notes

It is the caller's responsibility to make sure that any buffer being freed is not currently in use by the GPU; i.e. all calls to BCMSupport_SendPropertyBuffer have had a response received by the CPU (blocking SWI calls have completed, or async SWI calls have had the response callback called).

[RISC OS Open source page](#)

BCMSupport_MBSync

SWI &591C1 · BCMSupport

This SWI can be used to block until all responses (of a given type) have been received. This can be used, e.g. during your modules finalisation sequence to make sure that all pending responses have been received.

Before calling

R0 Flags:

Bit 0: 1 => Sync based on response mask/value pair

Bit 1: 1 => Sync based on callback R0

Bit 2: 1 => Sync based on callback R12

Bit 3: 1 => Sync based on callback pointer

Other bits reserved, should be zero

R1 Response mask

R2 Response value

R3 Response callback R0

R4 Response callback R12

R5 Response callback function pointer

On return

All registers preserved

Notes

For each flag specified in R0, all the criteria must match for BCMSupport to consider the pending response to be one that it needs to block waiting for. Entries which have a null callback function pointer can still match on the R0 and R12 values, allowing them to be used as unique tokens by your code.

To block for receipt of all messages, you can set bit 0 of R0 and set both R1 and R2 to zero.

This SWI is only guaranteed to block for the receipt of messages which were sent prior to the SWI being called. New messages which are sent after the SWI has begun executing may not be waited for.

This SWI must not be used from within one of the message callback functions.

[RISC OS Open source page](#)

BCMSupport_SendMBMessage

SWI &591C0 · BCMSupport

This is the lowest-level interface for sending messages via the mailbox. The word provided in R0 is written directly to the send channel.

Before calling

R0 Mailbox channel (bits 0-3) and data (bits 4-31)

R1 Flags:

Bit 0:

0 => Block and wait for response

1 => Don't block, call callback specified in R4-R6 on receipt of response

Bit 1:

0 => Block if the send buffer is full

1 => Return error if the send buffer is full

Other bits reserved, should be zero

R2 Response mask

R3 Response value

If R1 bit 0 is set:

R4 Response callback R0

R5 Response callback R12

R6 Response callback function pointer

R0 R4 value from send SWI

R1 Response word

R12 R5 value from send SWI

CPU in privileged mode

Interrupts disabled

On return

R0 Response word (If flags bit 0 is clear, and a response is expected (R2 != 0))

Else preserved

All other registers preserved

R0-R3, R12 can be corrupted

Notes

The values of R2 and R3 are used to identify the response in the receive channel, using the following logic:

```
if ((response AND R2) == R3)
{
    // this is my response
}
else
{
    // this is not my response
}
```

Specifying a response mask of 0 indicates that no response is expected.

Each sent message which is expecting a response is pushed onto a queue. When a response is received from the GPU the queue is searched in FIFO order until a match is found. By correctly setting the response mask and value in R2 and R3 this allows the interface to cope with mailbox channels which only respond to messages in-order (e.g. channels 0 and 1) versus those that reply out-of-order (e.g. channel 8)

If the call is operating in a blocking manner (R1 bit 0 or bit 1 are unset) then it must be called with interrupts enabled.

Any response which fails to match against an entry in the expected-responses list will be discarded.

Callback functions are called as follows:

The callback function pointer can also be null, which is taken as an indication that the response should be ignored.

Callback functions are called directly from the mailbox interrupt handler, and so must be quick and must not enable interrupts. Sending further messages from within a callback is discouraged (non-blocking sends are possible via making sure bits 0 and 1 are set, but because there is no guarantee the send will succeed the ability is of limited use)

If a non-blocking send is requested, and the mailbox is full, a "mailbox full" error is returned (error number &81F200)

[RISC OS Open source page](#)

BCMSupport_SendPropertyBuffer

SWI &591C4 · BCMSupport

This SWI is used to send a buffer via the mailbox property interface. The buffer must be filled with a sequence of tagged messages, as described on the Raspberry Pi GitHub wiki

Before calling

R0 Pointer to buffer (16 byte aligned)

R1 Flags:

Bit 0:

0 => Block and wait for response

1 => Don't block, call callback specified in R2-R4 on receipt of response

Bit 1:

0 => Block if the send buffer is full

1 => Return error if the send buffer is full

Other bits reserved, should be zero

If R1 bit 0 is set:

R2 Response callback R0**R3** Response callback R12**R4** Response callback function pointer**R0** R2 value from send SWI**R1** Buffer pointer (R0 value from send SWI)**R12** R3 value from send SWI

CPU in privileged mode

Interrupts disabled

On return

All registers preserved

R0-R3, R12 can be corrupted

Notes

The buffer pointer must be at least 16 byte aligned, and located in a region of non-cacheable, physically contiguous memory. BCMSupport_AllocPropertyBuffer can be used to allocate a buffer which meets these requirements. There is no requirement for the caller to use memory barriers after writing the request to the buffer or before reading the response; BCMSupport will perform the correct barrier operations on your behalf.

Care must be taken to ensure that the buffer is not freed or used for another purpose while the GPU is processing the contents.

If you want to avoid the hassle of managing your own non-cacheable buffers, consider using the simpler BCMSupport_SendTempPropertyBuffer call.

If the call is operating in a blocking manner (R1 bit 0 or bit 1 are unset) then it must be called with interrupts enabled.

Callback functions are called as follows:

Unlike BCMSupport_SendMBMessage, it is not possible to request an async operation with a null callback. This is because to do so would mean you have no way of knowing when it is safe to free or re-use the buffer. It is also recommended that once an operation has completed you check the buffer for any received error codes.

Callback functions are called directly from the mailbox interrupt handler, and so must be quick and must not enable interrupts. Sending further messages from within a callback is discouraged (non-blocking sends are possible via making sure bits 0 and 1 are set, but because there is no guarantee the send will succeed the ability is of limited use)

If a non-blocking send is requested, and the mailbox is full, a "mailbox full" error is returned (error number &81F200)

As with the standard mailbox messages, BCMSupport_MBSync can be used to wait for completion of (async) messages. However it is not currently possible to specify the buffer address to wait for; you are limited to using the callback R0/R12/PC values, or to block for all messages.

[RISC OS Open source page](#)

BCMSupport_SendTempPropertyBuffer

SWI &591C5 · BCMSupport

This is a simplified version of `BCMSupport_SendPropertyBuffer` that does not require the caller to preallocate a physically contiguous buffer. Instead, ordinary RAM can be used for the input and output. BCMSupport will take care of copying the input to a temporary buffer, sending the message, and then copying the output to your output buffer. In most cases a pre-allocated temporary buffer will be used, but if the tag sequence is too long or all the preallocated buffers are in use then an attempt will be made to (temporarily) allocate a new buffer from the heap.

Before calling

R0 Pointer to input buffer

R1 Pointer to output buffer

R2 Flags (reserved, should be zero)

On return

All registers preserved

Notes

There is no async option with this call, and it will always block until there is enough mailbox space available.

The input and output buffer pointers must be at least 4 byte aligned. The same buffer can be used for both input and output.

[RISC OS Open source page](#)

BlendTable_GenerateTable

SWI &56280 · BlendTable

This SWI generates a lookup table to allow for the efficient blending of colours between a source and destination palette at a given translucency. The blend equation is effectively as follows:

Before calling

R0 Flags:

Bit 0 = R2 is pointer to sprite, not name

Bit 1 = R4 is pointer to sprite, not name

Bit 2 = Lock table after creation

Other bits reserved, should be zero

R1 Source mode / sprite area

R2 Source palette pointer / sprite

R3 Destination mode / sprite area

R4 Destination palette pointer / sprite

R5 Translucency (0-256)

R6 One of:

0 to determine size of table

Or pointer to buffer to return table in

Or ignored if bit 2 of R0 set

On return

R0 Number of colours in source palette

R1 Number of colours in destination palette

R2 Size of buffer required/used

R6 Preserved if bit 2 of R0 clear, else pointer to cached table

result = (source*(256-translucency)+dest*translucency)/256

In practice

The returned lookup table is an array of bytes which should be indexed as follows to perform a blend between a source and destination palette index (BASIC syntax):

result = table?(source+dest*number_of_colours_in_source)

Notes

R1-R2, R3-R4 and the corresponding flags in bits 0 and 1 of R0 identify the source and destination palette using the same basic scheme as ColourTrans_ReadPalette. See ColourTrans mode identification for more information on the different methods of specifying the palettes to use.

If a locked table was requested by setting bit 2 of R0, the value returned in R6 will be a pointer to memory owned by BlendTable. Once you have finished using the table you must call BlendTable_UnlockTable in order to let BlendTable know that the memory can be safely reused for other purposes.

Whether locking is requested or not, the generated table will be stored in an internal cache in order to reduce the overheads of repeatedly calling the SWI with the same palettes and translucency values.

There is little point calling this SWI with a translucency value of 0 or 256 – it will just waste both CPU time and memory.

Instead, for a translucency of 0, ColourTrans_GenerateTable should be used to generate a direct lookup table for mapping the source palette to destination palette, and for a translucency of 256 it may be possible for your code to skip rendering altogether.

[RISC OS Open source page](#)

BlendTable_UnlockTable

SWI &56281 · BlendTable

This SWI should be used to release a table that was returned to you by the BlendTable_GenerateTable 'Lock table' operation.

Before calling

R0 Table pointer

On return

- All registers preserved

Notes

Failure to release locked tables will result in a memory leak.

[RISC OS Open source page](#)

Buffer_Create

SWI &42940 · BufferManager

This call claims an area of the RMA and registers it as a buffer. It can hold $R1 - 1$ bytes at most.

Before calling

R0 Buffer's flags

R1 Size of buffer to be created

R2 Preferred handle for the buffer, or -1 to get the Buffer Manager to generate one

On return

R0 Buffer handle

In practice

If $R2 < -1$, the given number is checked for uniqueness, if it already exists, a new one is generated.

Notes

Note that you do not need any complicated calculations to write data into the buffer – it's simply a block of memory. Get the address with `Buffer_GetInfo` and write to the buffer.

[RISC OS Open source page](#)

Buffer_Deregister

SWI &42943 · BufferManager

This deregisters the buffer specified by the handle given in **R0**. You should use this to deregister buffers registered with `Buffer_Register`. For buffers made using `Buffer_Create`, you should use `Buffer_Remove` instead.

Before calling

R0 Handle of buffer to be removed

On return

--

`Buffer_Register` `Buffer_Create` `Buffer_Remove`

[RISC OS Open source page](#)

Buffer_GetInfo

SWI &42947 · BufferManager

This call reads information about a particular buffer. Insertion and removal points wrap around, and the sum of free space and characters may not equal total buffer size.

Before calling

R0 Buffer number

On return

R0 Buffer flags

R1 Pointer to buffer start

R2 Pointer to buffer end

R3 Insertion offset

R4 Removal offset

R5 Bytes free

R6 Bytes used

[RISC OS Open source page](#)

Buffer_InternalInfo

SWI &42949 · BufferManager

This call converts a buffer handle into an internal buffer ID, and returns the address of a service routine that can be used to access the buffer quickly. The address of the routine and the R12 value are the same for all buffers.

Before calling

R0 Buffer handle

On return

R0 Internal buffer ID

R1 Address of service code

R2 Value to pass to service code in R12

Notes

If an invalid buffer handle is passed, an error is returned but R1 and R2 will be correct.

This call is available on RISC OS 3.5 or greater.

[RISC OS Open source page](#)

Buffer_LinkDevice

SWI &42945 · BufferManager

This call links routines to a buffer.

Before calling

R0 Buffer handle

R1 Pointer to wake up routine, or 0 for none

R2 Pointer to owner change routine, or 0 if the owner cannot be changed

R3 Routine's private word (R8)

R4 Workspace (R12)

On return

--

[RISC OS Open source page](#)

Buffer_ModifyFlags

SWI &42944 · BufferManager

This SWI modifies the buffer's flags, according to the following:

Before calling

R0 Buffer handle

R1 EOR mask

R2 AND mask

On return

R1 Old flags

R2 New flags

In practice

$\text{NewVal} = (\text{OldVal} \text{ AND } \text{R2}) \text{ EOR } \text{R1}$

To read: $\text{R1} = 0$, $\text{R2} = \&0000000F$ To set: $\text{R1} = \text{new value}$, $\text{R2} = \&FFFFFFF0$

Notes

Ensure that you do not modify reserved bits with this call. Any reserved bits should be clear in R1, set in R2.

[RISC OS Open source page](#)

Buffer_Register

SWI &42942 · BufferManager

This SWI is similar to Buffer_Create, but does not claim memory for you. Instead, it registers your existing memory as a buffer.

Before calling

R0 Buffer's flags word

R1 Start address of the buffer

R2 Byte beyond end of the buffer

R3 Preferred handle for the buffer, or -1 to get the Buffer Manager to generate one

On return

R0 Buffer handle

[RISC OS Open source page](#)

Buffer_Remove

SWI &42941 · BufferManager

This call removes the buffer specified by the handle given in R0. It releases the workspace in the RMA reserved for this buffer. You should use this to remove buffers created with Buffer_Create. For buffers made using Buffer_Register you should use Buffer_Deregister instead.

Before calling

R0 Buffer handle

On return

--

Notes

Any data in the buffer will be purged, and an access via InsV, RemV and CnpV will be ignored.

[RISC OS Open source page](#)

Buffer_Threshold

SWI &42949 · BufferManager

The threshold value in R1 is -1 to read, and 0 for none.

Before calling

R0 Buffer handle

R1 Threshold

On return

R1 Previous value

In practice

OS_UpCall 8 and OS_UpCall 9 are issued if bit 3 of the `buffer_flags` is set, and the buffer free space crosses the threshold.

[RISC OS Open source page](#)

Buffer_UnlinkDevice

SWI &42946 · BufferManager

This call unlinks routines from a buffer linked with `Buffer_LinkDevice`. It does not call the owner `changed=buffer_ownerchange` routine.

Before calling

R0 Buffer handle

On return

--

[RISC OS Open source page](#)

CD_AudioLevel

SWI &4125F · CDFSDriver

This call is deprecated. Do not use.

Before calling

R0-R6 ???

R7 CDFS Control Block indicating target device

On return

???

Notes

`CD_GetAudioParms` and `CD_SetAudioParms` can be used to control the audio volume level.

[RISC OS Open source page](#)

CD_AudioPause

SWI &4124D · CDFSDriver

The purpose of this call is to provide a method of pausing audio playback using the CD hardware.

Before calling

R0 0 to turn off pausing, if pause was in effect, or 1 to turn on pausing if audio was playing

R7 Pointer to CDFS Control Block

On return

[RISC OS Open source page](#)

CD_AudioStatus

SWI &41254 · CDFSDriver

The purpose of this call is to return the current audio status of the CD Drive.

Before calling

R7 Pointer to CDFS Control Block

On return

R0 Audio Status Value

R7 Preserved

Audio Status Values

Value / Meaning

0 – Meaning: Currently playing audio

1 – Meaning: Currently paused

2 – Meaning: Reserved (at present)

3 – Meaning: Audio play successfully completed

4 – Meaning: Error occurred during last audio play

5 – Meaning: Audio play has not been requested

Notes

None.

[RISC OS Open source page](#)

CD_ByteCopy

SWI &41262 · CDFSDriver

The original purpose of this call was to provide a fast memory copy routine, providing efficient way of copying any number of bytes, word-aligned or not.

Before calling

R1 Pointer to source

R2 Pointer to destination

R3 Number of bytes to copy

On return

R1 Preserved

R2 Preserved

R3 Preserved

Notes

It is recommended that you do not use this, as the overhead for calling the SWI is probably greater than any possible gain.

[RISC OS Open source page](#)

CD_CheckDrive

SWI &41250 · CDFSDriver

The purpose of this call is to check if a disc is present in the drive.

Before calling

R7 Pointer to CDFS Control Block

On return

R0 0 if CD in drive, error otherwise

Notes

None.

[RISC OS Open source page](#)

CD_CloseDrawer

SWI &4125B · CDFSDriver

The purpose of this call is to close the CD Drawer.

Before calling

R7 Pointer to CDFS Control Block

On return

--

Notes

Not all Drives support the function to close the drawer using software.

[RISC OS Open source page](#)

CD_Control

SWI &41257 · CDFSDriver

The purpose of this call is to initialise the CD Drive.

Before calling

R0 1

R7 Pointer to CDFS Control Block

On return

--

Notes

None.

[RISC OS Open source page](#)

CD_ConvertToLBA

SWI &41264 · CDFSDriver

The purpose of this call is to convert an address in the addressing mode supplied to logical block addressing (LBA, mode 0) form.

Before calling**R0** Addressing mode**R1** Address to convert**R7** Pointer to CDFS Control Block**On return****R1** Address in LBA format**Notes**

None.

[RISC OS Open source page](#)

CD_ConvertToMSF

SWI &41265 · CDFSDriver

The purpose of this call is to convert a block address to MSF, in the form &00MMSSFF.

Before calling**R0** Addressing mode**R1** Address in given mode**R7** Pointer to CDFS Control Block**On return****R1** Returned 32-bit value of MSF address**Notes**

Before use with CD_PlayAudio should have a value added to it to allow for lead-in time of audio CDs. (e.g. 512=2 sec).

[RISC OS Open source page](#)

CD_DiscChanged

SWI &41251 · CDFSDriver

The purpose of this call is to check if the CD drive has changed or not since the last call to this SWI.

Before calling

R7 Pointer to CDFS Control Block

On return

R0 Changed flag (0 if unchanged, 1 if changed)

Notes

None.

[RISC OS Open source page](#)

CD_DiscHasChanged

SWI &41256 · CDFSDriver

The purpose of this call is to mark the CD disc as having changed.

Before calling

R7 Pointer to CDFS Control Block

On return

--

Notes

None.

[RISC OS Open source page](#)

CD_DiscUsed

SWI &41253 · CDFSDriver

The purpose of this call is to return the size of CD currently in drive.

Before calling

R0 Addressing mode for returned size

R1 Pointer to 8-byte block

R7 Pointer to CDFS Control Block

On return

R0 Preserved

R1 Corrupted

R7 Preserved

8-byte block details:

Offset / Contents

0 – Contents: Size of CD

4 – Contents: Size of blocks on CD

Notes

The size of a CD is calculated based on $(((\text{min} * 60) + \text{sec}) * 75) + 151$ frames.

[RISC OS Open source page](#)

CD_DriveReady

SWI &41244 · CDFSDriver

The purpose of this call is to return the current state of drive, saying whether the drive is ready to accept commands or not.

Before calling

R7 Pointer to CDFS Control Block

On return

R0 Drive state (0 if ready, otherwise not ready)

Notes

None.

[RISC OS Open source page](#)

CD_DriveStatus

SWI &41243 · CDFSDriver

The purpose of this call is to return the current state of drive, saying whether the drive is ready to accept commands or not.

Before calling

R7 Pointer to CDFS Control Block

On return

R0 Status

Notes

None.

[RISC OS Open source page](#)

CD_EjectButton

SWI &41248 · CDFSDriver

This call enables or disables CD ejection. It is used by the *Lock command.

Before calling

R0 0 (to enable ejection), 1 (to disable ejection)

R7 Pointer to CDFS Control Block

On return

R0, R7 Preserved

Notes

Disabling ejection will prevent both software-initiated ejection (i.e. CD_OpenDrawer) and drive-initiated ejection (i.e. when the user presses the physical eject button on the drive).

[RISC OS Open source page](#)

CD_EnquireAddress

SWI &41249 · CDFSDriver

This call returns the current address of the head.

Before calling

R0 Addressing mode for returned head location

R7 Pointer to CDFS Control Block

On return

R0 Head address

R7 Preserved

Notes

Can be used while playing audio, to determine which section of audio is playing.

[RISC OS Open source page](#)

CD_EnquireDataMode

SWI &4124A · CDFSDriver

The purpose of this call is to query the data mode that is used by a particular disc sector.

Before calling

R0 Address mode

R1 CD address

R7 Pointer to CDFS Control Block

On return

R0 Data Mode

Notes

None.

[RISC OS Open source page](#)

CD_EnquireTrack

SWI &4124E · CDFSDriver

The purpose of this call is to return track range, or info on a specified track.

Before calling

R0 0 to return track range, or a valid track number

R1 Pointer to 5 byte track block

R7 Pointer to CDFS Control Block

On return

R0 Preserved

R1 Preserved

R7 Preserved

5 Byte Track Block

If **R0** = 0 on entry, the track range is returned in the block as follows:

Offset / Contents

0 – Contents: First track on CD (1-99)

1 – Contents: Last track on CD (1-99)

2 - 4 – Contents: Corrupted

If **R0** is a valid track number on entry, then information about that track is returned in the block:

Offset / Contents

0 – Contents: Logical block address of track start (ie Mode 0 address)

4 – Contents: Audio control bits specifying the type of information the track holds:

/ Bit / Meaning

– Bit: 0 · Meaning: 0 = audio, 1 = data

– Bit: 1 · Meaning: 0 = 4 channel sound, 1 = 2 channel sound

– Bit: 2-7 · Meaning: Reserved

Notes

Track numbers do not necessarily start at 1.

[RISC OS Open source page](#)

CD_GetAudioParms

SWI &41269 · CDFSDriver

This SWI can be used to read parameters relating to audio playback.

Before calling

R0 Reason code

R1-R6 Dependent on reason code

R7 Pointer to CDFS Control Block

On return

R0 -1 if reason code not supported (V flag clear)

Error pointer + V set if some other error

Else preserved

R1-R6 Dependent on reason code

Reason Codes

Value / Meaning

0 – Meaning: Get volume levels

Notes

CD_Supported indicates whether this SWI is supported.

[RISC OS Open source page](#)

CD_GetAudioParms 0

SWI &41269 · CDFSDriver

This SWI can be used to read the volume levels.

Before calling

R0 0 (Reason code)

R1 Pointer to word-aligned block to receive result

R7 Pointer to CDFS Control Block

On return

R0 -1 if reason code not supported (V flag clear)

Error pointer + V set if some other error

Else preserved, and block at R1 filled in

Result block

Word / Use

0 – Use: Channel 0 volume level

1 – Use: Channel 1 volume level

Notes

Volume levels are in the range 0 to 0xffff inclusive, where 0 = mute and 0xffff = max.

CD_Supported indicates whether this SWI is supported.

[RISC OS Open source page](#)

CD_GetParameters

SWI &41245 · CDFSDriver

The purpose of this call is to return the current data read parameters.

Before calling

R0 Pointer to 16 byte track block

R7 Pointer to CDFS Control Block

On return

R0 Buffer block updated

Notes

None.

[RISC OS Open source page](#)

CD_Identify

SWI &41263 · CDFSDriver

The purpose of this call is to identify the drive type of a CD-ROM drive.

Before calling

R7 Pointer to CDFS Control Block (drive type is ignored)

On return

R0 Preserved

R1 Preserved

R2 Drive type, or -1 if not recognised

In practice

When this call is passed onto registered CDFS Drivers, they receive additional values:

R0 / 36 byte SCSI inquiry data or 0

R1 – 36 byte SCSI inquiry data or 0: if R0=0 R1 points to an error block (the SCSI inquiry failed)

Notes

None.

[RISC OS Open source page](#)

CD_Inquiry

SWI &41255 · CDFSDriver

The purpose of this call is to return the CD-ROM device information.

Before calling

R0 Pointer to 36 byte block

R7 Pointer to CDFS Control Block

On return

R0 Buffer updated

36 Byte Block

Offset / Contents

0 – Contents: Device Type

4 – Contents: Length of block - 4

5 - 6 – Contents: Reserved

7 – Contents: Interface Flags

8 – Contents: Vendor identification, space padded to 8 characters

16 – Contents: Product identification, space padded to 16 characters

32 – Contents: Product revision level, space padded to 4 characters

Device Type

0 - 4 Device Type

0 – Meaning: Direct access device

5 – Meaning: CD-ROM device

7 – Meaning: Optical memory device

8 – Meaning: Medium changer device

5-7 Device qualifier

0 – Meaning: Device is connected

1 – Meaning: Device is not connected

3 – Meaning: Target doesn't support any device

15 – Meaning: Device is removable

16 - 18 – Meaning: ANSI-approved version supported

19 - 21 – Meaning: ECMA version supported

22 - 23 – Meaning: ISO version supported

24 - 27 Response format

0 – Meaning: Follows SCSI I format

1 – Meaning: Compatible with pre-SCSI I devices

2 – Meaning: Follows SCSI II format

28 - 29 – Meaning: Reserved

30 – Meaning: Device supports Terminate I/O Process message

31 – Meaning: Device supports asynchronous event notification

Interface Flags

0 – Meaning: Device responds to the reset condition with soft reset

1 – Meaning: Supports tagged command queuing

2 – Meaning: Reserved

3 – Meaning: Supports linked commands

- 4 – Meaning: Supports synchronous data transfer
- 5 – Meaning: Supports 16 bit data transfers
- 6 – Meaning: Supports 32 bit data transfers
- 7 – Meaning: Supports relative addressing mode

Notes

Do not rely too much on some of the SCSI specific values making sense, as not all drives are true SCSI devices.

[RISC OS Open source page](#)

CD_IsDrawerLocked

SWI &4125C · CDFSDriver

The purpose of this call is to check whether the CD drawer is locked.

Before calling

R7 Pointer to CDFS Control Block

On return

R0 0 if not locked, 1 if locked, or an error if not supported

Notes

None.

[RISC OS Open source page](#)

CD_LastError

SWI &4125E · CDFSDriver

The purpose of this call is to return the last error number from the CDFSDriver.

Before calling

--

On return

R0 Last error number

R7 Pointer to CDFS Control Block

Notes

None.

[RISC OS Open source page](#)

CD_OpenDrawer

SWI &41247 · CDFSDriver

This call stops the CD spinning, opens the drawer and/or ejects the caddy.

Before calling

R7 Pointer to CDFS Control Block

On return

R7 Preserved

Notes

An error is returned if the drawer has previously been locked using CD_EjectButton or *Lock.

[RISC OS Open source page](#)

CD_PlayAudio

SWI &4124B · CDFSDriver

This call attempts to play a section of CD as audio, specified by address. If the section contains non-audio information, the results are drive-dependent; you will not necessarily get an error returned, and some drives may still attempt to play the data. Calling CD_EnquireTrack to find out if the section you wish to play consists entirely of audio is recommended.

Before calling

R0 Addressing mode of parameters

R1 Start address of play section

R2 End address of play section

R7 Pointer to CDFS Control Block

On return

R0 Preserved

R1 Preserved

R2 Preserved

R7 Preserved

[RISC OS Open source page](#)

CD_PlayTrack

SWI &4124C · CDFSDriver

The purpose of this call is to attempt to play section of CD audio, specified by track.

Before calling

R0 Track number at which to start playing (1-99)

R1 255 to play to end of CD, or 254 to play to end of track

R7 Pointer to CDFS Control Block

On return

R0 Preserved

R1 Preserved

R7 Preserved

Notes

Returns error if track is non-audio.

[RISC OS Open source page](#)

CD_Prefetch

SWI &41259 · CDFSDriver

The purpose of this call is to cause the drive head to move to the indicated position and start prefetching data into the drives' cache, improving access times if the next read operation targets that address.

Before calling

R0 Addressing mode

R1 Address

R7 Pointer to CDFS Control Block

On return

--

Notes

CD_Supported can be used to determine whether the drive supports this operation.

[RISC OS Open source page](#)

CD_ReadAudio

SWI &41266 · CDFSDriver

The purpose of this call is to read audio data from disc, if the sector lies in an audio track.

Before calling

R0 Addressing mode

R1 Start address of section to grab

R2 Number of 2352-byte blocks to grab

R3 Pointer to destination buffer (multiple of 256)

R4 Flags:

Bits 0-7: 0 = read PCM, non-zero = reserved

Bits 8-31: Reserved (zero)

R7 Pointer to CDFS Control Block

On return

--

Notes

CD_Supported can be used to determine whether the drive supports this operation.

[RISC OS Open source page](#)

CD_ReadData

SWI &41241 · CDFSDriver

The purpose of this call is to read data from the CD, using the current mode .

Before calling

R0 Addressing mode

R1 Start address in given mode

R2 Number of blocks to read

R3 Pointer to destination buffer

R4 Number of bytes from each block to read (normally 2048)

R7 Pointer to CDFS Control Block

On return

--

Notes

Note for driver writers: the value supplied in R4 may not be valid when CDFS supplies it to the driver entry point.

See CD_SetParameters and CD_EnquireDataMode for more information on the current mode.

[RISC OS Open source page](#)

CD_ReadSubChannel

SWI &4124F · CDFSDriver

The purpose of this call is to return subchannel information encoded on the CD.

Before calling

R0 &40

R1 Pointer to subchannel block

R7 Pointer to CDFS Control Block

On return

R0 Preserved

R1 Preserved

R7 Preserved

Subchannel Block

Offset / Contents

0 – Contents: LBA address of current head position relative to track start

4 – Contents: Absolute LBA address

8 – Contents: Audio flags:

Bit 0 : 0 = Audio, 1 = data

Bit 1: 0 = 4 channels, 1 = 2 channels

9 – Contents: Track number

10 – Contents: Index number

Notes

None.

[RISC OS Open source page](#)

CD_ReadUserData

SWI &4124F · CDFSDriver

The purpose of this call is to return subchannel information encoded in the CD.

Before calling

R0 &40

R1 Pointer to subchannel block

R7 Pointer to CDFS Control Block

On return

R0 Preserved

R1 Preserved

R7 Preserved

Subchannel Block

Offset / Contents

0 – Contents: LBA address of current head position relative to track start

4 – Contents: Absolute LBA address

8 – Contents: Audio flags:

Bit 0 : 0 = Audio, 1 = data

Bit 1: 0 = 4 channels, 1 = 2 channels

9 – Contents: Track number

10 – Contents: Index number

Notes

None.

[RISC OS Open source page](#)

CD_Register

SWI &41260 · CDFSDriver

The purpose of this call is to register the soft-loadable driver with the CDFSDriver module.

Before calling

R0 Pointer to Driver Information Block

R1 Pointer to a special tag (&EE50EE50 = 1 word) followed by the CDFSDriver handler code

R2 Pointer to workspace for the soft-loadable module (passed in R12 to the handler)

On return

R0 Block updated

Notes

The block pointed to by R0 on entry must be in RAM, whereas anything else, i.e. the code for each CD _SWI, can be in ROM. The block must stay put in the memory so that CDFSDriver can check it when detecting drive types.

[RISC OS Open source page](#)

CD_Reset

SWI &4125A · CDFSDriver

The purpose of this call is to reset the specified drive.

Before calling

R7 Pointer to CDFS Control Block

On return

--

Notes

None.

[RISC OS Open source page](#)

CD_SCSIUserOp

SWI &4126B · CDFSDriver

When you want to issue a SCSI command to a drive, you should use this call, rather than SCSI_Op?. This will ensure that drives which aren't part of the SCSI stack (e.g. ATAPI drives) will receive the command.

Before calling

- R0** Data Transfer (Value of Bits 24 - 25)
- R1** Length of SCSI control block
- R2** SCSI control block
- R3** buffer for data transfer
- R4** Length of transfer
- R5** Timeout in centi-seconds (0 = default)
- R7** Pointer to CDFS Control Block

On return

- R3** Updated
- R4** Updated
- R5** Preserved
- R6** Preserved
- R7** Preserved
- R8** Preserved
- R9** Preserved

Flags

NZC = corrupted If V set R0-R2 are as for SWI SCSI_Op? or R0 error not in SCSI range and R1,R2 corrupt else R0-R2 = preserved

Data Transfer Values

Bit Value / Meaning

- 0** – Meaning: No transfer
- 1** – Meaning: Read
- 2** – Meaning: Write
- 3** – Meaning: Reserved

Notes

CD_SCSIUserOp must not be called in the background (for compatibility with ATAPI drives).

[RISC OS Open source page](#)

CD_SeekTo

SWI &41242 · CDFSDriver

The purpose of this call is to seeks the CD drive head to a given position.

Before calling

R0 Addressing mode

R1 Address to seek to

R7 Pointer to CDFS Control Block

On return

R0 -

R1 -

R7 -

Notes

The drive may not perform this physical action, at its discretion, but should behave as if it did.

[RISC OS Open source page](#)

CD_SetAudioParms

SWI &4126A · CDFSDriver

This SWI can be used to set parameters relating to audio playback.

Before calling

R0 Reason code

R1-R6 Dependent on reason code

R7 Pointer to CDFS Control Block

On return

R0 -1 if reason code not supported (V flag clear)

Error pointer + V set if some other error

Else preserved

R1-R6 Dependent on reason code

Reason Codes

Value / Meaning

0 – Meaning: Set volume levels

Notes

CD_Supported indicates whether this SWI is supported.

[RISC OS Open source page](#)

CD_SetAudioParms 0

SWI &4126A · CDFSDriver

This SWI can be used to set the audio volume levels.

Before calling

R0 0 (Reason code)

R1 Pointer to word-aligned parameter block

R7 Pointer to CDFS Control Block

On return

R0 -1 if reason code not supported (V flag clear)

Error pointer + V set if some other error

Else preserved

Parameter block

Word / Use

0 – Use: Channel 0 volume level

1 – Use: Channel 1 volume level

Notes

Volume levels are in the range 0 to 0xffff inclusive, where 0 = mute and 0xffff = max.

CD_Supported indicates whether this SWI is supported.

[RISC OS Open source page](#)

CD_SetParameters

SWI &41246 · CDFSDriver

The purpose of this call is to set the current data read parameters.

Before calling

R0 Pointer to 16 byte track block

R7 Pointer to CDFS Control Block

On return

--

Notes

None.

[RISC OS Open source page](#)

CD_StopDisc

SWI &41252 · CDFSDriver

The purpose of this call is to stop a CD playing, and wait until the CD stops spinning.

Before calling

R7 Pointer to CDFS Control Block

On return

R7 Preserved

[RISC OS Open source page](#)

CD_Supported

SWI &41256 · CDFSDriver

The purpose of this call is to check features the CD drive / driver supports.

Before calling

R7 Pointer to CDFS Control Block

On return

R0 Feature support flags

Feature Support Flags

0 - 2 – Meaning: Level to which CD_AudioControl? is supported (0 to 7)

– – Meaning: 0 = no support at all

– – Meaning: 1 = supported to level 0

– – Meaning: 7 = supported to level 6

3 – Meaning: Supports CD_Prefetch

4 – Meaning: Supports CD_CloseDrawer

5 – Meaning: Supports CD_AudioLevel

6-13 – Meaning: CD ROM speed - in multiples up to 255 x speed, 255 if unknown

14 – Meaning: Supports CD_ReadAudio

15 – Meaning: Supports CD_GetAudioParms and CD_SetAudioParms

Notes

None.

[RISC OS Open source page](#)

CD_Unregister

SWI &41261 · CDFSDriver

The purpose of this call is to remove the soft-loadable driver from the CDFSDriver module.

Before calling

R0 Pointer to Driver Information Block

R1 Pointer to a special tag (&EE50EE50 = 1 word) followed by the handler code

On return

--

Notes

The block pointed to by R0 on entry must be in RAM, whereas anything else, i.e. the code for each CD _ SWI, can be in the ROM.

[RISC OS Open source page](#)

CD_Version

SWI &41240 · CDFSDriver

The purpose of this call is to read the version of the CD module installed.

Before calling

On return

R0 Pointer to Version Block

Version Block

The following table details the layout of the returned block.

Offset / Contents

0 – Contents: Version number

4 – Contents: Version string

Notes

None.

[RISC OS Open source page](#)

CDFS_ConvertDriveToDevice

SWI &41E80 · CDFS

This call returns the device details for a specified CD drive number.

Before calling

R0 Drive number (0 = 1st CD drive)

On return

R0 Preserved

R1 &FFFFFFF if an error occurs, or Physical Drive Number

[RISC OS Open source page](#)

CDFS_DescribeDisc

SWI &41E86 · CDFS

This call returns information about the disc in the specified CD-ROM drive.

Before calling

R0 Drive number (0 = 1st CD drive)

R1 Word aligned 64 byte block

On return

R0 Preserved

R1 Preserved

In practice

Block pointed to by R1 updated:

+0 – Value: Disc type

+1 – Value: Disc mode

+2 – Value: Reserved

+3 – Value: LBA of Volume Description

+4 – Value: Size of disc

+8 – Value: LBA of Root Directory

+12 – Value: Size of Root Directory

+16 – Value: Block Size

+20 – Value: Disc name (0 terminated)

+56..64 – Value: Reserved

[RISC OS Open source page](#)

CDFS_GetBufferSize

SWI &41E82 · CDFS

Returns the size of the CDFS buffer stored in CMOS RAM. Buffer sizes are:

On return

R0 Buffer Size

R10 Corrupted

0 – Size: 0 KB

1 – Size: 8 KB

2 – Size: 16 KB

3 – Size: 32 KB

4 – Size: 64 KB

5 – Size: 128 KB

6 – Size: 256 KB

7 – Size: 512 KB

[RISC OS Open source page](#)

CDFS_GetNumberOfDrives

SWI &41E84 · CDFS

Returns the number of CD-ROM drives configured in CMOS RAM.

On return

R0 Number of CD-ROM drives

R10 Corrupted

[RISC OS Open source page](#)

CDFS_GiveFileType

SWI &41E85 · CDFS

Converts a PC style filename (eg README.TXT) into a RISC OS equivalent FileType number. If no matching filetype is found, R1=0 on exit.

Before calling

R0 Pointer to a filename

On return

R0 Preserved

R1 RISC OS FileType (0-&FFF)

Notes

This SWI is deprecated, use MimeMap_Translate? in preference. Originally CDFS managed its own list of filetype mappings, once the MimeMap module was introduced CDFS was changed to just act as a wrapper around that.

[RISC OS Open source page](#)

CDFS_SetBufferSize

SWI &41E81 · CDFS

Set the size of the CDFS buffer in CMOS RAM. Valid buffer sizes are:

Before calling

R0 Buffer Size (0-7)

On return

R0 Preserved

0 – Size: 0 KB

1 – Size: 8 KB

2 – Size: 16 KB

3 – Size: 32 KB

4 – Size: 64 KB

5 – Size: 128 KB

6 – Size: 256 KB

7 – Size: 512 KB

[RISC OS Open source page](#)

CDFS_SetNumberOfDrives

SWI &41E83 · CDFS

Set the number of the CD-ROM drives in CMOS RAM.

Before calling

R0 Number of CD-ROM drives (0-31)

On return

R0 Preserved

[RISC OS Open source page](#)

CDFS_Truncation

SWI &41E88 · CDFS

Set the filename truncation type used when shortening CD filenames? Truncation methods are:

Before calling

R0 0 to read current setting, or 1 to set truncation type

R1 Truncation type, if R0=1 on entry

On return

R0 Preserved

R1 If R0=0 on entry, R1=current truncation type, otherwise Preserved

Value / Truncation type

-1 – Truncation type: Use default for os version

0 – Truncation type: Truncate from right (default for RISC OS 2.00)

1 – Truncation type: Truncate from the left

2 – Truncation type: No truncation (default for RISC OS 3.00 and newer)

[RISC OS Open source page](#)

CDFS_WhereIsFile

SWI &41E87 · CDFS

Returns the LBA start address of a file.

Before calling

R0 Pathname

On return

R0 Preserved

R1 LBA start of file, or -1 if a directory

R2 Size of file in bytes, or Preserved if a directory

R10 Corrupted

[RISC OS Open source page](#)

ColourPicker_CloseDialogue

SWI &47703 · ColourPicker

The purpose of this call is to close a colour picker dialogue which is in progress.

Before calling

R0 Flags. (Reserved, all bits must be zero)

R1 Dialogue handle

On return

R0 Preserved

R1 Preserved

Notes

This may either be done by the client, or by the Wimp if the dialogue was set to be transient when created.

It is normally called in response to Message_ColourPickerCloseDialogueRequest.

[RISC OS Open source page](#)

ColourPicker_DeregisterModel

SWI &47701 · ColourPicker

This call is for internal use only and must not be used.

Before calling

--

On return

--

Notes

None.

[RISC OS Open source page](#)

ColourPicker_HelpReply

SWI &47707 · ColourPicker

The purpose of this call is to make the colour picker respond to a `Message_HelpRequest` with its own help text.

Before calling

R0 HelpReply flags (Reserved, must all be zero)

R1 Pointer to `Message_HelpRequest` message block, as returned from `Wimp_Poll`

On return

R0 Preserved

R1 Preserved

Notes

It is typically used by a client that wishes to replace part of the colour picker's help system.

Such a client would, on creating and opening a colour picker dialogue, force it to ignore help requests. The client would then selectively respond to help requests itself, or use this call to make the colour picker respond instead.

[RISC OS Open source page](#)

ColourPicker_ModelSWI

SWI &47708 · ColourPicker

This call is for internal use only and must not be used.

Before calling

--

On return

--

Notes

None.

[RISC OS Open source page](#)

ColourPicker_OpenDialogue

SWI &47702 · ColourPicker

The purpose of this call is to create and open a colour picker dialogue for a client.

Before calling

R0 Dialogue flags

R1 Pointer to colour picker block

On return

R0 Picker handle

R1 Window handle

Dialogue Flags

Bit / Value / Meaning

0 - 1 – Meaning: Dialogue type

0 · Normal Dialogue

1 · Menu Dialogue

2 · Toolbox dialogue (has Close and Back icons instead of OK and Cancel buttons)

3 · Submenu dialogue

4 - 31 – Value: 0 · Meaning: Reserved (must be zero)

Notes

The block passed need not be static, and defines the initial values of the window.

When using menu-like pickers you should always accept the Message_MenusDeleted otherwise things will break.

SWI is not re-entrant.

[RISC OS Open source page](#)

ColourPicker_ReadDialogue

SWI &47705 · ColourPicker

The purpose of this call is to read the current state of a colour picker dialogue.

Before calling

R0 ReadDialogue flags (Reserved, must all be zero)

R1 Dialogue handle as returned by ColourPicker_OpenDialogue

R2 Pointer to colour picker block

On return

R0 Preserved

R1 Required size of block, if queried

Notes

SWI is not re-entrant

[RISC OS Open source page](#)

ColourPicker_RegisterModel

SWI &47700 · ColourPicker

This call is for internal use only and must not be used.

Before calling

--

On return

--

Notes

None.

[RISC OS Open source page](#)

ColourPicker_SetColour

SWI &47706 · ColourPicker

This call is reserved for future expansion; you must not use it in your own code.

Before calling

--

On return

--

Notes

None.

[RISC OS Open source page](#)

ColourPicker_UpdateDialogue

SWI &47704 · ColourPicker

The purpose of this call is to update some or all of the contents of a colour picker dialogue.

Before calling

R0 Dialogue Update flags

R1 Pointer to colour picker block

On return

R0 Picker handle

R1 Window handle

Dialogue Update Flags

- 0 – Update part of dialogue when set: Whether the dialogue has a None button
- 1 – Update part of dialogue when set: Whether the None button is selected
- 2 – Update part of dialogue when set: Where Message_ColourPickerColourChanged is issued for
- 3 – Update part of dialogue when set: Visible area
- 4 – Update part of dialogue when set: Reserved (must be 0)
- 5 – Update part of dialogue when set: Window title
- 6 – Update part of dialogue when set: Colour setting
- 7 – Update part of dialogue when set: Colour model and setting (including optional data)
- 8 – Update part of dialogue when set: Whether the dialogue ignores Message_HelpRequest
- 9 – Update part of dialogue when set: Whether the dialogue passes on unhandled keypresses

Notes

The new values are taken from the passed colour picker block; all other parts of the block are ignored. The block need only be large enough to hold the highest offset field required.

The None button may be added or removed, and its setting adjusted. The title, setting and colour model may also be adjusted independently of each other.

If you are changing the visible area (ie bit 3 is set), you must fill in offsets 8 – 20 inclusive of the colour picker block including the reserved words. This is for future compatibility.

If bit 7 is set then bit 6 is ignored; the colour model is updated, and the RGB triplet is calculated from the data in the colour model block. If bit 6 is set and bit 7 is clear, then the colour setting is updated from the colour descriptor's initial RGB triplet and the colour model left unchanged, even if the current colour model isn't that in the colour descriptor.

SWI is not re-entrant.

[RISC OS Open source page](#)

ColourTrans_ColourNumberToGCOL

SWI &4074D · ColourTrans

The purpose of this call is to translate a colour number into a GCOL [colour] number.

Before calling

R0 Colour number

On return

R0 GCOL

Notes

This call should only be used within 256 colour mode. If used on a non-256 colour mode, the results of this call will be meaningless.

[RISC OS Open source page](#)

ColourTrans_ConvertCIEToRGB

SWI &40756 · ColourTrans

The purpose of this call is to convert the colour passed as CIE values to RGB values.

Before calling

R0 CIE x tristimulus value

R1 CIE y tristimulus value

R2 CIE z tristimulus value

On return

R0 Red component

R1 Green component

R2 Blue component

Notes

The parameters passed are fixed point 32-bit numbers. The lower 16 bits (0 – 15) are used below the decimal point, and the upper 16 bits above the decimal point.

[RISC OS Open source page](#)

ColourTrans_ConvertCMYKToRGB

SWI &4075B · ColourTrans

The purpose of this call is to convert the colour passed as CMYK values to RGB values.

Before calling

R0 Cyan component

R1 Magenta component

R2 Yellow component

R3 Key (black) component

On return

R0 Red component

R1 Green component

R2 Blue component

Notes

The parameters passed are fixed point 32-bit numbers. The lower 16 bits (0 – 15) are used below the decimal point, and the upper 16 bits above the decimal point.

[RISC OS Open source page](#)

ColourTrans_ConvertDeviceColour

SWI &40753 · ColourTrans

The purpose of this call is to convert a device colour to a standard colour.

Before calling

R1 24-bit device colour

R3 Pointer to calibration table, or 0 to use current calibration

On return

R2 24-bit standard colour

Notes

The 24-bit colours are passed in the &BBGGRR00 format.

[RISC OS Open source page](#)

ColourTrans_ConvertDevicePalette

SWI &40754 · ColourTrans

The purpose of this call is to convert a device palette to standard colours.

Before calling

R0 Number of colours to convert

R1 Pointer to table of 24-bit device colours

R2 Pointer to table to store standard colours

R3 Pointer to calibration table, or 0 to use current calibration

On return

- All registers preserved

Notes

The 24-bit colours are stored in the &BBGGRR00 format.

[RISC OS Open source page](#)

ColourTrans_ConvertHSVToRGB

SWI &40759 · ColourTrans

The purpose of this call is to convert the colour passed as CMYK values to RGB values.

Before calling

R0 Hue (0 – 360)

R1 Saturation

R2 Value

On return

R0 Red component

R1 Green component

R2 Blue component

Notes

The parameters passed are fixed point 32-bit numbers. The lower 16 bits (0 – 15) are used below the decimal point, and the upper 16 bits above the decimal point.

Hue values range from 0 – 360 (without any fractional element). Saturation and Value values range from 0 – 1 (including any fractional element).

Red, Green and Blue values returned are in the range 0 – 1.

An error will be generated if both the hue and saturation are both 0.

[RISC OS Open source page](#)

ColourTrans_ConvertRGBToCIE

SWI &40755 · ColourTrans

The purpose of this call is to convert the colour passed as RGB values to CIE values.

Before calling

R0 Red component

R1 Green component

R2 Blue component

On return

R0 CIE x tristimulus value

R1 CIE y tristimulus value

R2 CIE z tristimulus value

Notes

The parameters passed are fixed point 32-bit numbers. The lower 16 bits (0 – 15) are used below the decimal point, and the upper 16 bits above the decimal point.

[RISC OS Open source page](#)

ColourTrans_ConvertRGBToCMYK

SWI &4075A · ColourTrans

The purpose of this call is to convert the colour passed as RGB values to CMYK values.

Before calling

R0 Red component

R1 Green component

R2 Blue component

On return

R0 Cyan component

R1 Magenta component

R2 Yellow component

R3 Key (black) component

Notes

The parameters passed are fixed point 32-bit numbers. The lower 16 bits (0 – 15) are used below the decimal point, and the upper 16 bits above the decimal point.

[RISC OS Open source page](#)

ColourTrans_ConvertRGBToHSV

SWI &40758 · ColourTrans

The purpose of this call is to convert the colour passed as RGB values to HSV values.

Before calling

R0 Red component

R1 Green component

R2 Blue component

On return

R0 Hue (0 – 360)

R1 Saturation

R2 Value

Notes

The parameters passed are fixed point 32-bit numbers. The lower 16 bits (0 – 15) are used below the decimal point, and the upper 16 bits above the decimal point.

Hue values range from 0 – 360 (without any fractional element).

Hue is undefined for achromatic colours.

[RISC OS Open source page](#)

ColourTrans_GCOLToColourNumber

SWI &4074C · ColourTrans

The purpose of this call is to translate a GCOL number to a colour number.

Before calling

R0 GCOL

On return

R0 Colour number

Notes

This call should only be used within 256 colour mode. If used on a non-256 colour mode, the results of this call will be meaningless.

[RISC OS Open source page](#)

ColourTrans_GenerateTable

SWI &40763 · ColourTrans

The purpose of this call is to set up a translation table in a buffer that is used by scaled sprite plotting.

Before calling

R0 Source mode / sprite area

R1 Source palette pointer / sprite

R2 Destination mode, or -1 for current mode

R3 Destination palette pointer, or

-1 for current palette, or

0 for default for the mode

R4 Pointer to buffer, or 0 to return length

R5 Select Table Flags

R6 Pointer to transfer workspace (R12) (if R5 valid and bit 2 of R5 is set)

R7 Pointer to transfer function (if R5 valid and bit 2 of R5 is set)

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Required size of buffer (if R4=0 on entry), or preserved

R5 Preserved

R6 Preserved

R7 Preserved

Notes

Unlike `ColourTrans_SelectTable`, this call always uses the flags passed in R5.

R6 and R7 are only used if bit 2 of R5 is set.

Sometimes `ColourTrans` may decide that no translation table is necessary (e.g. source and dest have the same pixel format and palette), in which case the “measure table size” form of the call will return with R4=0. Take care to check for this before allocating memory and asking for the table to be filled in.

[RISC OS Open source page](#)

ColourTrans_InvalidateCache

SWI &40750 · `ColourTrans`

The purpose of this call is to inform `ColourTrans` that the palette has changed.

Before calling

--

On return

--

Notes

This call must be used whenever the palette has changed so that `ColourTrans` can update its own colour cache. This is not required, however, if a mode change has occurred as `ColourTrans` automatically updates detects this.

[RISC OS Open source page](#)

ColourTrans_MiscOp

SWI &4075F · `ColourTrans`

This call is for internal use only, and should not be used.

Before calling

--

On return

--

[RISC OS Open source page](#)

ColourTrans_ReadCalibration

SWI &40752 · `ColourTrans`

This purpose of this call is to read the calibration table, and copy it into a buffer.

Before calling

R0 Pointer to buffer (0 to read size of buffer)

On return

R0 Preserved

R1 Size of table if R0 = 0 on entry, else preserved

Notes

The call can also be used to read the size of the calibration table. It is recommended that the calibration's table size is first calculated and then a buffer large enough to copy the table is created.

[RISC OS Open source page](#)

ColourTrans_ReadPalette

SWI &4075C · ColourTrans

The purpose of this call is to read either the screen's palette, or a sprite's palette.

Before calling

R0 Source mode / sprite area

R1 Source palette pointer / sprite

R2 Pointer to buffer, or 0 to return size needed in R3

R3 Size of buffer (if R2 is non-zero)

R4 Flags

On return

R2 Incremented by words written

R3 Remaining size of buffer

Notes

All entries are returned as true 24-bit RGB, calibrated if required.

In 256 colour modes the full 256 palette entries are returned, rather than the 16 used by VIDC 1.

[RISC OS Open source page](#)

ColourTrans_ReturnColourNumber

SWI &40744 · ColourTrans

This purpose of this call is to return the closest colour number in the current screen mode and palette.

Before calling

R0 Palette entry

On return

R0 Colour number

Notes

The 24-bit palette entries are in the &BBGRR00 format.

[RISC OS Open source page](#)

ColourTrans_ReturnColourNumberForMode

SWI &40746 · ColourTrans

This purpose of this call is to return the closest colour number, given a specific palette entry with the destination screen mode and palette.

Before calling

- R0** Palette entry
- R1** Destination mode, or -1 for current mode
- R2** Destination palette pointer, or
-1 for current palette, or
0 for default for the current mode

On return

- R0** Colour number
- R1** Preserved
- R2** Preserved

Notes

The 24-bit palette entries are in the &BBGGRR00 format.

[RISC OS Open source page](#)

ColourTrans_ReturnFontColours

SWI &4074E · ColourTrans

This purpose of this call is to return the maximum range of colour than can be sensibly used, given the background, foreground and number of anti-aliasing colour desired.

Before calling

- R0** Font handle, or 0 for current font
- R1** Background palette entry
- R2** Foreground palette entry
- R3** Maximum foreground colour offset (0-14)

On return

- R0** Preserved
- R1** Background logical colour (preserved if in 256 colour mode)
- R2** Foreground logical colour
- R3** Maximum sensible colour offset (up to R3 on entry)

Notes

For any given palette entries passed in R1 and R2, it finds the best fit in the current palette, and then inspects the other available colours to deduce the maximum possible amount of anti-aliasing up to the limit passed in R3.

If anti-aliasing is desirable, R3 should equal 14 on entry, otherwise set to 0 for monochrome.

The values in R - R3 on exit are suitable for passing to Font_SetFontColours, or including in a font string in a command (18) sequence.

In 256 colour modes only 16 colours can be set before previously returned information becomes invalid - this is the source of the 16 colour change limitation in Font_Paint.

The 24-bit palette entries are in the &BBGRR00 format.

[RISC OS Open source page](#)

ColourTrans_ReturnGCOL

SWI &40742 · ColourTrans

This purpose of this call is to return the GCOL in the current mode that closest matches the palette entry in R0.

Before calling

R0 Palette entry

On return

R0 GCOL

Notes

The 24-bit palette entries are in the &BBGRR00 format.

GCOL is only of use for 256 colour screen modes.

[RISC OS Open source page](#)

ColourTrans_ReturnGCOLForMode

SWI &40745 · ColourTrans

This purpose of this call is to return the closest GCOL, given a specific palette entry with the destination screen mode and palette.

Before calling

R0 Palette entry

R1 Destination mode, or -1 for current mode

R2 Destination palette pointer, or

-1 for current palette, or

0 for default for the current mode

On return

R0 GCOL

R1 Preserved

R2 Preserved

Notes

The 24-bit palette entries are in the &BBGGRR00 format.

GCOL is only of use for 256 colour screen modes.

[RISC OS Open source page](#)

ColourTrans_ReturnOppColourNumber

SWI &40749 · ColourTrans

This purpose of this call is to return the furthest colour number in the current screen mode and palette.

Before calling

R0 Palette entry

On return

R0 Colour number

Notes

The 24-bit palette entries are in the &BBGGRR00 format.

[RISC OS Open source page](#)

ColourTrans_ReturnOppColourNumberForMode

SWI &4074B · ColourTrans

This purpose of this call is to return the furthest colour number, given a specific palette entry with the destination screen mode and palette.

Before calling

R0 Palette entry

R1 Destination mode, or -1 for current mode

R2 Destination palette pointer, or

-1 for current palette, or

0 for default for the current mode

On return

R0 Colour number

R1 Preserved

R2 Preserved

Notes

The 24-bit palette entries are in the &BBGGRR00 format.

[RISC OS Open source page](#)

ColourTrans_ReturnOppGCOL

SWI &40747 · ColourTrans

This purpose of this call is to return the furthest GCOL in the current mode, palette and palette entry.

Before calling

R0 Palette entry

On return

R0 GCOL

Notes

The 24-bit palette entries are in the &BBGGRR00 format.

GCOL is only of use for 256 colour screen modes.

[RISC OS Open source page](#)

ColourTrans_ReturnOppGCOLForMode

SWI &4074A · ColourTrans

This purpose of this call is to return the furthest GCOL, given a specific palette entry with the destination screen mode and palette.

Before calling

R0 Palette entry

R1 Destination mode, or -1 for current mode

R2 Destination palette pointer, or

-1 for current palette, or

0 for default for the current mode

On return

R0 GCOL

R1 Preserved

R2 Preserved

Notes

The 24-bit palette entries are in the &BBGGRR00 format.

GCOL is only of use for 256 colour screen modes.

[RISC OS Open source page](#)

ColourTrans_SelectGCOLTable

SWI &40741 · ColourTrans

The purpose of this call is to set up a GCOL translation table in a buffer.

Before calling

R0 Source mode / sprite area

R1 Source palette pointer / sprite

R2 Destination mode, or -1 for current mode

R3 Destination palette pointer, or

-1 for current palette, or

0 for default for the mode

R4 Pointer to buffer, or 0 to return length

R5 Flags

On return

All registers preserved

Notes

Given a source mode and palette, a destination mode and palette, and a buffer, sets up a list of GCOLs in the buffer. The values can be subsequently be used by passing them to GCOL? and TINT? commands.

[RISC OS Open source page](#)

ColourTrans_SelectTable

SWI &40740 · ColourTrans

The purpose of this call is to set up a translation table in a buffer that is used by scaled sprite plotting.

Before calling

R0 Source mode / sprite area

R1 Source palette pointer / sprite

R2 Destination mode, or -1 for current mode

R3 Destination palette pointer, or

-1 for current palette, or

0 for default for the mode

R4 Pointer to buffer, or 0 to return length

R5 Select Table Flags (only used if R0 & R1 indicate a sprite)

R6 Pointer to transfer workspace (R12) (if R5 valid and bit 2 of R5 is set)

R7 Pointer to transfer function (if R5 valid and bit 2 of R5 is set)

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Required size of buffer (if R4=0 on entry), or preserved

R5 Preserved

R6 Preserved

R7 Preserved

Notes

Unlike ColourTrans_GenerateTable this SWI call only uses R5-R7 if R0 & R1 describe a sprite. For mode numbers, mode descriptors, sprite mode words, and the current mode (-1), R5-R7 are ignored and the flags word is treated as 0.

R6 and R7 are only used if R5 is used and bit 2 of R5 is set.

Sometimes ColourTrans may decide that no translation table is necessary (e.g. source and dest have the same pixel format and palette), in which case the "measure table size" form of the call will return with R4=0. Take care to check for this before allocating memory and asking for the table to be filled in.

[RISC OS Open source page](#)

ColourTrans_SetCalibration

SWI &40751 · ColourTrans

This purpose of this call is to set the calibration table.

Before calling

R0 Pointer to calibration table

On return

--

Notes

The call copies the calibration table into the RMA, for use as the screen calibration table.

Experimentation suggest that R0 = 0 restores the default without error.

[RISC OS Open source page](#)

ColourTrans_SetColour

SWI &4075E · ColourTrans

This purpose of this call is to change the current plotting colours.

Before calling

R0 GCOL number

R3 Flags

R4 GCOL action

On return

- All registers preserved

Notes

GCOL is only of use in 256 colour screen modes.

[RISC OS Open source page](#)

ColourTrans_SetFontColours

SWI &4074F · ColourTrans

This purpose of this call is to calculate the best range of anti-aliasing colours and assign these as the colours to use by the Font Manager.

Before calling

R0 Font handle, or 0 for current font

R1 Background palette entry

R2 Foreground palette entry

R3 Maximum foreground colour offset (0-14)

On return

R0 Preserved

R1 Background logical colour (preserved if in 256 colour mode)

R2 Foreground logical colour

R3 Maximum sensible colour offset (up to R3 on entry)

Notes

Given a pair on palette entries, it calculates the best available range of anti-aliasing colours in the current palette.

[RISC OS Open source page](#)

ColourTrans_SetGCOL

SWI &40743 · ColourTrans

This purpose of this call is to calculate the closest GCOL in the current mode and palette and set it.

Before calling

R0 Palette entry

R3 Flags

R4 GCOL action (only the bottom 3 bits are valid)

On return

R0 GCOL

R2 Log2 of bits-per-pixel for current mode

R3 Initial value AND &80

R4 Preserved

[RISC OS Open source page](#)

ColourTrans_SetOppGCOL

SWI &40748 · ColourTrans

This purpose of this call is to calculate the furthest GCOL to the palette entry passed in R0 in the current mode and palette and set it.

Before calling

R0 Palette entry

R3 0 for foreground, or 128 for background

R4 GCOL action

On return

R0 GCOL

R2 Log2 of bits-per-pixel for current mode

R3 Initial value AND &80

R4 Preserved

Notes

The top three bits of R3 and R4 should be zero to allow for future expansion.

GCOL is only of use in 256 colour screen modes.

[RISC OS Open source page](#)

ColourTrans_SetOppTextColour

SWI &40762 · ColourTrans

This purpose of this call is to calculate the furthest GCOL in the current mode and palette to the palette entry passed in R0. It then sets the text colour to the calculated colour.

Before calling

R0 Palette entry

R3 Flags

On return

R0 GCOL

R3 Preserved

[RISC OS Open source page](#)

ColourTrans_SetTextColour

SWI &40761 · ColourTrans

This purpose of this call is to calculate the closest GCOL in the current mode and palette to the palette entry passed in R0. It then sets the text colour to the calculated colour.

Before calling

R0 Palette entry

R3 Flags

On return

R0 GCOL

R3 Preserved

[RISC OS Open source page](#)

ColourTrans_WriteCalibrationToFile

SWI &40757 · ColourTrans

This purpose of this call is to save the calibration table to a file.

Before calling

R0 Flags

R1 File handle to write to

On return

R0 Corrupted

Notes

It saves the calibration table by creating a list of *Commands which will recreate the current calibration.

[RISC OS Open source page](#)

ColourTrans_WriteLoadingsToFile

SWI &40760 · ColourTrans

This purpose of this call is to write a *Command to the given file to restore the error loadings.

Before calling

R1 File handle to write to

On return

All registers preserved

Notes

This allows desktop saving of the loadings.

[RISC OS Open source page](#)

ColourTrans_WritePalette

SWI &4075D · ColourTrans

The purpose of this call is to write the screen's palette, or sprite's palette.

Before calling

R0 pointer to sprite area, or -1 for screen

R1 sprite name/pointer (as specified in R4), or ignored for screen

R2 Pointer to palette data to write

R3 0 (reserved)

R4 Flags

On return

--

Notes

If R4 bit 2 is clear, 256 colour palettes are compacted to the 16 used by VIDC1 if possible, else all colours are written.

[RISC OS Open source page](#)

Debugger_Disassemble

SWI &40380 · Debugger

The purpose of this call is to disassemble a 32bit ARM instruction word. R1 should contain the address that the instruction came from - this allows the Debugger module to display absolute addresses for the target of PC-relative instructions (e.g. branches).

Before calling

R0 Instruction to disassemble

R1 Address to disassemble relative to

On return

R1 Pointer to null-terminated string containing disassembly

R2 Length of string (excluding terminator)

Notes

The string returned in R1 is located within the Debugger module's workspace. Therefore care must be taken to take a copy of the string before another call to the Debugger module overwrites it.

In addition to containing the assembler mnemonic that best represents the instruction, the disassembly may also contain a comment (marked with ';') that gives additional information, for example warnings about potential code bugs or potential CPU incompatibilities.

[RISC OS Open source page](#)

Debugger_DisassembleThumb

SWI &40381 · Debugger

The purpose of this call is to disassemble a 16bit Thumb instruction word. R1 should contain the address that the instruction came from - this allows the Debugger module to display absolute addresses for the target of PC-relative instructions (e.g. branches).

Before calling

R0 Instruction to disassemble

R1 Address to disassemble relative to

On return

R1 Pointer to null-terminated string containing disassembly

R2 Length of string (excluding terminator)

Notes

The string returned in R1 is located within the Debugger module's workspace. Therefore care must be taken to take a copy of the string before another call to the Debugger module overwrites it.

In addition to containing the assembler mnemonic that best represents the instruction, the disassembly may also contain a comment (marked with ';') that gives additional information, for example warnings about potential code bugs or potential CPU incompatibilities.

[RISC OS Open source page](#)

DeviceFS_CallDevice

SWI &42744 · DeviceFS

The purpose of this call is to make a call to a device with a specified register set.

Before calling

R0 Reason Code

R1 Device driver identification using following methods:

Device driver's handle

Path or Pointer

0 to broadcast to all devices

R2 Parameters passed to device driver, dependant on Reason Code

R3 Parameters passed to device driver, dependant on Reason Code

R4 Parameters passed to device driver, dependant on Reason Code

R5 Parameters passed to device driver, dependant on Reason Code

R6 Parameters passed to device driver, dependant on Reason Code

R7 Parameters passed to device driver, dependant on Reason Code

R12 Pointer to Workspace

On return

- Depends on device and reason code

Notes

This SWI calls the device entry point specified by the reason code. The device(s) to call can be specified by device handle, filename in the directory structure (\$ may be used), or 0 to broadcast to all devices.

[RISC OS Open source page](#)

DeviceFS_Deregister

SWI &42743 · DeviceFS

The purpose of this call is to deregister a device related to a particular device driver.

Before calling

R0 Device driver's handle

R1 Pointer to name of device to remove

On return

--

Notes

None.

[RISC OS Open source page](#)

DeviceFS_DeregisterObjects

SWI &42743 · DeviceFS

The purpose of this call is to deregister a single device related to a particular device driver.

Before calling

R0 Device driver's handle

R1 Pointer to name of device to remove

On return

--

Notes

Although this SWI name suggests that multiple devices can be removed, this is not the case. Only one device can be deregistered per call.

[RISC OS Open source page](#)

DeviceFS_ReceivedCharacter

SWI &42746 · DeviceFS

The purpose of this call is to inform DeviceFS that a device driver has received a character.

Before calling

R0 Byte received

R1 DeviceFS stream handle

On return

- C set => byte not transferred

C clear => byte transferred

Notes

DeviceFS will then process the character (buffer it, or unblock a waiting stream).

For speed DeviceFS_ReceivedCharacter and DeviceFS_TransmitCharacter do not validate the external handle passed - strange things will happen if bad handles are passed.

[RISC OS Open source page](#)

DeviceFS_Register

SWI &42740 · DeviceFS

The purpose of this call is to register a device driver and its associated devices with DeviceFS.

Before calling

R0 Device flags

R1 Pointer to list of devices to install

R2 Pointer to device driver entry point

R3 Private word (passed to entry point in R8)

R4 Workspace pointer (passed to entry point in R12)

R5 Pointer to special field validation string?

R6 Maximum RX devices (0 for none, -1 for unlimited)

R7 Maximum TX devices (0 for none, -1 for unlimited)

On return

R0 Device driver's handle

Global Device Flags

Bit / Meaning when set

0 – Meaning when set: Is block device (otherwise is character device)

1 – Meaning when set: Device is Full duplex

2 – Meaning when set: Monitor TX/RX available

3 – Meaning when set: Monitor EOF on reading

4 – Meaning when set: Issue device OS_UpCall with reason code 15 - 17

5 - 31 – Meaning when set: Reserved (must be zero)

Notes

If a device in the list should fail to register than all devices specified will be removed.

[RISC OS Open source page](#)

DeviceFS_RegisterObjects

SWI &42742 · DeviceFS

The purpose of this call is to register a list of additional devices with a device driver.

Before calling

R0 Device driver's handle

R1 Pointer to list of devices to install

On return

--

Notes

This call is an extension to the DeviceFS_Register SWI which itself allows devices to be registered at the same time as the device driver.

[RISC OS Open source page](#)

DeviceFS_Threshold

SWI &42745 · DeviceFS

The purpose of this call is to inform DeviceFS of the threshold value to use on buffered devices.

Before calling

R1 DeviceFS Stream handle

R2 Threshold to use, or -1 to read

On return

R2 Previous threshold used

Notes

This SWI sets the threshold at which halt and resume are called.

[RISC OS Open source page](#)

DeviceFS_TransmitCharacter

SWI &42747 · DeviceFS

The purpose of this call is to inform DeviceFS that a device driver wants to transmit a character.

Before calling

R1 DeviceFS stream handle

On return

R0 Byte to transmit

- C set => unable to read a character to transmit

Notes

This SWI is made by a device when it wants to transmit a character. DeviceFS will try to find character (unbuffer one, or read from a waiting stream).

For speed DeviceFS_TransmitCharacter and DeviceFS_ReceivedCharacter do not validate the external handle passed - strange things will happen if bad handles are passed.

[RISC OS Open source page](#)

DHCP_Execute

SWI &52E81 · DHCP

This SWI initialises a DHCP boot attempt on the specified interface. If the DHCP module is already controlling the specified interface, then this SWI has no effect.

Before calling

R0 Flags ·

Bit 0 = Immediate start

Bit 1 = Wait for interface to appear, equivalent to -w for *DHCPExecute

Bit 2 = Block until the interface is bound, equivalent to -b for *DHCPExecute

Bit 3 = On timeout assign a private IP address instead, equivalent to -p for *DHCPExecute

Other bits reserved, should be zero

R1 Pointer to interface name, such as "ey1"

On return

R2 Interface state

In practice

The Immediate start flag schedules the initial DHCP discover to be sent with a greatly reduced random backoff time, rather than the default random backoff of up to 10 seconds.

*DHCPExecute DHCP_SetOption DHCP_Inform

[RISC OS Open source page](#)

DHCP_GetOption

SWI &52E83 · DHCP

This SWI returns a copy of the DHCP option specified in R2. Option numbers are in the range 1 to 254 (inclusive) as described in RFC2132.

Before calling

R0 Flags, currently reserved, must be 0

R1 Pointer to interface name, such as "ey1"

R2 Option identifier

R3 Buffer

R4 Size of buffer

On return

R2 Interface state

R4 Reduced by the number of bytes written

In practice

In the BOUND, RENEWING and REBINDING state the buffer will contain the most recent DHCPACK message, in other states it will be the most recently sent message.

To find the size of buffer required, set the buffer pointer and size to 0 and call this SWI. No data will be copied, but R4 will have been reduced by the number of bytes that would have been, so allocate a buffer of at least -R4 bytes.

If the buffer is not sufficiently long, as much of the option as possible will be copied into it. If the buffer is overly long, no data will be written beyond the end of the option data.

[RISC OS Open source page](#)

DHCP_GetState

SWI &52E82 · DHCP

This SWI returns state information and obtains a copy, into the buffer given, of the DHCP packet which was used to initialise the specified interface.

Before calling

R0 Flags, currently reserved, must be 0

R1 Pointer to interface name, such as "ey1"

R3 Buffer

R4 Size of buffer

On return

R2 Interface state

R4 Reduced by the number of bytes written

In practice

In the BOUND, RENEWING and REBINDING state the buffer will contain the most recent DHCPACK message, in other states it will be the most recently sent message.

To find the size of buffer required, set the buffer pointer and size to 0 and call this SWI. No data will be copied, but R4 will have been reduced by the number of bytes that would have been, so allocate a buffer of at least -R4 bytes.

If the buffer is not sufficiently long, as much of the packet as possible will be copied into it. If the buffer is overly long, no data will be written beyond the end of the packet.

[RISC OS Open source page](#)

DHCP_Inform

SWI &52E85 · DHCP

This SWI informs a DHCP server that this station has had its IP address set by some other means, for example manually, but that some other details about the network are of interest.

Before calling

R0 Flags ·

Bit 0 = Set if address in R1 is a string, else it's a struct in_addr

Other bits reserved, should be zero

R1 Pointer to IP address

R2 Pointer to a list of DHCP options to request

R3 Size of list

In practice

The IP address may be provided either as a string "10.0.0.57" or pointer to a standard internet address structure.

The options are as for DHCP_SetOption? and may request information such as the preferred DNS, time servers, and so forth.

[RISC OS Open source page](#)

DHCP_SetOption

SWI &52E84 · DHCP

This SWI allows external software to add or remove options from a DHCP packet, without needing to know about the internal structure of a DHCP negotiation. If the option specified in R2 is already present in the specified interface's DHCP packet it is first deleted from the packet.

Before calling

R0 Flags, currently reserved, must be 0

R1 Pointer to interface name, such as "ey1"

R2 Option identifier

R3 Buffer

R4 Size of buffer

On return

R2 Interface state

In practice

Option numbers are in the range 1 to 254 (inclusive) as described in RFC2132. Some options are zero length, which may be signalled by a non-NULL pointer in R3 and a zero in R4.

Notes

An example opportunity to call this SWI is during the DynamicBootStart service call.

[RISC OS Open source page](#)

DHCP_Version

SWI &52E80 · DHCP

This SWI returns the version number of the DHCP module.

Before calling

R0 Flags, currently reserved, must be 0

On return

R0 Version number × 100

[RISC OS Open source page](#)

DMA_AllocateLogicalChannels

SWI &46147 · DMAManager

Since DMA controllers can now be added by third parties, a method of allocating logical channel numbers is required – this is what this SWI achieves. The logical channels are guaranteed to be unique within this session, even across reinitialisations of the DMAManager.

Before calling

R0 Number of logical channels to allocate

On return

R0 Number of first logical channel in allocation (any others are in a contiguous range above this number)

- Other registers preserved

Notes

Currently DMAManager reserves logical channel numbers 0-&FFF (inclusive) for use by DMA controllers/drivers located in the HAL. Therefore DMA channels allocated by this SWI will start their numbering at &1000. However, this range may change in future in order to accommodate the requirements of individual HALs. Software outside the HAL should not make any assumptions about what range of logical channel numbers it will be allocated.

This documentation only covers the RISC OS 5 version of the DMAManager SWIs. For other OS versions, consult the appropriate PRM volume (e.g. volume 5a).

[RISC OS Open source page](#)

DMA_DeregisterChannel

SWI &46141 · DMAManager

This SWI releases your hold on the indicated channel so that other clients can use it.

Before calling

R0 Channel handle

On return

- All registers preserved

In practice

Any active DMA transfers for the channel will be cancelled, with an error condition passed to the callback.

Notes

This documentation only covers the RISC OS 5 version of the DMAManager SWIs. For other OS versions, consult the appropriate PRM volume (e.g. volume 5a).

[RISC OS Open source page](#)

DMA_ExamineTransfer

SWI &46146 · DMAManager

This SWI is used to return the progress of a DMA transfer, giving the total number of bytes transferred. This will be a pessimistic figure, indicating the number of bytes that DMAManager can say for certainty have been transferred. Depending on how frequently the underlying DMA driver provides progress updates, the value may lag quite far behind the actual progress.

Before calling

R0 Flags:

Bits 0-31: Reserved (set to 0)

R1 DMA tag

On return

R0 Number of bytes transferred so far

- All other registers preserved

Notes

An error block is returned if the DMA tag is invalid.

This documentation only covers the RISC OS 5 version of the DMAManager SWIs. For other OS versions, consult the appropriate PRM volume (e.g. volume 5a).

[RISC OS Open source page](#)

DMA_QueueTransfer

SWI &46142 · DMAManager

Queues a DMA transfer request for a logical channel.

Before calling

R0 Flags:

Bit 0: 1=write transfer (memory to device), 0=read transfer (device to memory)

Bit 1: 1=scatter list is a circular buffer, 0=not circular

Bit 2: 1=call DMASync control routine, 0=don't call

Bits 3-31: Reserved (set to 0)

R1 Channel handle

R2 Value of R11 to be passed to control routines

R3 Pointer to word aligned scatter list

R4 Number of bytes to transfer, or 0 for infinite length transfer (if bit 1 of R0 set)

R5 Size of circular buffer (if bit 1 of R0 set)

R6 Number of bytes between calls to DMASync control routine (if bit 2 of R0 set)

On return

R0 DMA tag

- Other registers preserved

In practice

The returned DMA tag can be used with other DMAManager SWIs such as DMA_ExamineTransfer and DMA_TerminateTransfer in order to examine or control the state of the transfer.

Notes

This documentation only covers the RISC OS 5 version of the DMAManager SWIs. For other OS versions, consult the appropriate PRM volume (e.g. volume 5a).

[RISC OS Open source page](#)

DMA_RegisterChannel

SWI &46140 · DMAManager

This SWI allows you to register your intent to use a given logical DMA channel.

Before calling

R0 Flags:

Bits 0-3: Post-transfer channel delay

Bit 4: Disable burst transfers

Bit 5: Disable DRQ synchronisation to clock

Bits 6-31: Reserved (set to 0)

R1 Logical channel number

R2 DMA cycle speed (0-3)

R3 Transfer unit size (bytes)

R4 Pointer to vector of DMA callback routines

R5 Value to pass to callback routines in R12

R6 Peripheral read/receive physical address, or -1 to disallow reads

R7 Peripheral write/send physical address, or -1 to disallow writes

On return

R0 Channel handle

- Other registers preserved

In practice

Each logical channel can only be used by one client at any given time; if the channel you are requesting is already in use then an error will be returned.

Notes

This documentation only covers the RISC OS 5 version of the DMAManager SWIs. For other OS versions, consult the appropriate PRM volume (e.g. volume 5a).

[RISC OS Open source page](#)

DMA_ResumeTransfer

SWI &46145 · DMAManager

This SWI is used to resume a previously suspended DMA transfer.

Before calling

R0 Flags:

Bits 0-31: Reserved (set to 0)

R1 DMA tag

On return

- All registers preserved

Notes

A suspended transfer maintains its position in the queue, so a resumed transfer has priority over requests after it was suspended.

The DMAManager calls the DMA enable callback when the suspended transfer is restarted.

An error block is returned if the DMA tag is invalid, or the buffer is not suspended.

This documentation only covers the RISC OS 5 version of the DMAManager SWIs. For other OS versions, consult the appropriate PRM volume (e.g. volume 5a).

[RISC OS Open source page](#)

DMA_SuspendTransfer

SWI &46144 · DMAManager

This SWI is used to suspend the given active DMA transfer.

Before calling

R0 Flags:

Bit 0: 1=start next queued transfer, 0=do not start queued transfers

Bits 1-31: Reserved (set to 0)

R1 DMA tag

On return

- All registers preserved

Notes

The suspension of a transfer is achieved by the following steps:

- DMAManager calls the DMA disable callback, to allow the device to de-assert its DMA request line
- suspends the DMA request within the DMA controller
- updates the DMA Scatter List
- returns the request to its queue

If bit 0 of R0 is clear then no DMA requests for the same logical channel will be started until the suspended transfer is resumed or terminated.

An error block is returned if the DMA tag is invalid, or the specified DMA transfer is not in progress.

This documentation only covers the RISC OS 5 version of the DMAManager SWIs. For other OS versions, consult the appropriate PRM volume (e.g. volume 5a).

[RISC OS Open source page](#)

DMA_TerminateTransfer

SWI &46143 · DMAManager

This SWI is used to terminate a DMA transfer originally queued by DMA_QueueTransfer.

Before calling

R0 Pointer to an error block

R1 DMA tag

On return

- All registers preserved

Notes

Before a transfer is terminated, the following steps are followed:

- If the transfer is active: DMAManager will stop the transfer within the DMA controller. The DMA disable callback will be called (to allow the device to de-assert its DMA request line)
- The transfer will be removed from the queue associated with the logical channel, allowing the transfer behind it to start as necessary. If the transfer was blocking the logical channel (i.e. the transfer had previously been suspended by

DMA_SuspendTransfer with bit 0 of R0 clear) then the channel will be un-blocked and will resume processing its transfer queue

- The DMA completed callback will be called, with V set and R0 containing a pointer to the supplied error block

An error block is returned if the DMA tag is invalid.

This documentation only covers the RISC OS 5 version of the DMAManager SWIs. For other OS versions, consult the appropriate PRM volume (e.g. volume 5a).

[RISC OS Open source page](#)

DOSFS_DiscFormat

SWI &44B00 · DOSFS

The purpose of this call is to fill in a disc format structure with parameters for the specified format.

Before calling

R0 Pointer to Disc Format Block to be filled in

R1 SWI number to call to vet disc format (e.g. ADFS_VetFormat)

R2 Parameter in R1 to use when calling SWI

R3 Format specifier

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

This call fills in the disc format structure pointed to by R0 with the 'perfect' parameters for the specified format, taking no account of the abilities of the available hardware that will have to perform the format. Once filled in, this SWI calls the vetting SWI to check the format structure for achievability on the available hardware. The vetting SWI may generate an error if the format differs widely from what can be achieved; alternatively it may alter the format structure to the closest match that can be achieved. The vetting SWI then returns to this SWI, which checks whether the format block – as updated by the vetting SWI – is still an adequate match for the desired format. If it is, this SWI returns to its caller; otherwise it generates an error.

[RISC OS Open source page](#)

DOSFS_LayoutStructure

SWI &44B01 · DOSFS

The purpose of this call is to lay out in the specified image, all necessary structures to have a valid and empty disc.

Before calling

R0 Pointer to Structure specifier

R1 Pointer to bad block list. This list is terminated by a value of -1

R2 Pointer to disc name. This is terminated by a value of null

R3 Image file handle

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

In practice

It can be used:

- to layout a structure on a blank, formatted disc (in which case the specified image should be the whole disc image)
- to layout a partition in a file on a disc that has already been formatted (for example for the PC emulator).

Notes

If the given image format has no option to store a disc name then this parameter should be ignored.

The bad block list should be presented as an array of bad block addresses. Each address is four bytes long. The array is terminated by a -1 entry.

It is assumed that R0 gives enough information for the format - it may be that R0 contains many bit fields or points to a block of information - the choice is up to the image filing system module.

[RISC OS Open source page](#)

DragAnObject_Start

SWI &49C40 · DragAnObject

The purpose of this call is to start dragging an object.

Before calling

R0 Flags

R1 renderer called to render the object (SWI number or pointer to C/assembler function)

R2 Pointer to block holding registers for SWI / function parameters

R3 Pointer to 16-byte block containing drag box

R4 Pointer to optional block containing bounding box

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

Notes

As per the DragASprite_Start, this SWI starts the dragging process (calling DragASprite_Start and in turn Wimp_DragBox on the clients behalf), usually in response to a Drag (Mouse_Click) Event.

The renderer can be a C (or assembler) function or a SWI (determined by setting bits 16 and 17 in the flags word).

Example SWI renderer

```
void start_drag(...)
{
    _kernel_swi_regs regs, render;
    render.r[1] = (int) &icon;
    /* oh dear, bug in the wimp */
    render.r[4] = 0;
    render.r[5] = 0;
    regs.r[0] = some flags;
    regs.r[1] = Wimp_PlotIcon;
    regs.r[2] = (int) &render;
    regs.r[3] = (int) &bbox;
    _kernel_swi(DragAnObject_Start, ®s, ®s);
}
```

Example Code renderer

```
void _my_render(data)
{
    /* do the render */
}

void start_drag(...)
{
    _kernel_swi_regs regs;
    int render[4];
    /* render[0] ... render[3] will be passed to the function as parameters */
    render[0] = (int) data required by renderer;
    /* tell it we're a function and a module */
    regs.r[0] = some flags + (1<<16) + (1<<17);
    regs.r[1] = (int) _my_render;
    regs.r[2] = (int) &render;
    regs.r[3] = (int) &bbox;
    _kernel_swi(DragAnObject_Start, ®s, ®s);
}
```

[RISC OS Open source page](#)

DragAnObject_Stop

SWI &49C41 · DragAnObject

The purpose of this call is to terminate any current DragAnObject operation and releases any DragAnObject workspace claimed.

Before calling

--

On return

--

Notes

It is expected that this SWI will be called when the User_Drag_Box event is received by the application.

The current implementation of the call, simply redirects it to DragASprite_Stop

[RISC OS Open source page](#)

DragASprite_Start

SWI &42400 · DragASprite

The purpose of this call is to start dragging a sprite.

Before calling

R0 Drag Location Flags

R1 Sprite Area (0 for system, 1 for Wimp, otherwise an address)

R2 Pointer to sprite name

R3 Pointer to 16-byte block containing drag box

R4 Pointer to optional block containing bounding box

On return

All registers preserved

Notes

The sprite provided in R2 is copied so the original can be disposed of safely to free memory if required.

[RISC OS Open source page](#)

DragASprite_Stop

SWI &42401 · DragASprite

The purpose of this call is to terminate any current drag operation and releases any drag-a-sprite workspace claimed.

Before calling

--

On return

--

Notes

It is expected that this SWI will be called when the User_Drag_Box reason code is received by the application.

[RISC OS Open source page](#)

Draw_Fill

SWI &40702 · Draw

The purpose of this call is to draw a path and fill its inside.

Before calling

R0 Pointer to path block

R1 Fill style, or 0 for default

R2 Pointer to transformation matrix, or 0 for default

R3 Flatness, or 0 for default

On return

R0 Corrupt

R1 Preserved

R2 Preserved

R3 Preserved

Notes

The path block specifies the path parameters, such as line type, co-ordinates and so forth.

A value of 0 for the fill style uses the default fill style (value &30). This is equivalent of boundary centre, non-zero rule.

[RISC OS Open source page](#)

Draw_FillClipped

SWI &4070C · Draw

The purpose of this call is to draw a path and fill its inside, clipped to the path provided.

Before calling

R0 Pointer to path block

R1 Fill style, or 0 for default

R2 Pointer to transformation matrix, or 0 for default

R3 Flatness, or 0 for default

R4 Pointer to clip block

On return

R0 Corrupt

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

Notes

The path block specifies the path parameters, such as line type, co-ordinates and so forth.

A value of 0 for the fill style uses the default fill style (value &30). This is equivalent of boundary centre, non-zero rule.

[RISC OS Open source page](#)

Draw_FlattenPath

SWI &40708 · Draw

The purpose of this call is to convert an input path to a flattened output path.

Before calling

R0 Pointer to path block

R1 Pointer to output path block, or 0 to calculate size

R2 Flatness, or 0 for default

On return

R0 If R1=0, Calculated output buffer size

If R1=pointer, Pointer to end of path marker in output path

R1 Preserved

R2 Preserved

Notes

It is useful if you want to repeatedly stroke a path, without having to repeatedly spend time flattening it.

[RISC OS Open source page](#)

Draw_ProcessPath

SWI &40700 · Draw

The purpose of this call is to perform a multitude of different Draw operations on a any given draw path.

Before calling

R0 Pointer to path block

R1 Fill style (additional options)

R2 Pointer to transformation matrix, or 0 for default

R3 Flatness, or 0 for default

R4 Line thickness, or 0 for default

R5 Pointer to join/cap block, or 0 if none

R6 Pointer to dash block, or 0 if none

R7 Pointer to output path, or special value

On return

R0 Depend on value of R7 on entry

R7 = 0,1 or 2 then Corrupt

R7 = 3, Size of buffer

R7 = pointer, Pointer to new end of buffer marker

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

R5 Preserved

R6 Preserved

R7 Preserved

Notes

The fill style has additional bit settings compared to the other fill styles.

[RISC OS Open source page](#)

Draw_Stroke

SWI &40704 · Draw

The purpose of this call is to draw a path and display on screen.

Before calling

R0 Pointer to path block

R1 Fill style

R2 Pointer to transformation matrix, or 0 for default

R3 Flatness, or 0 for default

R4 Line thickness, or 0 for default

R5 Pointer to join/cap block, or 0 if none

R6 Pointer to dash block, or 0 if none

On return

R0 Corrupted

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

R5 Preserved

R6 Preserved

Notes

If thickness is zero, fill bits 3, 2 must not be 01. If thickness is non-zero, fill bits 2-5 must not be 0110. i.e. Filling just the boundary pixels.

[RISC OS Open source page](#)

Draw_StrokeClipped

SWI &4070E · Draw

The purpose of this call is to draw a path (clipped) on the screen.

Before calling

- R0** Pointer to path block
- R1** Fill style
- R2** Pointer to transformation matrix, or 0 for identity
- R3** Flatness, or 0 for default
- R4** Line thickness, or 0 for default
- R5** Pointer to join/cap block, or 0 if none
- R6** Pointer to dash block, or 0 if none
- R7** Pointer to clip block

On return

- R0** Corrupted
- R1** Preserved
- R2** Preserved
- R3** Preserved
- R4** Preserved
- R5** Preserved
- R6** Preserved
- R7** Preserved

Notes

If thickness is zero, fill bits 3, 2 must not be 01. If thickness is non-zero, fill bits 2-5 must not be 0110. i.e. Filling just the boundary pixels.

[RISC OS Open source page](#)

Draw_StrokePath

SWI &40706 · Draw

The purpose of this call is to draw a path and write the output as a path in the block.

Before calling

- R0** Pointer to path block
- R1** Pointer to an output path block, or 0 to calculate size
- R2** Pointer to transformation matrix, or 0 for default
- R3** Flatness, or 0 for default
- R4** Line thickness, or 0 for default
- R5** Pointer to join/cap block, or 0 if none
- R6** Pointer to dash block, or 0 if none

On return

R0 If R1 = 0, then calculated output buffer size, otherwise

Pointer to end of path marker in output path

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

R5 Preserved

R6 Preserved

Notes

If thickness is zero, fill bits 3, 2 must not be 01. If thickness is non-zero, fill bits 2-5 must not be 0110. i.e. Filling just the boundary pixels.

Draw_Stroke is similar to this call, however, it outputs the the screen rather than to a block.

[RISC OS Open source page](#)

Draw_TransformPath

SWI &4070A · Draw

The purpose of this call is to convert an input path into a transformed output path.

Before calling

R0 Pointer to path block

R1 Pointer to output path block, or 0 to calculate size

R2 Pointer to transformation matrix, or 0 for default

R3 0

On return

R0 If R1 = 0, then Corrupted, otherwise

Pointer to end of path marker in output path

R1 Preserved

R2 Preserved

R3 Preserved

Notes

This SWI is a subset of Draw_StrokePath, in that it only transforms a path. This can save the time taken to repeatedly transform the same path, or reduce errors by transforming before dashing and thickening.

[RISC OS Open source page](#)

DrawFile_BBBox

SWI &45541 · DrawFile

The purpose of this call is to return the bounding box (in Draw units) a given Draw file will occupy.

Before calling

R0 Flags (must be zero)

R1 Pointer to drawfile block

R2 Length of block

R3 Transformation matrix, or 0 to use identity

R4 Pointer to 4 word buffer for Bounding box in draw units

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

Notes

None.

[RISC OS Open source page](#)

DrawFile_DeclareFonts

SWI &45542 · DrawFile

The purpose of this call is to declare all fonts in a DrawFile by calling PDriver_DeclareFont?.

Before calling

R0 Flags

R1 Pointer to drawfile block?

R2 Length of block

On return

R0 Preserved

R1 Preserved

R2 Preserved

Flags

Bit / Meaning if set

0 – Meaning if set: do not download font (passed in R2 to PDriver_DeclareFont?)

Notes

This SWI should be called between PDriver_SelectJob? and PDriver_DrawPage? if the printer requires that fonts be declared with PDriver_DeclareFont?.

[RISC OS Open source page](#)

DrawFile_Render

SWI &45540 · DrawFile

The purpose of this call is to render a DrawFile to the screen.

Before calling

R0 Flags**R1** Pointer to drawfile block?**R2** Length of block**R3** Transformation matrix, or 0 to use identity**R4** Pointer to clipping rectangle in OS units, or 0 for none**R5** Flatness with which to render lines (if bit 2 of R0 set)

On return

R0 Preserved**R1** Preserved**R2** Preserved**R3** Preserved**R4** Preserved**R5** Preserved

Flags

Bit / Meaning if set

0 – Meaning if set: Render bounding boxes (as dotted red rectangles)**1** – Meaning if set: Do not render the objects themselves**2** – Meaning if set: Use R5 as flatness

Notes

This SWI renders a DrawFile at a given screen position where that position is defined as screen position 0, 0 with the x- and y-translations as specified in the transformation matrix. Hence, to render a non-rotated 1:1 draw file at (x, y) (screen coordinates in OS units) the transformation matrix is :

/

 $(1 \ll 16 - : 0)$ $(0 - : 1 \ll 16)$ $(256 * x - : 256 * y)$

The clipping rectangle is typically a redraw rectangle returned by the Wimp on a redraw window request. If R4 = 0, then the whole draw file is rendered. If non-zero, only objects which intersect the clipping rectangle are rendered.

[RISC OS Open source page](#)

Econet_AbandonAndReadReceive

SWI &40018 · Econet

The purpose of this call is to abandon a reception and returns information about it, including the size of data.

Before calling

R0 Handle

On return

R0 Econet status value

R1 0, or flag byte if r0=9 on exit

R2 Port number

R3 Station number

R4 Net number

R5 Pointer to data buffer

R6 Size of buffer (or size of data received if r0=9 on exit)

Notes

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_AbandonReceive

SWI &40003 · Econet

The purpose of this call is to abandon an RxCB.

Before calling

R0 Handle

On return

R0 Econet status value

Notes

Any data which was in transfer (r0=8 on exit) will be discarded and the block contents will remain the same.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_AbandonTransmit

SWI &40008 · Econet

The purpose of this call is to abandon an TxCB.

Before calling

R0 Handle

On return

R0 Econet status value

Notes

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_AllocatePort

SWI &40013 · Econet

The purpose of this call is to allocate a unique port number.

Before calling

--

On return

R0 Port number

Notes

This SWI asks the software for a unique port number which has not been claimed or allocated. When you have finished with the port, use Econet_DeAllocatePort to release it.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_ClaimPort

SWI &40015 · Econet

The purpose of this call is to claim a specific port number.

Before calling

R0 Port number

On return

--

Notes

When you have finished with the port, use `Econet_ReleasePort` to release it

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_ConvertStatusToError

SWI &4000C · Econet

The purpose of this call is to convert a status result from a prior Econet call to an error block and then generates an error.

Before calling

R0 Econet status value

R1 Pointer to error buffer

R2 Buffer size

R3 Station number

R4 Net number

On return

R0 Pointer to error block

V set

Notes

To prevent the error, call the X version of this SWI and read R0, or set the buffer address to 0 which will leave the string in the workspace.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_ConvertStatusToString

SWI &4000B · Econet

The purpose of this call is to convert a status to a string.

Before calling

R0 Econet status value

R1 Pointer to buffer for string

R2 Buffer size

R3 Station number

R4 Net number

On return

R0 Pointer to buffer

R1 Pointer to end address in buffer

R2 Space in buffer

Notes

This SWI converts a status result from a prior Econet call to a string. If the station or net is invalid then station information is not given.

If R0 is out of range this call will Data Abort/Address exception.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_CreateReceive

SWI &40000 · Econet

The purpose of this call is to create a Receive Control Block.

Before calling

R0 Port number

R1 Station number

R2 Net number

R3 Pointer to data buffer

R4 Buffer size

On return

R0 Handle

R2 0 if local net

Notes

This SWI creates a control block to receive data on an Econet port. The data buffer should be in RMA or some other permanently available area of memory, due to the way in which the data is read.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_DeAllocatePort

SWI &40014 · Econet

The purpose of this call is to deallocate a unique port number previously claimed using Econet_AllocatePort.

Before calling

R0 Port number

On return

--

Notes

You must not use this call for port numbers that have been previously claimed using `Econet_ClaimPort`; instead, you must call `Econet_ReleasePort`.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_DoImmediate

SWI &40017 · Econet

The purpose of this call is to create a TxCB for an immediate operation, polls it, reads its status, and abandons it.

Before calling

R0 Operation type

R1 Remote address, or procedure number

R2 Station number

R3 Net number

R4 pointer to data buffer

R5 Data size

R6 Repeat count

R7 Delay (in centi-seconds)

On return

R0 Handle

R1 Corrupted

R2 Pointer to buffer

R3 Station number

R4 Net number (or 0 if R3 on entry was the local net)

Notes

This SWI opens a transmission block and repeatedly polls until it finishes transmission or an error occurs. It abandons the block and returns the final status. It will loop in USR mode with IRQs enabled – a side effect is that callbacks may occur.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_DoTransmit

SWI &40009 · Econet

The purpose of this call is to create a TxCB, polls it, reads its status, and abandons it.

Before calling

R0 Flag byte

R1 Port number

R2 Station number

R3 Net number

R4 Pointer to data buffer
R5 Buffer size
R6 Retry count
R7 Delay (in centi-seconds)

On return

R0 Status value
R1 Corrupted
R2 Pointer to buffer
R3 Station number
R4 Net number (or 0 if R3 on entry was the local net)

Notes

As this SWI enables interrupts it should not be called from service or event handlers. It will loop in USR mode with IRQs enabled – a side effect is that callbacks may occur.

This SWI is not re-entrant.

[RISC OS Open source page](#)

Econet_EnumerateReceive

SWI &40005 · Econet

The purpose of this call is to return the handles of open RxCBs.

Before calling

R0 Index, 1 for first block

On return

R0 Handle, or 0 if no more blocks

Notes

This call should not be made from an IRQ or event routine as, although it will not fail, errors and omissions are likely to occur in the returned information.

[RISC OS Open source page](#)

Econet_EnumerateTransmit

SWI &4001F · Econet

The purpose of this call is to return the handles of open TxCBs.

Before calling

R0 Index, 1 for first block

On return

R0 Handle, or 0 if no more blocks

Notes

You should not make this call from an IRQ or event routine as, although it will not fail, the returned information may be inaccurate.

[RISC OS Open source page](#)

Econet_ExamineReceive

SWI &40001 · Econet

The purpose of this call is to read the status of a RxCB.

Before calling

R0 Handle

On return

R0 Econet status value

Notes

This is the transmit counterpart of Econet_PollTransmit.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_NetworkState

SWI &4001A · Econet

The purpose of this call is to return the state of the underlying transport to a given station.

Before calling

R0 Station number

R1 Net number

On return

R0 Preserved

R1 Preserved

R2 Transport state:

0 = fully functional

1 = no clock

Notes

The state returned is transport type dependent, but you may always assume that a value of zero means that the transport is fully functional.

You should only use the returned value as a hint to the exact state; in other words, it is suitable for display but not for decision making. Using this call is no substitute for proper error handling; to determine if a particular transmit will fail, you must do the transmit and be prepared for it to fail.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_PacketSize

SWI &4001B · Econet

The purpose of this call is to return the maximum packet size recommended on the underlying transport to a given station.

Before calling

R0 Station number

R1 Net number

On return

R0 Preserved

R1 Preserved

R2 Maximum permitted packet size (in bytes)

Notes

This call returns the maximum recommended packet size on the underlying transport to a given station. Larger packets will not necessarily be rejected, but their use is not recommended. The size returned is transport type dependent.

This call is intended for use by modules supplying protocols; you do not need to use it in application software. For maximum efficiency the protocol module should negotiate the packet size once. Since the recommended packet size may differ between the stations at either end of a transmission, the protocol module should interrogate both stations and take the lower value returned.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_PollTransmit

SWI &40007 · Econet

The purpose of this call is to read the status of a TxCB.

Before calling

R0 Handle

On return

R0 Econet status value

Notes

This is the transmit counterpart of Econet_ExamineReceive.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_PrintBanner

SWI &40010 · Econet

The purpose of this call is to displays the econet banner.

Before calling

--

On return

--

Notes

This is usually "Acorn Econet", but could be "Acorn Econet, no clock" if no clock is running.

This SWI is not re-entrant.

[RISC OS Open source page](#)

Econet_ReadLocalStationAndNet

SWI &4000A · Econet

The purpose of this call is to return a computer's station number and net number.

Before calling

--

On return

R0 Station number

R1 Net number

Notes

The net number will be zero if there are no bridges present

This SWI is not re-entrant.

[RISC OS Open source page](#)

Econet_ReadProtection

SWI &4000D · Econet

The purpose of this call is to read the current protection word for immediate operations.

Before calling

--

On return

R0 Current protection

Notes

This SWI call is deprecated. You should use `Econet_SetProtection` instead of this call.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_ReadReceive

SWI &40002 · Econet

The purpose of this call is to return information about a reception, including the size of data.

Before calling

R0 Handle

On return

R0 Status value

R1 0, or flag byte if R0 = 9 on exit

R2 Port number

R3 Station number

R4 Net number

R5 Pointer to data buffer

R6 Buffer size, or size of data received if R0 = 9 on exit

Notes

This SWI returns information about a control block, given its handle number. `Econet_ExamineReceive` performs a similar task, but more quickly

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_ReadStationNumber

SWI &4000F · Econet

The purpose of this call is to extract a station and/or net number from a supplied string.

Before calling

R1 Pointer to string to read

On return

R1 Address of terminating space/control character

R2 Station number, or - 1 if not found

R3 Net number, or -1 if not found

Notes

The string is expected to be of the form "nnn.sss".

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_ReadTransportName

SWI &4001C · Econet

The purpose of this call is to return the name of the underlying transport to a given station.

Before calling

R0 Station number

R1 Net number

On return

R0 Preserved

R1 Preserved

R2 Transport name

Notes

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_ReadTransportType

SWI &40011 · Econet

The purpose of this call is to return the name of the underlying transport to a given station.

Before calling

R0 Station number

R1 Net number

R2 2

On return

R0 Preserved

R1 Preserved

R2 Transport type:

0 Not known

1 Internet

2 Econet

3 Nexus

Notes

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_ReleasePort

SWI &40012 · Econet

The purpose of this call is to release a port previously claimed by Econet_ClaimPort.

Before calling

R0 Port number

On return

--

Notes

You must not use this call for port numbers that have been previously claimed using Econet_AllocatePort; instead, you must call Econet_DeAllocatePort.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_SetProtection

SWI &4000E · Econet

The purpose of this call is to set or read the protection word for immediate operations.

Before calling

R0 EOR mask word

R1 AND mask word

On return

R0 Old value

Notes

You should use R0=0, and R1=-1 to read the word.

This SWI Call replaces the now deprecated Econet_ReadProtection.

This SWI is not re-entrant.

[RISC OS Open source page](#)

Econet_StartImmediate

SWI &40016 · Econet

The purpose of this call is to create a TxCB and starts an immediate operation.

Before calling

R0 Operation type

R1 Remote address, or procedure number

R2 Station number
R3 Net number
R4 pointer to data buffer
R5 Data size
R6 Repeat count
R7 Delay (in centi-seconds)

On return

R0 Handle
R1 Corrupted
R2 Pointer to buffer
R3 Station number
R4 Net number (or 0 if R3 on entry was the local net)

Notes

The data buffer should be in RMA or some other permanently available area of memory, due to the way in which the data is read.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_StartTransmit

SWI &40006 · Econet

The purpose of this call is to create a Transmit Control Block and starts a transmission.

Before calling

R0 Flag byte
R1 Port number
R2 Station number
R3 Net number
R4 Pointer to data buffer
R5 Buffer size
R6 Retry count
R7 Delay (in centi-seconds)

On return

R0 Handle
R1 Corrupted
R2 Pointer to buffer
R3 Station number
R4 Net number (or 0 if R3 on entry was the local net)

Notes

The data buffer should be in RMA or some other permanently available area of memory, due to the way in which the data is read.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_Version

SWI &40019 · Econet

The purpose of this call is to return the version of software for the underlying transport to a given station.

Before calling

R0 Station number

R1 Net number

On return

R0 Preserved

R1 Preserved

R2 Version number * 100

Notes

Setting R0 and R1 to 0 on entry returns the version of the top level Econet software.

This SWI is re-entrant.

[RISC OS Open source page](#)

Econet_WaitForReception

SWI &40004 · Econet

The purpose of this call is to poll an RxCB, reads its status, and abandons it.

Before calling

R0 Handle

R0 Timeout in centi-seconds

R0 0 if escape should not abort

On return

R0 Status value

R1 0, or flag byte if R0 = 9 on exit

R2 Port number

R3 Station number

R4 Net number

R5 Pointer to data buffer

R6 Buffer size, or size of data received if R0 = 9 on exit

Notes

This SWI checks the port repeatedly until either aborted, or the data is received. The block will be abandoned after it has been received. As this SWI enables interrupts it should not be called from service or event handlers.

This SWI is not re-entrant.

[RISC OS Open source page](#)

FileCore_Create

SWI &40541 · FileCore

The purpose of this call is to create a new instantiation of an ADFS-like filing system.

Before calling

R0 Pointer to descriptor block

R1 Pointer to calling module's base

R2 Pointer to calling module's private word

R3 Drives configuration

R4 Suggested size for directory cache

R5 Suggested number of 1024 byte buffers or file cache

R6 Hard disc map sizes

On return

R0 Pointer to FileCore instance private word

R1 Address to call after completing background floppy op

R2 Address to call after completing background hard disc op

R3 Address to call to release FIQ after low level op

Notes

It must be called on initialisation by any filing system module that is adding itself to FileCore.

The only start up option currently supported is 'No directory state' which is indicated by setting bit 30. All other bits representing start up options must be clear.

If the filing system doesn't support background transfer of data, R5 must be zero.

The hard disc map size are given using 1 byte for each disc, with drive 4 in the low byte, and drive 7 in the high byte. The byte should contain 'map size'/256 (ie 2 for the old map). This is just a good guess and shouldn't involve starting up the drives to read from them. You might store this in the CMOS RAM.

You must store the FileCore instance private word returned by this SWI in your module's workspace; it's your module's means of identifying itself to FileCore.

When your module calls the addresses returned in R1-R3, it must be in SVC mode with R12 holding the value of R0 that this SWI returned. Interrupts need to be disabled. R0, R1, R3-R11 and R13 will be preserved by FileCore over these calls.

[RISC OS Open source page](#)

FileCore_DescribeDisc

SWI &40545 · FileCore

This call returns a disc record that describes the disc, in the block pointed to by R1.

Before calling

R0 Pointer to disc name (or :0 etc...)

R1 Pointer to disc descriptor data block

R8 Pointer to FileCore instance private word

On return

--

Notes

If the disc name specified is not in a drive it will be asked for via an error box or prompt. If the disc name is that of a drive and no disc is present then the Drive empty error will be reported.

[RISC OS Open source page](#)

FileCore_DiscardReadSectorsCache

SWI &40546 · FileCore

The purpose of this call is to discard the cache of read sectors created by FileCore_DiscOp 9.

Before calling

R6 Cache handle

On return

--

[RISC OS Open source page](#)

FileCore_DiscFormat

SWI &40547 · FileCore

The purpose of this call is to fill in a disc format structure with parameters for the specified format.

Before calling

R0 Pointer to disc format structure to be filled in

R1 SWI number to call to vet disc format (eg ADFS_VetFormat)

R2 Parameter in R1 to use when calling vetting SWI

R3 Format specifier

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

This call fills in the disc format structure pointed to by R0 with the 'perfect' parameters for the specified format, taking no account of the abilities of the available hardware that will have to perform the format. Once filled in, this SWI calls the vetting SWI to check the format structure for achievability on the available hardware.

The vetting SWI may generate an error if the format differs widely from what can be achieved; alternatively it may alter the format structure to the closest match that can be achieved. The vetting SWI then returns to this SWI, which checks whether the format block – as updated by the vetting SWI – is still an adequate match for the desired format. If it is, this SWI returns to its caller; otherwise it generates an error.

[RISC OS Open source page](#)

FileCore_DiscOp

SWI &40540 · FileCore

The purpose of this call is to provide a number of low-level disc operations.

Before calling

R1 Reason code + Flags

All other registers dependent on reason code

On return

R0 0 if no error, error number and VS otherwise

R1 Preserved

All other registers dependent on reason code

N, Z and C flag preserved

Notes

The action performed depends on the reason code value in R1.

Reason Codes

0 – Hex #: &00 · Action: Verify

1 – Hex #: &01 · Action: Read sectors

2 – Hex #: &02 · Action: Write sectors

3 – Hex #: &03 · Action: Read track/ID

4 – Hex #: &04 · Action: Write/format track

5 – Hex #: &05 · Action: Seek

6 – Hex #: &06 · Action: Restore

7 – Hex #: &07 · Action: Step in

8 – Hex #: &08 · Action: Step out

9 – Hex #: &09 · Action: Read sectors via cache

15 – Hex #: &0F · Action: Specify Hard disc

Flags

Bit / Meaning

4 – Meaning: Given alternative defect list

5 – Meaning: Given sector list

6 – Meaning: Ignore escape

7 – Meaning: Ignore timeout

8 - 31 – Meaning: 0 or bits 2-25 of the pointer to an alternative record (ie flags=flags OR ptr<<6)

The alternative record must be word aligned and within the first 64MB of addressable memory.

[RISC OS Open source page](#)

FileCore_DiscOp 0

SWI &40540 · FileCore

The purpose of this call is to verify sectors.

Before calling

R1 0 + Flags

R2 Disc address

R4 Size to verify

On return

R1 Preserved

R2 Next disc address

R4 Unverified

Notes

In RISC OS 3.60, the value in R2 was rounded down to be sector aligned.

[RISC OS Open source page](#)

FileCore_DiscOp 1

SWI &40540 · FileCore

The purpose of this call is to read sectors.

Before calling

R1 1 + Flags

R2 Disc address

R3 Pointer to block to read into

R4 Size

On return

R1 Preserved

R2 Next disc address

R3 Next free address

R4 Unread

Notes

In RISC OS 3.60, the value in R2 was rounded down to be sector aligned.

[RISC OS Open source page](#)

FileCore_DiscOp 15

SWI &40540 · FileCore

The purpose of this call is to specify a hard disc.. [more information required].

Before calling

R1 15 + Flags

R2 Disc address

On return

R1 Preserved

R2 Preserved?

Notes

[more information required].

[RISC OS Open source page](#)

FileCore_DiscOp 2

SWI &40540 · FileCore

The purpose of this call is to write sectors.

Before calling

R1 2 + Flags

R2 Disc address

R3 Pointer to block to write from

R4 Size to write

On return

R1 Preserved

R2 Next disc address

R3 Next free address

R4 Unwritten bytes

[RISC OS Open source page](#)

FileCore_DiscOp 3

SWI &40540 · FileCore

The purpose of this call is to read an ID (from a hard disc), or track (from a floppy).

Before calling

R1 3 + Flags

R2 Disc address

R3 Pointer to block to read into

On return

R1 Preserved

R2 Preserved ?

R3 Preserved ?

Notes

If the alternate defect list option bit (bit 4) is set in R1 on entry when reading a track/ID, then a whole track's worth of ID fields is read. This usage isn't available under RISC OS 2.

The call reads 4 bytes of sector ID information into the buffer pointed to by R3 for every sector on the track. The order of data is:

- Cylinder
- Head
- Sector number
- Sector size (0 = 128; 1 = 256, etc)

The operation is terminated after 200 mS (1 revolution)

The first sector ID transferred will normally be that following the index mark (it may be the second if there is abnormal interrupt latency from the index pulse interrupt). The first two ID's read may also be duplicated at the buffer end due to interrupt latency. Consequently the buffer should be at least 16 bytes longer than the maximum number of ID's expected (512 bytes at most).

The disc record provided is updated to return the actual number of sectors per track found (at offset 1). Note to use this option you MUST provide a valid defect list, which at a minimum is a word of &20000000 following after the disc record.

[RISC OS Open source page](#)

FileCore_DiscOp 4

SWI &40540 · FileCore

The purpose of this call is to write or format tracks.

Before calling

R1 4 + Flags

R2 Disc address

R3 Pointer to block to write, or 0 to format

R4 Format block

On return

R1 Preserved

R2 Preserved ?

R3 Preserved ?

R4 Preserved ?

Notes

If R3=0, this call formats a track of the specified disc, an error is generated, if the specified format is not possible to generate, or if the track requested is outside the valid range. The tracks are numbered from 0 to (number of tracks)-1. The mapping of the address is controlled by the disc structure record.

If R3<>0 then the data specified is written to the track.

[RISC OS Open source page](#)

FileCore_DiscOp 5

SWI &40540 · FileCore

Moves the drive head to the track the specified disc address is on.

Before calling

R1 5 + Flags

R2 Disc address

On return

R1 Preserved

R2 Preserved ?

In practice

An example would be to park the disc.

[RISC OS Open source page](#)

FileCore_DiscOp 6

SWI &40540 · FileCore

The drive seeks to track 0 as is recalibrated.

Before calling

R1 6 + Flags

R2 Disc address

On return

R1 Preserved

R2 Preserved ?

In practice

Note the disc address is only used for the drive number; the bottom 29 bits should be set to zero.

[RISC OS Open source page](#)

FileCore_DiscOp 7

SWI &40540 · FileCore

The purpose of this call is to step in a track for a floppy disc.

Before calling

R1 7 + Flags

R2 Disc address

On return

R1 Preserved

R2 Preserved ?

Notes

This call is used to step in a track. These are only supported by 1772 disc controllers. e.g. those found commonly during the 1980's, and should not be used so as to maintain compatibility.

[RISC OS Open source page](#)

FileCore_DiscOp 8

SWI &40540 · FileCore

The purpose of this call is to step out a track for a floppy disc.

Before calling

R1 8 + Flags

R2 Disc address

On return

R1 Preserved

R2 Preserved ?

Notes

This call is used to step out a track. These are only supported by 1772 disc controllers. e.g. those found commonly during the 1980's, and should not be used so as to maintain compatibility.

[RISC OS Open source page](#)

FileCore_DiscOp 9

SWI &40540 · FileCore

The purpose of this call is to read sectors via a cache held in RMA memory.

Before calling

R1 9 + Flags

R2 Disc address

R3 Pointer to block to read into

R4 Size to read

R6 Cache handle

On return

R1 Preserved

R2 Next disc address

R3 Next free address

R4 Unread bytes

R6 Cache handle

Notes

To start a sequence of these operations, set R6 (cache handle) to zero on entry. Its value will be updated on exit, and subsequent calls should use this new values. Bits 4-7 of R1 should be zero, but are ignored if set.

To discard the cache once finished, just call FileCore_DiscardReadSectorsCache.

Note: For RISC OS 3.60, the value in R2 will be rounded down to be sector aligned.

[RISC OS Open source page](#)

FileCore_DiscOp64

SWI &4054C · FileCore

The purpose of this call is to provide a number of low-level disc operations using the more advanced Disc Operation DiscOp64.

Before calling

R1 Reason code + Options

R2 Pointer to disc address

R3 Pointer to buffer (or scatter list)

R4 Length in bytes

R5 Pointer to alternative disc record

R6 Cache handle

R8 Pointer to Filecore instance private word

On return

R1 Preserved

R2 Preserved, disc address pointed to is updated to point to the next byte to be transferred

R3 Pointer to next buffer location to be transferred

R4 Number of bytes not transferred

R5 Preserved ?

R6 Preserved ?

R8 Preserved ?

Reason Codes

0 – Hex #: &00 · Action: Verify?

1 – Hex #: &01 · Action: Read sectors?

2 – Hex #: &02 · Action: Write sectors?

3 – Hex #: &03 · Action: Read track/ID?

4 – Hex #: &04 · Action: Write/format track?

5 – Hex #: &05 · Action: Seek?

6 – Hex #: &06 · Action: Restore?

7 – Hex #: &07 · Action: Step in?

8 – Hex #: &08 · Action: Step out?

9 – Hex #: &09 · Action: Read sectors via cache?

15 – Hex #: &0F · Action: Specify Hard disc?

Options

Bit / Meaning

4 – Meaning: Given alternative defect list

5 – Meaning: Given sector list

6 – Meaning: Ignore escape

7 – Meaning: Ignore timeout

8 - 31 – Meaning: Reserved, must be zero)

Notes

The action performed depends on the reason code value in R1.

This call is just like FileCore_DiscOp except that it attacks a number of shortcomings in old versions of FileCore:

- Alternative disc record pointers passed in to various SWIs currently only have a 26-bit range so will not work in 32 bit systems.
- FileCore_SectorDiscOp limits discs to 2^{29} sectors, i.e. 256GB for drives with 512 byte sectors.

In the future, this will allow for up to 16 exbibytes (over 16 billion GB) per disc, and many more discs per filing system.

As high (top-bit set) addresses are possible in a scatter list, only negative numbers smaller or equal to than -64K (ie $\geq$ unsigned &FFFF0000) should be interpreted as the end of list marker. This applies to any FileCore module for RISC OS 3.50 or later.

[RISC OS Open source page](#)

FileCore_Drives

SWI &40542 · FileCore

The purpose of this call is to return information on the filing system's drives.

Before calling

R8 Pointer to FileCore instance private word

On return

R0 Default drive
R1 Number of floppy drives
R2 Number of hard disc drives
R8 Preserved ?
[RISC OS Open source page](#)

FileCore_Features

SWI &4054D · FileCore

The purpose of this call is to return any additional features of FileCore to a client filing system prior to it registering.

Before calling

--

On return

R0 Bitmask of supported features

Notes

This call is only of use to filing systems which register as a FileCore filing system to allow it to select different code paths where necessary to communicate with older versions of FileCore.

With old versions of FileCore this will be an unknown SWI, but each new feature flag is defined to be set so:

SWI "XFileCore_Features"

MOVVS R0, #0; no new features at all

This call is only available from RISC OS 5.00 onwards.

[RISC OS Open source page](#)

FileCore_FloppyStructure

SWI &40544 · FileCore

The purpose of this call is to create a RAM image of a floppy disc map and root directory entry.

Before calling

R0 Pointer to buffer (must be $\geq 4k$ long)
R1 Pointer to disc record describing shape and format
R2 Flags
R3 Pointer to list of defects

On return

R0 Preserved ?
R1 Preserved ?
R2 Preserved ?

R3 Total size of structure created

Notes

The pointer to a list of defects is only needed for new map discs. You don't need to know a FileCore instance private word to use this call; instead the disc record tells FileCore which filing system is involved.

[RISC OS Open source page](#)

FileCore_FreeSpace

SWI &40543 · FileCore

The purpose of this call is to return the total free space on given disc, and the largest object that can be created on it.

Before calling

R0 Pointer to disc specifier (name, or :0, etc...)

R8 Pointer to FileCore instance private word

On return

R0 Total free space on disc

R1 Size of largest object that can be created

R8 Preserved ?

Notes

As this call cannot support large discs, the maximum value of &7FFFFFFF is a file 'at least 2GB' in size.

FileCore_FreeSpace64 provides support for larger discs, and should be used in preference.

[RISC OS Open source page](#)

FileCore_FreeSpace64

SWI &4054B · FileCore

The purpose of this call is to return the total free space on given disc, and the largest object that can be created on it.

Before calling

R0 Pointer to disc specifier (name, or :0, etc...)

R8 Pointer to FileCore instance private word

On return

R0 Free space (least significant word)

R1 Free space (most significant word)

R2 Size of largest object that can be created, or &7FFFFFFF if > 2GB

R8 Preserved ?

Notes

This call has support for 64-bit addressing, and thus can support large discs. This call should be used in preference to the earlier FileCore_FreeSpace that could only support much smaller discs.

[RISC OS Open source page](#)

FileCore_LayoutStructure

SWI &40548 · FileCore

The purpose of this call is to lay out into the specified file a set of structure corresponding to the identified format.

Before calling

R0 Identifier of particular format

R1 Pointer to bad block list (terminated by -1)

R2 Pointer to disc name (null terminated)

R3 Image file handle

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

The format identifier is a pointer to a disc record. An error is returned if the specified format can't map out defects, and there were defects in the defect list.

This call isn't available under RISC OS 2.

[RISC OS Open source page](#)

FileCore_MiscOp

SWI &40549 · FileCore

The purpose of this call is to provide a number of miscellaneous low-level functions for accessing drives.

Before calling

R0 Reason code

R1 Drive number

R8 Pointer to Filecore instance private word

All other registers dependent on reason code

On return

All other registers dependent on reason code

Reason Codes

- 0 – Hex #: &00 · Action: Mount
- 1 – Hex #: &01 · Action: Poll changed
- 2 – Hex #: &02 · Action: Lock floppy drive
- 3 – Hex #: &03 · Action: Unlock floppy drive
- 4 – Hex #: &04 · Action: Poll period
- 5 – Hex #: &05 · Action: Eject disc
- 6 – Hex #: &06 · Action: Read processed FileCore_Create block
- 7 – Hex #: &07 · Action: Read drive flags

Notes

The action performed depends on the reason code value in R0.

[RISC OS Open source page](#)

FileCore_MiscOp 0

SWI &40549 · FileCore

The purpose of this call is to mount a disc, reading in the data asked for.

Before calling

- R0** 0
- R1** Drive number
- R2** Disc address to read from
- R3** Pointer to buffer
- R4** Length to read into buffer
- R5** Pointer to disc record to fill in (floppies and floppy like hard discs only)
- R8** Pointer to Filecore instance private word

On return

- R0** Preserved
- R1** Preserved
- R2** Preserved
- R3** Preserved
- R4** Preserved
- R5** Preserved
- R8** Preserved ?

Notes

For a floppy disc, and for a hard disc where bit 4 of the descriptor block flags is set, this call asks the given filing system to first identify the disc's format. The suggested density to try first is given in the disc record; if this is not successful, the filing system should then try other densities. The following order is suggested:

- Quad density
- Double density
- Octal density
- Single density
- Double+ density

Once the filing system has identified the disc's format, it fills in the 'log2secsize', 'secspertrack', 'heads', 'density', 'lowsector' and 'root' values in the disc record.

- If 'log2seccsize' <=8, then it gives 'heads' the value (actual number of heads -1), and sets bit 6 of 'lowsector', so sides are treated as sequenced. Otherwise (i.e. when 'log2seccsize' >8) it gives 'heads' the value (actual number of heads), and clears bit 6 of low sector, so sides are treated as interleaved.
- The filing system clears bit 7 of 'lowsector'; this is used as an initial value, which FileCore subsequently corrects if necessary.

Having filled in the disc record, the filing system then reads in the data asked for.

For hard discs where bit 4 of the descriptor block flags is clear, this merely asks the given filing system to read in the data asked for. This typically necessitates it reading the boot block of the disc; if the disc doesn't have one, the filing system generates one itself.

[RISC OS Open source page](#)

FileCore_MiscOp 1

SWI &40549 · FileCore

The purpose of this call is to return information on the drive to see if a change has occurred.

Before calling

R0 1

R1 Drive number

R2 Sequence number

R8 Pointer to FileCore instance private word

On return

R0 Preserved ?

R1 Preserved ?

R2 Sequence number

R3 Result flag

R8 Preserved ?

Notes

The sequence number is to ensure no changes are lost due to reset being pressed. Both the given filing system and the FileCore incarnation should start with a sequence number of 0 for each drive. The filing system increments the sequence number with each change of state. If the filing system finds the entry sequence number doesn't match its copy it should return changed/maybe changed, depending on whether the disc changed line works/doesn't work.

[RISC OS Open source page](#)

FileCore_MiscOp 2

SWI &40549 · FileCore

The purpose of this call is to lock a floppy disc in a drive.

Before calling

R0 2

R1 Floppy drive number

R8 Pointer to FileCore instance private word

On return

R0 Preserved ?

R1 Preserved ?

R8 Preserved ?

Notes

It should at least ensure that the drive light stays on until unlocked. Note that locks are counted, so each 'lock drive' must be matched by an 'unlock drive'.

This call only works for floppy disc drives.

FileCore_MiscOp 3 reverses a single MiscOp 2 call.

[RISC OS Open source page](#)

FileCore_MiscOp 3

SWI &40549 · FileCore

The purpose of this call is to unlock a floppy disc in a drive.

Before calling

R0 3

R1 Floppy drive number

R8 Pointer to FileCore instance private word

On return

R0 Preserved ?

R1 Preserved ?

R8 Preserved ?

Notes

This call only works for floppy disc drives.

This call reverses a single FileCore_MiscOp 2 call.

[RISC OS Open source page](#)

FileCore_MiscOp 4

SWI &40549 · FileCore

The purpose of this call is to inform FileCore of the minimum period between polling for disc insertion under the given filing system.

Before calling

R0 4

R1 Drive number

R8 Pointer to FileCore instance private word

On return

R0 Preserved ?

R1 Preserved ?

R5 Minimum polling period (in centiseconds), or -1 if changed doesn't work

R6 Pointer to media type string (e.g. 'disc' for ADFS)

R8 Preserved ?

Notes

This is so that drive lights don't remain continuously illuminated.

The values are re-exported by FileCore in the UpCalls:

- 'Media not present' – see OS_UpCall 1
- 'Media not known' – see OS_UpCall 2

The value applies to all drives rather than a particular drive.

[RISC OS Open source page](#)

FileCore_MiscOp 5

SWI &40549 · FileCore

The purpose of this call is to causes a FileCore system to eject a disc.

Before calling

R0 5

R1 Drive number, bit 31 set if trying to find a disc

R4 Iteration count (if bit 31 of R1 set)

R5 Timeout period (if bit 31 of R1 set)

R8 Pointer to FileCore instance private word

On return

R0 Preserved ?

R1 Preserved ?

R4 Preserved ?

R5 Preserved ?

R8 Preserved ?

Notes

The following UpCalls will cause bit 31 to be set:

- 'Media not present' – see OS_UpCall 1
- 'Media not known' – see OS_UpCall 2

This indicates that you should choose a suitable drive to eject the disc from.

[RISC OS Open source page](#)

FileCore_MiscOp 6

SWI &40549 · FileCore

The purpose of this call is to read the FileCore_Create block used to initialise this filing system.

Before calling

R0 6

R8 Pointer to FileCore instance private word

On return

R0 Pointer to FileCore_Create block, with address in place of offsets

R8 Preserved ?

Notes

Available on RISC OS 3.60 onwards.

[RISC OS Open source page](#)

FileCore_MiscOp 7

SWI &40549 · FileCore

The purpose of this call is to read flags about the drive.

Before calling

R0 7

R1 Drive number

R8 Pointer to FileCore instance private word

On return

R0 Flag, if bit 0 set then Drive is locked

R1 Preserved ?

R8 Preserved ?

Notes

Available on RISC OS 3.60 onwards.

[RISC OS Open source page](#)

FileCore_SectorDiscOp

SWI &4054A · FileCore

The purpose of this call is to provide a number of low-level disc operations.

Notes

This SWI is identical to FileCore_DiscOp, except that the addresses given are in sectors, not bytes.

For information on how to use this call, please see FileCore_DiscOp.

In previous versions of FileCore (RISC OS 3.60-4.x / 6.x) this SWI is named FileCore_SectorOp

[RISC OS Open source page](#)

FilerAction_SendSelectedDirectory

SWI &40F80 · Filer_Action

The purpose of this call is to send a message specifying the selected directory.

Before calling

R0 Task handle to which to send the message

R1 Pointer to directory name (including null terminator)

On return

--

Notes

Sends the Wimp message Message_FilerSelectionDirectory

[RISC OS Open source page](#)

FilerAction_SendSelectedFile

SWI &40F81 · Filer_Action

The purpose of this call is to send a message specifying the selected files within a directory.

Before calling

R0 Task handle to which to send the message

R1 Pointer to selection name (including null terminator)

On return

--

Notes

This call accumulates the names of selected files that you pass to it. When it has received sufficient filenames to fill a Wimp message block it sends those names using the Wimp message Message_FilerAddSelection. The same message is used to send any unsent filenames if you subsequently call FilerAction_SendStartOperation.

[RISC OS Open source page](#)

FilerAction_SendStartOperation

SWI &40F82 · Filer_Action

The purpose of this call is to send a message containing information to start operation.

Before calling

- R0** Task handle to which to send the message
- R1** Reason code (as described below)
- R2** Option bits (as described below)
- R3** Pointer to operation specific data (see use cases below)
- R4** Length of operation specific data (see use cases below)

On return

- -

Reason Codes

- 0** – Action: Copy
- 1** – Action: Move (rename)
- 2** – Action: Delete
- 3** – Action: Set access
- 4** – Action: Set type
- 5** – Action: Count
- 6** – Action: Move (copy then delete original)
- 7** – Action: Copy local (within same directory)
- 8** – Action: Stamp files
- 9** – Action: Find File

Option Bits

Bit / Meaning when set

- 0** – Meaning when set: Verbose
- 1** – Meaning when set: Confirm
- 2** – Meaning when set: Force
- 3** – Meaning when set: Newer (as opposed to just Look)
- 4** – Meaning when set: Recurse (only applies to access)

Use Cases

Copy: Reason Code = 0

Entry

R3 · Pointer to name of destination directory (null terminated)

R4 · Length of name of destination directory (including null terminator)

Move: Reason Code = 1

Entry

R3 · Pointer to name of destination directory (null terminated)

R4 · Length of name of destination directory (including null terminator)

Delete: Reason Code = 2

Entry

R3 · Unused

R4 · 0

Set access: Reason Code = 3

Entry

R3 · Pointer to word containing required new access:

Bottom two bytes indicate the values to set

Top two bytes flag which bits are to be left alone

R4 · 4

Set type: Reason Code = 4

Entry

R3 · Pointer to word containing new type in bits 0-11

R4 · 4

Count: Reason Code = 5

Entry

R3 · Unused

R4 · 0

Copy Move: Reason Code = 6

Entry

R3 · Pointer to name of destination directory (null terminated)

R4 · Length of name of destination directory (including null terminator)

Copy Local: Reason Code = 7

Entry

R3 · Pointer to name of destination name (null terminated)

R4 · Length of name of destination name (including null terminator)

Stamp: Reason Code = 8

Entry

R3 · Unused

R4 · 0

Find: Reason Code = 9

Entry

R3 · Pointer to name of object to Find (null terminated)

R4 · Length of name of object to find (including null terminator)

Notes

This SWI sends any remaining leafnames with `Message_FilerAddSelection`, then starts the selected operation by constructing and sending `Message_FilerAction`.

Reason dependent data is always a pointer, even for word sized data. Usually the data is a destination pathname.

[RISC OS Open source page](#)

Filter_DeRegisterCopyFilter

SWI &42647 · FilterManager

The purpose of this call is to remove a `Wimp_BlockCopy` filter from the list.

Before calling

R0 Pointer to zero terminated filter name

R1 Address of `RectangleCopy` Filter Code

R2 Value to be passed to R12

On return

All registers preserved

Notes

All registers must be the same as those passed to `Filter_RegisterCopyFilter`.

[RISC OS Open source page](#)

Filter_DeRegisterPostFilter

SWI &42643 · FilterManager

The purpose of this call is to remove a filter from the list of post-filters.

Before calling

R0 Pointer to zero terminated filter name

R1 Address of Post Filter Code

R2 Value to be passed to R12

R3 Task handle of task to which filter was applied

R4 Poll Mask

On return

All registers preserved

Notes

All values must be the same as those passed to `Filter_RegisterPostFilter`

[RISC OS Open source page](#)

Filter_DeRegisterPostIconFilter

SWI &4264B · FilterManager

The purpose of this call is to remove a `Wimp_GetRectangle` icon plotting filter from the list of post-filters.

Before calling

R0 Pointer to zero terminated filter name

R1 Address of `Wimp_GetRectangle` Icon Filter Code

R2 Value to be passed to R12

R3 Task handle of task to which filter was applied

On return

All registers preserved

Notes

All registers must be the same as those passed to `Filter_RegisterPostIconFilter`

[RISC OS Open source page](#)

Filter_DeRegisterPostRectFilter

SWI &42649 · FilterManager

The purpose of this call is to add a remove a Wimp_GetRectangle filter from the list of post-filters.

Before calling

R0 Pointer to zero terminated filter name

R1 Address of Post Rectangle Filter Code

R2 Value to be passed to R12

R3 Task handle of task to which filter was applied

On return

All registers preserved

Notes

All registers must be the same as those passed to Filter_RegisterPostRectFilter.

[RISC OS Open source page](#)

Filter_DeRegisterPreFilter

SWI &42642 · FilterManager

The purpose of this call is to remove a filter from the list of pre-filters.

Before calling

R0 Pointer to zero terminated filter name

R1 Address of Pre Filter Code

R2 Value to be passed to R12

R3 Task handle of task to which filter was applied

On return

All registers preserved

Notes

All values must be the same as those passed to Filter_RegisterPreFilter

[RISC OS Open source page](#)

Filter_DeRegisterRectFilter

SWI &42645 · FilterManager

The purpose of this call is to add a remove a Wimp_GetRectangle filter from the list of pre-filters.

Before calling

R0 Pointer to zero terminated filter name

R1 Address of Pre Rectangle Filter Code

R2 Value to be passed to R12

R3 Task handle of task to which filter was applied

On return

All registers preserved

Notes

All registers must be the same as those passed to Filter_RegisterPostRectFilter.

[RISC OS Open source page](#)

Filter_RegisterCopyFilter

SWI &42646 · FilterManager

The purpose of this call is to add a new Wimp_BlockCopy filter to the list of pre-filters.

Before calling

R0 Pointer to zero terminated filter name

R1 Address of RectangleCopy Filter Code

R2 Value to be passed to R12

On return

All registers preserved

Notes

The address pointed to by R1 will be called whenever a task calls Wimp_BlockCopy with R12 equal to the value of R2 when this SWI is called.

[RISC OS Open source page](#)

Filter_RegisterPostFilter

SWI &42641 · FilterManager

The purpose of this call is to add a new filter to the list of post-filters.

Before calling

R0 Pointer to zero terminated filter name

R1 Address of Post Filter Code

R2 Value to be passed to R12

R3 Task handle of task to which filter is applied, or 0 for all tasks

R4 Poll Mask (value of 1 masks the event – same as Wimp_Poll)

On return

All registers preserved

Notes

The filter routine at the address pointed to by R1 will be called whenever the Wimp is about to return from Wimp_Poll. When registering the filter use R3 to control whether the filter is called for all tasks or only for a specific task.

See Post Filter Code for details of how the filter routine must operate.

[RISC OS Open source page](#)

Filter_RegisterPostIconFilter

SWI &4264A · FilterManager

The purpose of this call is to add a new Wimp_GetRectangle icon plotting filter to the list of post-filters.

Before calling

R0 Pointer to zero terminated filter name

R1 Address of Wimp_GetRectangle Icon Filter Code

R2 Value to be passed to filter code in R12

R3 Task handle of task to which filter is applied, or 0 for all tasks

On return

All registers preserved

Notes

The filter routine address pointed to by R1 will be called whenever the task who's handle is passed in R3 calls Wimp_GetRectangle. It will be called just after the windows icons have been plotted.

[RISC OS Open source page](#)

Filter_RegisterPostRectFilter

SWI &42648 · FilterManager

The purpose of this call is to add a new Wimp_GetRectangle filter to the list of post-filters.

Before calling

R0 Pointer to zero terminated filter name

R1 Address of Post Rectangle Filter Code

R2 Value to be passed to R12

R3 Task handle of task to which filter is applied, or 0 for all tasks

On return

All registers preserved

Notes

The address pointed to by R1 will be called whenever the task whose handle is passed in R3 calls Wimp_GetRectangle.

[RISC OS Open source page](#)

Filter_RegisterPreFilter

SWI &42640 · FilterManager

The purpose of this call is to add a new filter to the list of pre-filters.

Before calling

R0 Pointer to zero terminated filter name

R1 Address of Pre Filter Code

R2 Value to be passed to R12

R3 Task handle of task to which filter is applied, or 0 for all tasks

On return

All registers preserved

Notes

The address pointed to by R1 will be called with R12 equal to the value of R2, whenever the task whose handle is passed in R3 calls Wimp_Poll.

See Pre Filter Code for details of how the filter routine must behave.

[RISC OS Open source page](#)

Filter_RegisterRectFilter

SWI &42644 · FilterManager

The purpose of this call is to add a new Wimp_GetRectangle filter to the list of pre-filters. This filter is called just before the drawing of a rectangle begins, before the window background has been filled, and before the VDU graphics window has been set up.

Before calling

R0 Pointer to zero terminated filter name

R1 Address of Pre Rectangle Filter Code

R2 Value to be passed to R12

R3 Task handle of task to which filter is applied, or 0 for all tasks

On return

All registers preserved

Notes

The address pointed to by R1 will be called whenever the task who's handle is passed in R3 calls Wimp_GetRectangle.

[RISC OS Open source page](#)

Font_ApplyFields

SWI &400A7 · FontManager

This purpose of this call is to merge new fields into a new font identifier.

Before calling

R0 Pointer to original font identifier

R1 Pointer to fields to add (in font identifier)

R2 Pointer to output buffer (0 to return size)

R3 Size of output buffer

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Free space remaining in output buffer or size of buffer needed

Notes

Any existing field values in an existing font identifier are replaced when this call is used. By adding null fields, an existing field can be deleted. Fields not present in original identifier are placed in the result after fields present in the original identifier.

R2 on entry specified can also be used to return the size of the buffer if a value of zero is passed.

[RISC OS Open source page](#)

Font_CacheAddr

SWI &40080 · FontManager

This call returns the version number of the FontManager module, and information about the font cache.

Before calling

--

On return

R0 Version number x 100

R2 Size of font cache

R3 Bytes used in font cache

[RISC OS Open source page](#)

Font_Caret

SWI &40087 · FontManager

This call draws a caret at the given position.

Before calling

R0 Colour

R1 Height (OS units)

R2 Font_Paint Flags

R3 X co-ordinate

R4 Y co-ordinate

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

Notes

The colour is EORed onto the screen, so to obtain a specific colour, the background colour must be taken into consideration.

[RISC OS Open source page](#)

Font_CharBBox

SWI &4008E · FontManager

This purpose of this call is to return the bounding box for the given character of the given font.

Before calling

R0 Font handle

R1 Character

R2 Font_Paint Flags (only bit 4 is of use)

On return

R0 Preserved

R1 Minimum x (inclusive)

R2 Minimum y (inclusive)

R3 Maximum x (exclusive)

R4 Maximum y (exclusive)

Notes

The bounding box is the smallest box necessary to cover the character.

[RISC OS Open source page](#)

Font_ConverttoOS

SWI &40088 · FontManager

This purpose of this call is to convert co-ordinates from millipoints to OS units.

Before calling

R1 X millipoints

R2 Y millipoints

On return

R1 X OS units

R2 Y OS units

Notes

This call uses the current scale factor when converting. The default scale factor is 400 millipoints per OS unit.

To avoid overflow during the internal calculations, the input values must be in the range -536870912 (&E0000000) inclusive to 536870912 (&20000000) exclusive.

[RISC OS Open source page](#)

Font_Converttopoints

SWI &40089 · FontManager

This purpose of this call is to convert the co-ordinates from OS units to millipoints.

Before calling

R1 X OS units

R2 Y OS units

On return

R0 Corrupted

R1 X millipoints

R2 Y millipoints

Notes

This call uses the current scale factor when converting. The default scale factor is 400 millipoints per OS unit.

[RISC OS Open source page](#)

Font_CurrentFont

SWI &4008C · FontManager

This purpose of this call is to return the current colour settings of the Font Manager.

Before calling

--

On return

R0 Font handle

R1 Background colour

R2 Foreground colour

R3 Foreground colour offset

[RISC OS Open source page](#)

Font_CurrentRGB

SWI &400A3 · FontManager

This call returns the current colour settings after calling Font_Paint.

Before calling

--

On return

R0 Font handle

- R1** Background colour
- R2** Foreground colour
- R3** Maximum colour offset

Notes

If the colours were not set using RGB values it will generate the error Undefined RGB font colours.

Colours are returned in the form &BBGGRR00.

If R3 is zero, then the colours used are monochrome only (foreground / background). Any other value means the font is displayed using anti-aliasing.

[RISC OS Open source page](#)

Font_DecodeMenu

SWI &400A0 · FontManager

This call decodes the selection made from a font menu.

Before calling

- R0** Font_DecodeMenu Flags
- R1** Pointer to menu definition, as per Font_ListFonts
- R2** Pointer to menu selection, as per Menu_Selection within Wimp_Poll
- R3** Pointer to buffer to contain result, or 0 to return size
- R4** Size of buffer (if R3 <> 0)

On return

- R0** Preserved
- R1** Preserved
- R2** Pointer to rest of menu selections
- R3** Preserved
- R4** Size of buffer required, or 0 for no font selection

Notes

The definition of the font menu is returned in the same format as used by Font_ListFonts.

The format of the returned string depends on whether the names of the fonts or font encodings have been specified in the Messages_n_ file within the font directory.

If no Messages_n_ file was found (and hence the Font Manager has found the font names by scanning the directory tree) the name field is absent.

Details of the Font_DecodeMenu Result are available.

Experimentation with this call suggests that passing in a selection block pointing at the System Font will generate an error, rather than set R4 = 0.

[RISC OS Open source page](#)

Font_EnumerateCharacters

SWI &400A9 · FontManager

This purpose of this call is to determine which characters are present in a font, and which glyphs in the underlying font file characters map to.

Before calling

R0 Font handle (0 for current)

R1 Character code (0 to start)

R1 = 0 R1 = &20 (space) · R2 = -1

R1 = &10F (d-caron) R1 = &112 (E macron) · R2 = &151

R1 = &112 (E-macron) R1 = &113 (e macron) · R2 = &185

R1 = &113 (e-macron) R1 = &116 (E dot) · R2 = &195

R1 = &FB02 (f1 ligature) R1 = -1 · R2 = &FF

On return

R0 Corrupted

R1 Next character (-1 if this is the last)

R2 Internal character code of this character (-1 if unmapped)

Notes

This call works only by looking at encoding files – it cannot guarantee that a given character is actually defined in a font file, but it can return which characters definitely are not, by returning with R2 set to -1.

Examples

For the font \FHomerton.Medium\EUTF8, a call sequence might be:

So, in this example, we see that the f1 ligature character (Unicode FB02) is character &FF within the Homerton.Medium font file.

This call may be of use in conjunction with Font_ReadFontMetrics kerning data. Also, it would allow a character map program to scan through the UCS space to find defined characters.

[RISC OS Open source page](#)

Font_FindCaret

SWI &4008D · FontManager

This purpose of this call is to return information about which character is closest to the given offset.

Before calling

R1 Pointer to string

R2 X offset in millipoints

R3 Y offset in millipoints

On return

R1 Pointer to closest character

- R2** Width of string
- R3** Height of string
- R4** Length of string
- R5** Index of closest character

Notes

This call is deprecated. `Font_ScanString` should be used instead.

[RISC OS Open source page](#)

Font_FindCaretJ

SWI &40096 · FontManager

This purpose of this call is to return information about which character is closest to the given offset, within a justified string.

Before calling

- R1** Pointer to string
- R2** X offset in millipoints
- R3** Y offset in millipoints
- R4** X justification offset
- R5** Y justification offset

On return

- R1** Pointer to closest character
- R2** Width of string
- R3** Height of string
- R4** Length of string
- R5** Index of closest character

Notes

This call is deprecated. `Font_ScanString` should be used instead.

The supplied justification is divided on all spaces in the string.

[RISC OS Open source page](#)

Font_FindField

SWI &400A6 · FontManager

This call returns a pointer to the specified field with the font string.

Before calling

- R1** Pointer to font identifier
- R2** Character code of required qualifier

On return

R1 Pointer to value following qualifier within string (if present, else reserved)

R2 0 if field not present, otherwise preserved

Notes

The field is space terminated; all others are control character terminated.

[RISC OS Open source page](#)

Font_FindFont

SWI &40081 · FontManager

This call returns a handle for the given font name, and make it the current font.

Before calling

R1 Pointer to control terminated font name

R2 X points x 16

R3 Y points x 16

R4 X DPI, or 0 for default

R5 Y DPI, or 0 for default

On return

R0 Font handle

R4 X DPI

R5 Y DPI

Notes

When the font is no longer required, Font_LoseFont should be used. If R1 points to a string that does not contain any '\ characters it is assumed to be a font name. Otherwise it is treated as a font identifier.

[RISC OS Open source page](#)

Font_FutureFont

SWI &4008C · FontManager

This purpose of this call is to return what the state after the next call to Font_Paint would be.

Before calling

--

On return

R0 Font handle

R1 Background colour

R2 Foreground colour

R3 Foreground colour offset

Notes

Font_StringWidth must be called before this call is of use.

[RISC OS Open source page](#)

Font_FutureRGB

SWI &400A4 · FontManager

This call returns the current colour settings after calling any of the following: Font_ScanString, Font_StringWidth, Font_StringBBox, Font_FindCaret or Font_FindCaretJ.

Before calling

--

On return

R0 Font handle

R1 Background colour

R2 Foreground colour

R3 Maximum colour offset

Notes

If the colours were not set using RGB values it will generate the error undefined RGB font colours.

The colours are returned in the following form: &BBGRR00.

If the maximum colour offset is zero, then anti-aliasing is not used.

[RISC OS Open source page](#)

Font_ListFonts

SWI &40091 · FontManager

This SWI has two syntax forms:

In practice

- The basic form returns a list of all font/encoding names known to the Font Manager.
- The more complex form builds a menu definition of fonts/encodings known to the Font Manager.
- Font_ListFonts (Basic)
- Font_ListFonts (Menu)

[RISC OS Open source page](#)

Font_ListFonts (Basic)

SWI &400A7 · FontManager

This call returns, one at a time, the fonts or encodings found with a given path.

Before calling

R1 Pointer to buffer for identifier, or 0 to find required size

R2 Counter and flags

R3 Size of buffer in R1

R4 Pointer to buffer for font name, or 0 to find required size

R5 Size of buffer in R4

On return

R2 New count, or -1 if no more

R3 Required size of buffer in R1 if R1 = 0, otherwise length of data placed in buffer

R5 Required size of buffer in R4 if R4 = 0, otherwise length of data placed in buffer

Flags values

Bit / Meaning

0-15 – Meaning: Counter, or 0 for first call

16 – Meaning: Return font identifier in R1

17 – Meaning: Return local name in R4

18 – Meaning: Terminate with CR rather than 0

22 – Meaning: Return list of encodings rather than fonts

23-31 – Meaning: Reserved; must be 0

Notes

Names are returned alphabetically sorted. For this form of the call, only bits 16-18 and bit 22 of the flags in R2 should be used. All other bits must be clear.

Do not alter R2 between calls for finding required memory and finding actual data. If R2 = -1 on exit, then all fonts have been found and returned data is invalid.

[RISC OS Open source page](#)

Font_ListFonts (Menu)

SWI &400A7 · FontManager

This call creates a list of fonts arranged as a menu.

Before calling

R1 Pointer to buffer for font menu, or 0 to find required size

R2 Counter and flags

R3 Size of buffer in R1

R4 Pointer to buffer for indirected data, or 0 to find required size

R5 Size of buffer in R4

R6 Pointer to identifier to tick, or 0 for none, or 1 for System Font

On return

R2 New count, or -1 if no more

R3 Required size of buffer in R1 if R1 = 0, or 0 if the menu is null, otherwise length of data placed in buffer

R5 Required size of buffer in R4 if R4 = 0, otherwise length of data placed in buffer

Flags values

Bit / Meaning

0-15 – Meaning: Counter, or 0 for first call

19 – Meaning: Return menu definition in R1, indirected data in R4

20 – Meaning: Place System Font at head of menu

21 – Meaning: Tick font in R6, and its parent

22 – Meaning: Return list of encodings rather than fonts

23-31 – Meaning: Reserved; must be 0

Notes

For this form of the call, only bits 19-22 of the flags in R2 should be used. All other bits must be clear.

The returning definition is suitable for opening with Wimp_CreateMenu. Font_DecodeMenu can be used to work out which font was selected.

[RISC OS Open source page](#)

Font_LookupFont

SWI &400A8 · FontManager

This call returns information about a font as a bitmap in R2.

Before calling

R0 Font handle (0 for current handle)

R1 0

R2 0

On return

R0 Preserved

R1 Preserved

R2 Characteristics of font

R3 Corrupted

R4 Corrupted

R5 Corrupted

R6 Corrupted

R7 Corrupted

[RISC OS Open source page](#)

Font_LoseFont

SWI &40082 · FontManager

This call informs the Font Manager that the font (specified by R0) is no longer required.

Before calling

R0 Font handle

On return

--

[RISC OS Open source page](#)

Font_MakeBitmap

SWI &40099 · FontManager

This call creates a bitmap file stored in the font's directory.

Before calling

R1 Font handle, or pointer to identifier

R2 X points x 16

R3 Y points x 16

R4 X DPI

R5 Y DPI

R6 Flags

On return

--

Notes

A font of a particular size can be stored in the font's directory, which results in a faster caching of the font, especially for subpixel positioning.

[RISC OS Open source page](#)

Font_Paint

SWI &40086 · FontManager

This call paints a string to the screen using the currently selected font, or a specific font (as per flags).

Before calling

R0 Font handle (if applicable)

R1 Pointer to string (may include control characters)

R2 Flags

R3 X

R4 Y

R5 Pointer to block (if applicable)

R6 Pointer to transformation matrix (if applicable)

R7 Length (if applicable)

On return

- All registers preserved

Notes

Any VDU origin previously specified is ignored and absolute co-ordinates are assumed.

Underline and rubout do not work unless the transformed x-axis is parallel (not anti-parallel) to the output x-axis.

(Anti-parallel means being parallel, but in the opposite direction to the original.)

[RISC OS Open source page](#)

Font_ReadColourTable

SWI &40098 · FontManager

This call returns the 16 colours used to do anti-aliasing.

Before calling

R1 Pointer to a 16-byte buffer

On return

R1 Preserved

[RISC OS Open source page](#)

Font_ReadDefn

SWI &40083 · FontManager

This SWI has two syntax forms:

In practice

- The size form returns the size of buffer required to hold the information about the font
- The information form returns information about the given font
- Font_ReadDefn (information syntax)
- Font_ReadDefn (size syntax)

[RISC OS Open source page](#)

Font_ReadDefn (information syntax)

SWI &40083 · FontManager

This call returns information about the given font.

Before calling

R0 Font handle

R1 Pointer to buffer for font name

On return

R1 Preserved

R2 X points x16

R3 Y points x16

R4 X DPI

R5 Y DPI

R6 Age of font (number of font calls since this one was last accessed)

R7 Usage count

Notes

Before using this syntax form, the Font_ReadDefn (size syntax) should first be called to obtain the buffer size required to hold the font information returned by this syntax form.

[RISC OS Open source page](#)

Font_ReadDefn (size syntax)

SWI &40083 · FontManager

This call returns the size of buffer required to store information about the given font if using the Font_ReadDefn (information syntax) form of this call.

Before calling

R0 Font handle

R1 0

R3 'Full' (&4C4C5546)

On return

R1 Preserved

R2 Size of buffer required to store font information

R3 Corrupted

R4 Corrupted

R5 Corrupted

R6 Corrupted

R7 Corrupted

[RISC OS Open source page](#)

Font_ReadEncodingFilename

SWI &400A5 · FontManager

This call returns the filename of the encoding file, and is primarily of use to PDriverPS.

Before calling

R0 Font handle

R1 Pointer to buffer

R2 Length of buffer

On return

R0 Pointer to encoding filename (in buffer)

R1 Pointer to terminating 0 in buffer

R2 Free bytes in buffer

Notes

The filename form returned differs depending on whether the font has a public or private encoding. The table below shows the filename form for both encoding types.

Type / Filename

Public – Filename: font_prefix.Encoding.encoding

Private – Filename: font_prefix.font_name.Encoding

[RISC OS Open source page](#)

Font_ReadFontMax

SWI &4009C · FontManager

This purpose of this call is to read the current FontMax and FontMax1 to FontMax5 values.

Before calling

--

On return

R0 FontMax in bytes

R1 FontMax1 in points

R2 FontMax2 in points

R3 FontMax3 in points

R4 FontMax4 in points

R5 FontMax5 in points

R6 Corrupted (maybe)

R7 Corrupted (maybe)

Notes

The values returned may not necessarily be the configured values, as the call can return the current used values.

[RISC OS Open source page](#)

Font_ReadFontMetrics

SWI &4009F · FontManager

This call reads information in the font's IntMetrics file.

Before calling

- R0** Font handle
- R1** Pointer to buffer for bounding box array, 0 to read size
- R2** Pointer to buffer for X width data, 0 to read size
- R3** Pointer to buffer for Y width data, 0 to read size
- R4** Pointer to buffer for miscellaneous data, 0 to read size
- R5** Pointer to buffer for kerning data, 0 to read size
- R6** Pointer to buffer for new style kerning data, 0 to read size¹
- R7** 0

On return

- R0** File flags
- R1** Size of bounding box data
- R2** Size of X width data
- R3** Size of Y width data
- R4** Size of miscellaneous data
- R5** Size of kerning data
- R6** Size of new style kerning data
- R7** Corrupted (maybe)

In practice

- ¹ Available from version 3.41 onwards.

Notes

Currently not permitted on fonts which have a transformation matrix applied. All values are signed millipoints, unless specifically stated.

Width data is an array of 256 words, bounding box data an array of 256×4 words (x0, y0, x1, y1). This data is only returned for the first 256 characters of the font.

This call is of doubtful use for fonts with up to 2 billion characters, so should use Font_CharBBox or Font_ScanString instead of using R1 to R3 in this call.

The output values of R1 to R6 will be zero if no information is present in the file be examined.

[RISC OS Open source page](#)

Font_ReadFontPrefix

SWI &4009D · FontManager

This call returns the full path to the font data.

Before calling

R0 Font handle

R1 Pointer to buffer

R2 Length of buffer

On return

R0 Preserved

R1 Pointer to terminating 0 in buffer

R2 Free bytes in buffer

[RISC OS Open source page](#)

Font_ReadInfo

SWI &40084 · FontManager

This purpose of this call is to return the smallest possible box needed to cover any character in the given font.

Before calling

R0 Font handle

On return

R1 Minimum x (inclusive)

R2 Minimum y (inclusive)

R3 Maximum x (exclusive)

R4 Maximum y (exclusive)

In practice

The output is returned in OS units.

This call is deprecated. Font_CharBBox should be used instead.

[RISC OS Open source page](#)

Font_ReadScaleFactor

SWI &4008F · FontManager

This call returns the X and Y scale factors used when converting between OS co-ordinates and millipoints.

Before calling

--

On return

R1 X scale factor

R2 Y scale factor

Notes

To set values, `Font_SetScaleFactor` should be used.

[RISC OS Open source page](#)

Font_ReadThresholds

SWI &40094 · FontManager

This call reads the threshold values used by the Font Manager when painting.

Before calling

R1 Pointer to buffer to receive threshold values?

On return

R1 Preserved

Notes

The length of table required is one byte for each logical colour used (i.e $3 + \text{ABS}(\text{foreground_colour_offset})$), which can be read using `Font_CurrentFont`.

[RISC OS Open source page](#)

Font_ScanString

SWI &400A1 · FontManager

This call returns information on a string.

Before calling

R0 Initial font handle, or 0 for current

R1 Pointer to string

R2 Flags

R3 X co-ordinate of mouse click, or width of string

R4 Y co-ordinate of mouse click, or height of string

R5 Pointer to block

R6 Pointer to Font Paint Transformation Matrix

R7 Length of string

On return

R1 Pointer to where scan ended

R2 Preserved

R3 X of where scan ended

R4 Y of where scan ended

R5 Preserved

R6 Preserved

R7 Number of split characters in scanned part of string

Notes

How to use the call:

- How wide is a string? Use large values for R3, R4 and no split character.
- Given width, how much of a string will fit? Use correct width in R3, large value in R4 and (if needed) a split character in the buffer at R5.
- Address of a mouse click? Use correct width in R3, large value in R4 and (if needed) a split character in the buffer at R5.
- Graphical position of a caret? Set R7 to offset into string, use huge values for R3 and R4, and no split.

The flags in R2 determine how this call is used. Registers R0, R5, R6 and R7 are only used if bits 5, 6, 7 and 8 are set in flags in R2.

Unlike Font_Paint, this call only allows millipoints for all co-ordinates.

[RISC OS Open source page](#)

Font_SetColourTable

SWI &400A2 · FontManager

This call is for internal use only, and must not be called.

Before calling

--

On return

--

Notes

ColourTrans_SetFontColours or Font_Paint (with control sequence 19) can be used to set colours used by the Font Manager.

[RISC OS Open source page](#)

Font_SetFont

SWI &4008A · FontManager

This call sets the current font.

Before calling

R0 New font handle

On return

--

Notes

The font handle passed in R0 will be used in subsequent Font_Paint calls.

The font handle can be passed directly in R0 when using Font_Paint, and this should be used in preference to using this call.

[RISC OS Open source page](#)

Font_SetFontColours

SWI &40092 · FontManager

This call sets the font colours, and, optionally, the font handle.

Before calling

R0 Font handle (0 for current)

R1 Background colour

R2 Foreground colour

R3 Foreground colour offset

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

The foreground colour offsets passed in R3 are the colours used when anti-aliasing is used. The colours used will be from <foreground> to <foreground> + <offset>.

The offset can be from -14 to 14. If the offset is 0, then no anti-aliasing is used.

[RISC OS Open source page](#)

Font_SetFontMax

SWI &4009B · FontManager

This call sets the current FontMax and FontMax1 to FontMax5 values.

Before calling

R0 New FontMax in bytes
R1 New FontMax1 in points
R2 New FontMax2 in points
R3 New FontMax3 in points
R4 New FontMax4 in points
R5 New FontMax5 in points
R6 0 (reserved)
R7 0 (reserved)

On return

--

Notes

Setting these values does not change the configured values. Calling this SWI forces the Font Manager to search through the cache, checking to see if anything would have been cached differently based on these new values. If yes, then the Font Manager discards the relevant data, and will reload using the new settings when next required.

The values in R1-R5 should be in pixels * 72 * 16 (RISC OS 2 only).

[RISC OS Open source page](#)

Font_SetPalette

SWI &40093 · FontManager

This call sets the anti-alias palette.

Before calling

R1 Background logical colour
R2 Foreground logical colour
R3 Foreground colour offset
R4 Background physical colour
R5 Foreground physical colour
R6 &65757254 ('TRUE', to use 24-bit colours in R4 and R5)

On return

--

Notes

This call should be not used by any application that runs within the windowing environment (desktop). Instead, the Wimp_SetFontColours or ColourTrans_SetFontColours should be used instead.

If R6 = 'TRUE' (&65757254) then R4 and R5 have their top nibbles copied to their bottom nibbles, so &12345678 becomes &11335500.

[RISC OS Open source page](#)

Font_SetScaleFactor

SWI &40090 · FontManager

This call sets the values used when converting between OS co-ordinates and millipoints.

Before calling

R1 X scale factor

R2 Y scale factor

On return

R1 Preserved

R2 Preserved

Notes

Applications that run within the windowing environment (the desktop) should not use this call. This is because other applications may rely on current settings. If the call is used, then the current values should be restored afterwards.

The default value is 400 millipoints per OS unit.

To read values, Font_ReadScaleFactor should be used.

[RISC OS Open source page](#)

Font_SetThresholds

SWI &40095 · FontManager

The purpose of this call is to set the thresholds values used by the FontManager when painting.

Before calling

R1 Pointer containing threshold values? to set

On return

R1 Preserved

Notes

This is unlikely to be needed because the default should work well.

[RISC OS Open source page](#)

Font_StringBBox

SWI &40097 · FontManager

This call returns the smallest possible box that covers the whole string.

Before calling

R1 Pointer to string

On return

R1 Minimum x (inclusive)

R2 Minimum y (inclusive)

R3 Maximum x (exclusive)

R4 Maximum y (exclusive)

Notes

This call is deprecated. `Font_ScanString` should be used instead.

The box is returned in millipoints. This call cannot, due to rounding and anti-aliasing, be used to find size on screen.

[RISC OS Open source page](#)

Font_StringWidth

SWI &40085 · FontManager

The purpose of this call is to calculate how wide a string would in the current font under the given limits.

Before calling

R1 Pointer to string

R2 Max width in millipoints

R3 Max height in millipoints

R4 Split character, or -1 for none

R5 Max string length

On return

R1 Pointer to where scan ended

R2 Width of scanned string

R3 Height of scanned string

R4 Number of split characters found

R5 Length of scanned string

Notes

This call is deprecated. `Font_ScanString` should be used instead.

The call can be used to calculate any of the following:

- Finding the caret position in a string if the co-ordinates are known
- Finding the caret co-ordinates if you know the position

- Finding the length of a string when using justification
- Calculating where to split lines when formatting (set R4=32)
- Calculating the required data for justification

[RISC OS Open source page](#)

Font_SwitchOutputToBuffer

SWI &4009E · FontManager

This call switches the output from the Font Manager to a buffer and then saves it in the DrawFile format.

Before calling

R0 Flags if R1 > 0, else 0

R1 Pointer to word aligned buffer or,

8 to count buffer space needed

0 to switch to normal output

-1 to read state

On return

R0 Previous Flags

R1 Previous buffer pointer + size needed by Draw file structure

Notes

R1 on exit, points at the 0 terminator (it and the new size remaining are not counted in the data generated).

Each letter is converted to a filled path object, as are rubout boxes and the underlining.

Before calling the buffer must contain the following:

Offset / Contents

+0 – Contents: 0

+4 – Contents: Size of remaining buffer (bytes)

[RISC OS Open source page](#)

Font_UnCacheFile

SWI &4009A · FontManager

The purpose of this call is to delete a font cache, or re-cache it.

Before calling

R1 Pointer to full filename (as used by Filer)

R2 Recache flag (0 to remove cache, 1 to load new data)

On return

--

Notes

This is used when the font file is altered.

This SWI must be called twice. The first time to remove the old current font cache, and the second time to load the new data.

[RISC OS Open source page](#)

FPEmulator_Abort

SWI &40487 · FPEmulator

The purpose of this call is to handle a FPE Abort.

Before calling

R0 Context pointer (-2 for current context)

R1 R12 value associated with context (only relevant if R2 indicates a PC value inside the floating point undefined instruction handlers)

R2 PC value associated with the context (e.g. the PC value the IRQ handler would normally return to)

On return

R0 Pointer to frame associated with original non-recursive entry to the floating point system if R2 indicated a PC value inside the floating point undefined instruction handlers, otherwise zero

Notes

It is important in some cases that it should be possible to abort some partially complete floating point processing. Some examples:

- If an invalid address is given to an emulated LDF, STF, LFM or SFM instruction, it will result in an address exception or data abort occurring on a load or store instruction within the emulator. Had the address merely been that of a swapped-out piece of store, a virtual memory system could simply swap it in, take other standard remedial action and return to the offending instruction. However, since the address is actually invalid, this is not possible: typically, what is then wanted is that processing of the original LDF, STF, LFM or SFM should be abandoned and that some sort of error handler should be invoked, with the register values, etc., appropriate to the LDF, STF, LFM or SFM being made available to the error handler - not whatever values the emulator had in the registers at the time of the internal load or store instruction!
- If some sort of asynchronous signal is raised for a process while it is swapped out in the middle of some floating point processing, it may be important that you should be able to abandon the partially complete floating point processing.
- If FPEmulator_LoadContext is used to modify the values in a floating point context, any floating point processing associated with the old values in that context must be aborted. Failure to do this can result in all sorts of confusion within the floating point system - e.g. exceptions occurring on internal operations which simply shouldn't be able to occur and which therefore haven't been catered for.
- Some types of floating point exception handlers (e.g. Unix signals) don't have an IEEE-type trap handler interface, but instead want to look at the register values, etc., at the time the floating point exception was detected. Again, these shouldn't be the values that the floating point system happens to have in the registers when the trap handler code is called.

When floating point processing is to be abandoned, three important things may be required:

- Any pending exception must be cleared. A pending exception is one that has been detected by the hardware, but has not yet caused the undefined instruction vector to be entered. Note that if you're doing something like FPEmulator_LoadContext, this should be done even if there is no obvious floating point processing going on at the time. Also note that since there may be a pending exception on an internal calculation within the floating point system, it should also be done when there is some floating point processing going on at the time.

Finally, note that when a context is deactivated, any pending exception for it is stored away without causing the undefined instruction vector to be entered at the time: the actual processing will occur later. So it is possible for inactive contexts to contain pending exceptions, and not just for the currently active context.

- Any floating point processing that is currently going on should be abandoned. This means (a) that a suitable new value for the PC is required; (b) that any values put on the stack by the floating point system should be discarded; © that the values in the floating point context concerned should be made consistent with there being no current floating point processing.
- The register values at the time that the exception occurred must be made available.

This call cancels any pending exception and aborts any incomplete floating point processing associated with a given floating point context. It returns the frame pointer associated with the original non-recursive entry to the floating point undefined instruction handler, and restores any floating point registers that have been used for internal calculations. It does not restore contents of this original frame, of other mode registers or of floating point registers that have been updated as a result of the partial processing of the original non-recursive floating point instruction.

If register values are required, the stack frame has the following format:

[R0, -8]: (on RISC OS 4 or later) Value of SPSR_undef immediately after entry to the undefined instruction handler - i.e. value of CPSR before entry to the undefined instruction handler; [R0, -4]: (on RISC OS 4 or later) Value of CPSR immediately after entry to the undefined instruction handler; [R0, 0]: Value of user R0 immediately after entry to the undefined instruction handler; [R0, 4]: Value of user R1 immediately after entry to the undefined instruction handler; [R0, 8]: Value of user R2 immediately after entry to the undefined instruction handler; [R0, 12]: Value of user R3 immediately after entry to the undefined instruction handler; : : : : [R0, 52]: Value of user R13 immediately before entry to the undefined instruction handler; [R0, 56]: Value of user R14 immediately before entry to the undefined instruction handler; [R0, 60]: The return link from this undefined instruction call, including the PSR bits for RISC OS 3; [R0, 64]: This is the location pointed to by the stack pointer (R13_svc in RISC OS 3, R13_undef in RISC OS 4 or later) before entry to the undefined instruction handler.

Contexts are only available from FPEmulator 4.12 and later.

[RISC OS Open source page](#)

FPEmulator_ActivateContext

SWI &40481 · FPEmulator

The purpose of this call is to install a suitable undefined instruction handler, depending on the state of the specified context, what undefined instruction handlers are available and the hardware present.

Before calling

R0 New context pointer (or 0)

On return

--

Notes

This call also actually enables the floating point hardware if it is present and wanted; in this case, it also loads the context into the hardware.

The argument may also be zero, to indicate that the floating point system should enter the completely inactive state described under FPEmulator_ChangeContext.

You should normally use FPEmulator_ChangeContext in preference to this call.

See the notes under FPEmulator_DeactivateContext relating to restrictions on this pair of SWIs.

Contexts are only available from FPEmulator 4.12 and later.

[RISC OS Open source page](#)

FPEmulator_ChangeContext

SWI &40483 · FPEmulator

The purpose of this call is the standard way of changing context.

Before calling

R0 New context pointer (or 0)

On return

R0 Old context pointer

Notes

This call is the standard way of changing context. A floating point context pointer is passed to it as an argument. It switches over to using this as the current context and returns a pointer to the previous floating point context. It is an atomic equivalent to FPEmulator_DeactivateContext followed by FPEmulator_ActivateContext.

Either pointer may be zero: this indicates that the floating point system was/should be in a completely inactive state, with no current floating point context, its undefined instruction handlers not installed, etc.

Contexts are only available from FPEmulator 4.12 and later.

[RISC OS Open source page](#)

FPEmulator_ContextLength

SWI &40484 · FPEmulator

The purpose of this call is to return the size of a floating point context.

Before calling

--

On return

R0 Size of a floating point context

Notes

It is constant for any given implementation of the floating point system.

To create your own context, allocate a block of this size in the RMA or some other unpagged area of memory, then call FPEmulator_InitContext.

Contexts are only available from FPEmulator 4.12 and later.

[RISC OS Open source page](#)

FPEmulator_DeactivateContext

SWI &40481 · FPEmulator

The purpose of this call is to deactivate the current floating point context, ensuring that everything about the context is held in memory.

Before calling

--

On return

R0 Current context pointer

Notes

You should normally use FPEmulator_ChangeContext in preference to this call.

It returns the address of the current floating point context, or zero if the floating point system is in the completely disabled state described under FPEmulator_ChangeContext.

Note that these calls should be made in pairs, with an FPEmulator_ActivateContext call following each FPEmulator_DeactivateContext call. Also, you should not allow any floating point instructions to be executed between an FPEmulator_DeactivateContext call and the following FPEmulator_ActivateContext call, since executing a floating point instruction will re-activate the context and mean that the memory context may no longer be up-to-date.

Exception: it is legitimate to leave out the call to FPEmulator_DeactivateContext in the above sequence, but only if the old context is never going to be used again: it is likely to be in an inconsistent state if the FPEmulator_DeactivateContext call is omitted.

Contexts are only available from FPEmulator 4.12 and later.

[RISC OS Open source page](#)

FPEmulator_ExceptionDump

SWI &40486 · FPEmulator

The purpose of this call is to return a pointer to the register exception dump.

Before calling

--

On return

R0 Pointer to saved FP register dump

Notes

When a floating point exception trap is generated, the floating point system stores the current FP registers in an internal exception dump using FPEmulator_SaveContext. This call returns a pointer to that register dump. The Debugger module provides a ShowFPRregs command to view the dump contents.

On a floating point trap, the integer registers are also stored in the current exception register dump (OS_ChangeEnvironment 13) so they can be viewed using ShowRegs. Note that floating point registers are not stored on other exceptions, such as data aborts.

Contexts are only available from FPEmulator 4.12 and later.

[RISC OS Open source page](#)

FPEmulator_InitContext

SWI &40485 · FPEmulator

The purpose of this call is to initialise or re-initialise a floating point context.

Before calling

R0 Pointer to context

On return

--

Notes

It puts the context into a standard starting state, in which:

All floating point registers contain zeros.

The FPSR has the OFL, DVZ and IVO exception trap enable bits set. All other trap enable, system control and exception flag bits are clear.

This call may also initialise other less obvious context variables. To initialise a floating point context with other registers and/or FPSR values, use FPEmulator_InitContext and then switch to the context and load the desired values with LDF and/or WFS instructions.

Note that this call should not be made on a context that is currently active. If it is necessary to re-initialise the currently active context, make it no longer active (by using FPEmulator_ActivateContext or FPEmulator_ChangeContext), then re-initialise it, then reactivate it.

Contexts are only available from FPEmulator 4.12 and later.

[RISC OS Open source page](#)

FPEmulator_LoadContext

SWI &40488 · FPEmulator

The purpose of this call is to load a floating point context with a specified set of register and FPSR values, destroying any pending or incompletely processed exception in the process.

Before calling

R0 Pointer to context (-2 for current context)

R1 Pointer to 25-word area as described under FPEmulator_SaveContext

On return

--

Notes

This call should almost always be preceded by one to FPEmulator_Abort you can create chaos in the floating point system by calling FPEmulator_LoadContext while there is a pending or incompletely processed exception.

Contexts are only available from FPEmulator 4.12 and later.

This call did not work correctly until version 4.17 of the FPEmulator module.

[RISC OS Open source page](#)

FPEmulator_SaveContext

SWI &40489 · FPEmulator

The purpose of this call is to save the register and FPSR values held in a floating point context to a specified 25-word area of memory.

Before calling

R0 Pointer to context (-2 for current context)

R1 Pointer to 25-word area as described under FPEmulator_SaveContext

R2 PC value associated with the context

On return

R0 non-zero if there is a pending or incompletely processed exception associated with the context, zero if there isn't

Output Format

0 – Value: FPSR value

4 – Value: F0 value, as an extended precision number

16 – Value: F1 value, as an extended precision number

28 – Value: F2 value, as an extended precision number

40 – Value: F3 value, as an extended precision number

52 – Value: F4 value, as an extended precision number

64 – Value: F5 value, as an extended precision number

76 – Value: F6 value, as an extended precision number

88 – Value: F7 value, as an extended precision number

Notes

Note that it does this without processing any pending exceptions associated with the context, or completing the processing of any exception which is currently being processed. The routine returns a hint about whether there is a pending or incompletely processed exception associated with the context: if there is, the values returned may be rather odd and should not be relied upon as an accurate reflection of reality. For instance, if the FPA has a pending overflow exception, the destination register of the instruction concerned will often read as containing a very small number!

This call might be used by a debugger to view the current FP register contents. The PC value passed in would then be the address at which execution was interrupted – it is used to detect whether the floating point system was interrupted in the middle of processing an exception.

Contexts are only available from FPEmulator 4.12 and later.

This call did not work correctly until version 4.17 of the FPEmulator module.

[RISC OS Open source page](#)

FPEmulator_Version

SWI &40480 · FPEmulator

The purpose of this call is to return the version of the floating point emulator currently in use.

Before calling

--

On return

R0 Version of FPEmulator * 100

Notes

The PRMs incorrectly claim this to be in BCD.

[RISC OS Open source page](#)

Free_DeRegister

SWI &444C1 · Filer_Action

This SWI is used to remove the Desktop Free display that was previously installed with Free_Register.

Before calling

R0 Filing System Number (as passed to Free_Register call)

R1 Address of routine (as passed to Free_Register)

R2 R12 value (as passed to Free_Register)

On return

R0 Preserved

Notes

None.

[RISC OS Open source page](#)

Free_Register

SWI &444C0 · Filer_Action

This SWI is used to provide the Desktop Free display.

Before calling

R0 Filing System Number

R1 Address of routine to call to get free space information

R2 R12 on entry to the above routine

On return

All Registers preserved

Notes

Remove these handlers with Free_DeRegister.

[RISC OS Open source page](#)

Freeway_Enumerate

SWI &47A83 · Freeway

The purpose of this call is to allow a controlling application to enumerate currently held authenticated or unauthenticated objects of a given type, obtaining their names, location IP addresses, and descriptors if present.

Before calling

R0 Enumerate flags

R1 Type number

R2 Length of buffer for object name, or 0

R3 Pointer to buffer for object name, or 0

R4 Length of buffer for object descriptor

R5 Pointer to object descriptor, or 0 for none

R6 0 or undefined

R7 Enumeration reference (or 0 to start)

R8 Object authentication value if flags bit 0 is set

On return

R2 Length of held object name, including 0 terminator

R4 Length of held object descriptor

R6 IP address of computer which holds the object

R7 Next enumeration reference, or -1 to indicate end reached

- All other registers preserved

Enumerate Flags

Bit / Value / Meaning

0 – Value: 0 · Meaning: Enumerate unauthenticated objects only

1 · Enumerate authenticated objects only

1 - 31 – Value: 0 · Meaning: Reserved (must be zero)

Notes

If an authentication value is provided then only those objects whose actual authentication value matches the supplied value are enumerated. If no authentication value is supplied then only unauthenticated objects are enumerated. The length of the held object name or descriptor may be read without filling in buffers by setting R3=0 or R5=0 respectively, on entry.

However in this case there is no guarantee that the object attributes will not have changed, or that the object will still exist, if the SWI is called again some time later with the same enumeration reference.

If R7 is returned -1 then there were no further known objects of that type - the object name buffer will not have been filled in, and R6 is undefined.

An error is returned if the type number is unknown, or if a supplied name or descriptor buffer is too short. In the latter cases the actual name and descriptor lengths are returned in R2 and R4.

[RISC OS Open source page](#)

Freeway_Read

SWI &47A82 · Freeway

The purpose of this call is to read information about the attributes of an object.

Before calling

R0 Read flags

R1 Type number

R2 Pointer to object name, 0 terminated

R3 Length of buffer for object descriptor

R4 Pointer to object descriptor, or 0 for none

R5 Object authentication value if present, otherwise undefined

On return

R3 Length of held object descriptor

R5 IP address of computer which holds the object

Read Flags

Bit / Value / Meaning

0 – Value: 1 · Meaning: Authentication value provided

3 – 31 – Value: 0 · Meaning: Reserved (must be zero)

Notes

The type number and object name must be provided. The SWI returns the IP address of the holder, and optionally the descriptor. The length of a held object descriptor may be read by setting R4 to 0 on entry. However in this case there is no guarantee that the object attributes will not have changed, or that the object will still exist, if the SWI is called again some time later.

An error is returned if the type number is unknown, or the object name is unknown, or if a supplied authentication value does not match the object's own authentication value, or if a supplied object descriptor buffer is too short; in this case the actual length is returned in R3.

[RISC OS Open source page](#)

Freeway_Register

SWI &47A80 · Freeway

The purpose of this call is to register or deregister interest in objects of a given Freeway type.

Before calling

R0 Register flags

R1 Type number

R2 Object authentication value if flags bit 1 is set

R2 Pointer to type name if flags bit 2 is set

On return

--

Register Flags

Bit / Value / Meaning

0 – Value: 0 · Meaning: Register interest

1 · Deregister interest

1 – Value: 0 · Meaning: Interested in unauthenticated objects

1 · Interested in authenticated objects

2 – Value: 0 · Meaning: R3 not valid

1 · R3 is pointer to type name (fewer than 16 characters)

3 – 31 – Value: 0 · Meaning: Reserved (must be zero)

Notes

This SWI enables a controlling application to register interest in authenticated or unauthenticated objects of a particular Freeway type with the local Freeway software, and also to give a 'name' to that type (this name is of no significance to the software, it's just for the convenience of the human user).

Freeway will hold information arriving from the network about a remote object only if one or more registrations of interest have been made locally in that object's type. If the object is authenticated then at least one of those registrations must have included an authentication value which matches the object's own. An error is returned if insufficient free memory exists.

[RISC OS Open source page](#)

Freeway_Serial

SWI &47A85 · Freeway

This SWI call is for internal use only.

Before calling

On return

Notes

Do not use this SWI.

[RISC OS Open source page](#)

Freeway_Status

SWI &47A84 · Freeway

-

Before calling

On return

Notes

-
[RISC OS Open source page](#)

Freeway_Write

SWI &47A81 · Freeway

The purpose of this call is to add or remove a locally held object of a given type.

Before calling

R0 Write flags

R1 Type number

R2 Pointer to object name, 0 terminated

R3 Length of object descriptor

R4 Pointer to object descriptor, or 0 for none or when flags bit 0 is set

R5 Object authentication value if flags bit 1 is set, otherwise undefined

On return

- -

Write Flags

Bit / Meaning when set

0 – Meaning when set: Remove object (otherwise add)

1 – Meaning when set: Object is authenticated

2 - 31 – Meaning when set: Reserved (must be zero)

Notes

If the object is unauthenticated then other computers are notified immediately, otherwise notification is withheld until a remote computer requests it. An error is returned if the type number is not known (i.e. if `Freeway_Register` has not been called with this type number), or if an object is added and a remote object of the given name and type already exists, or if an object is removed and no local object of the given name and type is currently held, or if no memory exists to store information about the object.

If flags bit 0 is set and the object named is already held as a local object, the object's descriptor and authentication value are updated if they differ from those passed to the SWI.

Controlling applications should be prepared to receive `UpCall_Freeway`, reason code object deleted, referring to any object previously added successfully. This is to cover the possibility of Freeway deciding that the information held about an object is unreliable and so deciding to remove it unilaterally, e.g. if a remote object is created with the same name as a local object.

[RISC OS Open source page](#)

FSLock_ChangeStatus

SWI &44782 · FSLock

The purpose of this call is to change one or more of the lock status, the password and the locked filing system.

Before calling

R0 New Lock status

R1 Pointer to current file locking password

R2 Pointer to new file locking password

R3 New locked filing system number

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

This SWI can only be called by number; not by name.

SWI is not re-entrant.

[RISC OS Open source page](#)

FSLock_Status

SWI &44781 · FSLock

The purpose of this call is to return the current lock status, and the locked filing system's number.

Before calling

--

On return

R0 Lock status

R1 Locked filing system number (undefined if lock status = 0)

Notes

This SWI can only be called by number; not by name.

SWI is not re-entrant.

[RISC OS Open source page](#)

FSLock_Version

SWI &44780 · FSLock

The purpose of this call is to return information describing the FSLock module.

Before calling

--

On return

R0 FSLock version × 100

R1 Pointer to FSLock's module workspace

Notes

SWI is re-entrant.

[RISC OS Open source page](#)

GPIO_Features

SWI &58FAF · GPIO

This call returns additional features supported by the GPIO module.

On return

R0 = Bitmask of supported features

bits 0-31: reserved, equal to zero

[RISC OS Open source page](#)

GPIO_ReadData

SWI &58F80 · GPIO

Reads the state of the given GPIO pin, or returns an error if the pin number is not valid.

Before calling

R0 Logical GPIO pin number

On return

R0 = 0 if pin low, 1 if pin high

[RISC OS Open source page](#)

GPIO_ReadOE

SWI &58F82 · GPIO

Reads the “Output Enable” state of the given GPIO pin, or returns an error if the pin number is not valid.

Before calling

R0 Logical GPIO pin number

On return

R0 = 0 if pin set as Output, 1 if pin set as Input

[RISC OS Open source page](#)

GPIO_WriteData

SWI &58F81 · GPIO

Writes the state of the given GPIO pin, or returns an error if the pin number is not valid.

Before calling

R0 Logical GPIO pin number

R1 = 0 (set low), 1 (set high)

[RISC OS Open source page](#)

GPIO_WriteEvent

SWI &58FAC · GPIO

Writes the event state of the given GPIO pin, or returns an error if the pin number is not valid.

Before calling

R0 Logical GPIO pin number

R1 0 (clear event), 1 (set event)

[RISC OS Open source page](#)

GPIO_WriteOE

SWI &58F83 · GPIO

Writes the “Output Enable” state of the given GPIO pin, or returns an error if the pin number is not valid.

Before calling

R0 Logical GPIO pin number

R1 0 (Output) or 1 (Input)

[RISC OS Open source page](#)

Hourglass_Colours

SWI &406C6 · Hourglass

The purpose of this call is to allow the colours used by the hourglass to be set or read.

Before calling

R0 New colour value for pointer colour 1, or -1 for no change

R1 New colour value for pointer colour 3, or -1 for no change

On return

R0 Old colour 1

R1 Old colour 3

- All other registers preserved

Notes

Colours are specified as &xxBBGRR.

With the default pointer shape, colour 1 is cyan (the hourglass frame and 'sand') and colour 3 is blue (hourglass body, percentage number, and LEDs)

The colours are reset each time the hourglass appears.

[RISC OS Open source page](#)

Hourglass_LEDs

SWI &406C5 · Hourglass

The purpose of this call is to control the display of the hourglass 'LEDs'. These LEDs can be used to convey additional state to the user, perhaps as a debugging aid.

Before calling

R0 EOR mask

R1 AND mask

On return

R0 Old LED state

- All other registers preserved

Notes

The current version of the hourglass supports two LEDs (controlled by bits 0 and 1 of the LED mask), which both start in the off state when the hourglass is enabled.

The LED state is updated using the standard pattern:

$\text{new_value} = (\text{old_value} \text{ AND } \text{and_mask}) \text{ EOR } \text{eor_mask}$

[RISC OS Open source page](#)

Hourglass_Off

SWI &406C1 · Hourglass

The purpose of this call is to disable the display of the hourglass mouse pointer.

Before calling

--

On return

- All registers preserved

Notes

The module retains a count of how many times Hourglass_On has been called; Hourglass_Off must be called an equal number of times in order for the hourglass to actually vanish. This allows for code which uses the hourglass to safely call other code which uses the hourglass. Hourglass_Smash can also be used to force the hourglass to vanish; this is most useful when dealing with errors, to ensure the mouse pointer returns to a user-friendly state.

[RISC OS Open source page](#)

Hourglass_On

SWI &406C0 · Hourglass

The purpose of this call is to enable the display of the hourglass mouse pointer.

Before calling

--

On return

- All registers preserved

Notes

Internally the hourglass module delays the activation of the hourglass by about a third of a second. This delay is intended to prevent situations where the hourglass briefly flashes on the screen before vanishing again, due to an operation completing quicker than the calling code expects (e.g. due to running on a faster computer, or due to having less data to process). This means that, in general, code which wants to use the hourglass should avoid making any time estimations itself - if there's a task which operates on an arbitrary amount of user-supplied data, or one which is known to be computationally expensive on slow computers, then the code should simply activate the hourglass at the start of the task and let the module decide whether the processing has taken long enough that the hourglass is worth displaying to the user. However, if necessary, Hourglass_Start can be used to enable the hourglass with a custom delay.

At the end of your processing you must take care to call Hourglass_Off to disable the hourglass again.

The module retains a count of how many times Hourglass_On has been called; Hourglass_Off must be called an equal number of times in order for the hourglass to vanish. This allows for code which uses the hourglass to safely call other code which uses the hourglass. Hourglass_Smash can also be used to force the hourglass to vanish; this is most useful when dealing with errors, to ensure the mouse pointer returns to a user-friendly state.

[RISC OS Open source page](#)

Hourglass_Percentage

SWI &406C4 · Hourglass

The purpose of this call is to control the display of a percentage progress figure on the hourglass.

Before calling

R0 Percentage value (0-99), or any other value to disable percentage display

On return

- All registers preserved

Notes

If the hourglass is currently on, this call can be used to enable/update the percentage figure which is shown beneath the hourglass body.

If the hourglass is used in a nested manner (e.g. Hourglass_On is called while the hourglass is visible) then only the nesting level which last successfully enabled the percentage display will be in control of it. The percentage will remain visible for the duration of the child, but it will have no control over it.

The percentage display will automatically disable when Hourglass_Off is used to leave the nesting level at which it was enabled.

[RISC OS Open source page](#)

Hourglass_Smash

SWI &406C2 · Hourglass

The purpose of this call is to force the hourglass to vanish.

Before calling

--

On return

- All registers preserved

Notes

Normally hourglass enable/disable is controlled via the Hourglass_On and Hourglass_Off SWIs. The module retains a count of how many times Hourglass_On has been called, and Hourglass_Off must be called an equal number of times in order for the hourglass to vanish. This allows for code which uses the hourglass to safely call other code which uses the hourglass.

Hourglass_Smash is designed to be used in situations where you don't know if the hourglass is currently on, or you don't know how many times Hourglass_On has been called - e.g. when dealing with errors, to ensure the mouse pointer returns to a user-friendly state.

[RISC OS Open source page](#)

Hourglass_Start

SWI &406C3 · Hourglass

The purpose of this call is to enable the display of the hourglass mouse pointer, with a specified delay.

Before calling

R0 Start delay, in centiseconds

On return

- All registers preserved

Notes

This SWI is functionally equivalent to Hourglass_On, except that it allows the caller to specify the delay to use before the hourglass should be made visible. See Hourglass_On for more details on controlling the hourglass.

A delay of 0 is (effectively) treated as an infinite startup delay, locking the hourglass off until Hourglass_Off is called the correct number of times to 'disable' it again.

[RISC OS Open source page](#)

HTTP_AddCookie

SWI &83FBD · AcornHTTP

Cookies set through a response received from a web server can be moved from the unread queue to the cookie store using HTTP_ConsumeCookie. This call is intended to allow additional cookies to be set, for example, through the execution of Javascript.

Before calling

R0 Flags ·

Bit 0 => · Set for secure

Bits 1-31 => · Reserved. Must be zero ·

R1 Pointer to name ·

R2 Pointer to value ·

R3 Pointer to expiry date-time ·

R4 Pointer to path ·

R5 Pointer to domain ·

On return

- No exit defined

Notes

Values R1-R5 should be pointers to NULL-terminated strings. The date-time format is expected to be that used for the Set-Cookie HTTP response header field.

Cookies created using this call are not saved persistently in the cookies file in <Choices\$Write>.WWW.AcornHTTP.Cookies

[RISC OS Open source page](#)

HTTP_ConsumeCookie

SWI &83FBE · AcornHTTP

Unless `Browse$AcceptAllCookies` is set to a value which is not "off", any cookies found in the headers of responses received by AcornHTTP will be added to a temporary queue to be processed.

Before calling

R0 Flags ·

Bit 0 => · Set for accept, unset for reject

Bits 1-31 => · Reserved. Must be zero ·

R1 Session? Unused? ·

R2 The handle of a queued cookie as returned by `HTTP_EnumerateCookies` ·

On return

- No exit defined

In practice

For each cookie in the queue, the client is expected to indicate whether the cookie should be accepted or rejected. If the cookie is accepted it will be moved to the cookie store. Either way, the cookie will be removed from the queue.

Notes

Because the cookie has been removed from the queue, it is not possible to resume enumeration of the queue by passing the same cookie handle in R1 on entry to `HTTP_EnumerateCookies`.

If a client wishes to process the entire queue of cookies, it is therefore simplest to alternate calls to `HTTP_EnumerateCookies` and `HTTP_ConsumeCookie`, passing R1=0 on entry to `HTTP_EnumerateCookies` every time.

The module does not have support for the Public Suffix List

[RISC OS Open source page](#)

HTTP_DeregisterMethod

SWI &83FBC · AcornHTTP

This call deregisters a non-standard HTTP method.

Before calling

R0 Pointer to the name of non-standard HTTP method

R1 Method flags (as were set when registering)

On return

- No exit defined

[RISC OS Open source page](#)

HTTP_EnumerateCookies

SWI &83FBF · AcornHTTP

All cookies received during an HTTP response are added to a queue, and a flag is set in the R0 status value returned by URL_Status and URL_ReadData to indicate to the client that cookies have been found. This call allows a client to enumerate the cookies which have been queued or which are already in the cookie store. Cookies which are in the queue can then either be accepted (moved to the store) or rejected using HTTP_ConsumeCookie.

Before calling

R0 Flags ·

Bit 0 => · Set to enumerate stored cookies; unset to enumerate queued cookies

Bits 1-31 => · Reserved. Must be zero ·

R1 Unique cookie handle. 0 for initial call. ·

On return

R0 Cookie flags ·

Bit 0 => · Set if cookie has the Secure attribute.

Bits 1-31 => · Reserved. Must be zero ·

R1 Unique cookie handle. 0 for no more cookies.

R2 Total number of cookies created.

R3 Number of cookies in the queue.

R4 Pointer to domain name string.

R5 Pointer to name string.

R6 Pointer to value string.

R7 Pointer to path string.

In practice

If queued cookies are left unprocessed, the cookie queue will eventually reach its length limit, whereupon adding further cookies will cause the earlier cookies to be discarded.

Notes

The cookie enumeration will not complete successfully if any cookies are added or removed from the queue or the store after enumeration has started. In particular this means that if a cookie is retrieved by enumerating the queue and is then processed with HTTP_ConsumeCookie, enumeration cannot be resumed. It is therefore simplest, if processing all members of the queue, to reset R1 to 0 before every call to HTTP_EnumerateCookies.

The return values in R4-R7 are all pointers to NULL-terminated strings. If R1 is zero on exit then R4-R7 will all be zero also.

Limitations

The cookie queue and the store are shared across all clients.

The expiry of the cookie does not appear to be available to the client.

It appears that the HttpOnly cookie attribute is unsupported.

Cookies will be moved automatically from the queue to the store if Browse\$AcceptAllCookies is set and is not equal to "off".

[RISC OS Open source page](#)

HTTP_GetData

SWI &83F80 · AcornHTTP

This page currently only documents the flags and settings specific to AcornHTTP in R0 and R2. The other registers are exactly as described in the Protocol_GetData documentation.

Before calling

R0 Flags ·

- Bit 0 set · R6 · alternative User Agent identifier
- Bit 1 set · R5 = length of data pointed to by R4 (if R4 not zero)
- Bits 2-7 · Reserved: must be zero
- Bit 29 · Reserved for internal use (secure connection flag)
- Bit 30 · If set, AcornHTTP will not add cookies to the request
- Bit 31 · Reserved for use by URL module (if set, this is a proxy request)

R2 Bits 0-7 · request method (not a bitfield):

- 1 · GET
- 2 · HEAD
- 3 · Reserved – do not use
- 4 · POST
- 5,6,7 · Reserved – do not use
- 8 · PUT

Bits 8-15 · type of data wanted (if R0 bit 1 set, otherwise in bits 0-31 of R5):

- 0 · Body only
- 1 · Head only
- 2 · Both

Bits 16-31 · reserved and should be zero.

[RISC OS Open source page](#)

HTTP_RegisterMethod

SWI &83FBB · AcornHTTP

AcornHTTP is designed to be very flexible but it is limited to the classical HTTP methods.

Before calling

R0 Pointer to the name of non-standard HTTP method ·

R1 Method flags ·

- Bits 0-1 => · 0= Caller must set up everything
- 1= Behave like PUT/POST
- 2= Behave like GET
- 3= Unused
- Bits 2-31 => · Reserved. Must be zero ·

On return

R2 Opaque method number to pass in R2 in calls to URL_GetURL

In practice

Since its original design a couple of other HTTP methods have been defined for various purposes e.g. for WebDAV, CalDAV etc. which are more or less popular. Many methods are in fact similar to the standard set (GET/PUT etc) and so can be handled by changing AcornHTTP quite simply.

Every client which wants to use AcornHTTP with a non-standard HTTP method registers the method details via this SWI.

When a client no longer makes usage of the registered HTTP non-standard method especially when quitting it has to deregister its method(s). This allows other applications to reuse the method number for their own purposes.

Notes

Any number in method range 1-127 may be returned as a valid result except the method numbers which are assigned to HTTP standard methods (GET, HEAD, POST, PUT, OPTIONS, TRACE, DELETE).

[RISC OS Open source page](#)

IIC_Control

SWI &240 · IIC

The purpose of this call is to allow software to perform transfers on the primary IIC bus that are present in the system.

Before calling

R0 Device address (bits 1-7), R/W flag (bit 0 set to read, clear to write)

R1 Pointer to block

R2 Size of block (number of bytes to transfer)

On return

R0-R2 Corrupted

In practice

There is no support for repeated start conditions, so for most simple transfers this SWI would be called twice – the first time to write a value (such as a register offset) and then to read one or more bytes of data from the device.

Errors

Possible errors are:

&20300 / No acknowledge from IIC device

&20301 – No acknowledge from IIC device: IIC system busy / IIC operation in progress

Notes

The benefit of IIC_Control is that it is simple to use. However, for more flexibility including support for multiple IIC buses and the ability to perform a complete interaction as an atomic transfer (via repeated start), please refer to OS_IICOp.

On older systems, IIC_Control disables interrupts during the actual transfer. On newer systems (RISC OS 5), IIC_Control internally calls OS_IICOp.

[RISC OS Open source page](#)

InverseTable_Calculate

SWI &4BF40 · InverseTable

This SWI returns pointers to two lookup tables for converting palette indices to RGB values (via R0) and back again (via R1, the 'inverse table')

Before calling

- None

On return

R0 Pointer to palette RGB lookup table

R1 Pointer to RGB palette lookup table

In practice

On exit R0 is a pointer to a 1KB lookup table, with one word per entry, to convert from the current screen palette to a standard 15bpp colour value. I.e. the red component in bits 0-4, green in bits 5-9, and blue in bits 10-14.

R1 will be a pointer to a 32KB lookup table giving the inverse mapping, i.e. converting a 15bpp colour value back to the nearest palette index. In this case each table entry is a byte, giving the resultant palette index.

Notes

This SWI only operates in 256 colour screen modes. Any other colour depth will result in a 'Bad MODE' error.

The tables pointed to by R0 and R1 are held inside InverseTable's workspace and should not be modified by the user. The tables will persist until the next mode change, palette change, or change in screen output redirection.

There is practically zero overhead in calling this SWI in situations in which the tables do not need to be recalculated. Therefore it's recommended that you simply call this SWI to fetch the table pointers as and when needed, rather than attempt to cache the pointers (or the table contents) yourself.

See also InverseTable_SpriteTable, to generate a pair of lookup tables for use with an arbitrary (256 colour) sprite/palette.

[RISC OS Open source page](#)

InverseTable_SpriteTable

SWI &4BF41 · InverseTable

This SWI is an enhanced version of InverseTable_Calculate that allows for the generation of lookup tables for an arbitrary sprite/palette, instead of just the current screen mode.

Before calling

R0 Pointer to empty, word-aligned 1KB buffer to fill with palette RGB table

R1 Pointer to empty 32KB buffer to fill with RGB palette table

R2 Mode / sprite area

R3 Palette pointer / sprite

On return

R0 Pointer to palette RGB lookup table

R1 Pointer to RGB palette lookup table

In practice

On entry R0 and R1 must point to empty buffers suitable for storing the generated tables.

On exit R0 and R1 will either point to the buffers that were supplied (with the buffers filled with the requested data), or alternately they may point to a table that was already cached by InverseTable internally. In this case the same rules as for InverseTable_Calculate apply - the returned pointers are only guaranteed to be valid until the next mode/palette/redirection change. If this is an issue for your application you can simply copy the returned table into the buffers which you passed in on entry.

Notes

This SWI only works with 256 colour modes/sprites. Any other colour depth will result in a 'Bad MODE' error.

R2 and R3 identify a mode + palette or a sprite area + sprite pointer, using the same methods as ColourTrans_ReadPalette. See ColourTrans mode identification for more details. Note that with this SWI it is not possible to specify a sprite using its name.

This call has a higher processing cost than InverseTable_Calculate, as InverseTable does not cache any custom-generated tables internally.

[RISC OS Open source page](#)

Joystick_CalibrateBottomLeft

SWI &43F42 · Joystick

Make this call when the joystick is know to be at the bottom left corner of its possible travel, in order that the joystick driver can calibrate the range.

Before calling

--

On return

--

Notes

Having made a call to this SWI or Joystick_CalibrateTopRight then the other corner must also be calibrated as a pair too. Until that time Joystick_Read and ADVAL will return an error.

[RISC OS Open source page](#)

Joystick_CalibrateTopRight

SWI &43F41 · Joystick

Make this call when the joystick is know to be at the top right corner of its possible travel, in order that the joystick driver can calibrate the range.

Before calling

--

On return

Notes

Having made a call to this SWI or Joystick_CalibrateBottomLeft then the other corner must also be calibrated as a pair too. Until that time Joystick_Read and ADVAL will return an error.

[RISC OS Open source page](#)

Joystick_Read

SWI &43F40 · Joystick

The purpose of this call is to read the most recent buffered value from the joystick hardware.

Before calling

R0 Flags:

Bits 0-7: Stick number

Bits 8-15: Reason code

Other bits reserved, should be zero

On return

Registers dependent on reason code

Reason Codes

/ Hex # / Action

0 – Hex #: &00 · Action: Read 8 bit joystick

1 – Hex #: &01 · Action: Read 16 bit joystick

[RISC OS Open source page](#)

Joystick_Read 0

SWI &43F40 · Joystick

The purpose of this call is to read the latest 8 bit joystick value.

Before calling

R0 0 + stick number in bits 0-7

On return

R0 State:

Bits 0-7: Signed Y value (-127 down, +127 up)

Bits 8-15: Signed X value (-127 left, +127 right)

Bits 16-23: Buttons (such as the fire button)

Bits 24-31: Reserved

Notes

For digital joysticks, fixed values of ± 64 will be returned when the switches are activated. Due calibration drift in analogue circuitry it is recommended to use the band -&20 to +&20 as an "at rest" region.

Prior to a call to this SWI the joystick hardware may be disabled or powered down. As it is the buffered value that is returned, triggering a new conversion to be scheduled, the very first call to this SWI will return X=0 Y=0.

[RISC OS Open source page](#)

Joystick_Read 1

SWI &43F40 · Joystick

The purpose of this call is to read the latest 16 bit joystick value.

Before calling

R0 256 + stick number in bits 0-7

On return

R0 Position:

Bits 0-15: Unsigned Y value (0 down, 65535 up)

Bits 16-31: Unsigned X value (0 left, 65535 right)

R1 Bits 0-31: Buttons (such as the fire button)

Notes

Due calibration drift in analogue circuitry it is recommended to use the band &6000 to &A000 as an “at rest” region.

Prior to a call to this SWI the joystick hardware may be disabled or powered down. As it is the buffered value that is returned, triggering a new conversion to be scheduled, the very first call to this SWI will return X=0 Y=0.

Prior to 23-11-2021, R1 returned only 8 buttons with bits 8-31 being reserved.

[RISC OS Open source page](#)

JPEG_FileInfo

SWI &49981 · SpriteExtend

This call is used to read the attributes of a JPEG image held on a filing system.

Before calling

R0 Flags:

Bit 0: Return dimensions

Bit 1: Return SOF type

Other bits reserved, should be zero

R1 Pointer to control-character terminated filename

On return

R0 Flags:

Bit 0: Image is greyscale

Bit 1: Transformed plots are not supported

Bit 2: Pixel density is only a ratio

Bits 3-6: SOF type (if requested)

R2 Width in pixels

R3 Height in pixels

R4 X pixel density

R5 Y pixel density

R6 Size of extra workspace needed to render

- Other registers preserved

[RISC OS Open source page](#)

JPEG_Info

SWI &49980 · SpriteExtend

This call is used to read the attributes of a JPEG image held in memory. It also does a partial validation of the data so can return an error if it appears invalid.

Before calling

R0 Flags:

Bit 0: Return dimensions

Bit 1: Return SOF type

Other bits reserved, should be zero

R1 Pointer to JPEG image in memory

R2 Length of data (bytes)

On return

R0 Flags:

Bit 0: Image is greyscale

Bit 1: Transformed plots are not supported

Bit 2: If set, pixel density is a ratio, otherwise DPI

Bits 3-6: SOF type (if R0 bit 1 set on entry)

R2 Width in pixels (if R0 bit 0 set on entry)

R3 Height in pixels (if R0 bit 0 set on entry)

R4 X pixel density (see R0 bit 2 on exit)

R5 Y pixel density (see R0 bit 2 on exit)

R6 Size of extra workspace needed to render

- Other registers preserved

[RISC OS Open source page](#)

JPEG_PDriverIntercept

SWI &49986 · SpriteExtend

This call is to allow the printer drivers to request that SpriteExtend passes on any calls to perform JPEG plotting to PDriver_JPEGSWI? instead of handling them itself. The calls passed on are JPEG_PlotScaled, JPEG_PlotTransformed, JPEG_PlotFileScaled, and JPEG_PlotFileTransformed only.

Before calling

R0 Flags:

Bit 0: Pass on plotting calls

Bit 1: Use translation tables

On return**R0** Previous flags value[RISC OS Open source page](#)

JPEG_PlotFileScaled

SWI &49983 · SpriteExtend

This call is used to plot a scaled JPEG image to screen.

Before calling**R0** Pointer to control character terminated filename**R1** X coordinate**R2** Y coordinate**R3** Pointer to scale factors, or 0 for 1:1 scaling**R4** Flags:

Bit 0: Dither output

Bit 1: Use full error diffused dithering

Bit 3: Use colour mapping

Bits 4-11: Translucency (0 = opaque, 255 = 1/256 visibility)

Other bits reserved, should be zero

R5 Pointer to Colour Mapping Descriptor, if relevant**On return**

- All registers preserved

Notes

Coordinates are in OS units, and specify the bottom-left corner of the image.

1:1 scaling maps directly from source pixels to destination pixels; the DPI and eigen values of the source and destination are ignored.

This call is effectively just a convenience wrapper around JPEG_PlotScaled.

[RISC OS Open source page](#)

JPEG_PlotFileTransformed

SWI &49985 · SpriteExtend

This call is used to plot a transformed JPEG image to screen.

Before calling**R0** Pointer to control character terminated file name**R1** Flags:

Bit 0: R2 is pointer to destination coordinates, else pointer to matrix

Bit 1: Dither output

Bit 2: Use full error diffused dithering

Bit 4: Use colour mapping

Bits 5-12: Translucency (0 = opaque, 255 = 1/256 visibility)

Other bits reserved, should be zero

R2 Pointer to coordinate block or matrix, as defined by R0

R3 Pointer to Colour Mapping Descriptor, if relevant

On return

- All registers preserved

Notes

If R1 bit 0 is set, R2 is treated as a pointer to a Draw module Transformation Matrix describing the transform to apply to the image. If bit 0 is clear then R2 is treated as a pointer to a block giving the screen coordinates of a parallelogram to transform the image into:

0 – Value: X0 coordinate, in 1/256th OS units

1 – Value: Y0 coordinate, in 1/256th OS units

2 – Value: X1 coordinate, in 1/256th OS units

3 – Value: Y1 coordinate, in 1/256th OS units

4 – Value: X2 coordinate, in 1/256th OS units

5 – Value: Y2 coordinate, in 1/256th OS units

6 – Value: X3 coordinate, in 1/256th OS units

7 – Value: Y3 coordinate, in 1/256th OS units

Note that the default version of SpriteExtend supplied with RISC OS 5 does not support the plotting of transformed JPEGs. However it will perform a scaled plot if it detects that the transformation can be reduced to a set of scale factors compatible with JPEG_PlotScaled.

This call is effectively a convenience wrapper for JPEG_PlotTransformed.

[RISC OS Open source page](#)

JPEG_PlotScaled

SWI &49982 · SpriteExtend

This call is used to plot a scaled JPEG image to screen.

Before calling

R0 Pointer to JPEG image in memory

R1 X coordinate

R2 Y coordinate

R3 Pointer to scale factors, or 0 for 1:1 scaling

R4 Length of data (bytes)

R5 Flags:

Bit 0: Dither output

Bit 1: Use full error diffused dithering

Bit 3: Use colour mapping

Bits 4-11: Translucency (0 = opaque, 255 = 1/256 visibility)

Other bits reserved, should be zero

R6 Pointer to Colour Mapping Descriptor, if relevant

On return

- All registers preserved

Notes

Coordinates are in OS units, and specify the bottom-left corner of the image.

1:1 scaling maps directly from source pixels to destination pixels; the DPI and eigen values of the source and destination are ignored.

[RISC OS Open source page](#)

JPEG_PlotTransformed

SWI &49984 · SpriteExtend

This call is used to plot a transformed JPEG image to screen.

Before calling

R0 Pointer to JPEG image in memory

R1 Flags:

Bit 0: R2 is pointer to destination coordinates, else pointer to matrix

Bit 1: Dither output

Bit 2: Use full error diffused dithering

Bit 4: Use colour mapping

Bits 5-12: Translucency (0 = opaque, 255 = 1/256 visibility)

Other bits reserved, should be zero

R2 Pointer to coordinate block or matrix, as defined by R0

R3 Length of data (bytes)

R4 Pointer to Colour Mapping Descriptor, if relevant

On return

- All registers preserved

Notes

If R1 bit 0 is set, R2 is treated as a pointer to a Draw module Transformation Matrix describing the transform to apply to the image. If bit 0 is clear then R2 is treated as a pointer to a block giving the screen coordinates of a parallelogram to transform the image into:

0 – Value: X0 coordinate, in 1/256th OS units

1 – Value: Y0 coordinate, in 1/256th OS units

2 – Value: X1 coordinate, in 1/256th OS units

3 – Value: Y1 coordinate, in 1/256th OS units

4 – Value: X2 coordinate, in 1/256th OS units

5 – Value: Y2 coordinate, in 1/256th OS units

6 – Value: X3 coordinate, in 1/256th OS units

7 – Value: Y3 coordinate, in 1/256th OS units

Note that the default version of SpriteExtend supplied with RISC OS 5 does not support the plotting of transformed JPEGs. However it will perform a scaled plot if it detects that the transformation can be reduced to a set of scale factors compatible with JPEG_PlotScaled.

[RISC OS Open source page](#)

Message_ColourPickerResetColourRequest

SWI &47704 · WindowManager

This message is sent by ColourPicker whenever the user clicks Adjust on the Cancel button of the dialogue.

Message Block

+20 · ColourPicker dialogue handle

Notes

The client should respond by calling ColourPicker_UpdateDialogue, to reset the dialogue so that it displays the colour currently in effect. This is the last colour selected by clicking Adjust on the dialogue's OK icon, or failing that, the colour the dialogue showed when first opened

[RISC OS Open source page](#)

MessageTrans_CloseFile

SWI &41504 · MessageTrans

This SWI is used to close a file that was previously opened with MessageTrans_OpenFile.

Before calling

R0 Pointer to 4-word data structure

On return

--

Notes

None.

[RISC OS Open source page](#)

MessageTrans_CopyError

SWI &41508 · MessageTrans

This SWI is used to copy an error message to one of the internal buffers used by MessageTrans_ErrorLookup.

Before calling

R0 Pointer to SWI Error Block

On return

R0 Pointer to error buffer used

Notes

None.

[RISC OS Open source page](#)

MessageTrans_Dictionary

SWI &41509 · MessageTrans

This SWI is used to return the address of first byte of the kernel dictionary.

Before calling

--

On return

R0 Pointer to the first byte of the kernel's dictionary

Notes

None.

[RISC OS Open source page](#)

MessageTrans_EnumerateTokens

SWI &41505 · MessageTrans

This SWI is used to look up tokens that match the given wildcarded token specification.

Before calling

R0 Pointer to 4-word data structure (cannot be 0)

R1 Pointer to token (possibly wildcarded)

R2 Pointer to buffer for result

R3 Size of buffer

R4 Index (0 for first time)

On return

R0 Preserved

R1 Preserved

R2 0 if no more matching tokens, otherwise preserved

R3 Length of result (excluding terminator)

R4 Next index

Wildcard token /

? – : 1 character wildcard

* – : 0 or more characters

Notes

The token returned in the buffer has the same terminator as the wildcard in R1.

[RISC OS Open source page](#)

MessageTrans_ErrorLookup

SWI &41506 · MessageTrans

This SWI is used to translate a message token within an error block, with optional parameter substitution.

Before calling

R0 Pointer to error block

R1 Pointer to 4-word data structure, or 0 for global messages

R2 Pointer to buffer for result, or 0 to use internal buffer

R3 Size of buffer

R4 Pointer to parameter 0, or 0 if none

R5 Pointer to parameter 1, or 0 if none

R6 Pointer to parameter 2, or 0 if none

R7 Pointer to parameter 3, or 0 if none

On return

R0 Pointer to error buffer used

V flag set

Notes

This SWI is like MessageTrans_Lookup except that it is passed a token in an error block in R0. The data structure is in R1, not R0. If the token is not found in the message file specified, it will be searched for in the global message file.

If R2 = 0 on entry MessageTrans will use an internal buffer. There are 2 for calls from IRQ routines, and 10 for foreground processes, which it cycles through.

To check that your error message was looked up (rather than MessageTrans itself reporting an error) verify that the error number in the returned block is the same as that passed in.

[RISC OS Open source page](#)

MessageTrans_FileInfo

SWI &41500 · MessageTrans

This SWI is used to provide information about a message file.

Before calling

R1 Pointer to filename

On return

R0 flag word:

bit 0 set >= file is held in memory (can be accessed directly)

bit 1 - 31 reserved (you can safely ignore them)

R2 Size of buffer required to hold file

Notes

If the file is held in memory, and you require read-only access, you need not use a buffer to access it.

[RISC OS Open source page](#)

MessageTrans_GSLookup

SWI &41507 · MessageTrans

This SWI is used to translate a message token into a string, with optional parameter substitution.

Before calling

R0 Pointer to 4-word data structure, or 0 for global messages

R1 Pointer to Token

R2 Pointer to buffer for result

R3 Size of buffer

R4 Pointer to parameter 0, or 0 if none

R5 Pointer to parameter 1, or 0 if none

R6 Pointer to parameter 2, or 0 if none

R7 Pointer to parameter 3, or 0 if none

On return

R0 Preserved

R1 Pointer to token-terminator

R2 Pointer to result string (read only with no substitution if R2 = 0 on entry)

R3 Size of result excluding terminator

Notes

This SWI is like MessageTrans_Lookup except that the result is then GStran's'd (OS_GStran's) before return. If R2 = 0 on entry this call is identical to MessageTrans_LookUp.

[RISC OS Open source page](#)

MessageTrans_Lookup

SWI &41502 · MessageTrans

This SWI is used to translate a message token into a string, with optional parameter substitution.

Before calling

R0 Pointer to 4-word data structure, or 0 for global messages

R1 Pointer to Token

R2 Pointer to buffer for result

R3 Size of buffer

R4 Pointer to parameter 0, or 0 if none

R5 Pointer to parameter 1, or 0 if none

R6 Pointer to parameter 2, or 0 if none

R7 Pointer to parameter 3, or 0 if none

On return

R0 Preserved

R1 Pointer to token-terminator

R2 Pointer to result string (read only with no substitution if R2 = 0 on entry)

R3 Size of result excluding terminator

Notes

If R2 = 0 on entry, you'll get a pointer to a read-only string, and no parameter substitution will be done, also the string will be terminated by 10, and not 0.

If R2 != 0, %0 to %3 in the string will be replaced by the strings in R4 to R7. Prior to this call, you must have called MessageTrans_OpenFile.

If the token is not found in the message file specified, it will be searched for in the global message file.

If R2 = 0 on entry and the file is not stored in your own workspace, then the pointer will become invalid after Service_MessageFileClosed is issued - your application must trap this service call so it can re-read pointers.

[RISC OS Open source page](#)

MessageTrans_MakeMenus

SWI &41503 · MessageTrans

This SWI is used to create a menu structure from a skeleton menu of tokens.

Before calling

R0 Pointer to data structure

R1 Pointer to menu definition

R2 Pointer to buffer for result

R3 Size of buffer

On return

R0 Preserved

R1 Preserved

R2 Pointer to the end of the data within the buffer provided in R2 upon entry

R3 Bytes remaining in buffer

Notes

If the menu item is writeable and indirected, the corresponding 3 words in the destination buffer should be set to correct values before calling this SWI.

If the menu item is indirected and not writeable then a pointer into the message file will be stored in the menu structure created.

If the message file is held in RMA this pointer (and hence the menu structure) will become invalid after Service_MessageFileClosed is issued - your application must trap this service call and delete the menu if it is issued.

[RISC OS Open source page](#)

MessageTrans_OpenFile

SWI &41501 · MessageTrans

This SWI is used to open a message-file. If R2 = 0, then the data-structure in R0 and the filename in R1 both must be in the RMA.

Before calling

R0 Pointer to file descriptor (held in the RMA if R2 = 0 on entry)

R1 Pointer to filename (held in the RMA if R2 = 0 on entry)

R2 Pointer to buffer to hold the file data, or value of 0 (allocate some space in RMA, or use file directly if possible)

On return

--

Notes

An application may decide that it would like to buffer the file in its own workspace (rather than the RMA) if it needs to be loaded, or use the file directly if it is already in memory. To do this:

```
SYS "MessageTrans_FileInfo",,filename$ TO flags%,size%
```

```
IF flags% AND 1 THEN buffer%=0 ELSE buffer%=FNalloc(size%)
```

```
SYS "OS_Module",6,,,17+LENfilename$ TO ,,filedesc%
```

```
$(filedesc%+16)=filename$
```

```
SYS "MessageTrans_OpenFile",filedesc%,filedesc%+16,buffer%
```

where FNalloc() allocates a buffer of a given size, by using the Wimp_SlotSize or "END=" command. In fact the filename and file descriptor only need to be in the RMA if R2=0 on entry to MessageTrans_OpenFile.

[RISC OS Open source page](#)

NetFS_ConvertDate

SWI &4004B · NetFS

The purpose of this call is to convert a file server time and date to a RISC OS time and date.

Before calling

R0 Pointer to file server format date and time

R1 Pointer to 5 byte output buffer

On return

R1 Preserved

Notes

This SWI converts from a 5 byte file server format date and time to a standard RISC OS 5 byte time block.

SWI is re-entrant.

[RISC OS Open source page](#)

NetFS_DoFSOp

SWI &40048 · NetFS

The purpose of this call is to Ccmmmand the current file server to perform an operation.

Before calling

R0 File server function

R1 Buffer

R2 Number of bytes to send from buffer to file server

R3 Total size of buffer

On return

R0 Return condition from file server

R3 Number of bytes written to buffer by file server

Notes

If an error is returned you should copy it, as its buffer may be overwritten by subsequent use of NetFS.

SWI is not re-entrant.

[RISC OS Open source page](#)

NetFS_DoFSOpToGivenFS

SWI &4004C · NetFS

The purpose of this call is to command a given file server to perform an operation.

Before calling

R0 File server function

R1 Buffer

R2 Number of bytes to send from buffer to file server

R3 Total size of buffer

R4 Station number

R5 Net number

On return

R0 Return condition from file server

R3 Number of bytes written to buffer by file server

Notes

If an error is returned you should copy it, as its buffer may be overwritten by subsequent use of NetFS.

SWI is not re-entrant.

[RISC OS Open source page](#)

NetFS_EnableCache

SWI &40051 · NetFS

The purpose of this call is to enable a suspended event task.

Before calling

--

On return

--

Notes

This SWI restarts the automatic update of the list of filesystems, which is automatically suspended when NetFS_EnumerateFSList is called.

SWI is not re-entrant.

[RISC OS Open source page](#)

NetFS_EnumerateFS

SWI &4004A · NetFS

The purpose of this call is to list all file servers to which the NetFS software is currently logged on.

Before calling

R0 Offset in file server list of first item to read

R1 Pointer to buffer

R2 Size of buffer in bytes

R3 Number of items to read

On return

R0 Offset of next item to read, or -1 if finished

R3 Number of items read

Notes

The information is returned as a series of 20 byte blocks – if you are logged on to your current file server it will be returned last. Apart from this the order is undefined. To return user ID information in the list use NetFS_EnumerateFSContexts.

SWI is not re-entrant.

[RISC OS Open source page](#)

NetFS_EnumerateFSContexts

SWI &4004E · NetFS

The purpose of this call is to list all file servers, including User ID to which the NetFS software is currently logged on.

Before calling

R0 Offset in file server list of first item to read

R1 Pointer to buffer

R2 Size of buffer in bytes

R3 Number of items to read

On return

R0 Offset of next item to read, or -1 if finished

R2 Space remaining in buffer

R3 Number of items read

Notes

The information is returned as a series of 44 byte blocks.

SWI is not re-entrant.

[RISC OS Open source page](#)

NetFS_EnumerateFSList

SWI &40049 · NetFS

The purpose of this call is to list all file servers to which the NetFS software is currently logged on.

Before calling

R0 Offset in file server list of first item to read

R1 Pointer to buffer

R2 Size of buffer in bytes

R3 Number of items to read

On return

R0 Offset of next item to read, or -1 if finished

R3 Number of items read

Notes

This SWI is used to list information about file servers held by NetFS – the information returned by the *ListFS command. The information is returned as a series of 20 byte blocks, in alphabetical order. To ensure that the list does not change during enumeration the SWI disables the automatic update of the list – you must restart this after enumeration by calling NetFS_EnableCache.

SWI is not re-entrant.

[RISC OS Open source page](#)

NetFS_GetObjectUID

SWI &40050 · NetFS

The purpose of this call is to get a unique identifier for an object.

Before calling

R0 Pointer to canonical object name

R3 Pointer to canonical special field

On return

R0 File object type

R2 Load address

R3 Exec address

R4 Object length

R5 Object attribute

R6 Least significant word of UID

R7 Most significant word of UID

Notes

This SWI is similar to OS_File 5, except that it also returns a 64 bit UID for the object. This is guaranteed to be unique over all file servers on all networks.

SWI is not re-entrant.

[RISC OS Open source page](#)

NetFS_ReadCurrentContext

SWI &40044 · NetFS

This call is not implemented.

Before calling

--

On return

R0 Corrupted

R1 Corrupted

R2 Corrupted

Notes

Please disregard this SWI Call.

SWI is not re-entrant.

[RISC OS Open source page](#)

NetFS_ReadFSName

SWI &40042 · NetFS

The purpose of this call is to read the name of the your current file server.

Before calling

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R1 Free bytes in buffer

Notes

If your are not logged on, this call returns a null string.

SWI is re-entrant.

[RISC OS Open source page](#)

NetFS_ReadFSNumber

SWI &40040 · NetFS

The purpose of this call is to return the full station number of your current file server.

Before calling

--

On return

R0 Station number

R1 Net number

Notes

This SWI returns the full station number of your current file server, or R0 = R1 = 0 if your current file server is not set.

SWI is re-entrant.

[RISC OS Open source page](#)

NetFS_ReadFSTimeouts

SWI &40046 · NetFS

The purpose of this call is to read the current values for timeouts used by NetFS.

Before calling

--

On return

R0 Transmit count

R1 Transmit delay

R2 Machine peek count

R3 Machine peek delay

R4 Receive delay

R5 Broadcast delay

Notes

This SWI reads the current communication timeouts. All delays are in centiseconds.

SWI is re-entrant.

[RISC OS Open source page](#)

NetFS_ReadUserId

SWI &4004F · NetFS

The purpose of this call is to return the current user ID if logged on to the current file server.

Before calling

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Corrupted

R1 Pointer to terminator

R2 Free bytes in buffer (including terminator)

Notes

This SWI reads the user ID if logged onto the current file server, or returns a null string (single zero) if you are not logged on.

SWI is re-entrant.

[RISC OS Open source page](#)

NetFS_SetCurrentContext

SWI &40045 · NetFS

This call is not implemented.

Before calling

--

On return

R0 All registers preserved

Notes

Please disregard this SWI Call.

SWI is re-entrant.

[RISC OS Open source page](#)

NetFS_SetFSName

SWI &40043 · NetFS

The purpose of this call is to set the name of the your current file server.

Before calling

R0 Pointer to buffer containing name

On return

--

Notes

An error is generated if you are not logged onto the specified file server. It restores context (e.g. the current directory).

SWI is not re-entrant.

[RISC OS Open source page](#)

NetFS_SetFSNumber

SWI &40041 · NetFS

The purpose of this call is to set the full station number of your current file server.

Before calling

R0 Station number

R1 Net number

On return

R0 Corrupted

R1 Corrupted

Notes

This SWI sets the full station number used by NetFS as the current file server. It restores context (e.g. the current directory).

SWI is not re-entrant.

[RISC OS Open source page](#)

NetFS_SetFSTimeouts

SWI &40047 · NetFS

The purpose of this call is to set the current values for timeouts used by NetFS.

Before calling

R0 Transmit count

R1 Transmit delay

R2 Machine peek count

R3 Machine peek delay

R4 Receive delay

R5 Broadcast delay

On return

--

Notes

This SWI sets the current communication timeouts. All delays are in centiseconds.

SWI is re-entrant.

[RISC OS Open source page](#)

NetFS_UpdateFSList

SWI &4004D · NetFS

The purpose of this call is to add names of discs to the list of names held by NetFS.

Before calling

R0 Station number

R1 Net number

On return

R0 Corrupted

R1 Corrupted

Notes

This SWI fetches the names of discs on the given file server (or all servers if $R0 = R1 = 0$), to ensure that the list from NetFS_EnumerateFS is current.

SWI is not re-entrant.
[RISC OS Open source page](#)

NetPrint_BindPSName

SWI &40206 · NetPrint

The purpose of this call is to convert a printer server's name to its address, providing it is free.

Before calling

R0 Pointer to name

On return

R1 Station number

R2 Net number

Notes

If it cannot be found, an error is returned.

SWI is not re-entrant
[RISC OS Open source page](#)

NetPrint_ConvertStatusToString

SWI &40208 · NetPrint

The purpose of this call is to translate a status value returned from NetPrint_ListServers into the local language.

Before calling

R0 Pointer to status block

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Status Block for NetPrint_ConvertStatusToString

Offset / Contents

0 – Contents: Status value

1 – Contents: Optional station number

2 – Contents: Optional net number

Notes

This SWI converts the given Status to a null terminated string in the supplied buffer.

SWI is re-entrant

[RISC OS Open source page](#)

NetPrint_ListServers

SWI &40207 · NetPrint

The purpose of this call is to return the names of all printer servers.

Before calling

- R0** Reason code
- R1** Pointer to buffer
- R2** Size of buffer
- R3** Time to take before returning (centiseconds)

Reason Codes

Value / Contents

- 0** – Contents: Names and numbers
- 1** – Contents: Names only
- 2** – Contents: Names, numbers, status

Notes

This SWI returns the names of all printer servers. The format of the data returned in the buffer depends on the code passed in R0.

SWI is not re-entrant

[RISC OS Open source page](#)

NetPrint_ListServers 0

SWI &40207 · NetPrint

The purpose of this call is to return the names and numbers of all printer servers.

Before calling

- R0** 0 (reason code)
- R1** Pointer to buffer
- R2** Size of buffer
- R3** Time to take before returning (centiseconds)

On return

- R0** Number of entries returned
- R1** Preserved
- R2** Preserved
- R3** Return code:
 - 0 = Timed out
 - 1 = Buffer full

Returned Buffer Format

Offset / Contents

0 – Contents: Station number

1 – Contents: Net number

2 – Contents: Server name (zero terminated)

Notes

SWI is not re-entrant

[RISC OS Open source page](#)

NetPrint_ListServers 1

SWI &40207 · NetPrint

The purpose of this call is to return the names only of all printer servers.

Before calling

R0 1 (reason code)

R1 Pointer to buffer

R2 Size of buffer

R3 Time to take before returning (centiseconds)

On return

R0 Number of entries returned

R1 Preserved

R2 Preserved

R3 Return code:

0 = Timed out

1 = Buffer full

Returned Buffer Format

Offset / Contents

0 – Contents: Server name (zero terminated)

Notes

Only names are returned using this call. They are sorted by name (case insensitive), with no duplicates.

SWI is not re-entrant

[RISC OS Open source page](#)

NetPrint_ListServers 2

SWI &40207 · NetPrint

The purpose of this call is to return the names, numbers and status of all printer servers.

Before calling

R0 2 (reason code)

R1 Pointer to buffer

R2 Size of buffer

R3 Time to take before returning (centiseconds)

On return

R0 Number of entries returned

R1 Preserved

R2 Preserved

R3 Return code:

0 = Timed out

1 = Buffer full

Returned Buffer Format

Offset / Contents

0 – Contents: Station number

1 – Contents: Net number

2 – Contents: Status value

3 – Contents: Station number for status

4 – Contents: Net number for status

5 – Contents: Server name (zero terminated)

Notes

SWI is not re-entrant

[RISC OS Open source page](#)

NetPrint_ReadPSName

SWI &40202 · NetPrint

The purpose of this call is to read the name of your current printer server.

Before calling

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

This call returns a null string if your printer server is only stored as a number.

SWI is re-entrant

[RISC OS Open source page](#)

NetPrint_ReadPSNumber

SWI &40200 · NetPrint

The purpose of this call is to return the full station number of your current printer server.

Before calling

--

On return

R0 Station number

R1 Net number

Notes

This SWI returns the full station number of your current printer server, or **R0** = **R1** = 0 if your current printer server is stored only as a name.

SWI is re-entrant

[RISC OS Open source page](#)

NetPrint_ReadPSTimeouts

SWI &40204 · NetPrint

The purpose of this call is to read the current values for timeouts used by NetPrint.

Before calling

--

On return

R0 Transmit count

R1 Transmit delay

R2 Machine peek count

R3 Machine peek delay

R4 Receive delay

R5 Broadcast delay

Notes

All delays are in centiseconds.

SWI is re-entrant

[RISC OS Open source page](#)

NetPrint_SetPSName

SWI &40203 · NetPrint

The purpose of this call is to set by name the printer server used as your current one.

Before calling

R0 Pointer to name

On return

R0 Preserved

Notes

SWI is not re-entrant

[RISC OS Open source page](#)

NetPrint_SetPSNumber

SWI &40203 · NetPrint

The purpose of this call is to set by name the printer server used as your current one.

Before calling

R0 Pointer to name

On return

R0 Preserved

Notes

SWI is not re-entrant

[RISC OS Open source page](#)

NetPrint_SetPSTimeouts

SWI &40205 · NetPrint

The purpose of this call is to set the current values for timeouts used by NetPrint.

Before calling

R0 Transmit count

R1 Transmit delay

R2 Machine peek count

R3 Machine peek delay

R4 Receive delay

R5 Broadcast delay

On return

- -

Notes

All delays are in centiseconds.

SWI is re-entrant

[RISC OS Open source page](#)

OS_AbortTrap

SWI &67

The abort trapping system allows for data and prefetch aborts to be handled at a higher level than previously possible (e.g. via OS_ClaimProcessorVector or the environment handlers). They are designed to be easy to use for purposes such as implementing on-demand page mapping or virtual hardware devices; i.e. where the code can interact with the address range as if it were normal memory, with the OS and AbortTrap silently handling any aborts by emulating the aborting instruction and then resuming execution.

Before calling

R0 Reason code:

0 = Register handler

1 = Deregister handler

R1 Low address (inclusive)

R2 High address (exclusive)

R3 Address of handler routine

R4 R12 value for handler routine

R0 Bits 0-3: Reason code:

0 = Store operation. Bit 4 indicates if the access is privileged (1) or unprivileged (0).

1 = Load operation. Bit 4 indicates if the access is privileged (1) or unprivileged (0).

2 = Memory map operation. Bits 4-9 indicate the required permissions.

3+ Reserved

Bits 4+: Flags

R1 Pointer to buffer to use as data source (reason 0) or destination (reason 1). Invalid for memmap requests (reason 2).

R2 Base address of memory being accessed

R3 Length of memory being accessed (bytes)

R12 R4 value specified when routine was registered

- Entered in SVC mode with interrupts disabled

On return

- All registers preserved

- On success, V clear and the buffer contents updated (if applicable)

On failure, V set and R0 pointing to an appropriate error block

All other registers preserved

In practice

When a data abort occurs, the kernel will decode the instruction to determine the type of access (e.g. read or write), the address range being accessed, and which registers the data should be transferred in/out of. The kernel will then attempt to emulate the load/store operation, by invoking the relevant AbortTrap handler(s) for the region.

Handlers have three ways of responding to requests:

- They can comply with the request. If enough handlers comply, then the kernel will apply the necessary register updates (e.g. performing base register writeback) and return from the abort handler, resuming execution as if the instruction had never generated the abort.
- They can fail with a serious error (bit 31 of error number set). This will cause the abort processing to be halted and the error to be raised in a similar manner to unhandled data/prefetch aborts (i.e. the exception registers block will be populated, OS_ReadSysInfo 7 will be updated, the privileged mode stacks will be flattened, and the error will be raised via OS_GenerateError).
- They can decline to handle the request, by returning with a non-serious error. In this case, the kernel will check to see if there are any other handlers for that region, without raising the error. Note that if there are multiple handlers which cover the same region, the order in which the kernel will try them is undefined.

AbortTrap handlers are invoked after the OS_ClaimProcessorVector handlers, and before the data/prefetch abort environment handlers. This means that if there's an abort which AbortTrap isn't able to handle (e.g. there's a part of the address range which isn't covered by a handler, or where the handler has declined the request), then the kernel will restore the original registers and control will pass to the data/prefetch environment handler.

AbortTrap handlers will only be invoked for translation faults (no memory at the target address) or permission faults (e.g. attempting to write to read-only memory). Other types of fault (alignment faults, or hardware faults like cache parity errors) will not result in handlers being invoked, and there is no guarantee that unpredictable instructions (e.g. LDM which loads the base register but also applies writeback) will be processed. AbortTrap processing will not be performed if there is insufficient space in the SVC stack or the kernel believes the stack pointer is corrupt.

For aborts which cross page boundaries, it's possible that only one of the pages is responsible for generating the abort, and the other page would have been able to service the request directly without aborting. The kernel will detect this and make sure that AbortTrap handlers are only invoked for pages where the abort was necessary; for pages where no abort would be generated, the kernel will emulate the access itself by directly copying to/from the relevant address range.

When the kernel invokes an AbortTrap handler, it can ask it to do one of two things:

- Emulate a memory access by transferring data in/out of a buffer provided by the kernel. This is used for most load/store operations. For store operations, the buffer will have been filled with the values of the registers which are being stored. For load operations, the values which the AbortTrap handler writes to the buffer will be copied to the relevant registers.
- Map in memory with the indicated (minimum) permissions. This is used for prefetch aborts, and for some advanced instructions such as LDREX/STREX (which cannot safely be emulated via normal load/store instructions). For this type of request, if all the necessary handlers comply, then the kernel will return from the exception and re-execute the aborting instruction, allowing the CPU to execute it natively.

AbortTrap handlers

Care must be taken to ensure that an error is returned if the handler is called with unsupported/unrecognised reason code or flags.

For memmap operations, the permission flags are:

Bit / Meaning

- 4 – Meaning: Executable in user mode
- 5 – Meaning: Writeable in user mode
- 6 – Meaning: Readable in user mode
- 7 – Meaning: Executable in privileged modes
- 8 – Meaning: Writeable in privileged modes
- 9 – Meaning: Readable in privileged modes

This is the same pattern as used by OS_Memory 18.

Notes

Where possible, it's preferable to use Abortable Dynamic Areas instead of directly registering with OS_AbortTrap.

Code which registers AbortTrap handlers may also need to respond to Service_ValidateAddress, in order for the OS & software to consider the address range as valid.

Regions which handlers are asked to service have no size or alignment guarantees, other than the minimum size being one byte, and the requested region being fully within the low/high addresses specified when the handler was registered.

For handler reasons 0 & 1, bytes should be transferred in linear order; the kernel will take care of any rotation required by pre-ARMv7 rotated loads.

SWP/SWPB instructions are emulated as a load operation (handler reason 1) followed by a store operation (handler reason 0). If the handler maps in the page as part of the read operation, the kernel may directly emulate the store by copying the data itself.

Currently, data aborts from Thumb mode are not handled and will be passed straight on to the abort environment handler.

StrongARM abort handling errata are not dealt with, so care must be taken when using handlers on afflicted CPUs (see Notable CPU bugs).

For operations which cross multiple regions (e.g. multiple AbortTrap regions or directly-accessible memory regions), and where the overall result is a failure:

- Some AbortTrap read or write operations may have been performed
- For write instructions, some directly-accessible memory may have been updated

This behaviour is broadly consistent with the ARM abort model.

This call is available in RISC OS Select and RISC OS 5.29+. Differences between RISC OS 5 and RISC OS Select are:

- RISC OS 5 supports all ARMv3-ARMv8 load/store instructions, including FPA, VFP and NEON
- RISC OS 5 adds support for mmap operations
- RISC OS 5 adds support for processing prefetch aborts via AbortTrap

[RISC OS Open source page](#)

OS_AddCallback

SWI &54

The purpose of this call is to add a transient Callback to the list.

Before calling

R0 Pointer to address to call

R1 Value of R12 to be called with

On return

R0 Preserved

R1 Preserved

Notes

This call places a transient callback to the list of functions which want to be called as soon as RISC OS is not busy. It is usually used by an interrupt routine that needs to do complex processing that would take too long in an interrupt, or that needs to call a non-re-entrant SWI.

OS_AddCallback tells RISC OS that the interrupt routine wishes to be 'called back'. This can happen when RISC OS is being threaded out of - that is, when it enters User mode with interrupts enabled (either after a SWI is called, or after an interrupt).

It can also happen when RO is idling, for example, while it is waiting in OS_ReadC. Note that, unlike non-transient callbacks, there is no requirement for the SVC stack to be empty for a transient callback to be triggered.

Transient CallBacks can safely be used by many clients.

A routine called by this mechanism is called in SVC mode with interrupts enabled, must preserve ALL registers and return by MOV PC,R14.

[RISC OS Open source page](#)

OS_AddToVector

SWI &47

The purpose of this call is to add a routine to the list of those that claim a vector.

Before calling

R0 Vector number

R1 Pointer to address of claiming routine

R2 Value to be passed in R12 when the routine is called

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

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.

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.

This call provides the same functionality as OS_Claim, except that this call does not remove earlier identical instances (i.e. those which have identical workspace and code addresses).

[RISC OS Open source page](#)

OS_AMBControl

SWI &70

This SWI is for internal use only. You must not use it.

Before calling

--

On return

--

[RISC OS Open source page](#)

OS_Args

SWI &09

The purpose of this call is to read or write to an open file.

Before calling

R0 Reason Code

R1 File handle

R2 Attribute (per reason code)

On return

Different per reason code

Reason Codes

0 – Description: Read pointer/FS

1 – Description: Write pointer

2 – Description: Read extent

3 – Description: Write extent

4 – Description: Read allocated size

5 – Description: Read EOF status

6 – Description: Reserve space

7 – Description: Read pathname of open file

8 – Description: Inform of changed open file

9 – Description: IOCTL

254 – Description: Read info on file handle

255 – Description: Flush file(s)

Notes

Each reason code specifies a different use for this call.

[RISC OS Open source page](#)

OS_Args 0

SWI &09 · OS_Args

The purpose of this call is to return the temporary filing system number (**R1** = 0), or file pointer (**R1** <> 0).

Before calling

R0 0 (reason code)

R1 File handle, or 0

On return

R0 Temporary Filing System Numbers 1 (if **R1**=0 on entry)

R1 Preserved

R2 File pointer (if **R1** <> 0 on entry)

Notes

1 FileSwitch versions between 1.70 (RISC OS 3.00) and 2.75 will return 0×40000000 if no temporary filing system is active. For backwards compatibility, treat values 0 and 0×40000000 as meaning 'none'.

[RISC OS Open source page](#)

OS_Args 1

SWI &09 · OS_Args

The purpose of this call is to set the file sequential pointer.

Before calling

R0 1 (reason code)

R1 File handle

R2 New pointer

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

If the pointer is set outside the current extent, more space is then allocated (and filled with zeros).

[RISC OS Open source page](#)

OS_Args 2

SWI &09 · OS_Args

The purpose of this call is to return the extent of a file.

Before calling

R0 2 (reason code)

R1 File handle

On return

R0 Preserved

R1 Preserved

R2 Extent of file

[RISC OS Open source page](#)

OS_Args 254

SWI &09 · OS_Args

The purpose of this call is to return information on the given file handle.

Before calling

R0 254 (reason code)

R1 File handle

On return

R0 Stream status word

R1 Preserved

R2 Filing System Information Word

[RISC OS Open source page](#)

OS_Args 255

SWI &09 · OS_Args

The purpose of this call is to ensure that any buffered data on the given file(s) is written to the filing system.

Before calling

R0 255 (reason code)

R1 File handle, or 0 for all open files on the temporary filing system.

On return

R0 Preserved

R1 Preserved

R2 Preserved

[RISC OS Open source page](#)

OS_Args 3

SWI &09 · OS_Args

The purpose of this call is to set the extent of a file.

Before calling

R0 3 (reason code)

R1 File handle

R2 Extent

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

Any new space allocated when setting the extent will be filled with zeros.

[RISC OS Open source page](#)

OS_Args 4

SWI &09 · OS_Args

The purpose of this call is to return the allocated size of a file.

Before calling

R0 4 (reason code)

R1 File handle

On return

R0 Preserved

R1 Preserved

R2 Allocated size of the file in bytes

Notes

The call returns the space set aside on the media for the given handle, regardless of its extent. Often this will be larger, for example to avoid fragmenting open files.

The value return will always be equal to, or greater than the file extent.

[RISC OS Open source page](#)

OS_Args 5

SWI &09 · OS_Args

The purpose of this call is to return the status of EOF.

Before calling

R0 5 (reason code)

R1 File handle

On return

R0 Preserved

R1 Preserved

R2 0 if not EOF, else EOF

Notes

If the end of the file has been reached, then a value of EOF will be returned, else the value returned will be EOF.

[RISC OS Open source page](#)

OS_Args 6

SWI &09 · OS_Args

The purpose of this call is to set the required minimum size of a file.

Before calling

R0 6 (reason code)

R1 File handle

R2 Size to ensure (in bytes)

On return

R0 Preserved

R1 Preserved

R2 Space reserved (in bytes)

Notes

The size of the file will be at least the size provided in R2 on entry. The size ensured may however be larger than requested, for example to reduce fragmentation.

[RISC OS Open source page](#)

OS_Args 7

SWI &09 · OS_Args

The purpose of this call is to return the pathname of a file.

Before calling

R0 7 (reason code)

R1 File handle

R2 Pointer to buffer

R5 Size of buffer

On return

R0 Preserved

R1 Preserved

R2 Preserved

R5 Free bytes in buffer

[RISC OS Open source page](#)

OS_Args 8

SWI &09 · OS_Args

The purpose of this call is to inform of a change to an image stamp by an image filing system.

Before calling

R0 8 (reason code)

R1 File handle

R2 New image stamp

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

An image filing system uses this call when it has changed a disc's image stamp.

FileSwitch passes this on to the handler of discs, which typically does nothing more than storing the new image stamp in the disc's record.

[RISC OS Open source page](#)

OS_Args 9

SWI &09 · OS_Args

The purpose of this call is to perform an IOCTL function.

Before calling

R0 9 (reason code)

R1 File handle

R2 IOCTL Parameter Block

On return

R0 Preserved

R1 Preserved

R2 Preserved

[RISC OS Open source page](#)

OS_BGet

SWI &0A

The purpose of this call is to read a byte from the current sequential file pointer.

Before calling

R1 File handle

On return

R0 Byte read if C clear, undefined if C set

R1 Preserved

[RISC OS Open source page](#)

OS_BinaryToDecimal

SWI &28

The purpose of this call is to convert a 32-bit integer to a string.

Before calling

R0 Integer

R1 Pointer to buffer

R2 Size of buffer

On return

R0 Preserved

R1 Preserved

R2 Length of string

Notes

Another call `OS_ConvertInteger4` also provides the same functionality as this call.

[RISC OS Open source page](#)

OS_BPut

SWI &0B

The purpose of this call is to write a byte to the current sequential file pointer.

Before calling

R0 Byte

R1 File handle

On return

R0 Preserved

R1 Preserved

[RISC OS Open source page](#)

OS_BreakCtrl

SWI &18

The purpose of this call is to set up the address of the BreakPoint handler and the BreakPoint register save block.

Before calling

R0 pointer to BreakPoint register save block (0 to read)

R1 pointer to BreakPoint handler (0 to read)

On return

R0 pointer to previous BreakPoint register save block

R1 pointer to previous BreakPoint handler

Notes

This call is deprecated, and should not be used. OS_ChangeEnvironment should be used instead.

[RISC OS Open source page](#)

OS_BreakPt

SWI &17

The purpose of this call is to cause a BreakPoint trap, and the BreakPoint handler to be entered.

Before calling

--

On return

--

Notes

When this is executed, all user mode registers are saved into a block and the BreakPoint handler is called. The saved registers are only guaranteed to be correct for user mode.

The default handler displays the message 'Break point at &xxxxx' and calls OS_Exit. This SWI would be placed in code by the debugger at required breakpoints.

The handler can be changed with OS_ChangeEnvironment.

[RISC OS Open source page](#)

OS_Byte

SWI &06 · Controlling sources: Hdr:OsBytes managed by the allocations manager

This SWI has many different uses.

Before calling

R0 Reason codes

Notes

Different parts of the Operating System rely on this SWI.

The value in R0 upon entry determines the function of the call, and is known as the reason code.

For calls with reason codes between 166 and 255:

NewVal = (OldVal AND R2) EOR R1

To read: / R1=0 / R2=&FF

To write: – R1=0: R1=New Value · R2=&FF: R2=0

Reason Codes

- 0 – Hex: &00 · Meaning: Display OS version info
- 1 – Hex: &01 · Meaning: Write user flag
- 2 – Hex: &02 · Meaning: Specify input stream
- 3 – Hex: &03 · Meaning: Specify output stream
- 4 – Hex: &04 · Meaning: Cursor key status
- 5 – Hex: &05 · Meaning: Write printer driver type
- 6 – Hex: &06 · Meaning: Write printer ignore char
- 7 – Hex: &07 · Meaning: Write RS232 receive rate
- 8 – Hex: &08 · Meaning: Write RS232 transmit rate
- 9 – Hex: &09 · Meaning: Write duration of first flash colour
- 10 – Hex: &0A · Meaning: Write duration of second flash colour
- 11 – Hex: &0B · Meaning: Write keyboard auto-repeat delay
- 12 – Hex: &0C · Meaning: Write keyboard auto-repeat rate
- 13 – Hex: &0D · Meaning: Disable event
- 14 – Hex: &0E · Meaning: Enable event
- 15 – Hex: &0F · Meaning: Flush buffer
- 16 – Hex: &10 · Meaning: Store number of channels to be sampled
- 17 – Hex: &11 · Meaning: Return to the caller, doing nothing
- 18 – Hex: &12 · Meaning: Reset function keys
- 19 – Hex: &13 · Meaning: Wait for vsync
- 20 – Hex: &14 · Meaning: Reset font definitions
- 21 – Hex: &15 · Meaning: Flush selected buffer
- 25 – Hex: &19 · Meaning: Reset group of font definitions
- 70 – Hex: &46 · Meaning: Read/Write country number
- 71 – Hex: &47 · Meaning: Read/Write alphabet or keyboard number
- 106 – Hex: &6A · Meaning: Select pointer/Activate mouse
- 112 – Hex: &70 · Meaning: Write VDU driver screen bank
- 113 – Hex: &71 · Meaning: Write display hardware screen bank
- 114 – Hex: &72 · Meaning: Write shadow/non-shadow state
- 117 – Hex: &75 · Meaning: Read VDU status
- 118 – Hex: &76 · Meaning: Reflect keyboard status in LEDs
- 120 – Hex: &78 · Meaning: Temporarily lock auto-repeat

- 121 – Hex: &79 · Meaning: Keyboard scan
- 122 – Hex: &7A · Meaning: Keyboard scan from 16 decimal
- 124 – Hex: &7C · Meaning: Clear escape condition
- 125 – Hex: &7D · Meaning: Set escape condition
- 126 – Hex: &7E · Meaning: Acknowledge escape condition
- 127 – Hex: &7F · Meaning: Check for end of file
- 128 – Hex: &80 · Meaning: Return the switch state and last channel converted, or a channel's uncalibrated position
- 129 – Hex: &81 · Meaning: Scan for a particular key
- 134 – Hex: &86 · Meaning: Read text cursor position
- 135 – Hex: &87 · Meaning: Read char at text cursor, and screen mode
- 138 – Hex: &8A · Meaning: Insert byte into buffer
- 139 – Hex: &8B · Meaning: Write filing system options
- 143 – Hex: &8F · Meaning: Issue module service call
- 144 – Hex: &90 · Meaning: Set vertical screen shift, and interlace
- 145 – Hex: &91 · Meaning: Get char from buffer
- 152 – Hex: &98 · Meaning: Examine buffer status
- 153 – Hex: &99 · Meaning: Insert char into input buffer
- 156 – Hex: &9C · Meaning: Read/Write serial port state
- 160 – Hex: &A0 · Meaning: Read VDU variable value
- 161 – Hex: &A1 · Meaning: Read non-volatile CMOS memory
- 162 – Hex: &A2 · Meaning: Write non-volatile CMOS memory
- 163 – Hex: &A3 · Meaning: Read/Write general graphics info
- 165 – Hex: &A5 · Meaning: Read output cursor position
- 176 – Hex: &B0 · Meaning: 50 Hz counter
- 177 – Hex: &B1 · Meaning: Read input source
- 178 – Hex: &B2 · Meaning: Read/Write keyboard semaphore
- 181 – Hex: &B5 · Meaning: Read/Write serial input interpretation status
- 182 – Hex: &B6 · Meaning: Read/Write Nolgnore state
- 188 – Hex: &BC · Meaning: Read the current channel
- 189 – Hex: &BD · Meaning: Read the number of channels to be sampled
- 190 – Hex: &BE · Meaning: Read the resolution of conversion
- 191 – Hex: &BF · Meaning: Read/Write RS423 busy flag
- 192 – Hex: &C0 · Meaning: Read serial port state
- 193 – Hex: &C1 · Meaning: Read/Write flash counter
- 194 – Hex: &C2 · Meaning: Read duration of second flash colour
- 195 – Hex: &C3 · Meaning: Read duration of first flash colour
- 196 – Hex: &C4 · Meaning: Read/Write keyboard auto-repeat delay
- 197 – Hex: &C5 · Meaning: Read/Write keyboard auto-repeat rate
- 198 – Hex: &C6 · Meaning: Read/Write *Exec file handle
- 199 – Hex: &C7 · Meaning: Read/Write *Spool file handle
- 200 – Hex: &C8 · Meaning: Read/Write Break and Escape effect
- 201 – Hex: &C9 · Meaning: Read/Write keyboard disable flag
- 202 – Hex: &CA · Meaning: Read/Write keyboard status byte
- 203 – Hex: &CB · Meaning: Read/Write serial input buffer minimum space
- 204 – Hex: &CC · Meaning: Read/Write serial ignore flag
- 211 – Hex: &D3 · Meaning: Read/Write bell channel
- 212 – Hex: &D4 · Meaning: Read/Write bell sound volume
- 213 – Hex: &D5 · Meaning: Read/Write bell frequency
- 214 – Hex: &D6 · Meaning: Read/Write bell duration
- 216 – Hex: &D8 · Meaning: Read/Write length of function key string
- 217 – Hex: &D9 · Meaning: Read/Write paged mode line count
- 218 – Hex: &DA · Meaning: Read/Write bytes in VDU queue

- 219 – Hex: &DB · Meaning: Read/Write Tab key code
- 220 – Hex: &DC · Meaning: Read/Write escape character
- 221 – Hex: &DD · Meaning: Read/Write interpretation of input values &C0-&CF
- 222 – Hex: &DE · Meaning: Read/Write interpretation of input values &D0-&DF
- 223 – Hex: &DF · Meaning: Read/Write interpretation of input values &E0-&EF
- 224 – Hex: &E0 · Meaning: Read/Write interpretation of input values &F0-&FF
- 225 – Hex: &E1 · Meaning: Read/Write function key interpretation
- 226 – Hex: &E2 · Meaning: Read/Write Shift Fn interpretation
- 227 – Hex: &E3 · Meaning: Read/Write Ctrl Fn interpretation
- 228 – Hex: &E4 · Meaning: Read/Write Shift Ctrl Fn interpretation
- 229 – Hex: &E5 · Meaning: Read/Write escape key status
- 230 – Hex: &E6 · Meaning: Read/Write escape effects
- 236 – Hex: &EC · Meaning: Read/Write character destination status
- 237 – Hex: &ED · Meaning: Read/Write cursor key status
- 238 – Hex: &EE · Meaning: Read/Write numeric keypad interpretation
- 240 – Hex: &F0 · Meaning: Read country number
- 241 – Hex: &F1 · Meaning: Read/Write user flag
- 242 – Hex: &F2 · Meaning: Read serial baud rates
- 243 – Hex: &F3 · Meaning: Read timer switch state
- 245 – Hex: &F5 · Meaning: Read/Write printer driver type
- 246 – Hex: &F6 · Meaning: Read/Write printer ignore character
- 247 – Hex: &F7 · Meaning: Read/Write Break key actions
- 250 – Hex: &FA · Meaning: Read VDU driver screen bank number
- 251 – Hex: &FB · Meaning: Read display screen bank number
- 253 – Hex: &FD · Meaning: Read last break type
- 254 – Hex: &FE · Meaning: Set effect of Shift-Ctrl on numeric keypad
- 255 – Hex: &FF · Meaning: Read/Write boot option

[RISC OS Open source page](#)

OS_Byte 0

SWI &06 · OS_Byte

The purpose of this call is to return information about either the Operating System or machine.

Before calling

R0 0 (reason code)

R1 0 to return OS version, or

1 to return machine information

On return

R0 Pointer to OS version string as an error block

R1 Preserved

R2 Corrupted

V flag set

R0 Preserved

R1 Machine type

R2 Corrupted

Notes

If R1=0 on entry, this call displays the OS version by returning it as an error string, with V set.

If R1=1 this call returns the machine type in R1 - all RISC OS machines return 6.

[RISC OS Open source page](#)

OS_Byte 1

SWI &06 · OS_Byte

The purpose of this call is to write a location that is unused by the OS, which can then be used to pass results between applications.

Before calling

R0 1

R1 New value

On return

R0 Preserved

R1 Machine type

R2 Corrupted

Notes

The location may be read (and written) with OS_Byte 241.

This is a legacy call provided for compatibility with the 8-bit range of Acorn machines. System variables now provide a much better way of passing results, and should be used in instead.

[RISC OS Open source page](#)

OS_Byte 10

SWI &06 · OS_Byte

The purpose of this call is to set the duration of the second flash colour in frames-per-second.

Before calling

R0 10 (&A)

R1 New duration (in video frames, or 0 for 'forever')

On return

R0 Preserved

R1 Previous duration

R2 Corrupted

Notes

A duration of 0 means display the second colour constantly. The current duration may be read with OS_Byte 194, and may also be set with VDU 23,10?.

The first flash colour duration can be set using OS_Byte 9.

[RISC OS Open source page](#)

OS_Byte 106

SWI &06 · OS_Byte

The purpose of this call is to set the pointer shape of one of four possible shapes, or simply activate the mouse.

Before calling

R0 106 (&6A)

R1 Pointer shape and linkage flag

On return

R0 Preserved

R1 Shape and linkage flag before overwritten

R2 Corrupted

Notes

You can define four 'pointer buffers' using OS_Word 21, each holding a different shape definition for the mouse pointer. This call allows you to select one of these definitions for future use, or to turn off the pointer depending on the bottom seven bits of R1.

[RISC OS Open source page](#)

OS_Byte 11

SWI &06 · OS_Byte

The purpose of this call is to write the keyboard auto-repeat delay in centiseconds.

Before calling

R0 11 (&0B)

R1 Delay period in centiseconds

On return

R0 Preserved

R1 Previous delay period

R2 Corrupted

Notes

You must hold down each key on the keyboard for a number of centiseconds before it begins to autorepeat.

This call enables you to change the initial delay from the default set by *Configure Delay. This variable may also be read or set by OS_Byte 196. This setting is persisted in the machines CMOS RAM.

The auto-repeat rate is set by using OS_Byte 12

[RISC OS Open source page](#)

OS_Byte 112

SWI &06 · OS_Byte

The purpose of this call is to select the bank of screen memory which is to be used by the VDU drivers.

Before calling

R0 112 (&70)

R1 Bank number

On return

R0 Preserved

R1 Previous bank number

R2 Corrupted

Notes

If R1 equals 0, then the default bank for the current mode (1 or 2) will be used. Any other value means to set the screenbank which starts at Screenstart+(R1*ModeSize). The maximum bank number is TotalScreenSize/ModeSize. The default bank is for a non-shadow mode 1, for a shadow mode 2.

[RISC OS Open source page](#)

OS_Byte 113

SWI &06 · OS_Byte

The purpose of this call is to select the bank of screen memory which is to be used by the display hardware.

Before calling

R0 113 (&71)

R1 Bank number, 0 for default

On return

R0 Preserved

R1 Value before being overwritten

R2 Corrupted

Notes

The value may be overwritten with OS_Byte 251.

[RISC OS Open source page](#)

OS_Byte 114

SWI &06 · OS_Byte

The purpose of this call is to write the shadow/non-shadow state to memory.

Before calling

R0 114 (&72)

R1 0 = modes will be shadow

1 = modes will be non-shadow

On return

R0 Preserved

R1 Previous shadow state

R2 Corrupted

Notes

This call determines whether future changes to numbered screen modes will be forced to use shadow state. OS_Byte 112 and OS_Byte 113 control the use of shadow banks.

Shadow state requires twice the amount of RAM than the equivalent non-shadow mode since two copies of the screen are stored in memory.

[RISC OS Open source page](#)

OS_Byte 117

SWI &06 · OS_Byte

The purpose of this call is to read the VDU status.

Before calling

R0 117 (&75)

On return

R0 Preserved

R1 Status flags

Notes

This call returns the content of the VDU status byte. This byte gives information on the way in which characters are output.

[RISC OS Open source page](#)

OS_Byte 118

SWI &06 · OS_Byte

The purpose of this call is to update the status of the keyboard LEDs to reflect that of the keyboard.

Before calling

R0 118 (&76)

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

Notes

The settings of Caps Lock, Scroll Lock and Num Lock are held in a location referred to as the keyboard status byte. See OS_Byte 202 for details of this. Under normal circumstances they are shown by the LEDs on the keyboard, but these are not updated when the status is changed using OS_Byte 202. This call ensures that the current contents of the keyboard status byte are reflected in the LEDs.

[RISC OS Open source page](#)

OS_Byte 12

SWI &06 · OS_Byte

The purpose of this call is to write the keyboard auto-repeat rate in centiseconds.

Before calling

R0 12 (&0C)

R1 Repeat rate in centiseconds (0 to set repeat delay/rate to default)

On return

R0 Preserved

R1 Previous repeat rate

R2 Corrupted

Notes

After the auto-repeat delay specified by OS_Byte 11, each key will repeat until released at the rate passed to this call.

This enables you to change the initial rate from the default set by *Configure Repeat. This variable may also be set/read by OS_Byte 197. This setting is persisted in the machines CMOS RAM.

The auto-repeat delay is set by using OS_Byte 11

[RISC OS Open source page](#)

OS_Byte 120

SWI &06 · OS_Byte

The purpose of this call is to temporarily lock keyboard auto-repeat.

Before calling

R0 120 (&78)

R1 0

R2 0

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

Notes

This call is kept for compatibility with the BBC/Master 128. It locks the auto-repeat mechanism for the duration of a key being down (to prevent multiple input).

[RISC OS Open source page](#)

OS_Byte 121

SWI &06 · OS_Byte

The purpose of this call is to check which keys are being pressed.

Before calling

R0 121 (&79)

R1 Key to start at, or key EOR &80 for single key

On return

R0 Preserved

R1 For range check: key pressed, or &FF if none pressed

For single key: &FF; if key pressed, or other if not

R2 Corrupted

Notes

This call can also be used to determine which mouse button has been clicked.

OS_Byte 122 is identical to this call, except that it ignores any key that has a value of less than 16. This includes Shift, Ctrl, Alt and mouse keys.

[RISC OS Open source page](#)

OS_Byte 122

SWI &06 · OS_Byte

The purpose of this call is to check which keys are being pressed (ignoring the Shift, Ctrl, Alt and mouse keys).

Before calling

R0 122 (&7A)

On return

R0 Preserved

R1 Key, or &FF if none pressed

R2 Corrupted

Notes

This call is equivalent to OS_Byte 121 with R1=16. It scans the keyboard ignoring Shift, Control, Alt and mouse buttons. Any key that has a value below 16 is ignored.

[RISC OS Open source page](#)

OS_Byte 124

SWI &06 · OS_Byte

The purpose of this call is to clear the escape function

Before calling

R0 124 (&7C)

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

Notes

This call clears any escape condition by calling the escape handler with R11=0, then returns.

OS_Byte 125 is used to set the escape condition, and OS_Byte 126 is used to acknowledge it.

[RISC OS Open source page](#)

OS_Byte 125

SWI &06 · OS_Byte

The purpose of this call is to set the 'Escape condition' flag.

Before calling

R0 125 (&7D)

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

Notes

This call sets an escape condition (calling the escape handler), but does not generate an escape event.

OS_Byte 124 is used to clear the escape condition, and OS_Byte 126 is used to acknowledge it.

[RISC OS Open source page](#)

OS_Byte 126

SWI &06 · OS_Byte

The purpose of this call is to acknowledge the escape condition and attempt to clear it.

Before calling

R0 126 (&7E)

On return

R0 Preserved

R1 0 if no escape condition

255 if escape condition cleared

R2 Corrupted

Notes

If an escape condition exists, it performs the escape condition side effects if these have been enabled with OS_Byte 230, and then calls the escape handler to indicate that the escape has been cleared.

OS_Byte 124 is used to clear the escape condition, and OS_Byte 125 is used to set it.

[RISC OS Open source page](#)

OS_Byte 127

SWI &06 · OS_Byte

The purpose of this call is to check for End of File.

Before calling

R0 127 (&7F)

R1 File handle

On return

R0 Preserved

R1 Non-zero for EOF

R2 Corrupted

Notes

This call is now deprecated. Please use OS_Args 5 instead.

[RISC OS Open source page](#)

OS_Byte 128

SWI &06 · OS_Byte

The purpose of this call is to return the mouse coordinates, or number of bytes in an input buffer, or number of free bytes in an output buffer.

Before calling

R0 128 (&80)

R1 Reason code

On return

R0 Preserved

R1 If R1 = 1 on entry: state of switch 0 on joysticks 0 (in bit 0) and 1 (in bit 1)

Otherwise: Bits 0-7 give LSB of answer

R2 If R1 = 1 on entry: Number of last channel converted

Otherwise: Bits 0-23 give bits 8-31 of answer

Notes

This call checks the (buffered) mouse position or the free space in system buffers, depending on the reason code passed in R1. It is recommended to use OS_Mouse to read the mouse position, as this reads the current (unbuffered) position.

[RISC OS Open source page](#)

OS_Byte 129

SWI &06 · OS_Byte

This call has four specific uses that are determined by the values passed in R1 and R2.

Before calling

R0 129 (&81)

R1 Dependant upon operation

R2 Dependant upon operation

In practice

- OS_Byte 129 (read a key within time limit)
- OS_Byte 129 (read OS version)
- OS_Byte 129 (scan keyboard for range of keys)
- OS_Byte 129 (scan keyboard for single key)

[RISC OS Open source page](#)

OS_Byte 129 (OS)

SWI &06 · OS_Byte

The purpose of this call is to return a value that specifies the version of the Operating System.

Before calling

R0 129 (&81)

R1 0

R2 &FF

On return

R0 Preserved

R1 OS Version

R2 0

[RISC OS Open source page](#)

OS_Byte 129 (scan range)

SWI &06 · OS_Byte

The purpose of this call is to scan the keyboard for a range of keys.

Before calling

R0 129 (&81)

R1 &01 – &7F lowest keyboard scan code EOR &7F

R2 &FF

On return

R0 Preserved

R1 Key number, or 255 if none pressed

R2 Corrupted

[RISC OS Open source page](#)

OS_Byte 129 (scan single)

SWI &06 · OS_Byte

The purpose of this call is to scan the keyboard for a single key.

Before calling

R0 129 (&81)

R1 &80 - &FF keyboard scan code EOR &FF

R2 &FF

On return

R0 Preserved

R1 0 if not pressed, or 255 if pressed

R2 0 if not pressed, or 255 if pressed

[RISC OS Open source page](#)

OS_Byte 129 (time)

SWI &06 · OS_Byte

The purpose of this call is to read a key within a specified time limit.

Before calling

R0 129 (&81)

R1 0-255 (time limit low byte)

R2 &00 - &7F (time limit high byte)

On return

R0 Preserved

R1 ASCII code of character read, undefined if not read

R2 0 character read

27 escape condition

255 timeout

Notes

This call will extract a character from the keyboard buffer/current function key expansion if one is available, else will wait for $(R2 * 256 + R1)$ centiseconds, where the maximum wait is 327.67 seconds, as $R2 \leq 127$.

Cursor edit mode will operate if enabled with OS_Byte 4.

If an escape condition is detected the application should either acknowledge it with OS_Byte 126 or clear it with OS_Byte 124.

The Wimp depends on undefined behaviour: the C flag is clear when a character has been read, and set otherwise. It also tests R1 on exit, in particular expecting R1 = &FF on timeout, rather than using the documented result in R2. New code should use R2 to determine the outcome.

[RISC OS Open source page](#)

OS_Byte 13

SWI &06 · OS_Byte

The purpose of this call is to disable an event.

Before calling

R0 13 (&0D)

R1 Event Number

On return

R0 Preserved

R1 Old enable state (0 means was already disabled)

R2 Corrupted

Notes

Event enable/disable states are reference-counted, to ensure that events are only broadcast to the system if at least one program is interested in receiving them.

This call decreases the count of enable requests for a particular event being called through the EventV (event vector). When the count reaches 0 the event is actually disabled.

Care should be taken to match OS_Byte 13 calls with OS_Byte 14 as the event will be locked on if the count goes negative. This is a safety measure to prevent the event being inadvertently silenced if a bug causes OS_Byte 13 to be called too many times.

It is possible to safely read the event enable count by making a call to OS_Byte 14 followed by a call to OS_Byte 13. However unless you can control or predict when other systems are going to enable or disable the event, this information may be of limited use.

Events are enabled by calling OS_Byte 14

[RISC OS Open source page](#)

OS_Byte 134

SWI &06 · OS_Byte

The purpose of this call is to return the text cursor position.

Before calling

R0 134 (&86)

On return

R0 Preserved

R1 Position in x direction

R2 Position in y direction

Notes

The text cursor position is returned unless cursor editing is in progress, in which case the position returned is that of the input cursor. OS_Byte 165 reads the position of the cursor irrespective of cursor editing mode.

[RISC OS Open source page](#)

OS_Byte 135

SWI &06 · OS_Byte

The purpose of this call is to return the screen mode and ASCII code of the character at the text cursor position.

Before calling

R0 135 (&87)

On return

R0 Preserved

R1 ASCII value of character (0 if unreadable)

R2 Mode Specifier

Notes

If cursor editing is in progress, it returns the character code returned by the character at the input cursors position. i.e. the character would be copied as input the next time the Copy key is pressed).

If a numbered mode is used, the returned mode number never has bit 7 set, even if it's a shadow mode.

[RISC OS Open source page](#)

OS_Byte 138

SWI &06 · OS_Byte

The purpose of this call is to insert a byte into a buffer.

Before calling

R0 138 (&8A)

R1 Buffer number

R2 Byte to insert

On return

R0 Preserved

R1 Preserved

R2 Preserved

C flag set if buffer was full, otherwise clear

Notes

While inserting 9 bytes into the mouse buffer you must disable interrupts to prevent a real mouse transition from being entered into the middle of your data.

Note that characters 0 and 128-255 are inserted into the keyboard buffer as two bytes 0,n to distinguish them from function keys – if you are inserting strings into the keyboard buffer you should be careful to check for top bit set characters and precede them with a 0 byte, else you may erroneously insert function key keypresses – 'l' is equivalent to pressing F12, 'ü' to pressing Control, Shift F12, which in the desktop will respectively generate a CLI prompt, and initiate desktop shutdown.

[RISC OS Open source page](#)

OS_Byte 139

SWI &06 · OS_Byte

The purpose of this call is to set filing system options.

Before calling

R0 139 (&8B)

R1 Option

R2 Value

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

Notes

This call is deprecated. Please use OS_FSControl 10 instead.

[RISC OS Open source page](#)

OS_Byte 14

SWI &06 · OS_Byte

The purpose of this call is to enable an event.

Before calling

R0 14 (&0E)

R1 Event Number

On return

R0 Preserved

R1 Old enable state (0 means was already disabled)

R2 Corrupted

Notes

Event enable/disable states are reference-counted, to ensure that events are only broadcast to the system if at least one program is interested in receiving them.

This call enables a particular event so that it is broadcast through EventV (the event vector). When your program is no longer interested in the event, it must call OS_Byte 13 so that the event can be silenced if no other programs are interested in it.

Under RISC OS 5, if the reference counter for an event reaches its maximum value then the counter will be locked to that value, preventing further OS_Byte 13 / OS_Byte 14 calls from having any effect. This is a safety feature to prevent the event being inadvertently disabled, e.g. if 300 calls to OS_Byte 14 are followed by 299 calls to OS_Byte 13. Currently each counter is a single byte (giving a maximum value of 255), but this may change in future OS versions.

[RISC OS Open source page](#)

OS_Byte 143

SWI &06 · OS_Byte

The purpose of this call is to issue a module service call.

Before calling

R0 143 (&8F)

R1 Service type

R2 Argument for service

On return

R0 Preserved

R1 Preserved

R2 Possible return argument

Notes

This call is deprecated. Please use OS_ServiceCall instead.

[RISC OS Open source page](#)

OS_Byte 144

SWI &06 · OS_Byte

The purpose of this call is to specify the vertical screen alignment and interlace options after the next mode change.

Before calling

R0 144 (&90)

R1 Vertical screen shift (as a signed 8 bit number)

R2 Interlace flag, 0=on, 1=off

On return

R0 Preserved

R1 Previous vertical screen shift

R2 Previous interlace flag

Notes

This call is equivalent to *TV.

[RISC OS Open source page](#)

OS_Byte 145

SWI &06 · OS_Byte

The purpose of this call is to extract the next byte from the given buffer

Before calling

R0 145 (&91)

R1 Buffer number

On return

R0 Preserved

R1 Preserved

R2 Byte removed

C flag set if buffer was empty on entry, clear if byte was successfully removed

Notes

The buffer number is not validated – undefined results (rather than an error) occur if a bad buffer number is passed.

[RISC OS Open source page](#)

OS_Byte 15

SWI &06 · OS_Byte

The purpose of this call is to flush all buffers, or the current input buffer.

Before calling

R0 15 (&0F)

R1 0 to flush all buffers

1 to flush the current input buffer

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

Notes

The contents of either all buffers or only the current input buffer are discarded, depending on R1.

Individual buffers may be flushed using OS_Byte 21.

[RISC OS Open source page](#)

OS_Byte 152

SWI &06 · OS_Byte

The purpose of this call is to check whether a given buffer is empty, and if a byte is available return it without removing it from the buffer.

Before calling

R0 152 (&98)

R1 Buffer number

On return

R0 Preserved

R1 Preserved

R2 Next byte in buffer, corrupted if buffer empty

C flag set if buffer is empty, otherwise clear

Notes

The buffer number is not validated – undefined results (rather than an error) occur if a bad buffer number is passed.

[RISC OS Open source page](#)

OS_Byte 153

SWI &06 · OS_Byte

The purpose of this call is to insert a character into either the keyboard or serial buffer.

Before calling

R0 153 (&99)

R1 Input source

R2 Byte to insert

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

C flag set if buffer was full, otherwise clear

Notes

If the current escape character (set with OS_Byte 220) is inserted, and escape generation is not disabled (see OS_Byte 229 and OS_Byte 200) then an escape condition occurs either as

- an escape event is generated (if enabled with OS_Byte 14) or
- the escape handler is called.

The effects set by OS_Byte 230 occur when the escape condition is acknowledged with OS_Byte 126.

When inserting into the keyboard buffer, the same rules must be followed as for OS_Byte 138 when dealing with characters outside the range 1-127.

[RISC OS Open source page](#)

OS_Byte 156

SWI &06 · OS_Byte

The purpose of this call is to read or write the serial port state.

Before calling

R0 156 (&9C)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 State before being overwritten

R2 Keyboard disable flag (see OS_Byte 201)

Notes

This call is now deprecated. Please use OS_SerialOp 0 and OS_SerialOp 1 to change the receive interrupt/control bit instead.

When writing, the effects depend on R1:

0 - 1 – Value: 0 - 2 · Effect: No effect

3 · Reset transmit, receive, control registers

2 - 4 – Value: 0 · Effect: 7 bit, even parity, 2 stop bits (7E2)

1 · 7 bit, odd parity, 2 stop bits (7O2)

2 · 7 bit, even parity, 1 stop bits (7E1)

3 · 7 bit, odd parity, 1 stop bit (7O1)

4 · 8 bit, no parity, 2 stop bits (8N2)

5 · 8 bit, no parity, 1 stop bits (8N1)

6 · 8 bit, even parity, 1 stop bits (8E1)

7 · 8 bit, odd parity, 1 stop bit (8O1)

5 - 6 – Value: 0 · Effect: RTS low, transmit interrupt disabled

1 · RTS low, transmit interrupt enabled

2 · RTS high, transmit interrupt disabled

3 · RTS low, transmit interrupt disabled, break level on transmit data

7 – Value: 0 · Effect: Receive interrupt disabled

1 · Receive interrupt enabled

You should only write bit 7 with this call (and really should be using OS_Byte 2). The value can also be read with OS_Byte 192.

The default value for bits 2-4 is stored in OS_Byte CMOS 16.

[RISC OS Open source page](#)

OS_Byte 16

SWI &06 · OS_Byte

The purpose of this call is to store the number of channels to be sampled by the ADC on the user port / I/O Podule.

Before calling

R0 16 (&10)

R1 Number of channels to be sampled (0 - 4)

On return

R0 Preserved

Notes

If the value passed is greater than 4, it is stored as 4. This stored value can be read using OS_Byte 189.

(On the I/O Podule this call also set the number of channels to be sampled; but this is not possible on the joystick interface's hardware.)

[RISC OS Open source page](#)

OS_Byte 160

SWI &06 · OS_Byte

The purpose of this call is to read the VDU variable value.

Before calling

R0 160 (&A0)

R1 VDU variable

On return

R0 Preserved

R1 value of variable

R2 value of next variable (entry R1 + 1)

Notes

This call is provided for compatibility with the BBC/Master 128, and reads various VDU variables. Note that it does not use the same numbers for variables as OS_ReadVduVariables, and that that SWI can read many more variables.

Only the first 16 MOS VDU vars were implemented in Arthur. 3rd party modules such as Cerilica VDUVars160 implement the full set - see VDU 23,16.

[RISC OS Open source page](#)

OS_Byte 161

SWI &06 · OS_Byte

The purpose of this call is to read a setting stored in the CMOS ram, which is used to stored configuration information.

Before calling

R0 161 (&A1)

R1 CMOS byte number

On return

R0 Preserved

R1 Preserved

R2 Value

Notes

OS_Byte 162 is used to write values to the CMOS ram.

[RISC OS Open source page](#)

OS_Byte 162

SWI &06 · OS_Byte

The purpose of this call is to write a setting to the CMOS ram, which is used to store configuration information.

Before calling

R0 162 (&A2)

R1 CMOS byte number

R2 Value

On return

R0 Preserved

R1 Preserved

R2 Corrupted

Notes

OS_Byte 161 is used to read values from the CMOS ram.

[RISC OS Open source page](#)

OS_Byte 163

SWI &06 · OS_Byte

This call has four specific uses that are determined by the values passed in R2.

Before calling

R0 163 (&A3)

R1 242

R2 Dependant upon operation

In practice

- OS_Byte 163 (reset dot-dash pattern and length)
- OS_Byte 163 (set dot-dash pattern repeat length)
- OS_Byte 163 (return status information)
- OS_Byte 163 (return information on current sprite)

[RISC OS Open source page](#)

OS_Byte 163 (reset)

SWI &06 · OS_Byte

The purpose of this call is to reset the dot-dash pattern and length used for graphics.

Before calling

R0 163 (&A3)

R1 242

R2 0

On return

R0 Preserved

R1 Preserved

R2 Preserved

[RISC OS Open source page](#)

OS_Byte 163 (return sprite information)

SWI &06 · OS_Byte

The purpose of this call is to return information on the current sprite.

Before calling

R0 163 (&A3)

R1 242

R2 66

On return

R0 Preserved

R1 Width (in pixels)

R2 Height (in pixels)

[RISC OS Open source page](#)

OS_Byte 163 (return status)

SWI &06 · OS_Byte

The purpose of this call is to return status information.

Before calling

R0 163 (&A3)

R1 242

R2 65

On return

R0 Preserved

R1 Packed Information

R2 System sprite area size (in bytes)

[RISC OS Open source page](#)

OS_Byte 163 (set)

SWI &06 · OS_Byte

The purpose of this call is to set the dot-dash pattern repeat length used for graphic operations.

Before calling

R0 163 (&A3)

R1 242

R2 1 – 64

On return

R0 Preserved

R1 Preserved

R2 Preserved

[RISC OS Open source page](#)

OS_Byte 165

SWI &06 · OS_Byte

The purpose of this call is to return the position of the output cursor.

Before calling

R0 165 (&A5)

On return

R0 Preserved

R1 Position in x direction

R2 Position in y direction

Notes

This returns the position, even whilst cursor editing is in progress.

OS_Byte 134 reads the position of the text cursor.

[RISC OS Open source page](#)

OS_Byte 17

SWI &06 · OS_Byte

The purpose of this call is to return to the caller, doing nothing

Before calling

R0 17 (&11)

R1 Channel number on which to force ADC conversion (0 - 4) - not implemented???

On return

R0 Preserved

Notes

(On the I/O Podule this call also set the number of channels to be sampled; but this is not possible on the joystick interface's hardware.)

[RISC OS Open source page](#)

OS_Byte 176

SWI &06 · OS_Byte

The purpose of this call is to read or set the 50Hz counter.

Before calling

R0 176 (&B0)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Value before being overwritten

R2 Corrupted

Notes

This call reads or sets the "50Hz" counter, a 1 byte counter which is decreased every vSync. As a consequence the rate of this counter is actually mode and monitor dependent, and should not be used for accurate timing.

[RISC OS Open source page](#)

OS_Byte 177

SWI &06 · OS_Byte

The purpose of this call is to read a character from either the keyboard or serial buffer.

Before calling

R0 177 (&B1)

R1 0

R2 255

On return

R0 Preserved

R1 Input source

R2 Corrupted

Notes

This returns the number of the buffer from which character input gets characters assuming Exec is not enabled – see OS_Byte 198).

You must not use any numbers than 0 and 255 for R1 and R2 – to change this variable use OS_Byte 2.

[RISC OS Open source page](#)

OS_Byte 178

SWI &06 · OS_Byte

The purpose of this call is to read or write the keyboard semaphore.

Before calling

R0 178 (&B2)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Value before being overwritten

R2 Corrupted

Notes

This call is now deprecated, and should not be used.

[RISC OS Open source page](#)

OS_Byte 18

SWI &06 · OS_Byte

The purpose of this call is to reset the currently assigned definitions to each function key.

Before calling

R0 18 (&12)

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

Notes

It unsets the system variables Key\$0 to Key\$15 and cancels any current function key expansion.

Function key expansion can also be cancelled with OS_Byte 216.

[RISC OS Open source page](#)

OS_Byte 181

SWI &06 · OS_Byte

The purpose of this call is to read or sets the state of serial interpretation

Before calling

R0 181 (&B5)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Status before being overwritten

R2 No ignore state (see OS_Byte 182)

Status Codes

Status / Meaning

0 – Meaning: Characters read from the serial buffer are given the same interpretation as those read from the keyboard buffer

1 – Meaning: No keyboard interpretation is performed

Notes

The status value of 1 is the default setting.

If a status value of 0 then;

- Current escape character generates an escape condition
- Function keys are expanded
- Escape events and character entering buffer events are generated

[RISC OS Open source page](#)

OS_Byte 182

SWI &06 · OS_Byte

The purpose of this call is to read or write the printer NoIgnore state.

Before calling

R0 182 (&B6)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Status before being overwritten

R2 No ignore state (see OS_Byte 182)

Notes

If bit 7 is clear then the ignore character set by OS_Byte 6 or OS_Byte 246 is filtered from the printer output stream. If bit 7 is set, then the printer stream passes all characters. The default state of this bit is stored in OS_Byte CMOS 15.

The default setting of this flag is controlled by *Configure Ignore and may be changed temporarily using *Ignore.

[RISC OS Open source page](#)

OS_Byte 188

SWI &06 · OS_Byte

The purpose of this call is to read the current channel.

Before calling

R0 188 (&BC)

On return

R0 Preserved

R1 Current Channel (1 - 4)

Notes

(On the I/O Podule this call does the same.)

[RISC OS Open source page](#)

OS_Byte 189

SWI &06 · OS_Byte

The purpose of this call is to read the number of channels to be sampled.

Before calling

R0 189 (&BD)

On return

R0 Preserved

R1 Number of channels to be sampled (0 – 4)

Notes

The number of channels to be stored using OS_Byte 16.

(On the I/O Podule this call does the same.)

[RISC OS Open source page](#)

OS_Byte 19

SWI &06 · OS_Byte

The purpose of this call is to wait for the vertical sync.

Before calling

R0 19 (&13)

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

Notes

This call takes the control of the ARM until the electron beam of the monitor has reached its last position and jumps to the top left of the screen.

[RISC OS Open source page](#)

OS_Byte 190

SWI &06 · OS_Byte

The purpose of this call is to read the resolution of sampled conversion.

Before calling

R0 190 (&BE)

On return

R0 Preserved

R1 0 (default conversion, which is 16 bit)

Notes

This is always returned as 0, meaning the default for the hardware, which is 16 bit.

(On the I/O Podule this call can return 0 for the default (which for its hardware is 12 bit), or 8 for 8 bit conversion, or 12 for 12 bit conversion; however, hardware limitations mean that conversion can only be guaranteed to 8 bits.)

[RISC OS Open source page](#)

OS_Byte 191

SWI &06 · OS_Byte

The purpose of this call is to read or write the serial busy flag.

Before calling

R0 191 (&BF)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Value before being overwritten

R2 See serial state (see OS_Byte 192)

Notes

The use of this call is deprecated.

[RISC OS Open source page](#)

OS_Byte 192

SWI &06 · OS_Byte

The purpose of this call is to read the serial port state.

Before calling

R0 192 (&C0)

R1 0

R2 255

On return

R0 Preserved

R1 Communication state

R2 Flash counter (see OS_Byte 193)

Notes

This call is deprecated. Please use OS_SerialOp 0 or OS_SerialOp 1 instead.

It is equivalent to a read operation with OS_Byte 156. It is provided for backwards compatibility only.

This call should not be used to write the value - doing so would make the value in memory inconsistent with the hardware.

[RISC OS Open source page](#)

OS_Byte 193

SWI &06 · OS_Byte

The purpose of this call is to read or write the flash counter.

Before calling

R0 193 (&C1)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Value before being overwritten

R2 Corrupted

Notes

This call reads or sets the count-down timer for flashing colours. Provided that the flash period is not infinite, the counter is loaded with the duration of the first flash colour and counts down by one every VSync until it reaches zero. At this point the colours are swapped and the counter loaded with the second flash period. etc.

[RISC OS Open source page](#)

OS_Byte 194

SWI &06 · OS_Byte

The purpose of this call is to read the duration of the second flash colour.

Before calling

R0 194 (&C2)

R1 EOR value

R2 255

On return

R0 Preserved

R1 Duration (video frames)

R2 Corrupted

Notes

This call reads the duration of the second flash colour, as set with OS_Byte 10 or VDU 23,10?.

This call should not be used to write the value – doing so would only change the value in memory, without affecting the hardware.

The duration of the first flash colour can be read by OS_Byte 195.

[RISC OS Open source page](#)

OS_Byte 195

SWI &06 · OS_Byte

The purpose of this call is to read the duration of the first flash colour.

Before calling

R0 195 (&C3)

R1 EOR value

R2 255

On return

R0 Preserved

R1 Duration (video frames)

R2 Corrupted

Notes

This call reads the duration of the first flash colour, as set with OS_Byte 9 or VDU 23,9?. This call should not be used to write the value – doing so would only change the value in memory, without affecting the hardware.

The duration of the second flash colour can be read by OS_Byte 194.

[RISC OS Open source page](#)

OS_Byte 196

SWI &06 · OS_Byte

The purpose of this call is to read or write the keyboard auto-repeat delay.

Before calling

R0 196 (&C4)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Value before being overwritten

R2 Keyboard auto-repeat rate (see OS_Byte 197)

Notes

To read or write the keyboard's auto-repeat rate, use OS_Byte 197.

See OS_Byte 11 for more information.

[RISC OS Open source page](#)

OS_Byte 197

SWI &06 · OS_Byte

The purpose of this call is to read or write the keyboard auto-repeat rate.

Before calling

R0 197 (&C5)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Value before being overwritten

R2 Corrupted

Notes

Note, that *FX 12,0 resets the auto repeat rate/delay to default *FX 197,0 sets the auto repeat rate to zero. See OS_Byte 12 for more information.

To read or write the keyboard's auto-repeat delay, use OS_Byte 196.

[RISC OS Open source page](#)

OS_Byte 198

SWI &06 · OS_Byte

The purpose of this call is to read or write the *Exec file handle.

Before calling

R0 198 (&C6)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous file handle

R2 Corrupted

Notes

This call reads or sets the *Exec file handle. A handle of 0 means that Exec is disabled. A non-zero handle means that Exec is enabled, and hence is the current input stream.

[RISC OS Open source page](#)

OS_Byte 199

SWI &06 · OS_Byte

The purpose of this call is to read or write the *Spool file handle.

Before calling

R0 199 (&C7)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous file handle

R2 Corrupted

Notes

This call reads or sets the *Spool file handle. A handle of 0 means that spooling is disabled, a non-zero handle means that spooling is enabled (unless disabled with OS_Byte 3).

Changing the spool handle to an invalid file handle can really ruin your day (every call to OS_WriteC etc. will return an error, including the calls used to report the error to screen...).

[RISC OS Open source page](#)

OS_Byte 2

SWI &06 · OS_Byte

The purpose of this call is to select the device from which all subsequent input is taken by OS_ReadC.

Before calling

R0 2

R1 Stream selection

On return

R0 Preserved

R1 Value before being overwritten

R2 Corrupted

Notes

On exit only 0 or 1 can be returned, so 0 may also mean 2. The state of this variable can be read with OS_Byte 177.

If Exec is enabled (see OS_Byte 198) then the Exec file handle is used as the current input stream.

[RISC OS Open source page](#)

OS_Byte 20

SWI &06 · OS_Byte

The purpose of this call is to reset the VDU font definitions.

Before calling

R0 20 (&14)

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

Notes

This call is used to reset the shape of the printable ASCII characters in the range of 32-127, which may be modified with VDU 23,32? - 255. To reset the whole or a smaller group of character definitions, use OS_Byte 25.

[RISC OS Open source page](#)

OS_Byte 200

SWI &06 · OS_Byte

The purpose of this call is to read or write the Break and Escape effect.

Before calling

R0 200 (&C8)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 State before being overwritten

R2 Keyboard disable flag (see OS_Byte 201)

Notes

This call can read, or change, the effects of a reset (including resets caused by Break) and of Escape.

The bottom two bits of R1 have the following significance:

Bit / Value / Effect

0 – Value: 0 · Effect: Normal escape action

1 · Escape disabled unless caused by OS_Byte 125

1 – Value: 0 · Effect: Normal reset action

1 · Power on reset (only if bits 2-7 of R1 are zero) This means a value of %00000001x causes a memory clear (no matter what kind of reset was pressed)

[RISC OS Open source page](#)

OS_Byte 201

SWI &06 · OS_Byte

The purpose of this call is to enable or disable the keyboard.

Before calling

R0 201 (&C9)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Flag before being overwritten

R2 Corrupted

Notes

This call enables or disables the keyboard. 0 will enable the keyboard (as normal). Any other value will stop the interrupt routine inserting key presses into the keyboard buffer.

To read or write information about the status of the keyboard, please use OS_Byte 202.

[RISC OS Open source page](#)

OS_Byte 202

SWI &06 · OS_Byte

The purpose of this call is to read or write information about the keyboard's status.

Before calling

R0 202 (&CA)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Status before being overwritten

R2 Serial input threshold (see OS_Byte 203)

Notes

The keyboard status byte holds information on the current status of the keyboard, such as the setting of Caps Lock. This call enables you to read and change these settings. Note that this call does not update the LEDs. If you don't stamp this state with OS_Byte 118, the LEDs won't be switched on or off. The default state of Caps Lock is stored in OS_Byte CMOS 11.

The keyboard can be enabled or disabled by OS_Byte 201.

[RISC OS Open source page](#)

OS_Byte 203

SWI &06 · OS_Byte

The purpose of this call is to read or write the serial buffer threshold.

Before calling

R0 203 (&CB)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Value before being overwritten

R2 Serial ignore flag (see OS_Byte 204)

Notes

This call reads or sets the serial buffer threshold. While the free space is below this value the serial hardware will attempt to halt incoming data. OS_SerialOp 0 controls the handshaking method. The default threshold is 9 characters.

[RISC OS Open source page](#)

OS_Byte 204

SWI &06 · OS_Byte

The purpose of this call is to enable or disable serial input buffering.

Before calling

R0 204 (&CC)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Flag before being overwritten

R2 Corrupted

Notes

0 will enable buffering of received data. Any other value will prevent buffering, but errors will still generate events if enabled with OS_Byte 14.

[RISC OS Open source page](#)

OS_Byte 21

SWI &06 · OS_Byte

The purpose of this call is to flush a specific buffer.

Before calling

R0 21 (&15)

R1 Buffer number

On return

R0 Preserved

R1 Preserved

R2 Corrupted

Notes

OS_Byte 15 is used to flush all the buffers, or the current input buffer.

[RISC OS Open source page](#)

OS_Byte 211

SWI &06 · OS_Byte

The purpose of this call is to read or write the sound channel for the bell (VDU 7?).

Before calling

R0 211 (&D3)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous channel

R2 Bell volume (see OS_Byte 212)

Notes

The default channel is 1.

To read/write the bell volume use OS_Byte 212. To read/write the frequency use OS_Byte 213. To read/write the duration use OS_Byte 214.

[RISC OS Open source page](#)

OS_Byte 212

SWI &06 · OS_Byte

The purpose of this call is to read or write the sound volume for the bell.

Before calling

R0 212 (&D4)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous volume

R2 Bell frequency (see OS_Byte 213)

Notes

This call reads or sets the bell volume, the range 128 (loud) – 248 (soft).

*Configure Loud sets the volume to &90. *Configure Quiet sets the volume to &D0.

To read/write the bell channel use OS_Byte 211. To read/write the frequency use OS_Byte 213. To read/write the duration use OS_Byte 214.

[RISC OS Open source page](#)

OS_Byte 213

SWI &06 · OS_Byte

The purpose of this call is to read or write the sound frequency (pitch) for the bell.

Before calling

R0 213 (&D5)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous frequency

R2 Bell duration (see OS_Byte 214)

Notes

This call reads or sets the bell pitch, which is a BBC style frequency in the range 0-255. The default pitch is 100.

To read/write the bell channel use OS_Byte 211. To read/write the volume use OS_Byte 212. To read/write the duration use OS_Byte 214.

[RISC OS Open source page](#)

OS_Byte 214

SWI &06 · OS_Byte

The purpose of this call is to read or write the duration for the bell.

Before calling

R0 214 (&D6)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous duration

R2 Corrupted

Notes

The duration values are always in increments of 5 centiseconds. The default duration is 6.

To read/write the bell channel use OS_Byte 211. To read/write the volume use OS_Byte 212. To read/write the frequency use OS_Byte 213.

[RISC OS Open source page](#)

OS_Byte 216

SWI &06 · OS_Byte

The purpose of this call is to read or write the count of characters left in the currently active function key definition.

Before calling

R0 216 (&D8)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous length

R2 Paged mode line count (see OS_Byte 217)

Notes

If this count is non-zero then OS_ReadC is reading from a function key definition rather than the current input stream.

Writing a 0 length will cancel the current function key expansion – the effect of writing any other value is indeterminate.

[RISC OS Open source page](#)

OS_Byte 217

SWI &06 · OS_Byte

The purpose of this call is to read or write the count of lines displayed since the last halt when in paged mode? (VDU 14).

Before calling

R0 217 (&D9)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous count

R2 Corrupted

Notes

This call reads or sets the count of lines displayed since the last page halt when in. Usually this call is used to set the count to 0, before requesting user input – OS_ReadLine makes this call automatically.

[RISC OS Open source page](#)

OS_Byte 218

SWI &06 · OS_Byte

The purpose of this call is to read or write the count of bytes in the VDU queue.

Before calling

R0 218 (&DA)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous count

R2 Corrupted

Notes

The value of the count is -(bytes needed to complete VDU sequence) held in 1 byte, so if two bytes are needed to complete the current sequence then count will be &FE.

Usually this call is used to set the count to 0, which abandons any current VDU sequence. This stops error messages being garbled by a partially completed VDU sequence.

[RISC OS Open source page](#)

OS_Byte 219

SWI &06 · OS_Byte

The purpose of this call is to read or write the character that the tab key produces.

Before calling

R0 219 (&DB)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous value

R2 Corrupted

Notes

If $128 \leq \text{value} \leq 255$ then the character inserted in the buffer is modified by shift and control:

Key / Modification

Shift – Modification: Value EOR &10

Control – Modification: Value EOR &20

[RISC OS Open source page](#)

OS_Byte 220

SWI &06 · OS_Byte

The purpose of this call is to read or write the character that causes an escape condition when read from the input stream.

Before calling

R0 220 (&DC)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous value

R2 Corrupted

Notes

The default is 27 (generated by the escape key).

[RISC OS Open source page](#)

OS_Byte 221 - 228

SWI &06 · OS_Byte

The purpose of this call is to read or write the interpretation of input values &80-&FF, which includes codes generated by the function keys.

Before calling

R0 221 - 228 (&DD - &E5)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous value

R2 Corrupted

Meaning of value passed by R0

221 · &C0 - &CF – Default Value: 1 · Effect: Function keys (Print, F1-F9, cursor keys and copy if acting as function keys)

222 · &D0 - &FF – Default Value: &80 · Effect: Return buffer code - 16

223 · &E0 - &FF – Default Value: &90 · Effect: Return buffer code - 16

224 · &F0 - &FF – Default Value: 0 · Effect: Discard

225 · &80 - &8F – Default Value: 1 · Effect: Function keys (F10-F12, Insert)

226 · &90 - &9F – Default Value: &D0 · Effect: Return buffer code

227 · &A0 - &AF – Default Value: &E0 · Effect: Return buffer code

228 · &B0 - &BF – Default Value: &F0 · Effect: Return buffer code

Notes

The 128 buffer codes are divided into 8 blocks, and the block affected is chosen by R0. The value generated via R1 and R2 determines the interpretation of buffer codes:

Value / Interpretation

0 – Interpretation: Discard

1 – Interpretation: Expand function key number (code MOD 16)

2 – Interpretation: Insert 0 byte followed by code

3 – 255 – Interpretation: Insert (code MOD 16) + value

[RISC OS Open source page](#)

OS_Byte 229

SWI &06 · OS_Byte

The purpose of this call is to enable or disable the generation of escape conditions, and to read the current setting.

Before calling

R0 229 (&E5)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 State before being overwritten

R2 Escape effects (see OS_Byte 230)

Notes

Escape conditions may be caused by pressing the current escape character or by inserting it into the input buffer with OS_Byte 153.

If the value in R1 passed is zero, which is the default, then escape conditions are enabled. Any non-zero value will disable them. When they're disabled, the current escape character set by OS_Byte 220 will pass through the input stream unaltered.

[RISC OS Open source page](#)

OS_Byte 230

SWI &06 · OS_Byte

The purpose of this call is to enable or disable the generation of escape conditions, and to read the current setting.

Before calling

R0 230 (&E6)

R1 EOR value

R2 AND value

On return**R0** Preserved**R1** State before being overwritten**R2** Corrupted**Notes**

This call reads or sets the escape effects. A non-zero state disables escape effects. A state of 0 (the default) enables escape effects – when escape is acknowledged (see OS_Byte 126).

- Active output buffers are flushed – see OS_Byte 21
- Current *Exec file closed – see OS_Byte 198
- VDU paged line count cleared – see OS_Byte 217
- Terminates any sound being produced

[RISC OS Open source page](#)

OS_Byte 236

SWI &06 · OS_Byte

The purpose of this call is to read or write the character destination status.

Before calling**R0** 236 (&EC)**R1** EOR value**R2** AND value**On return****R0** Preserved**R1** Previous state**R2** Cursor key status (see OS_Byte 237)**Notes**

The state may also be written with OS_Byte 3.

The cursor key status can be read or written to by using OS_Byte 237.

[RISC OS Open source page](#)

OS_Byte 237

SWI &06 · OS_Byte

The purpose of this call is to read or write the cursor key status.

Before calling**R0** 237 (&ED)**R1** EOR value**R2** AND value

On return**R0** Preserved**R1** Previous state**R2** Numeric keypad interpretation (see OS_Byte 238)**Notes**

The state may also be written with OS_Byte 4.

[RISC OS Open source page](#)

OS_Byte 238

SWI &06 · OS_Byte

The purpose of this call is to read or set the numeric keypad interpretation.

Before calling**R0** 238 (&EE)**R1** EOR value**R2** AND value**On return****R0** Preserved**R1** Previous base number**R2** Corrupted**Notes**

Keypad keys generate offsets from this base value, which with the default base number of 48 give the key presses shown:

- 0 · +0** – Key code Generated: 0 · Offset with Num Lock: +157 · Key Code Generated: Insert
- 1 · +1** – Key code Generated: 1 · Offset with Num Lock: +91 · Key Code Generated: End/Copy
- 2 · +2** – Key code Generated: 2 · Offset with Num Lock: +94 · Key Code Generated: Down
- 3 · +3** – Key code Generated: 3 · Offset with Num Lock: +110 · Key Code Generated: Page Down
- 4 · +4** – Key code Generated: 4 · Offset with Num Lock: +92 · Key Code Generated: Left
- 5 · +5** – Key code Generated: 5 · Offset with Num Lock: (ignored)
- 6 · +6** – Key code Generated: 6 · Offset with Num Lock: +93 · Key Code Generated: Right
- 7 · +7** – Key code Generated: 7 · Offset with Num Lock: -18 · Key Code Generated: Home
- 8 · +8** – Key code Generated: 8 · Offset with Num Lock: +95 · Key Code Generated: Up
- 9 · +9** – Key code Generated: 9 · Offset with Num Lock: +111 · Key Code Generated: Page Up
- . · -2** – Key code Generated: . · Offset with Num Lock: +79 · Key Code Generated: Delete
- / · -1** – Key code Generated: / · Offset with Num Lock: unchanged
- * · -6** – Key code Generated: * · Offset with Num Lock: unchanged
- # · -13** – Key code Generated: # · Offset with Num Lock: unchanged
- · -3** – Key code Generated: - · Offset with Num Lock: unchanged
- + · -5** – Key code Generated: + · Offset with Num Lock: unchanged
- Enter · -35** – Key code Generated: Return · Offset with Num Lock: unchanged

[RISC OS Open source page](#)

OS_Byte 240

SWI &06 · OS_Byte

The purpose of this call is to write the country number used by the international module.

Before calling

R0 240 (&F0)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Country number

R2 User flag (see OS_Byte 241)

Notes

This value can be written (and read) with OS_Byte 70.

[RISC OS Open source page](#)

OS_Byte 241

SWI &06 · OS_Byte

The purpose of this call is to alter a location that is unused by the OS, which can be used to pass results between applications.

Before calling

R0 241 (&F1)

R1 EOR value

R1 AND value

On return

R0 Preserved

R1 Value before being overwritten

R2 Corrupted

Notes

The location may also be written with OS_Byte 1. System variables now provide a much better way of passing results.

[RISC OS Open source page](#)

OS_Byte 242

SWI &06 · OS_Byte

The purpose of this call is to read the serial baud rate.

Before calling

R0 242 (&F2)

R1 0

R2 255

On return

R0 Preserved

R1 Baud rates

R2 Timer switch (see OS_Byte 243)

Notes

This call is deprecated. You should use OS_SerialOp 5 and OS_SerialOp 6 instead.

The baud rates returned are in a backwards compatible format.

[RISC OS Open source page](#)

OS_Byte 243

SWI &06 · OS_Byte

The purpose of this call is to read the timer switch state.

Before calling

R0 243 (&F3)

R1 0

R2 255

On return

R0 Preserved

R1 Switch state

R2 Corrupted

Notes

This call is deprecated.

This call is obsolete – it reads which of the two copies of the centisecond clock is read/written to by OS_Word 3 and OS_Word 4, and which is updated on the 100Hz interrupt. In effect it provides a location that changes 100 times per second.

[RISC OS Open source page](#)

OS_Byte 245

SWI &06 · OS_Byte

The purpose of this call is to read or set the path to use for printer output.

Before calling

R0 245 (&F5)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Value before being overwritten

R2 Corrupted

Notes

The printer type may also be written with OS_Byte 5. The default type is set held within OS_Byte CMOS 15.

This call does not wait for the printer buffer to empty first. Because of this, it does not enable interrupts, and so may be used to read the printer type from within an interrupt routine.

[RISC OS Open source page](#)

OS_Byte 246

SWI &06 · OS_Byte

The purpose of this call is to read or set the printer ignore character.

Before calling

R0 246 (&F6)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Value before being overwritten

R2 Corrupted

Notes

The printer ignore character may also be set using OS_Byte 6.

To unset the ignore character (NoIgnore) you cannot use this call, you need to use OS_Byte 182.

The default ignore character is stored in OS_Byte CMOS 14?, the NoIgnore state in OS_Byte CMOS 15 and can be set with *Configure Ignore.

[RISC OS Open source page](#)

OS_Byte 247

SWI &06 · OS_Byte

The purpose of this call is to read and change the result of pressing Break.

Before calling

R0 247 (&F7)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Value before being overwritten

R2 Corrupted

Notes

The value byte alters Break and modifiers of it as follows:

Bits / Key combination

0,1 – Key combination: Break

2,3 – Key combination: Shift Break

4,5 – Key combination: Ctrl Break

6,7 – Key combination: Ctrl Shift Break

Each two bit numbers can mean one of these:

Value / Meaning

00 – Meaning: Act as reset

01 – Meaning: Act as escape key

10 – Meaning: No effect

11 – Meaning: Undefined

The default is %00000001, so Break causes an escape condition, together with Shift and/or Ctrl it acts like the reset button.

Please note that this SWI also alters the CMOS when writing a new status.

[RISC OS Open source page](#)

OS_Byte 25

SWI &06 · OS_Byte

The purpose of this call is to reset the definitions for a group of characters in the VDU font.

Before calling

R0 25 (&19)

R1 Group to restore

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

Notes

This call restores the characters eventually redefined by the VDU 23,32-355 command in the range of 32-255. Which characters are to be reset is given in R1.

To reset the entire VDU font, use OS_Byte 20.

[RISC OS Open source page](#)

OS_Byte 250

SWI &06 · OS_Byte

The purpose of this call is to read the screen bank used by the display, as set by OS_Byte 112

Before calling

R0 250 (&FA)

R1 0

R2 255

On return

R0 Preserved

R1 VDU driver screen bank

R2 Display screen bank (see OS_Byte 251)

Notes

This call should not be used to write the value – doing so would only change the value in memory, without affecting the hardware.

[RISC OS Open source page](#)

OS_Byte 251

SWI &06 · OS_Byte

The purpose of this call is to read the screen bank used by the display, as set by OS_Byte 113.

Before calling

R0 251 (&FB)

R1 0

R2 255

On return

R0 Preserved

R1 Display screen bank

R2 Corrupted

Notes

This call should not be used to write the value – doing so would only change the value in memory, without affecting the hardware.

[RISC OS Open source page](#)

OS_Byte 253

SWI &06 · OS_Byte

The purpose of this call is to read the last reset type that occurred.

Before calling

R0 253 (&FD)

R1 0

R2 255

On return

R0 Preserved

R1 Break type

R2 Effect of shift on keypad (see OS_Byte 254)

[RISC OS Open source page](#)

OS_Byte 254

SWI &06 · OS_Byte

The purpose of this call is to enable or disable the effect of Shift and Ctrl on the numeric keypad or to read the current state.

Before calling

R0 254 (&FE)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Value before being overwritten

R2 Corrupted

Notes

These keys may modify the code just before it is inserted into the input buffer.

If R1 is zero, then Shift and Ctrl are enabled. Any non-zero value will disable them. This is the default.

If Shift/Ctrl are enabled, then the following action occurs depending on the value generated by a keypad key:

* key value $\geq$ &80 · Shift exclusive ORs the value with &10 Ctrl exclusive ORs the value with &20

* key value $<$ &80 · Shift and Ctrl still have no effect

If NumLock is off, then a *Fx 254 0 makes the keypad act exactly like the cursor keys, including Shift, Ctrl or both.

The base value used for numeric keypad interpretation can be read and written with OS_Byte 238.

[RISC OS Open source page](#)

OS_Byte 255

SWI &06 · OS_Byte

The purpose of this call is to read or set the current auto boot flag in OS_Byte CMOS Settings.

Before calling

R0 255 (&FF)

R1 EOR value

R2 AND value

On return

R0 Preserved

R1 Previous state

R2 Corrupted

Notes

This call is deprecated. You should use OS_FSControl 10 instead.

Hum. Yes. This is what the PRM says, and it's utter garbage.

This call reads or sets the soft copy, which defaults to the contents of the CMOS ram. But you can change it. And OS_FSControl 10 sets the boot option, which is something else (even if machine will try to boot, if the boot option on the disc is off, nothing happens.) OS_FSControl 10 cannot read anything. It's write only.

[RISC OS Open source page](#)

OS_Byte 3

SWI &06 · OS_Byte

The purpose of this call is to select the devices to which all subsequent output from OS_WriteC (and related SWIs) should be sent.

Before calling

R0 3

R1 Bit mask for output streams

On return

R0 Preserved

R1 Value before being overwritten

R2 Corrupted

Notes

The state of this variable can be read and written with OS_Byte 236.

Enabling the serial device and printer with this OS_Byte and setting the printer to the serial device with OS_Byte 5 causes characters to be placed in both serial and printer buffers, which both are then fed to the serial port.

[RISC OS Open source page](#)

OS_Byte 4

SWI &06 · OS_Byte

The purpose of this call is to alter the effect of the four arrow keys and the End key.

Before calling

R0 4

R1 New state

On return

R0 Preserved

R1 Old state

R2 Corrupted

Notes

The value of R1 determines their state.

OS_Byte 237 may be used to write and read this state.

The 'End' key was also known as the 'Copy' key on Acorn branded keyboards.

[RISC OS Open source page](#)

OS_Byte 5

SWI &06 · OS_Byte

The purpose of this call is to determine the path to use for printer output.

Before calling

R0 5

R1 Printer driver type

On return

R0 Preserved

R1 Previous printer driver type

R2 Corrupted

Notes

The type may be read (and written) with OS_Byte 245. Values in the range 0-7 can be set as the default type in OS_Byte CMOS 15 and *Configure Print.

If a serial printer is in use, setting bit 0 in OS_Byte 3 will mean that the characters are printed twice.

[RISC OS Open source page](#)

OS_Byte 6

SWI &06 · OS_Byte

The purpose of this call is to write the printer ignore character

Before calling

R0 6

R1 ASCII code of ignore character

On return

R0 Preserved

R1 Previous ignore character

R2 Corrupted

Notes

It may be read (and written) with OS_Byte 246. To unset the ignore character (Nolgnore) you cannot use this call, you need to use OS_Byte 182. The state can also be changed by *Ignore. The settings are saved to the CMOS RAM, to make them persistent.

[RISC OS Open source page](#)

OS_Byte 7

SWI &06 · OS_Byte

The purpose of this call is to write the RS232 receive rate.

Before calling

R0 7

R1 Baud rate code

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

Notes

The use of this call has been deprecated. Please use OS_SerialOp 5 instead.

[RISC OS Open source page](#)

OS_Byte 70

SWI &06 · OS_Byte

The purpose of this call is to read or write the country number used by the International module.

Before calling

R0 70 (&46)

R1 Country number, or 127 to read

On return

R0 Preserved

R1 0 if invalid country number passed country number before being overwritten

R2 Corrupted

Notes

This value can also be read with OS_Byte 240.

[RISC OS Open source page](#)

OS_Byte 71

SWI &06 · OS_Byte

The purpose of this call is to read or write the alphabet or keyboard layout number used by the international module.

Before calling

R0 71 (&47)

R1 Value may be one of the following:

0 - 126 Set alphabet number

127 Read alphabet number

128 - 254 Set keyboard layout number

255 Read keyboard layout number

On return

R0 Preserved

R1 0 if invalid alphabet or keyboard number passed, or country/keyboard number before being overwritten

R2 Corrupted

Notes

Although keyboard layout numbers are passed in with 128 added, they are returned as the actual value.

Keyboard layout numbers have a 1:1 mapping with country/territory numbers.

[RISC OS Open source page](#)

OS_Byte 8

SWI &06 · OS_Byte

The purpose of this call is to write the RS232 transmit rate.

Before calling

R0 8

R1 Baud rate code

On return

R0 Preserved

R1 Corrupted

R2 Corrupted

Notes

The use of this call has been deprecated. Please use OS_SerialOp 6 instead.

The default rate can be set using *Configure Baud

[RISC OS Open source page](#)

OS_Byte 9

SWI &06 · OS_Byte

The purpose of this call is to set the duration of the first flash colour in frames-per-second.

Before calling

R0 9

R1 New duration (in video frames, or 0 for 'forever')

On return

R0 Preserved

R1 Previous duration

R2 Corrupted

Notes

A duration of 0 means display the first colour constantly. The current duration may be read with OS_Byte 195, and may also be set with VDU 23,9?.

The second flash colour duration can be set using OS_Byte 10.

[RISC OS Open source page](#)

OS_CallAfter

SWI &3B

The purpose of this call is to call a specified memory address after a delay, in centi-seconds.

Before calling

R0 Delay in centi-seconds

R1 Address to call

R2 Value of R12 to call code with

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

The routine will be called on the (R0+1)th clock tick to be seen once the routine has been registered. E.g. if R0 is 10 then it's guaranteed that a full 10 centiseconds will have passed before the routine is called. The minimum supported R0 value is 1.

A routine called by this mechanism is called in SVC mode with interrupts disabled, must preserve ALL registers and return by MOV PC, R14. The routine can enable interrupts, but must take care to restore the previous state on exit.

To cancel the settings of this call, use OS_RemoveTickerEvent.

[RISC OS Open source page](#)

OS_CallASWI

SWI &6F

The purpose of this call, is to call a run-time determined SWI.

Before calling

R0-R9 As required for target SWI

R10 Target SWI number

On return

R0-R9 As defined for target SWI

R10 Preserved

Notes

This SWI allows a target SWI number to be determined at run time, and passed in a register. This removes the need for a common idiom of dynamic code, in language library SWI veneers for example. In an APCS-R library, use OS_CallASWIR12.

Note that this call is merely an alias for calling the target SWI. It has no entry/exit conditions of its own, except for the special use of R10. To call a target SWI with X bit set, use the X form of the SWI number in R10; there is no distinction between OS_CallASWI and XOS_CallASWI.

You cannot use this SWI to call itself, that is OS_CallASWI or OS_CallASWIR12, since there is no defined final target SWI in this case.

You cannot usefully call OS_BreakPt or OS_CallAVector, as it will corrupt the processor flags before entering the target SWI.

For future compatibility, you should always use this SWI in preference to any local construction for calling a SWI by number. For compatibility with older versions of RISC OS, a new CallASWI module will be made available.

Note that this SWI calling mechanism is almost certainly faster than any other alternative implementation, including the original `_kernel_swi` and `_swix` code contained in older versions of the SharedCLibrary. The new SharedCLibrary now simply uses `OS_CallASWIR12` for `_kernel_swi` and `_swix`.

`OS_CallASWI` cannot be called from BASIC as BASIC only passes registers R0-R7 via its SYS instruction. It would not be useful anyway.

The CallASWI module provides this functionality for versions of RISC OS 3.70 or earlier.

[RISC OS Open source page](#)

OS_CallASWIR12

SWI &71

The purpose of this call, is to call a run-time determined SWI.

Before calling

R0-R9 As required for target SWI

R12 Target SWI number

On return

R0-R9 As defined for target SWI

R12 Preserved

Notes

This call is identical to `OS_CallASWI`, except that it uses R12 to specify the target SWI. This may be more convenient in some environments. In particular under APCS-R, R10 is the stack limit pointer, which must be preserved at all times.

If a SWI called using `OS_CallASWI` were to abort or generate an error the run-time library would usually examine R10 and decide that it had no stack to handle the abort or error. Therefore APCS-R libraries must use `OS_CallASWIR12` (R12 being a scratch register under APCS-R).

The CallASWI module provides this functionality for versions of RISC OS 3.70 or earlier.

[RISC OS Open source page](#)

OS_CallAVector

SWI &34

The purpose of this call, is to call a vector directly.

Before calling

R0-R8 Vector routine parameters

R9 Vector number

V and C flags in R15 = flags to pass to vector

On return

Dependent on vector called

Notes

This calls the vector given in R9. This is used for calling vectored routines which don't have any other entry point such as some calls to RemV or CnpV.

Don't use this to call ByteV vector or some others, as the vector handler expects entry conditions you may not provide.

[RISC OS Open source page](#)

OS_CallBack

SWI &15

OS_CallBack sets up the address of the CallBack handler and its register save block.

Before calling

R0 Address of the register save block, 0 to read

R1 Address of the callback handler, 0 to read

On return

R0 Address of previous register save block

R1 Address of previous callback handler

Notes

The register save block must be word-aligned and 16 words long.

This call is deprecated and should not be used. Please use OS_ChangeEnvironment instead. Internally, this SWI just calls OS_ChangeEnvironment

This sets up the address of the CallBack handler and the register save block, zero for either value meaning no change - hence you can read the current value.

Once a CallBack handler has been installed, you can request callbacks via OS_SetCallBack.

For information on callbacks refer to CallBack Handler.

[RISC OS Open source page](#)

OS_CallEvery

SWI &3C

The purpose of this call is to call a specified address every R0+1 centiseconds.

Before calling

R0 Time to call in centi-seconds

R1 Pointer to address to call

R2 Value of R12 to be called with

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

Sets a CallBack routine to be triggered every time a specified delay has elapsed.

For more details on how this call functions, please see OS_AddCallBack.

To cancel the settings of this call, use OS_RemoveTickerEvent. Note that it is not possible to remove a ticker event which is currently executing.

Because the minimum supported R0 value is 1, it is not possible to use this call to register a routine which is called on every tick. Software which wishes to do so should instead use OS_Claim to register directly with TickerV.

[RISC OS Open source page](#)

OS_ChangedBox

SWI &5A

The purpose of this call is to determine which areas of the screen have changed.

Before calling

R0 0 = Disable changed box calculations

1 = Enable changed box calculations

2 = Reset changed box to null rectangle

-1 = Read changed box information

On return

R0 Previous enable state in bit 0

R1 Pointer to a fixed block of 5 words

Notes

The data pointed to by R1 on exit at offsets 4 - 16 are only valid if the change box calculations were in an enabled state immediately after the call; otherwise they are undefined.

This call checks which areas of the screen were changed by VDU commands and returns the smallest rectangle which covers all changed pixels.

[RISC OS Open source page](#)

OS_ChangeDynamicArea

SWI &2A

The purpose of this call is change the size of a dynamic area.

Before calling

R0 Area to alter

R1 Amount to move in bytes (signed integer)

On return

R0 Preserved

R1 Amount the area has changed in bytes (unsigned)

Notes

Let you enlarge (R1 positive) or shrink (R1 negative) the selected area.

If you need to release as much space as possible from an area, try to reduce its size by 16 MBytes.

Note, that normally this cannot be used while the application work area is being used; main purpose is for the OS, modules and utilities.

This SWI issues OS_UpCall 257 (moving memory); if the upcall is claimed when an application is running in the application workspace then the memory movement is allowed.

[RISC OS Open source page](#)

OS_ChangeEnvironment

SWI &40

The purpose of this call is to install a handler.

Before calling

R0 Handler number

R1 New address, or 0 to read

R2 Value of R12 when code is called, or 0 to read

R3 Pointer to buffer if appropriate, or 0 to read

On return

R0 Preserved

R1 Previous address

R2 Previous R12

R3 Previous buffer pointer

Notes

This is a single routine which performs the actions of OS_Control, OS_SetEnv, OS_CallBack, OS_BreakCtrl and OS_UnusedSWI. In fact all of those routines use this call and are only kept for backwards compatibility, so you should use this call in preference.

Note, OS_ChangeEnvironment vectors through ChangeEnvironmentV. A routine linked into this vector can stop the change from happening by setting R1 to zero and passing the call on.

If you set up any handlers, you must provide Error, Exit and UpCall handlers so that you can restore all handlers on any kind of program exit (even X SWIs can generate serious errors).

Handlers are specific to the current task and are swapped automatically by the Wimp when it switches tasks.

Consult the documentation for each individual handler to determine what extra buffer pointer (R3), if any, is required.

[RISC OS Open source page](#)

OS_ChangeRedirection

SWI &5E

The purpose of this call is to read or write the file handles used by OS_CLI to redirect input/output.

Before calling

R0 New file handle for input

0 = not redirected

-1 = read

R1 New file handle for output

0 = not redirected

-1 = read

On return

R0 Old file handle for input (or 0 for standard input)

R1 Old file handle for output (or 0 for standard output)

Notes

This call is mainly provided for the use of the TaskManager, but you may also find the call useful.

Calling OS_Exit automatically removes any redirection (and closes the associated file handles) before control passes to the exit handler.

[RISC OS Open source page](#)

OS_CheckModeValid

SWI &3F

The purpose of this call is to check if it is possible to change to a specified screen mode.

Before calling

R0 Screen mode to check (either mode number or Mode Specifier Block)

On return

If C flag unset, then Screen mode is valid

R0 Preserved

If C flag set, then Screen mode is invalid

R0 -1 = Screen mode is non-existent

-2 = Not enough memory

R1 Alternative screen mode to use

-2 = Unable to select alternative screen mode

Notes

This finds out whether you may change to a given mode or whether it is invalid, for example if the mode isn't supported by the current monitor type or if there is no suitable MDF entry for a Mode Specifier Block.

If the mode is invalid then the returned value in R0 will indicate the reason why, and R1 may contain a suggested alternative mode.

[RISC OS Open source page](#)

OS_Claim

SWI &1F

The purpose of this call is to add a routine to the list of those that claim a vector.

Before calling

R0 Vector number

R1 Pointer to address of claiming routine

R2 Value to be passed in R12 when the routine is called

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

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. If you wish to add multiple 'identical' routines use OS_AddToVector.

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.

[RISC OS Open source page](#)

OS_ClaimDeviceVector

SWI &4B

The purpose of this call is to claim a device vector.

Before calling

R0 Device number and flag in bit 31

R1 Address of device driver routine

R2 Value to be passed in R12 when driver is called

R3 Address of interrupt status if R0 = module "IRQ" or "FIQ as IRQ" on entry

R4 Interrupt mask to use, if R0 = module "IRQ" or "FIQ as IRQ" on entry

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

Notes

This installs the device driver. If the same driver has already been installed on the vector then the old copy is removed from the vector. This does not enable interrupts from the device. The previous driver is added to the list of earlier claimants. Your driver is called if the interrupt controller receives an interrupt from the appropriate device and your driver was the last to claim the vector.

For IO expansion cards (device numbers 8 or 13) there may be multiple cards all sharing the same interrupt line (and therefore device number) the values in R3 and R4 will be used to determine the true owner. A byte read will be performed of the interrupt status and your driver will be called if (status AND R4) is non zero. From RISC OS 5.00, bits 8-15 of R4 can specify an EOR mask to permit active low interrupts to be supported, ie. when ((status EOR R4[15:8]) AND R4[7:0]) is non zero.

Under RISC OS 5, your driver routine will be called as follows:

- The ARM is in IRQ32 mode and interrupts are disabled
- R0 is the device number (with no flags in the high bits)
- R12 has R2's value when claiming the vector
- R13 points to the IRQ stack. The word pointed to by R13 is a return address for claiming the vector.
- R14 contains a return address for passing on the call to the next device driver.

On exit:

- Processor mode and interrupt state must be preserved
- R1-R3, R12, and R14 may be corrupted by the call.
- R0 must be preserved if you are passing on an unhandled shared IRQ. If you are claiming the IRQ, it can be corrupted.
- For non-shared IRQs, returning via R14 is functionally equivalent to returning via the stacked return address. This also means that R0 doesn't need to be preserved when returning via R14.

If the interrupt is caused by your device, your routine should:

- Service the interrupt
- Stop the device from generating interrupts, when necessary
- Call HAL_IRQClear to ensure the IRQ controller resumes its processing
- Return to the kernel by popping the pre-prepared return address from the stack

If the interrupt is not caused by your device (i.e. it's a different device which is on the same shared IRQ line as your device), your routine should return to the kernel by MOV PC,R14 or similar. The kernel will then call the next routine on the device vector chain.

Note that if an interrupt is received for a device which has no drivers registered, or (for a shared IRQ line) if the end of the chain is reached without any driver handling the IRQ, the kernel will automatically disable the IRQ via HAL_IRQDisable. This behaviour is necessary so that devices which request an interrupt when no driver is loaded will not hang the system. However it does mean that a buggy device driver on a shared IRQ line may accidentally cause that IRQ line to be disabled for all drivers on that line – be aware of this when developing/debugging your drivers.

You should ensure:

- A very high execution speed
- If your routine takes more than 100µs, you should re-enable interrupts, if previously disabled, to ensure other devices' interrupts can be serviced – e.g. disc ops. With interrupts enabled your routine must cope with being re-entered.
- Saving R14_svc to a stack before calling SWIs and reloading it after it returns to prevent double use of R14_svc
- Strictly avoiding the use of non re-entrant SWIs, because the supervisor stack would become corrupted if used
- Clearing the interrupt flags when finishing
- You cope with SWI-error handling. You must use XSWIs. As there is no-one to pass the error on to, you must either handle it yourself or store an error indicator, so that the next SWI call (or the current, if already threaded) to this module will generate an error.

[RISC OS Open source page](#)

OS_ClaimProcessorVector

SWI &69

The purpose of this call is to claim a processor vector.

Before calling

R0 bits 0-7: Vector number

bit 8: 0 to release, or 1 to claim

bits 9-31: reserved

R1 Address of replacement routine

R2 If releasing: Address of your routine

On return

R0 Preserved

R1 If claiming, address of original routine, else preserved

R2 Preserved

Notes

This call is used to allow low-level code to claim the ARM's hardware vectors, which are called when interrupts and aborts occur.

The address of your routine must be provided on release to ensure claims and releases happen in the correct order. If the address currently on the vector differs from that in R2, an error is returned.

The FIQ vector cannot be claimed through this call. Instead, `Service_ClaimFIQ` or `Service_ClaimFIQinBackground` must be used.

Routines installed via `OS_ClaimProcessorVector` are entered directly from the corresponding hardware vector. This means that:

- On 26bit OS versions, they will be entered in a 32bit CPU mode.
- Data and prefetch abort handlers need to be careful about accessing application space; if lazy task swapping is enabled, this may trigger a recursive abort, potentially leaving the machine stuck in an abort loop.
- On ARMv6 and newer, care is needed to ensure the "exclusive monitor" is left in a safe state. See the ARM ARM or the Hardware Vectors documentation for more detail.

This SWI is only available on RISC OS 3.5 and later. To claim a processor vector on earlier OS versions, you must directly patch the processor vector table at &0. If your code needs to pass on to the previous claimant, then care will be needed to make sure you decode the old handler instruction correctly (typically it will either be a branch instruction or a PC-relative LDR of the PC)

[RISC OS Open source page](#)

OS_ClaimScreenMemory

SWI &41

The purpose of this call is to claim spare memory allocated as screen memory.

Before calling

R0 0 for claim, or 1 for release

R1 Bytes required

On return

If C flag set, memory not claimed, if clear then memory claimed

R0 Preserved

R1 Bytes available

R2 Start address

Notes

Can only be claimed if no other bank than bank 1 has been used.

While claimed, you must now cause the screen to scroll.

[RISC OS Open source page](#)

OS_CLI

SWI &05

The purpose of this call is to process a command, as if typed in the CLI.

Before calling

R0 Pointer to string terminated by Null, Linefeed or Return

On return

R0 Preserved

Notes

When it is called, it performs the following actions:

- Checks the stack space, if there's not enough free memory to establish the stack the error 'No room for supervisor stack' is returned.
- Checks the command line length. Prior to RISC OS 4, a CLI-command has to be less than 257 bytes long, including the terminator. If it's longer, the call is ignored without returning an error. Later OS versions increase the limit to 1024 bytes including terminator.
- Then execute the command as any other *Command typed in CLI

[RISC OS Open source page](#)

OS_Confirm

SWI &59

The purpose of this call is to return a Yes or No answer.

Before calling

--

On return

R0 Result

Notes

This call should not be used within the Desktop environment, and should only be used by applications outside the desktop, if ever.

C flag is set if escape pressed Z flag is set if result was in any way 'Yes'.

This call asks the user for confirmation of an action by changing the pointer and waiting until a key or mouse button is pressed.

The values returned are:

- Select button returns the value 121 (character y)
- Menu and Adjust buttons return value 110 (character n)
- Keypresses return the lower-case ascii value.

[RISC OS Open source page](#)

OS_Control

SWI &0F

The purpose of this call is to read or write address handlers.

Before calling

R0 Address of error handler, or 0 to read

R1 Pointer to buffer for the error handler, or 0 to read

R2 Address of escape state change handler, or 0 to read

R3 Address of event handler, or 0 to read

On return

R0 Previous error handler address

R1 Previous buffer address

R2 Previous escape routine address

R3 Previous event handler address

Notes

This call is deprecated, and should not be used. OS_ChangeEnvironment should be used instead.

[RISC OS Open source page](#)

OS_ConvertBinary1

SWI &DD

The purpose of this call is to convert a value into a string of 8 binary characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output string is padded with leading zeros, so it has a fixed length of 8 characters.

[RISC OS Open source page](#)

OS_ConvertBinary2

SWI &DE

The purpose of this call is to convert a value into a string of 16 binary characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output string is padded with leading zeros, so it has a fixed length of 16 characters.

[RISC OS Open source page](#)

OS_ConvertBinary3

SWI &DF

The purpose of this call is to convert a value into a string of 24 binary characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output string is padded with leading zeros, so it has a fixed length of 24 characters.

[RISC OS Open source page](#)

OS_ConvertBinary4

SWI &E0

The purpose of this call is to convert a value into a string of 32 binary characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output string is padded with leading zeros, so it has a fixed length of 32 characters.

[RISC OS Open source page](#)

OS_ConvertCardinal1

SWI &D5

The purpose of this call is to convert a value into an unsigned string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from 0 to 255.

The string is not padded with zeros, so is of variable length.

[RISC OS Open source page](#)

OS_ConvertCardinal2

SWI &D6

The purpose of this call is to convert a value into an unsigned string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from 0 to 65,535. The commas are not output.

The string is not padded with zeros, so is of variable length.

Every three digits from the right, a space is inserted. e.g. 65 535.

[RISC OS Open source page](#)

OS_ConvertCardinal3

SWI &D7

The purpose of this call is to convert a value into an unsigned string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from 0 to 16,777,215. The commas are not output.

The string is not padded with zeros, so is of variable length.

Every three digits from the right, a space is inserted. e.g. 16 777 215.

[RISC OS Open source page](#)

OS_ConvertCardinal4

SWI &D8

The purpose of this call is to convert a value into an unsigned string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from 0 to 4,294,967,295 (10 characters in length). The commas are not output.

The string is not padded with zeros, so is of variable length.

Every three digits from the right, a space is inserted. e.g. 4 294 967 295.

[RISC OS Open source page](#)

OS_ConvertDateAndTime

SWI &C1

The purpose of this call is to convert a five-byte value representing centiseconds since 1900, into a string.

Before calling

- R0** Pointer to 5-byte time block
- R1** Pointer to buffer for resulting string
- R2** Size of buffer
- R3** Pointer to format string

On return

- R0** Pointer to buffer (R1 on entry)
- R1** Pointer to terminator
- R2** Free bytes in buffer
- R3** Preserved

[RISC OS Open source page](#)

OS_ConvertFileSize

SWI &EC

The purpose of this call is to convert a value to an amount of space in the form x bytes, x Kilobytes, or x Megabytes.

Before calling

- R0** File size
- R1** Pointer to buffer for resulting string
- R2** Size of buffer

On return

- R0** Pointer to buffer (R1 on entry)
- R1** Pointer to terminator
- R2** Free bytes in buffer

[RISC OS Open source page](#)

OS_ConvertFixedFileSize

SWI &EB

The purpose of this call is to convert a value to an amount of space in the form x bytes, x Kilobytes, or x Megabytes.

Before calling

R0 File size

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output string is a fixed length, with the start padded with spaces. The call OS_ConvertFileSize performs the identical operation as this call, except the output is variable in length.

[RISC OS Open source page](#)

OS_ConvertFixedNetStation

SWI &E9

The purpose of this call is to convert the network and station number to a string.

Before calling

R0 Pointer to net and station number block

R1 Pointer to buffer for resulting string

R2 Free bytes in buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output is in the form nnn.sss.

If the network is 0 then:

- The first four characters are spaces
- Leading zeros in the station number are converted to spaces otherwise Leading zeros in the network number are converted to spaces Leading zeros in the station number remain as zeros

[RISC OS Open source page](#)

OS_ConvertHex1

SWI &D0

The purpose of this call is to convert a value into a string of 1 hex character.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The string does not include an ampersand ('&') character, but is padded with leading zeros, so is of fixed length.

[RISC OS Open source page](#)

OS_ConvertHex2

SWI &D1

The purpose of this call is to convert a value into a string of 2 hex characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The string does not include an ampersand ('&') character, but is padded with leading zeros, so is of fixed length.

[RISC OS Open source page](#)

OS_ConvertHex4

SWI &D2

The purpose of this call is to convert a value into a string of 4 hex characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The string does not include an ampersand ('&') character, but is padded with leading zeros, so is of fixed length.

[RISC OS Open source page](#)

OS_ConvertHex6

SWI &D3

The purpose of this call is to convert a value into a string of 6 hex characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The string does not include an ampersand ('&') character, but is padded with leading zeros, so is of fixed length.

[RISC OS Open source page](#)

OS_ConvertHex8

SWI &D4

The purpose of this call is to convert a value into a string of 8 hex characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The string does not include an ampersand ('&') character, but is padded with leading zeros, so is of fixed length.

[RISC OS Open source page](#)

OS_ConvertInteger1

SWI &D9

The purpose of this call is to convert a value into a signed string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from -128 to 127 (4 characters).

If the most significant bit is set (of the number bytes used), the number is taken to be negative, and a leading '-' is produced.

The string is not padded with zeros, so is of variable length.

[RISC OS Open source page](#)

OS_ConvertInteger2

SWI &DA

The purpose of this call is to convert a value into a signed string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from -32,768 to 32,767 (6 characters). The commas are not output.

If the most significant bit is set (of the number bytes used), the number is taken to be negative, and a leading '-' is produced.

The string is not padded with zeros, so is of variable length.

[RISC OS Open source page](#)

OS_ConvertInteger3

SWI &DB

The purpose of this call is to convert a value into a signed string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from -8,388,608 to 8,388,607 (8 characters). The commas are not output.

If the most significant bit is set (of the number bytes used), the number is taken to be negative, and a leading '-' is produced.

The string is not padded with zeros, so is of variable length.

[RISC OS Open source page](#)

OS_ConvertInteger4

SWI &DC

The purpose of this call is to convert a value into a signed string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from -2,147,483,648 to 2,147,483,647 (11 characters). The commas are not output.

If the most significant bit is set (of the number bytes used), the number is taken to be negative, and a leading '-' is produced.

The string is not padded with zeros, so is of variable length.

[RISC OS Open source page](#)

OS_ConvertNetStation

SWI &EA

The purpose of this call is to convert the network and station number to a string.

Before calling

R0 Pointer to net and station number block

R1 Pointer to buffer for resulting string

R2 Free bytes in buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output is in the form nnn.sss, but suppresses zeros and spaces to yield the shortest string – hence if the network is 0 no network number or "." is placed in the buffer.

[RISC OS Open source page](#)

OS_ConvertSpacedCardinal1

SWI &E1

The purpose of this call is to convert a value into an unsigned string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from 0 to 255.

The string is not padded with zeros, so is of variable length.

[RISC OS Open source page](#)

OS_ConvertSpacedCardinal2

SWI &E2

The purpose of this call is to convert a value into an unsigned string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from 0 to 65,535 (6 characters). Commas are not output.

The string is not padded with zeros, so is of variable length.

Every three digits from the right, a space is inserted. e.g. 65 535.

[RISC OS Open source page](#)

OS_ConvertSpacedCardinal3

SWI &E3

The purpose of this call is to convert a value into an unsigned string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from 0 to 16,777,215 (10 characters). Commas are not output.

The string is not padded with zeros, so is of variable length.

Every three digits from the right, a space is inserted. e.g. 16 777 215

[RISC OS Open source page](#)

OS_ConvertSpacedCardinal4

SWI &E4

The purpose of this call is to convert a value into an unsigned string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from 0 to 4,294,967,295 (13 characters). Commas are not output.

The string is not padded with zeros, so is of variable length.

Every three digits from the right, a space is inserted. e.g. 4 294 967 295

[RISC OS Open source page](#)

OS_ConvertSpacedInteger1

SWI &E5

The purpose of this call is to convert a value into a signed string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from -128 to 127 (4 characters).

If the most significant bit is set (of the number bytes used), the number is taken to be negative, and a leading '-' is produced.

The string is not padded with zeros, so is of variable length.

[RISC OS Open source page](#)

OS_ConvertSpacedInteger2

SWI &E6

The purpose of this call is to convert a value into a signed string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from -32,768 to 32,767 (6 characters). Commas are not output.

If the most significant bit is set (of the number bytes used), the number is taken to be negative, and a leading '-' is produced.

The string is not padded with zeros, so is of variable length.

Every three digits from the right, a space is inserted. e.g. 32 767

[RISC OS Open source page](#)

OS_ConvertSpacedInteger3

SWI &E7

The purpose of this call is to convert a value into a signed string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from -8,388,608 to 8,388,607 (10 characters). Commas are not output.

If the most significant bit is set (of the number bytes used), the number is taken to be negative, and a leading '-' is produced.

The string is not padded with zeros, so is of variable length.

Every three digits from the right, a space is inserted. e.g. 8 388 608

[RISC OS Open source page](#)

OS_ConvertSpacedInteger4

SWI &E8

The purpose of this call is to convert a value into a signed string of decimal characters.

Before calling

R0 Value to be converted

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

Notes

The output can be a value from -2,147,483,648 to 2,147,483,647 (14 characters). Commas are not output.

If the most significant bit is set (of the number bytes used), the number is taken to be negative, and a leading '-' is produced.

The string is not padded with zeros, so is of variable length.

Every three digits from the right, a space is inserted. e.g. 2 147 483 647

[RISC OS Open source page](#)

OS_ConvertStandardDateAndTime

SWI &C0

The purpose of this call is to convert a five-byte value representing centiseconds since 1900, into a string.

Before calling

R0 Pointer to 5-byte time block

R1 Pointer to buffer for resulting string

R2 Size of buffer

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to terminator

R2 Free bytes in buffer

The format output is per Sys\$DateFormat.

[RISC OS Open source page](#)

OS_ConvertVariform

SWI &ED

This is a general-purpose call for converting numbers to strings. It is a more flexible and feature-rich version of the existing OS_Convert* SWIs (OS_ConvertInteger4, OS_ConvertHex8, etc.)

Before calling

R0 Pointer to input value (word aligned)

R1 Pointer to output buffer

R2 Buffer length, or negative to read required length

R3 Bytes to use from input value (or nibbles for hex output)

R4 Type to convert to

On return

R0 Pointer to buffer (R1 on entry)

R1 Pointer to null terminator

R2 Bytes free in buffer, or NOT space required if R2 negative on entry

Notes

This call first became available in RISC OS 5.17. It is also available in CallASWI 0.06 and newer.

[RISC OS Open source page](#)

OS_CRC

SWI &5B

The purpose of this call is to calculate the cyclic-redundancy check for a block of data.

Before calling

R0 CRC continuation value, or 0

R1 Pointer to start of block

R2 Pointer to end of block

R3 Increment

On return

R0 CRC value

R1 Preserved

R2 Preserved

R3 Preserved

Notes

The value calculated is only 16 bit.

The SWI adds R3 to R1 until it equals R2, ie if R1 never equals R2 you have an infinite loop

The byte at R2 is not included in the CRC value.

[RISC OS Open source page](#)

OS_DelinkApplication

SWI &4D

The purpose of this call is to remove any vectors that an application is using.

Before calling

R0 Pointer to buffer

R1 Buffer size in bytes

On return

R0 Preserved

R1 Number of bytes left in buffer

Notes

When an application running at &8000 is going to be swapped out, it must remove all vectors that it uses. Otherwise, if the kernel attempted to call the vector it would jump into whatever happened to be at this memory address in whatever application was running at the time – probably with disastrous results.

The buffer is used to store details of the vectors used, so they can be restored afterwards with OS_RelinkApplication.

Each vector requires 12 bytes of storage and the list is terminated by a single byte. If the space left is 0, you are to allocate another buffer and repeat the call; the first buffer is still valid, only the vectors which didn't fit into it are placed into the second buffer. When relinking you must pass all buffers used by this call.

Note that this call will only delink the standard software vectors. Ticker events, hardware/processor vectors, IRQ handlers and environment handlers must be preserved by other means (when running under the Wimp, it will preserve environment handlers for you).

[RISC OS Open source page](#)

OS_DynamicArea

SWI &66

The purpose of this call is to perform operations on user dynamic areas.

Before calling

R0 Reason code

- 0** Creates a new dynamic area
- 1** Removes a previously created dynamic area
- 2** Return information on a dynamic area
- 3** Enumerates dynamic areas
- 4** Renumbers dynamic areas
- 5** Return total free space (bytes)
- 6** Internal use only
- 7** Internal use only
- 8** Set maximum size clamps
- 9** Ensure sparse area is mapped to valid memory
- 10** Release a region of a Sparse area
- 11** Lock area (RISCOS Ltd LockedDAs)
- 12** Unlock area (RISCOS Ltd LockedDAs)
- 13** Resize locked area (RISCOS Ltd LockedDAs)
- 14** Describe heap area (RISCOS Ltd HeapDAs)
- 15** Claim block in heap area (RISCOS Ltd HeapDAs)
- 16** Release block in heap area (RISCOS Ltd HeapDAs)
- 17** Resize block in heap area (RISCOS Ltd HeapDAs)
- 18** Read size of block in heap area (RISCOS Ltd HeapDAs)
- 19** Change domain state of area (RISCOS Ltd DomainDAs)
- 20** Locate area by address
- 21** PMP physical operation
- 22** PMP logical operation
- 23** PMP resize operation
- 24** Return information on a dynamic area/PMP
- 25** Examine PMP pages
- 26** Adjust application space limit
- 27** Return total free space (pages)
- 28** Internal use only

On return

R0 Preserved

Notes

This call was introduced in RISC OS 3.50. Reason codes 6 to 10 were introduced in RISC OS 3.80. Reason codes 11 to 19 were introduced by RISC OS Select, and are not currently supported by RISC OS 5. Reason codes 20 to 25 were introduced by RISC OS 5.23. Reason codes 26 to 28 were introduced by RISC OS 5.27.

[RISC OS Open source page](#)

OS_DynamicArea 0

SWI &66 · OS_DynamicArea

The purpose of this call is to create a new dynamic area.

Before calling

R0 0 (reason code)

R1 -1 (or new area number 128-255; reserved for internal use)

R2 Initial logical size of area, in bytes

R3 -1 (or base of area; reserved for internal use)

R4 Area flags vary by OS; see note

R5 Maximum logical size of area in bytes, or -1 for unlimited (see notes)

R6 Handler routine

R7 Pointer to workspace for handler, or -1 for base of area

R8 Pointer to name of area (limited to 31-chars in 3.80+)

R9 Initial maximum physical size of PMP in pages (if area is a PMP; otherwise, unused)

On return

R1 Allocated area number

R3 Base of area

R5 Maximum logical size of area

R9 Maximum physical size of area (if area is a PMP; otherwise, preserved)

All other registers preserved

Notes

Area numbers allocated are greater than or equal to 256.

The setting R5 as -1 is strongly deprecated due to the problems of address space exhaustion on large memory machines. Also, a maximum size of -1 has no valid meaning for Sparse areas. New code should set the smallest maximum size that is sensible for the use required. Because old code may often use -1, a clamp may be set via OS_DynamicArea 8 in order to limit the address space exhaustion problem.

Area flags in b12-15 vary by OS version – specifically if OS_Byte 129 (OS) returns &AA.

PMPs

When creating a PMP, arguments must be set as follows:

- The “Requires specific pages” flag (bit 8) must be set
- The “Sparse” (bit 10), doubly-mapped (bit 6) and DMA auto-alloc (bit 15) flags must be clear
- The initial logical size (R2) must be zero
- A handler routine (R6) must be provided

When the PMP is created it will have no pages assigned to it.

If necessary, the maximum physical size specified in R9 will be clamped to the total number of pages in the system - therefore you can specify -1 in order to create the largest possible PMP. However this will also require the kernel to allocate a large amount of memory for storing the page list, so to reduce memory wastage it's recommended to dynamically adjust the maximum physical size of the PMP in order to avoid situations where a large PMP only has a small number of pages allocated to it.

The maximum logical size of a PMP will be clamped using the same rules as for regular dynamic areas. If the maximum logical size was zero then the PMP will have no logical space allocated to it and will return with R3 equal to zero.

[RISC OS Open source page](#)

OS_DynamicArea 1

SWI &66 · OS_DynamicArea

The purpose of this call is to remove a previously created dynamic area.

Before calling

R0 1 (reason code)

R1 Area number

On return

R0 Preserved

R1 Preserved

Notes

Before the area is removed, it attempts to shrink it to zero size by calling OS_ChangeDynamicArea internally.

If OS_ChangeDynamicArea returns an error, then the area will be grown back to its original size using OS_ChangeDynamicArea, and this call will return with an error. If OS_ChangeDynamicArea successfully reduced the area to zero size, then it will be removed.

Once the area has been removed Service_DynamicAreaRemove is issued to inform the rest of the system about this change.

An error is returned if the area was not removed for any reason.

Sparse areas shall be released via OS_DynamicArea 10 before removal. If the attempt fails, the area is not removed, but no attempt is made to restore the mapping distribution as before the removal.

[RISC OS Open source page](#)

OS_DynamicArea 10

SWI &66 · OS_DynamicArea

The purpose of this call is to allow a region of a Sparse area to be released as free memory.

Before calling

R0 10 (reason code)

R1 Area number

R2 Base of region to release

R3 Size of region to release

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

The region ($\text{base} \leq x \leq \text{size}-1$) must be entirely within the address range of the Sparse area. There are no restrictions on the distribution of any mapped memory within the region before the call.

Note that although arbitrary alignment of base and size is allowed, memory can only be released with a granularity equal to the page size of the machine (as return by `OS_ReadMemMapInfo`). Fragments below this granularity and not released are not accumulated across calls. Hence, release of sub-page sized regions is not useful.

[RISC OS Open source page](#)

OS_DynamicArea 2

SWI &66 · OS_DynamicArea

The purpose of this call is to return information relating to a dynamic area.

Before calling

R0 2 (reason code)

R1 Area number

On return

R0 Preserved

R1 Preserved

R2 Current size of area, in bytes

R3 Base logical address of area

R4 Area flags

R5 Maximum size of area in bytes

R6 Pointer to dynamic area handler routine, or 0 if no routine

R7 Pointer to workspace for handler

R8 Pointer to name of area

Notes

For doubly-mapped areas, **R3** on exit from this call returns the address of the boundary between the first and second copies of the area, whereas `OS_ReadDynamicArea` returns the start address of the first copy (i.e. the lowest valid address in the area).

For Sparse Dynamic Areas, the current size is calculated from the number of pages that are currently mapped in. Therefore it is not guaranteed to describe the logical extent of the area.

For Physical Memory Pool DAs, the values in **R2** and **R5** will correspond to the physical properties of the dynamic area, not the logical properties. Therefore the values are completely unrelated to the base address returned in **R3** (and if an area has a maximum logical size of zero, the base address will be zero). If necessary the values will be clamped at `2GB-PageSize` in order to try and avoid any issues relating to signed number overflow. To be able to query a PMP without this clamping, or to be able to query the logical size values, `OS_DynamicArea 24` should be used.

[RISC OS Open source page](#)

OS_DynamicArea 20

SWI &66 · OS_DynamicArea

This call can be used to locate the dynamic area or system area that encompasses the given address.

Before calling

R0 20 (reason code)

R1 Logical address

On return

R0 Area type:

0: Dynamic Area

1: System area

Other values: Reserved

R1 Area number

In practice

For dynamic areas, the area number returned in R1 will be the dynamic area number. For system areas, the area number will correspond to one of the area numbers recognised by OS_Memory 16.

Notes

If the address is not located in any area then an error will be returned.

This call only checks that the address is within the minimum and maximum bounds of the area. It does not verify that there is valid memory at that address. To check that there is valid memory at an arbitrary address, use a call such as OS_Memory 24 or OS_ValidateAddress.

[RISC OS Open source page](#)

OS_DynamicArea 21

SWI &66 · OS_DynamicArea

This call is used to claim/release physical memory pages in a Physical Memory Pool.

Before calling

R0 21 (reason code)

R1 Area number

R2 Pointer to page list

R3 Number of entries in page list

On return

R2 Advanced to first entry not processed (or end of list)

R3 Number of entries not processed

Notes

Each entry in the page list is 12 bytes in length, structured as follows:

0 – : PMP page index (From zero to the max physical size of PMP)

1 – : Physical page number:

-2: Kernel picks page number

-1: Release the page

Other: A standard physical page number, e.g. as returned by OS_Memory 12

2 – : Page flags:

Bit 7: Page was previously reserved via OS_Memory 23

Bit 15: Lock page to prevent it from being claimed by another dynamic area

Other bits: Reserved (ignored). Specify zero for future compatibility.

The entries in the page list will be processed in order. For each entry the kernel will compare the state given in the list with the current state of that PMP page index and perform the appropriate action – claiming a page, releasing a page, or altering its flags. If the state of the page matches the state given in the list (or is a close enough match) then no action will be taken. E.g. if a page had previously been claimed by providing a specific page number, and then a new request is made for that page with the magic page number -2, no action will be performed. However if a different specific page number is given then the old page will be released and the new page will be claimed. Note that when pages are swapped in this manner the contents of the page will not be preserved.

If an error occurs then R2 and R3 will be updated to point to the first entry that has not been processed. However this may not be the entry that generated the error (it might be caused by an entry further on in the list).

When requesting specific page numbers, it's recommended to also set bit 15 of the page flags, to prevent other dynamic areas from claiming the page. This will produce behaviour that is consistent with standard dynamic areas which request specific physical pages via their PreGrow handler. When locking a page in this manner, bit 7 must be specified if the page had been previously reserved via OS_Memory 23. This is a simple safety feature aimed at reducing the risk of buggy dynamic area handlers accidentally claiming pages which had been reserved by someone else.

When releasing a page, the flags specified in word 2 of the page list entry are ignored.

The initial state of a page after it has been allocated to a PMP is for it to not be mapped in to the logical address space. To map or unmap pages within your logical address space you must use OS_DynamicArea 22.

Attempting to release or swap a page which is currently mapped in will fail – you must first unmap the page via OS_DynamicArea 22.

[RISC OS Open source page](#)

OS_DynamicArea 22

SWI &66 · OS_DynamicArea

This call is used to map/unmap pages within the logical portion of a Physical Memory Pool.

Before calling

R0 22 (reason code)

R1 Area number

R2 Pointer to page list

R3 Number of entries in page list

On return

R2 Advanced to first entry not processed (or end of list)

R3 Number of entries not processed

Notes

Each entry in the page list is 12 bytes in length, structured as follows:

0 – : DA page number indicating the logical address to alter (Zero-based)

1 – : PMP page index:

-1: Unmap any page from the indicated location

Other: Index of PMP page to map in to the location

2 – : Page flags:

Bits 0-5, 12-14: As per Memory Page Access Flags

Bit 15: Lock page to prevent it from being claimed by another dynamic area

Other bits: Reserved (ignored). Specify zero for future compatibility.

The entries in the page list will be processed in order. For each entry the kernel will compare the state given in the list with the current state of that DA page number and perform the appropriate action – mapping in a physical page, unmapping a physical page, or altering its flags. If the state of the page matches the state given in the list then no action will be taken.

If an error occurs then R2 and R3 will be updated to point to the first entry that has not been processed. However this may not be the entry that generated the error (it might be caused by an entry further on in the list).

When unmapping a page, the flags specified in word 2 of the page list entry are ignored – if you need to change the flags of a page when mapping it out either change the flags before mapping it out, or issue a `OS_DynamicArea 21` call afterwards.

Attempting to map in a PMP page which does not have a physical RAM page allocated to it (via `OS_DynamicArea 21`) will fail.

Currently a PMP page can only be mapped in at one location at a time. So if page X is currently mapped in to location Y, and you submit a request to map it in at location Z, then the page tables will be updated so that location Y generates a translation fault before location Z is updated to map to page X. As far as the sequential order of operations is performed, this all happens within the context of the entry that requests the page to be moved.

For performance reasons, when making large numbers of changes you should map out any cacheable pages first, then non-cacheable pages, then map in new pages (in any order). The kernel will scan ahead through the list to work out how many pages (and of what type) are being mapped out so that it can work out whether it should perform global or per-page cache/TLB invalidation.

Bit 15 of the flags as specified here maps directly to bit 15 of the flags that are set via `OS_DynamicArea 21` – the flag is associated with the physical page, not with its logical mapping.

[RISC OS Open source page](#)

OS_DynamicArea 23

SWI &66 · `OS_DynamicArea`

This call is used to alter the maximum physical size of a Physical Memory Pool.

Before calling

R0 23 (reason code)

R1 Area number

R2 Resize amount (positive/negative page count)

On return

R2 Amount size of area has changed by (unsigned page count)

Notes

This allows you to change the value of R9 that was passed to `OS_DynamicArea 0` when the PMP was created.

Compared to regular dynamic areas, PMPs have extra memory overheads that are directly related to the maximum physical size of the area. So for large PMPs which may spend most of their time empty (e.g. RAMFS) it's recommended to dynamically adjust the physical size of the PMP to avoid wasting memory.

Note that when shrinking a PMP, the shrink will only succeed if the last N entries of the page list are unclaimed pages. If there are pages allocated then they must be released first (via `OS_DynamicArea 21`).

If the SWI returns an error, R2 will still have been updated to indicate the amount of the change that succeeded.

[RISC OS Open source page](#)

OS_DynamicArea 24

SWI &66 · OS_DynamicArea

The purpose of this call is to return information relating to a dynamic area. Where possible this call should be preferred over the older `OS_DynamicArea 2`, as it is able to correctly report the physical and logical sizes of Physical Memory Pool dynamic areas.

Before calling

R0 24 (reason code)

R1 Area number

On return

R0 Preserved

R1 Preserved

R2 Current logical size of area, in bytes

R3 Base logical address of area

R4 Area flags

R5 Maximum logical size of area, in bytes

R6 Current physical size of area, in pages

R7 Maximum physical size of area, in pages

R8 Pointer to name of area

Notes

For doubly-mapped areas, R3 on exit from this call returns the address of the boundary between the first and second copies of the area, whereas `OS_ReadDynamicArea` returns the start address of the first copy (i.e. the lowest valid address in the area).

For sparse and PMP DAs, the current logical size is calculated from the number of pages that are currently mapped in. Therefore it is not guaranteed to describe the logical extent of the area.

For non-PMP DAs, the physical size values returned in R6 and R7 will be derived from the logical size values.

Unlike `OS_DynamicArea 2`, this call cannot be used to read the handler address and parameter.

[RISC OS Open source page](#)

OS_DynamicArea 25

SWI &66 · OS_DynamicArea

This call is used to query the status of pages within a Physical Memory Pool.

Before calling

R0 25 (reason code)

R1 Area number

R2 Pointer to input/output page list

R3 Number of entries in page list

On return

- All registers preserved

Page list updated

Notes

Each entry in the page list is 16 bytes in length, structured as follows:

Word /

0 – : PMP page index

1 – : Physical page number

2 – : Dynamic area page number

3 – : Page flags:

Bits 0-5, 12-14: As per Memory Page Access Flags

Bit 15: Page is locked to prevent it from being claimed by another dynamic area

Other bits: Reserved

On entry, for each array entry either the PMP page index, phys page number, or DA page number must be provided, with the other page numbers set to -1. Page flags are ignored.

On exit, if the page is a member of the PMP, the entries will be filled in as appropriate. If the page isn't mapped in (and it was a lookup by PMP page index/phys page number) the DA page index will be set to -1. If no physical page is allocated (or the indicated page isn't a member of the PMP) the page flags will be set to 0.

[RISC OS Open source page](#)

OS_DynamicArea 26

SWI &66 · OS_DynamicArea

RISC OS 5 has had a boundary between application space and dynamic areas. Application space cannot extend above the boundary address, and dynamic areas cannot be placed below it.

Traditionally the boundary address is fixed at ROM compile time, but this call allows it to be moved at runtime, giving users and programs more control over the memory map. E.g. compatibility aid software which wants to mimic the memory map used by 26bit versions of RISC OS can use this call to lower the limit, allowing the creation of dynamic areas at 26bit-era addresses.

Before calling

R0 26 (reason code)

R1 New address boundary between application space and dynamic areas, or 0 to read

On return**R0** Preserved**R1** Previous address boundary**Notes**

A “Bad parameters” error (&1EA) will be returned if the provided address is outside the compile-time limits supported by the kernel. An “Overlapping areas” error (&1C6) will be returned if the address is within the compile-time limits, but conflicts with memory/address space reserved by an existing application or dynamic area.

Note that OS_DynamicArea 0 will prefer to place dynamic areas at the lowest possible free address. Thus care should be taken if software wants to be able to lower the limit and then raise it again at a later date. E.g. the software could create “filler” dynamic areas which will reserve all the free space, and then kill those dynamic areas just before it restores the application space limit.

Any adjustment to the boundary will be reflected in the R2 value returned by OS_ReadDynamicArea when inspecting application space (area -1).

This call was introduced in RISC OS 5.27

[RISC OS Open source page](#)

OS_DynamicArea 27

SWI &66 · OS_DynamicArea

The purpose of this call is to return the total free space, in pages.

Before calling**R0** 27 (reason code)**R1** Number of single shrinkable areas to exclude from sum, or -1 to include all shrinkable areas**On return****R0** Preserved**R1** Preserved**R2** Total amount of free space (pages)**In practice**

This call supersedes OS_DynamicArea 5, which returns a 32-bit byte count of free free space, thus preventing it from returning an accurate value if there are many gigabytes of free RAM.

[RISC OS Open source page](#)

OS_DynamicArea 3

SWI &66 · OS_DynamicArea

The purpose of this call is to enumerate dynamic areas.

Before calling**R0** 3 (reason code)**R1** Area number, -1 to start

On return

R0 Preserved

R1 Next area number, or -1 for no more

Notes

This allows an application to find out what dynamic areas are defined.

-1 is passed on entry to start the enumeration process. The call is then repeated until -1 is returned on exit, which confirms the end of the process. This also means that the application slot dynamic area cannot be enumerated by this call.

[RISC OS Open source page](#)

OS_DynamicArea 4

SWI &66 · OS_DynamicArea

The purpose of this call is to renumber dynamic areas.

Before calling

R0 4 (reason code)

R1 Old area number

R2 New area number

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

This call is intended for system use only, and should not be used.

On completion, Service_DynamicAreaRenumber is issued to inform the rest of the system about this change.

An error is returned if the area specified by the old area number does not exist or if the new number clashes with an existing area.

[RISC OS Open source page](#)

OS_DynamicArea 5

SWI &66 · OS_DynamicArea

The purpose of this call is to return the total free space, in bytes.

Before calling

R0 5 (reason code)

R1 Number of single shrinkable areas to exclude from sum, or -1 to include all shrinkable areas

On return

R0 Preserved

R1 Preserved

R2 Total amount of free space (bytes)

In practice

The returned value will be clamped at 2GB-PageSize in order to try and avoid any issues relating to signed number overflow. To avoid this clamping, use OS_DynamicArea 27.

[RISC OS Open source page](#)

OS_DynamicArea 8

SWI &66 · OS_DynamicArea

The purpose of this call is to set clamps on maximum size for subsequently created dynamic areas.

Before calling

R0 8 (reason code)

R1 Clamp on maximum size of (non-Sparse) areas created by OS_DynamicArea 0 with R5 = -1, or 0 to read only

R2 Clamp on maximum size of (non-Sparse) areas created by OS_DynamicArea 0 with R5 > 0, or 0 to read only

R3 Clamp on maximum size of Sparse areas created by OS_DynamicArea 0 with R4 bit 10 set, or 0 to read only

On return

R0 Preserved

R1 Previous clamp for OS_DynamicArea 0 with R5 = -1

R2 Previous clamp for OS_DynamicArea 0 with R5 > 0

R3 Previous clamp for OS_DynamicArea 0 with R4 bit 10 set

Notes

Specifying -1 in R1 or R2 means that the respective clamp is the RAM limit of the machine (default). Specifying a figure larger than the RAM limit is equivalent to specifying -1.

Specifying -1 for R3 is invalid, since there is no concept of RAM limit for sparse area maximum size. Ergo, the effective limit is then the size of the largest fragment of logical address space free at creation time.

[RISC OS Open source page](#)

OS_DynamicArea 9

SWI &66 · OS_DynamicArea

The purpose of this call is to ensure that a region of a Sparse area is mapped to valid memory.

Before calling

R0 9 (reason code)

R1 Area number

R2 Base of region to claim

R3 Size of region to claim

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

The region ($\text{base} \leq x \leq \text{size}-1$) must be entirely within the address range of the Sparse area. There are no restrictions on the distribution of any mapped memory within the region before the call.

Note that although arbitrary alignment of base and size is allowed, the granularity of the mapping actually performed is the page size of the machine (as return by `OS_ReadMemMapInfo`).

[RISC OS Open source page](#)

OS_EnterOS

SWI &16

The purpose of this call is to set the processor to Supervisor mode.

Before calling

--

On return

All registers preserved

Notes

Interrupt and Fast Interrupt status is unaltered. Note that SWI calls while in SVC mode will alter R14, and will use R13 as stack.

To return to User mode on an ARM6 or later:

```
MSR CPSR_c, #0x10000 ; USR32
```

alternatively to return to either USR32 or USR26

```
MRS R14, CPSR
```

```
BIC R14, R14, #0x1111 ; Only sets usermode
```

```
MSR CPSR_c, R14
```

To return to User mode pre ARM6 (deprecated):

```
TEQP PC, #0
```

```
MOV R0, R0 ; Anything that doesn't use banked R8-R14
```

alternatively to return to user mode leaving the other PSR bits constant

```
BIC R14, PC, #0x11 ; Only sets usermode
```

```
TEQP R14, #0
```

```
MOV R0, R0 ; Anything that doesn't use banked R8-R14
```

[RISC OS Open source page](#)

OS_EnterUSR26

SWI &74

The purpose of this call is to set the processor to User mode.

Before calling

--

On return

All registers preserved

Notes

The interrupt status is unaltered.

This SWI returns in 26-bit user mode. Behaviour is undefined unless this SWI is called from 32-bit user mode. This SWI cannot be called from an address above 64M.

[RISC OS Open source page](#)

OS_EnterUSR32

SWI &73

The purpose of this call is to set the processor to User mode.

Before calling

--

On return

All registers preserved

Notes

The interrupt status is unaltered.

This SWI returns in 32-bit user mode. Behaviour is undefined unless this SWI is called from 26-bit user mode (normal RISC OS mode). This SWI cannot be called from an address above 64M. This SWI does not use the normal SWI exit code, and does not check for callbacks.

Once in 32-bit user mode, all code is subject to the restrictions of the ARM 32-bit instruction set. Beware of instructions that are illegal in 32-bit mode, and of instructions that behave differently.

Because 32-bit user mode is not the normal RISC OS user mode, there are additional restrictions:

- 32-bit user mode code must not call any SWIs, except OS_EnterUSR26 to return to normal RISC OS user mode
- 32-bit user mode code can jump to addresses above 64M (code in dynamic areas), but must jump back below 64M before calling OS_EnterUSR26.
- Although transient callbacks can occur (via interrupt returns, the current callback handler is not called because the API for callback handlers is not 32-bit aware.
- If an abort occurs in 32-bit mode code, the register dump information will be as if from 26-bit mode; an abort address above 64M cannot be reported properly. Instead, you should use OS_ReadSysInfo 7 to read the 32-bit PC and PSR for the last abort.

[RISC OS Open source page](#)

OS_EvaluateExpression

SWI &2D

The purpose of this call is to evaluate a string and return the value into a given buffer as either a string or integer.

Before calling

R0 Pointer to string

R1 Pointer to buffer

R2 Length of buffer

On return

R0 Preserved

R1 0 if integer returned, else preserved

R2 Integer if R1=0, else length of string

Notes

This SWI takes a string, evaluates it, and places result in given buffer. Type of the result is given in R1 :

0 / Integer returned in R2

Not 0 – Integer returned in R2: Pointer to string, length in R2

For a comprehensive list of expressions that can be used with this SWI please see the Expressions page.

Floating point numbers cannot be used within expressions.

[RISC OS Open source page](#)

OS_Exit

SWI &11

The purpose of this call is to pass control over to the most recent exit handler.

Before calling

R0 Pointer to error block

R1 "ABEX" (&58454241) If Return code and error block provided

R2 Return code

On return

Never returns

Notes

If R1 <> "ABEX", or R0 does not appear to be a valid error pointer, then Sys\$ReturnCode is set to 0.

If R1 = "ABEX", and $0 \leq R2 \leq \text{Sys\$RCLimit}$, then Sys\$ReturnCode is set to R2.

If R1 = "ABEX" and R2 is outside this range, then Sys\$ReturnCode is still set to R2, and the error pointer given in R0 is passed to OS_GenerateError. This has the effect of causing the Error Handler to be invoked, rather than the exit handler.

[RISC OS Open source page](#)

OS_ExitAndDie

SWI &50

The purpose of this call is to pass control over to the most recent exit handler, and kill the module as pass by R3.

Before calling

R0 Pointer to error block

R1 "ABEX" (&58454241) If Return code and error block provided

R2 Return code

R3 Pointer to module name

On return

Never returns

Notes

This SWI provides the same functionality as OS_Exit but in addition kills the module as passed by R3.

If R1 <> "ABEX", or R0 does not appear to be a valid error pointer, then Sys\$ReturnCode is set to 0.

If R1 = "ABEX", and 0 <= R2 <= Sys\$RCLimit, then Sys\$ReturnCode is set to R2.

If R1 = "ABEX" and R2 is outside this range, then Sys\$ReturnCode is still set to R2, and the error pointer given in R0 is passed to OS_GenerateError. This has the effect of causing the Error Handler to be invoked, rather than the exit handler.

[RISC OS Open source page](#)

OS_File

SWI &08

The purpose of this call is to perform operations on whole files.

Before calling

R0 Reason code

Other registers dependent of reason code

On return

R0 Preserved

Other registers dependent of reason code

Reason Codes

0 – Action: Save block of memory as a file

1 – Action: Write catalogue info for named object

2 – Action: Write load address for named object

3 – Action: Write exec address for named object

4 – Action: Write attributes for named object

5 – Action: Read catalogue info for object

- 6 – Action: Delete named object
- 7 – Action: Create empty un-stamped file
- 8 – Action: Create directory
- 9 – Action: Date stamp named object
- 10 – Action: Save block of memory as a typed file
- 11 – Action: Create empty file
- 12 – Action: Load named file given path
- 13 – Action: Read catalogue info for object given path
- 14 – Action: Load named file given path variable
- 15 – Action: Read catalogue info for object given path variable
- 16 – Action: Load named file without path
- 17 – Action: Read catalogue info for object without path
- 18 – Action: Set type of named object
- 19 – Action: Generate error message
- 20 – Action: Read catalogue info for stamped object
- 21 – Action: Read catalogue info for stamped object given path
- 22 – Action: Read catalogue info for stamped object given path variable
- 23 – Action: Read catalogue info for stamped object without path
- 24 – Action: Read block size of file
- 255 – Action: Load named file

Other languages documentation

Français

[RISC OS Open source page](#)

OS_File 0

SWI &08 · OS_File

The purpose of this call is to save a block of memory as a file.

Before calling

- R0 0 (reason code)
- R1 Pointer to filename
- R2 Load address
- R3 Execution address
- R4 Start address
- R5 End address (exclusive)

On return

All registers preserved

Notes

An error is returned if any of the following conditions are met: object is locked, open or is a directory.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_File 1

SWI &08 · OS_File

The purpose of this call is to write the load address, execution address and file attributes for a named object.

Before calling

R0 1 (reason code)

R1 Pointer to filename

R2 Load address

R3 Execution address

R5 Attributes

On return

All registers preserved

Notes

There are several different reason codes that can be used to write various catalogue information using the OS_File call.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_File 10

SWI &08 · OS_File

The purpose of this call is to save a block of memory as a file.

Before calling

R0 10 (reason code)

R1 Pointer to filename

R2 File type

R4 Start address

R5 End address (exclusive)

On return

All registers preserved

Notes

An error is returned if any of the following conditions are met: object is locked, open or is a directory.

[RISC OS Open source page](#)

OS_File 11

SWI &08 · OS_File

The purpose of this call is to create an stamped file of a given size with a particular file type.

Before calling

R0 11 (reason code)

R1 Pointer to non-wildcarded object name

R2 File type

R4 0

R5 Size of file

On return

All registers preserved

[RISC OS Open source page](#)

OS_File 12

SWI &08 · OS_File

The purpose of this call is to load a named file, using a path string.

Before calling

R0 12 (reason code)

R1 Pointer to filename

R2 Load address if (R3 AND &FF = 0)

R3 Bits 0-7 clear to use R2 as load address, else use file's load address/filetype

Bit 30 is used by the SaferOSFile extension

Bit 31 set to specify the file is code and should be synchronised

R4 Pointer to path string

R5 Length of buffer at R2 (if R3b30 set and SaferOSFile extension present)

On return

R0 Object type

R1 Preserved

R2 Load address

R3 Execution address

R4 Object length

R5 Object attributes

Notes

An error will be generated if the file is not found, is a directory, no read access, or a bad load address.

If the file is stamped, and loading at load address (R3 AND &FF <> 0) the variable Alias\$@LoadType_xxx will be used, and it is possible that (like OS_FSControl 4) this call may not return.

There are several different reason codes that can be used to load files using the OS_File call. They differ depending on how the path used to find the file.

In addition:

- R3 bit 31 is not supported (or necessary) before RISC OS 3.70.
- As of RISC OS 3.71, files stamped &F95 (Code) are always Synchronised after loading

[RISC OS Open source page](#)

OS_File 13

SWI &08 · OS_File

The purpose of this call is to read catalogue information for a named object using a path string.

Before calling

R0 13 (reason code)

R1 Pointer to filename

R4 Pointer to path string

On return

R0 Object type

R1 Preserved

R2 Load address

R3 Execution address

R4 Object length

R5 Object attributes

R2 – R5 are corrupted if no object is found

Notes

There are several different reason codes that can be used to read various catalogue information using the OS_File call.

[RISC OS Open source page](#)

OS_File 14

SWI &08 · OS_File

The purpose of this call is to load a named file, using a path variable.

Before calling

R0 14 (reason code)

R1 Pointer to filename

R2 Load address if (R3 AND &FF = 0)

R3 Bits 0-7 clear to use R2 as load address, else use file's load address/filetype

Bit 30 is used by the SaferOSFile extension

Bit 31 set to specify the file is code and should be synchronised

R4 Pointer to path variable

R5 Length of buffer at R2 (if R3b30 set and SaferOSFile extension present)

On return

- R0** Object type
- R1** Preserved
- R2** Load address
- R3** Execution address
- R4** Object length
- R5** Object attributes

Notes

An error will be generated if the file is not found, is a directory, no read access, or a bad load address.

If the file is stamped, and loading at load address (R3 AND &FF <> 0) the variable Alias\$@LoadType_xxx will be used, and it is possible that (like OS_FSControl 4) this call may not return.

There are several different reason codes that can be used to load files using the OS_File call. They differ depending on how the path used to find the file.

In addition:

- R3 bit 31 is not supported (or necessary) before RISC OS 3.70.
- As of RISC OS 3.71, files stamped &F95 (Code) are always Synchronised after loading

[RISC OS Open source page](#)

OS_File 15

SWI &08 · OS_File

The purpose of this call is to read catalogue information for a named object using a path variable.

Before calling

- R0** 15 (reason code)
- R1** Pointer to filename
- R4** Pointer to path variable

On return

- R0** Object type
 - R1** Preserved
 - R2** Load address
 - R3** Execution address
 - R4** Object length
 - R5** Object attributes
- R2 - R5 are corrupted if no object is found

Notes

There are several different reason codes that can be used to read various catalogue information using the OS_File call.

[RISC OS Open source page](#)

OS_File 16

SWI &08 · OS_File

The purpose of this call is to load a named file, using no path.

Before calling

R0 16 (reason code)

R1 Pointer to filename

R2 Load address if (R3 AND &FF = 0)

R3 Bits 0-7 clear to use R2 as load address, else use file's load address/filetype

Bit 30 is used by the SaferOSFile extension

Bit 31 set to specify the file is code and should be synchronised

R5 Length of buffer at R2 (if R3b30 set and SaferOSFile extension present)

On return

R0 Object type

R1 Preserved

R2 Load address

R3 Execution address

R4 Object length

R5 Object attributes

Notes

An error will be generated if the file is not found, is a directory, no read access, or a bad load address.

If the file is stamped, and loading at load address (R3 AND &FF <> 0) the variable Alias\$@LoadType_xxx will be used, and it is possible that (like OS_FSControl 4) this call may not return.

There are several different reason codes that can be used to load files using the OS_File call. They differ depending on how the path used to find the file.

In addition:

- R3 bit 31 is not supported (or necessary) before RISC OS 3.70.
- As of RISC OS 3.71, files stamped &F95 (Code) are always Synchronised after loading.

[RISC OS Open source page](#)

OS_File 17

SWI &08 · OS_File

The purpose of this call is to read catalogue information for a named object, using no path identifier.

Before calling

R0 17 (reason code)

R1 Pointer to filename

On return

R0 Object type

R1 Preserved

R2 Load address

R3 Execution address

R4 Object length

R5 Object attributes

R2 - R5 are corrupted if no object is found

Notes

No path is used to find the named object referenced by R1.

There are several different reason codes that can be used to read various catalogue information using the OS_File call.

[RISC OS Open source page](#)

OS_File 18

SWI &08 · OS_File

The purpose of this call is to write file type for a named object.

Before calling

R0 18 (reason code)

R1 Pointer to filename

R2 File type

On return

All registers preserved

Notes

There are several different reason codes that can be used to write various catalogue information using the OS_File call.

[RISC OS Open source page](#)

OS_File 19

SWI &08 · OS_File

The purpose of this call is to generate an error based on the filename.

Before calling

R0 19 (reason code)

R1 Pointer to filename

R2 Object block

On return

R0 Pointer to error block

[RISC OS Open source page](#)

OS_File 2

SWI &08 · OS_File

The purpose of this call is to write the load address information for a named object.

Before calling

R0 2 (reason code)

R1 Pointer to filename

R2 Load address

On return

All registers preserved

Notes

There are several different reason codes that can be used to write various catalogue information using the OS_File call.

[RISC OS Open source page](#)

OS_File 20

SWI &08 · OS_File

The purpose of this call is to read catalogue information for a named file, using File\$Path.

Before calling

R0 20 (reason code)

R1 Pointer to filename

On return

R0 Object type

R1 Preserved

R2 Load address (without top three bytes if file is typed)

R3 Execution address

R4 Object length

R5 Object attributes

R6 Object filetype

Notes

R2 to R6 may be corrupted if the object is not found.

The value returned in R6 is 4096 (&1000) for images and directories, and 8192 (&2000) for applications.

A value of -1 returned in R6 can also mean:

- An untyped file, when R2 and R3 contain addresses
- No file found, when R0 = 0

[RISC OS Open source page](#)

OS_File 21

SWI &08 · OS_File

The purpose of this call is to read catalogue information for a named file, using a path string.

Before calling

R0 21 (reason code)

R1 Pointer to filename

R4 Pointer to path string

On return

R0 Object type

R1 Preserved

R2 Load address (without top three bytes if file is typed)

R3 Execution address

R4 Object length

R5 Object attributes

R6 Object filetype

Notes

R2 to R6 may be corrupted if the object is not found.

The value returned in R6 is 4096 (&1000) for images and directories, and 8192 (&2000) for applications.

A value of -1 returned in R6 can also mean:

- An untyped file, when R2 and R3 contain addresses
- No file found, when R0 = 0

[RISC OS Open source page](#)

OS_File 22

SWI &08 · OS_File

The purpose of this call is to read catalogue information for a named file, using a path variable.

Before calling

R0 22 (reason code)

R1 Pointer to filename

R4 Pointer to path variable

On return

R0 Object type

R1 Preserved

R2 Load address (without top three bytes if file is typed)

R3 Execution address

R4 Object length

R5 Object attributes

R6 Object filetype

Notes

R2 to R6 may be corrupted if the object is not found.

The value returned in R6 is 4096 (&1000) for images and directories, and 8192 (&2000) for applications.

A value of -1 returned in R6 can also mean:

- An untyped file, when R2 and R3 contain addresses
- No file found, when R0 = 0

[RISC OS Open source page](#)

OS_File 23

SWI &08 · OS_File

The purpose of this call is to read catalogue information for a named file, using no path.

Before calling

R0 23 (reason code)

R1 Pointer to filename

On return

R0 Object type

R1 Preserved

R2 Load address (without top three bytes if file is typed)

R3 Execution address

R4 Object length

R5 Object attributes

R6 Object filetype

Notes

R2 to R6 may be corrupted if the object is not found.

The value returned in R6 is 4096 (&1000) for images and directories, and 8192 (&2000) for applications.

A value of -1 returned in R6 can also mean:

- An untyped file, when R2 and R3 contain addresses
- No file found, when R0 = 0

[RISC OS Open source page](#)

OS_File 24

SWI &08 · OS_File

The purpose of this call is to find the natural block size of a given file.

Before calling

R0 24 (reason code)

R1 Pointer to filename

On return

R0 Preserved

R1 Preserved

R2 Natural block size in bytes

Notes

When you create or extend a file, disc space can only be allocated in chunks of a minimum size (512 bytes, for example). If your file is just 1 byte long for example, 511 bytes of disc space may have to be wasted after it if the natural block size is 512.

[RISC OS Open source page](#)

OS_File 255

SWI &08 · OS_File

The purpose of this call is to load a named file, using File\$Path variable.

Before calling

R0 255 (reason code)

R1 Pointer to filename

R2 Load address if (R3 AND &FF = 0)

R3 Bits 0-7 clear to use R2 as load address, else use file's load address/filetype

Bit 30 is used by the SaferOSFile extension

Bit 31 set to specify the file is code and should be synchronised

R5 Length of buffer at R2 if R3b30 set and SaferOSFile extension present

On return

R0 Object type

R1 Preserved

R2 Load address

R3 Execution address

R4 Object length

R5 Object attributes

Notes

An error will be generated if the file is not found, is a directory, no read access, or a bad load address.

If the file is stamped, and loading at load address (R3 AND &FF <> 0) the variable Alias\$@LoadType_xxx will be used, and it is possible that (like OS_FSControl 4) this call may not return.

There are several different reason codes that can be used to load files using the OS_File call. They differ depending on how the path used to find the file.

In addition:

- R3 bit 31 is not supported (or necessary) before RISC OS 3.70.
- As of RISC OS 3.71, files stamped &F95 (Code) are always Synchronised after loading

[RISC OS Open source page](#)

OS_File 3

SWI &08 · OS_File

The purpose of this call is to write the execution address information for a named object.

Before calling

R0 3 (reason code)

R1 Pointer to filename

R3 Execution address

On return

All registers preserved

Notes

There are several different reason codes that can be used to write various catalogue information using the OS_File call.

[RISC OS Open source page](#)

OS_File 4

SWI &08 · OS_File

The purpose of this call is to write the file attribute information for a named object.

Before calling

R0 4 (reason code)

R1 Pointer to filename

R5 Attributes

On return

All registers preserved

Notes

There are several different reason codes that can be used to write various catalogue information using the OS_File call.

[RISC OS Open source page](#)

OS_File 5

SWI &08 · OS_File

The purpose of this call is to read catalogue information for a named object using the File\$Path for finding the object.

Before calling

R0 5 (reason code)

R1 Pointer to filename

On return

R0 Object type

R1 Preserved

R2 Load address

R3 Execution address

R4 Object length

R5 Object attributes

R2 – R5 are corrupted if no object is found

Notes

There are several different reason codes that can be used to read various catalogue information using the OS_File call.

[RISC OS Open source page](#)

OS_File 6

SWI &08 · OS_File

The purpose of this call is to delete a named object.

Before calling

R0 6 (reason code)

R1 Pointer to non-wildcarded object name

On return

R0 Object type

R1 Preserved

R2 Load address

R3 Execution address

R4 Object length

R5 Object attributes

R2 – R5 are corrupted if no object is found

Notes

It is not an error if the object does not exist. An error is generated if object is locked, open or a non-empty directory.

[RISC OS Open source page](#)

OS_File 7

SWI &08 · OS_File

The purpose of this call is to create an unstamped file of a given size with a particular file type.

Before calling

R0 7 (reason code)

R1 Pointer to non-wildcarded object name

R2 Load address

R3 Execution address

R4 0

R5 Size of file

On return

All registers preserved

[RISC OS Open source page](#)

OS_File 8

SWI &08 · OS_File

The purpose of this call is to create a directory.

Before calling

R0 8 (reason code)

R1 Pointer to non-wildcarded object name

R4 Number of entries, or 0 for default

On return

All registers preserved

Notes

It is not an error if the directory already exists.

Other languages documentation

Français?

[RISC OS Open source page](#)

OS_File 9

SWI &08 · OS_File

The purpose of this call is to write the date/time stamp information for a named object to the current date/time.

Before calling

R0 9 (reason code)

R1 Pointer to object name

On return

All registers preserved

Notes

The date/time stamp is stored using two 32-bit fields that would otherwise store the load address and execution address for a file. Information on how the date/time stamp is stored is available [here](#)

The file type is set to @FFD if not already.

There are several different reason codes that can be used to write various catalogue information using the OS_File call.

[RISC OS Open source page](#)

OS_Find

SWI &0D

The purpose of this call is to either open or close files.

Before calling

R0 Reason code, and path bits if reason $\neq 0$

R1 Filename for opening files, handle otherwise

R2 Pathname, if 'path used' is 1 or 2

On return

R0 File handle or 0 if object doesn't exist (if opening and R0 bit 3 is 0 on entry)

Reason Codes

R0 / Action

&00 – Action: Closes file R1, or all files if R1 = 0

&40 – Action: Open existing file with read-only access

&80 – Action: Create empty file with read/write access

&C0 – Action: Open existing file with read/write access

[RISC OS Open source page](#)

OS_FindMemMapEntries

SWI &60

The purpose of this call is to return the logical address to which the requested page numbers are mapped, ie address page.

Before calling

R0 Pointer to request block, terminated with -1

On return

R0 Block at R0 updated

Notes

On entry, the logical address field must be set. You may supply probable page numbers, which, if correct will make this call return more quickly than it might otherwise. If you have no idea what the page number might be, you should set the page number to zero on entry. The protection value is ignored on entry. If the page number is -1 on exit, then the memory map entry was not found; in this case, the protection level will always be 3. Otherwise the request list has been updated with the page number and protection level for the given logical address.

[RISC OS Open source page](#)

OS_FSControl

SWI &29

The purpose of this call is control the filing system manager and the filing system themselves.

Before calling

R0 Reason code

Other registers dependent on reason code

On return

R0 Preserved

Other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

Reason Codes

- 0 – Hex #: &00 · Action: Set current dir
- 1 – Hex #: &01 · Action: Set library dir
- 2 – Hex #: &02 · Action: Inform of start of new app
- 3 – Hex #: &03 · Action: Run Type (Internal use only)
- 4 – Hex #: &04 · Action: Run a file
- 5 – Hex #: &05 · Action: Catalogue a dir
- 6 – Hex #: &06 · Action: Examine current dir
- 7 – Hex #: &07 · Action: Catalogue library
- 8 – Hex #: &08 · Action: Examine library

- 9 – Hex #: &09 · Action: Examine objects
- 10 – Hex #: &0A · Action: Set FS options
- 11 – Hex #: &0B · Action: Set temporary FS
- 12 – Hex #: &0C · Action: Add a FS
- 13 – Hex #: &0D · Action: Check for FS
- 14 – Hex #: &0E · Action: Select FS
- 15 – Hex #: &0F · Action: Boot from FS
- 16 – Hex #: &10 · Action: Remove FS
- 17 – Hex #: &11 · Action: Add secondary module
- 18 – Hex #: &12 · Action: Convert filetype number to name
- 19 – Hex #: &13 · Action: Restore current FS
- 20 – Hex #: &14 · Action: Read location of temp FS
- 21 – Hex #: &15 · Action: Return a file handle
- 22 – Hex #: &16 · Action: Close all open files
- 23 – Hex #: &17 · Action: Shutdown FS
- 24 – Hex #: &18 · Action: Set attributes of objects
- 25 – Hex #: &19 · Action: Rename object
- 26 – Hex #: &1A · Action: Copy objects
- 27 – Hex #: &1B · Action: Wipe objects
- 28 – Hex #: &1C · Action: Count objects
- 29 – Hex #: &1D · Action: Create Handle (Internal use only)
- 30 – Hex #: &1E · Action: Read location of temporary FS secondary module
- 31 – Hex #: &1F · Action: Convert filetype name to number
- 32 – Hex #: &20 · Action: Output list of object names & info
- 33 – Hex #: &21 · Action: Convert a filing system number to a name
- 34 – Hex #: &22 · Action: Set Contexts (Reserved)
- 35 – Hex #: &23 · Action: Add an image FS
- 36 – Hex #: &24 · Action: Remove an image FS
- 37 – Hex #: &25 · Action: Canonicalise path
- 38 – Hex #: &26 · Action: Extract filetype from file information
- 39 – Hex #: &27 · Action: Set User Root Directory
- 40 – Hex #: &28 · Action: Swap current and previous directories
- 41 – Hex #: &29 · Action: Return defect list for an image file
- 42 – Hex #: &2A · Action: Map out defect from an image file
- 43 – Hex #: &2B · Action: Unset the Current Directory
- 44 – Hex #: &2C · Action: Unset the User Root Directory
- 45 – Hex #: &2D · Action: Unset the Library Directory
- 46 – Hex #: &2E · Action: Return used space map of image file
- 47 – Hex #: &2F · Action: Read boot option for disc or image file
- 48 – Hex #: &30 · Action: Write boot option for disc or image file
- 49 – Hex #: &31 · Action: Read free space for disc or image file
- 50 – Hex #: &32 · Action: Rename disc or image file
- 51 – Hex #: &33 · Action: Request update of image stamp
- 52 – Hex #: &34 · Action: Find object at offset within image file
- 53 – Hex #: &35 · Action: Set given directory to path unchecked
- 54 – Hex #: &36 · Action: Read path of given directory
- 55 – Hex #: &37 · Action: Read free space for a large disc or image file
- 56 – Hex #: &38 · Action: Read defect list for a large disc or image file
- 57 – Hex #: &39 · Action: Map out defect from large disc or image
- 58 – Hex #: &3A · Action: Enumerate file handles

[RISC OS Open source page](#)

OS_FSControl 0

SWI &29 · OS_FSControl

The purpose of this call is set the current directory and filing systems.

Before calling

R0 0 (reason code)

R1 Pointer to wildcarded filename

On return

R0 Preserved

R1 Preserved

Notes

If a pointer to "" is used, the directory is set to the User Root Directory (URD).

To unset the CSD use OS_FSControl 43.

[RISC OS Open source page](#)

OS_FSControl 1

SWI &29 · OS_FSControl

The purpose of this call is set the Library Directory (Lib) and filing system.

Before calling

R0 1

R1 Pointer to (wildcarded) filename

On return

R0 Preserved

R1 Preserved

Notes

If a pointer to "" is used, then the default for this filing system is used.

On FileCore-based systems, the default is to search &.Library, \$.Library and @.

To unset the Library Directory use OS_FSControl 45.

[RISC OS Open source page](#)

OS_FSControl 10

SWI &29 · OS_FSControl

The purpose of this call is to set filing system options.

Before calling

R0 10 (&0A)

R1 Option

R2 Value

On return

R0 Preserved

Other registers dependent on reason code

Notes

Sets the options on the temporary filing system. See OS_FSControl 48 to set the boot option (4) for a given disc.

[RISC OS Open source page](#)

OS_FSControl 11

SWI &29 · OS_FSControl

The purpose of this call is set the temporary filing system from the prefix string in R1.

Before calling

R0 11 (&0B)

R1 Pointer to (wildcarded) filename

On return

R0 Preserved

R1 Updated to point beyond filing system specifier

R2 If -1 then no filing system specifier present (call has no effect)

If ≥ 0 then filing system number to restore

R3 Pointer to special field, 0 if not present

Notes

It is used by OS_CLI to set the temporary filing system for the command's duration. After the command the current filing system is restored with OS_FSControl 19.

Examples

blah.blah vlsgr – Registers out: R1 = R1 in

R2 = -1

R3 = 0

<FS>:blah – Registers out: R1 past ':'

R2 = <number of filing system

R3 = 0

<FS>#fred:blah – Registers out: R1 past ':'

R2 = <number of filing system

R3 past '#'

<path>:blah – Registers out: R1 = R1 in

R2 = -1

R3 = 0

<path>#fred:blah – Registers out: R1 = R1 in

R2 = -1

R3 past '#'

#fred:blah – Registers out: R1 = R1 in

R2 = -1

R3 past '#'

[RISC OS Open source page](#)

OS_FSControl 12

SWI &29 · OS_FSControl

The purpose of this call is to make FileSwitch aware of a new Filing System.

Before calling

R0 12 (&0C)

R1 Pointer to module base

R2 Offset of filing system information block

R3 Pointer to private word

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

[RISC OS Open source page](#)

OS_FSControl 13

SWI &29 · OS_FSControl

The purpose of this call is to check for a presence of a filing system.

Before calling

R0 13 (&0D)

R1 Filing system number or pointer to name

R2 Dependant on R1

On return

R0 Preserved

R1 Filing system number

R2 Pointer to filing system control block or 0 if not found

Notes

The FS name match is case-insensitive.

If R2 is 0, the FS name is taken to be terminated with any control character or the characters: "#", ":", or "-". If R2 is not 0, then the FS name is terminated by any control character.

The FS control block that is pointed to by R2 on exit is for the internal use of FileSwitch, and you should not use or alter it. You should only test R2 for equality (or not) with 0.

An error is returned if the FS name contains bad chars or is badly terminated.

[RISC OS Open source page](#)

OS_FSControl 14

SWI &29 · OS_FSControl

The purpose of this call is to select a filing system as both the current and temporary filing system.

Before calling

R0 14 (&0E)

R1 Filing system number or pointer to name, or 0 for none

On return

R0 Preserved

R1 Preserved

[RISC OS Open source page](#)

OS_FSControl 15

SWI &29 · OS_FSControl

The purpose of this call is to boot from the current filing system.

Before calling

R0 15 (&0F)

On return

R0 Preserved – may not return if boot runs an application

Notes

The actions taken are dependant on the boot setting (see OS_Byte 255 and option 4 setting – the latter is set on the temporary filing system with OS_FSControl 10).

[RISC OS Open source page](#)

OS_FSControl 16

SWI &29 · OS_FSControl

The purpose of this call is to remove a Filing System from FileSwitch.

Before calling

R0 16 (&10)

R1 Pointer to filing system name

On return

R0 Preserved

R1 Preserved

Notes

The name is control character terminated.

If the FS is current then the current and temporary filing system numbers are set to 0, and the old FS number is stored. If it is added again before a new current FS is selected then it will be reselected as current. If the FS is the temporary FS (but not the current FS) then the temporary FS is set to the current FS.

[RISC OS Open source page](#)

OS_FSControl 17

SWI &29 · OS_FSControl

The purpose of this call is to add secondary modules which provide additional filing system commands?.

Before calling

R0 17 (&11)

R1 Pointer to filing system name

R2 Pointer to secondary system name

R3 Secondary system private word

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

An example of adding extra filing system commands is FileCore; it adds the commands provided by ADFS.

[RISC OS Open source page](#)

OS_FSControl 18

SWI &29 · OS_FSControl

The purpose of this call is to decode a filetype into text.

Before calling

R0 18 (&12)

R2 Filetype (bits 0-11)

On return

R0 Preserved

R2 First four chars

R3 Final four chars

Notes

This SWI issues Service_LookupFileType. If unclaimed, it builds a default string. The string is 8 characters long, suffixed with spaces as necessary.

[RISC OS Open source page](#)

OS_FSControl 19

SWI &29 · OS_FSControl

The purpose of this call is to set the temporary filing system back to the current filing system.

Before calling

R0 19 (&13)

On return

R0 Preserved

Notes

It is called by OS_CLI after a command has run, and by the kernel before the error handler is called.

[RISC OS Open source page](#)

OS_FSControl 2

SWI &29 · OS_FSControl

The purpose of this call is to inform the Operating System and current application, that a new application is starting.

Before calling

R0 2

R1 pointer to command tail

R2 currently active object pointer to write

R3 pointer to command name

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

Although registers are preserved, the call may not return.

This SWI allows you to fake starting an application. It performs the following actions:

- Issues OS_UpCall 256 to inform the current application – if the upcall is claimed then the error “Unable to start application” is returned.
- Issues Service_NewApplication to inform modules. If the service is claimed, then the error cannot be returned to you (as your application will now have been killed) so the error “Unable to start application” is passed to OS_GenerateError, which results in the error being sent to your parent’s error handler.
- Unless the exit handler is below memory limit then all handlers below memory limit are reset to their defaults (OS_ReadDefaultHandler).
- The currently active object pointer is set; the current time and command tail are set (OS_WriteEnv).

[RISC OS Open source page](#)

OS_FSControl 20

SWI &29 · OS_FSControl

The purpose of this call is to read the location of the primary module for the temporary filing system, and its private word. (For example, if ADFS were the temporary filing system, this call would return FileCore’s address, whereas OS_FSControl 30 would return the address of ADFS – the secondary module. However, if NetFS were the temporary filing system, this call would return its address.)

Before calling

R0 20 (&14)

On return

R0 Preserved

R1 Pointer to primary module base

R2 Pointer to private word

Notes

If there is no temporary filing system this call will return values for the current filing system. If there is no current filing system R1=0, R2 is undefined. The secondary module values can be read with OS_FSControl 30.

[RISC OS Open source page](#)

OS_FSControl 21

SWI &29 · OS_FSControl

The purpose of this call is to return the internal file handle for a filing system.

Before calling

R0 21 (&15)

R1 file handle

On return

R0 Preserved

R1 Filing system file handle

R2 Filing system information word

Notes

This SWI returns the internal file handle (as returned by FSEntry_Open) for a given file handle. If the fileswitch handle is invalid R1=0 on return.

This SWI should only be used to implement a filing system.

[RISC OS Open source page](#)

OS_FSControl 22

SWI &29 · OS_FSControl

The purpose of this call is to close all open files.

Before calling

R0 22 (&16)

On return

R0 Preserved

Notes

This SWI flushes and closes all open files on all filing systems.

It does not stop if an individual close reports an error, but will only return without error if all individual closes were error free.

It is equivalent to the *Shut command.

[RISC OS Open source page](#)

OS_FSControl 23

SWI &29 · OS_FSControl

The purpose of this call is to shutdown all filing systems.

Before calling

R0 23 (&17)

On return

R0 Preserved

Notes

This SWI flushes and closes all open files on all filing systems. It does not stop if an individual close reports an error, but will only return without error if all individual closes were error free.

It informs all filing systems of the shutdown – this means that:

- NetFS logs off from all file servers
- FileCore dismounts all discs
- disc heads are parked

It is equivalent to the *ShutDown command.

[RISC OS Open source page](#)

OS_FSControl 24

SWI &29 · OS_FSControl

The purpose of this call is to set the attributes of an object.

Before calling

R0 24 (&18)

R1 Pointer to (wildcarded) filename

R2 Attribute string

On return

R0 Preserved

Notes

This SWI sets the access attributes for all leaf objects that match the filename.

It is equivalent to the *Access command.

The attribute string is exactly the same as those for the *Access command.

[RISC OS Open source page](#)

OS_FSControl 25

SWI &29 · OS_FSControl

The purpose of this call is to rename a file.

Before calling

R0 25 (&19)

R1 Pointer to original filename

R2 Pointer to new filename

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

It can only rename objects to new names on the same filing system and disc – otherwise use perform a move, followed by a delete operation (see OS_FSControl 26).

There is no SWI to discover which disc a given file is on – hence it is impossible to ascertain whether this SWI will fail without actually calling it.

[RISC OS Open source page](#)

OS_FSControl 26

SWI &29 · OS_FSControl

The purpose of this call is to copy objects.

Before calling

R0 26 (&1A)

R1 Pointer to wildcarded filename of files to copy

R2 Pointer to destination

R3 Flags

R4 Start load

R5 Start exec

R6 End load

R7 End exec

R8 Pointer to descriptor

On return

All registers preserved

Notes

This SWI allows you to copy files from one directory to another or as a new name. The destination leafname may not be wildcarded, unless it is "*" signifying "take destination leafname from source leafname".

The start and end load and exec parameters give a range of date stamps to copy between. The filetype field is ignored.

It is not possible to choose a combination of flags to perform the same operation as “no confirm” in the desktop, as the desktop will overwrite files without confirmation, but won't force the overwriting of locked files, whereas this SWI will only overwrite files if it either has force set, or confirm set and gets positive confirmation.

[RISC OS Open source page](#)

OS_FSControl 27

SWI &29 · OS_FSControl

The purpose of this call is to wipe objects.

Before calling

R0 27 (&1B)

R1 Pointer to wildcarded filename of files to wipe

R3 Flags

R4 Start load

R5 Start exec

R6 End load

R7 End exec

On return

All registers preserved

Notes

It is equivalent to the *Wipe command.

[RISC OS Open source page](#)

OS_FSControl 28

SWI &29 · OS_FSControl

The purpose of this call is to count objects. It is equivalent to the *Count command.

Before calling

R0 28 (&1C)

R1 Pointer to wildcarded filename of files to count

R3 Flags

R4 Start load

R5 Start exec

R6 End load

R7 End exec

R8 Pointer to descriptor

On return

R2 Total length

R3 Number of files

All registers preserved

Notes

The descriptor block, introduced in FileSwitch 2.73, should be used when counting discs larger than 4 Gbytes since the register R2 alone cannot return more than this. If the descriptor block is not used and the result overflowed R2 the value 0×7FFFFFFF is substituted (prior to this change the count total modulo 4 Gbytes was returned).

[RISC OS Open source page](#)

OS_FSControl 30

SWI &29 · OS_FSControl

The purpose of this call is to read the location of the secondary module for the temporary filing system.

Before calling

R0 30 (&1E)

On return

R0 Preserved

R1 Pointer to secondary module base of temporary filing system

R2 Pointer to private word of temporary filing system

Notes

If there is no temporary filing system this call will return values for the current filing system. If there is no current filing system, or no secondary module R1=0, R2 is undefined.

The primary module values can be read with OS_FSControl 20.

[RISC OS Open source page](#)

OS_FSControl 31

SWI &29 · OS_FSControl

The purpose of this call is to convert a string to a filetype number.

Before calling

R0 31 (&1F)

R1 Pointer to filetype string

On return

R0 Preserved

R1 Preserved

R2 Filetype

Notes

The string is control terminated, and can be a filetype-name ("Obey"), or a filetype-number ("FEB").

An error is generated if conversion impossible.

[RISC OS Open source page](#)

OS_FSControl 32

SWI &29 · OS_FSControl

The purpose of this call is to output a list of object names and information.

Before calling

R0 32 (&20)

R1 Pointer to wildcarded pathname

On return

R0 Preserved

R1 Preserved

Notes

It is equivalent to the *FileInfo command.

[RISC OS Open source page](#)

OS_FSControl 33

SWI &29 · OS_FSControl

The purpose of this call is to convert a filing system number to a filing system name.

Before calling

R0 33 (&21)

R1 Filing system number

R2 Pointer to buffer

R3 Length of buffer

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

This call converts the filing system number passed in R1 to a filing system name. The name is stored in the buffer pointed to by R2, and is null-terminated. If FileSwitch does not know of the filing system number you pass it, a null string is returned.

[RISC OS Open source page](#)

OS_FSControl 35

SWI &29 · OS_FSControl

The purpose of this call is to register an image filing system with the Operating System.

Before calling

R0 35 (&23)

R1 Module base address

R2 Offset to ImageFS information block

R3 Value to pass to routines in R12

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

The module should make this call when it initialises.

R1 and R2 give the location of an image filing system information block, which is used by FileSwitch to communicate with the image filing system module. It contains both information about the image filing system, and the location of entry points to the module's code.

The private word pointer passed in R3 is stored by FileSwitch. When it makes a call to the image filing system module, the private word is passed in R12. Normally, this private word is the workspace pointer for the module.

[RISC OS Open source page](#)

OS_FSControl 36

SWI &29 · OS_FSControl

The purpose of this call is to unregister/remove an image filing system from the list maintained by FileSwitch.

Before calling

R0 36 (&24)

R1 Image filing system's filetype

On return

R0 Preserved

R1 Preserved

Notes

It calls the image filing system to close open files, flush buffers, and so on. You should use it in the finalise entry of an image filing system module.

Image filing systems must be removed on any type of finalisation call, and added on any kind of initialisation. The reason for this is that FileSwitch keeps pointers into the image filing system module code, which may be moved as a result of tidying the module area or other such operations.

Image filing systems can only be removed by passing the filing system's filetype. Attempting to use the filing system number will result in an error being returned.

Modules must not complain about errors in filing system removal. Otherwise, it would be impossible to reinitialise the module after reinitialising the filing system manager.

[RISC OS Open source page](#)

OS_FSControl 37

SWI &29 · OS_FSControl

The purpose of this call is to take a filename (which may contain wildcards) and convert it into a complete pathname, including disc and directory name when not provided.

Before calling

R0 37 (&25)

R1 Pointer to pathname to convert

R2 Pointer to buffer

R3 Pointer to name of path variable, or 0 for none

R4 Pointer to ctrl-terminated comma separated path string, or 0 for none

R5 Buffer size

On return

R5 Number of spare bytes in buffer (inc. term)

Notes

If you call once with R2 and R5 set to zero, R5 will minus the return the number of bytes needed. Now you can call again having allocated 1-R5 bytes from somewhere (not just -R5 as you need to allow space for the terminator).

Examples

fred may convert to SCSI::Gamma\$.Utils.Fred

[RISC OS Open source page](#)

OS_FSControl 38

SWI &29 · OS_FSControl

The purpose of this call is to return the filetype of an object.

Before calling

R0 38 (&26)

R1 Pointer to pathname

R2 Start load

R3 Start exec

R4 Object length

R5 Attributes

R6 Object type

On return

R2 Filetype, or

-1 = untyped

&1000 = directory

&2000 = application

[RISC OS Open source page](#)

OS_FSControl 39

SWI &29 · OS_FSControl

The purpose of this call is to set the User Root Directory (URD) to the given pathname.

Before calling

R0 39 (&27)

R1 Pointer to pathname

On return

R0 Preserved

R1 Preserved

Notes

The URD represented by the character '&'.

To unset the URD use OS_FSControl 44.

[RISC OS Open source page](#)

OS_FSControl 4

SWI &29 · OS_FSControl

The purpose of this call is run a file.

Before calling

R0 4

R1 Pointer to (wildcarded) filename

On return

R0 Preserved

R1 Preserved

Notes

Files with load and exec addresses will be loaded and executed, files of type &FF8 (Absolute) will be loaded and executed at &8000.

Utilities (type &FFC) will be loaded into RMA and run.

All other filetypes will be searched for in Alias\$@RunType_xxx, where xxx is the filetype. i.e &FA8.

It is possible that this call will not exit depending on the execution of the file.

[RISC OS Open source page](#)

OS_FSControl 40

SWI &29 · OS_FSControl

The purpose of this call is to set to exchange the current and previous directories.

Before calling

R0 40 (&28)

On return

R0 Preserved

Notes

It is the equivalent of *Back command. i.e. swaps the previous directory with the current directory.

[RISC OS Open source page](#)

OS_FSControl 41

SWI &29 · OS_FSControl

The purpose of this call is to return the defect list of an image.

Before calling

R0 41 (&29)

R1 Pointer to name of image (null terminated)

R2 Pointer to buffer

R5 Buffer size

On return

All registers preserved

Notes

This SWI fills the buffer with the defect list of an image. The buffer is filled with words giving byte offsets of defects, terminated by &20000000.

To map out a defect from an image, use OS_FSControl 42.

A newer implementation of this call for handling larger discs (64 bit values) is OS_FSControl 56.

[RISC OS Open source page](#)

OS_FSControl 42

SWI &29 · OS_FSControl

The purpose of this call is to map out a defect from an image.

Before calling

R0 42 (&2A)

R1 Pointer to name of image (null terminated)

R2 Byte offset of defect start

On return

All registers preserved

Notes

To return a list of defects from an image, use OS_FSControl 41.

OS_FSControl 57 provides exactly the same functionality as this call, but supports 64 bits values, and thus can support larger discs.

[RISC OS Open source page](#)

OS_FSControl 43

SWI &29 · OS_FSControl

The purpose of this call is to unset the Currently Selected Directory (CSD).

Before calling

R0 43 (&2B)

On return

All registers preserved

Notes

To set the CSD use OS_FSControl 0.

[RISC OS Open source page](#)

OS_FSControl 44

SWI &29 · OS_FSControl

The purpose of this call is to unset the User Root Directory (URD).

Before calling

R0 44 (&2C)

On return

All registers preserved

Notes

To set the URD use OS_FSCControl 39.

[RISC OS Open source page](#)

OS_FSCControl 45

SWI &29 · OS_FSCControl

The purpose of this call is to unset the Library Directory (Lib).

Before calling

R0 45 (&2D)

On return

All registers preserved

Notes

To set the Library Directory use OS_FSCControl 1.

[RISC OS Open source page](#)

OS_FSCControl 46

SWI &29 · OS_FSCControl

The purpose of this call is to returns an image file's used space map.

Before calling

R0 46 (&2E)

R1 Pointer to name of image (null terminated)

R2 Pointer to buffer

R5 Buffer size

On return

All registers preserved

Notes

This SWI writes the image used space map to the buffer, 1 bits signifying used blocks, 0 bits free blocks. Bit 0 in a byte comes before bit 7.

This SWI sets bits up to min(buffer size, file size / block size). Block size for a file can be read with OS_File 24. For a non-image file the buffer will be filled with ones.

[RISC OS Open source page](#)

OS_FSControl 47

SWI &29 · OS_FSControl

The purpose of this call is to read the boot option of the disc or image file that holds a specified object.

Before calling

R0 47 (&2F)

R1 Pointer to name of object

On return

R0 Preserved

R1 Preserved

R2 Boot option

[RISC OS Open source page](#)

OS_FSControl 48

SWI &29 · OS_FSControl

The purpose of this call is to set the boot option for a disc or image.

Before calling

R0 48 (&30)

R1 Pointer to name of object

R2 New boot option

On return

R0 Preserved

R1 Preserved

R2 Preserved

[RISC OS Open source page](#)

OS_FSControl 49

SWI &29 · OS_FSControl

The purpose of this call is to reads the free space on the disc or image file that holds a specified object.

Before calling

R0 49 (&31)

R1 Pointer to name of object

On return

R0 Free space

R1 Size of largest object that can be created

R2 Size of disc

Notes

For large discs, you should use OS_FSControl 55.

[RISC OS Open source page](#)

OS_FSControl 5

SWI &29 · OS_FSControl

The purpose of this call is to catalogue a directory.

Before calling

R0 5

R1 Pointer to (wildcarded) filename

On return

R0 Preserved

R1 Preserved

Notes

This command is equivalent of issuing a *Cat command.

If a pointer to "" is used, the current directory is catalogued.

[RISC OS Open source page](#)

OS_FSControl 50

SWI &29 · OS_FSControl

The purpose of this call is to rename the disc or image file containing the given object.

Before calling

R0 50 (&32)

R1 Pointer to name of object

R2 New disc name

On return

R0 Preserved

R1 Preserved

R2 Preserved

[RISC OS Open source page](#)

OS_FSControl 51

SWI &29 · OS_FSControl

The purpose of this call is to request that an image stamp be updated.

Before calling

R0 51 (&33)

R1 Pointer to name of object

R2 0 (stamp when updated)

1 (stamp now)

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

This SWI is called by a handler of discs (such as FileCore) when it wants an image filing system. e.g. DOSFS to update the image stamp (Disc Cycle ID).

[RISC OS Open source page](#)

OS_FSControl 52

SWI &29 · OS_FSControl

The purpose of this call is to find the name and type of object that uses a particular offset within an image.

Before calling

R0 52 (&34)

R1 Pointer to name of image file or disc

R2 Offset into image file or disc

R3 Pointer to buffer to receive object name

R4 Buffer length

On return

R0 Preserved

R1 Preserved

R2 Object Type

R3 Preserved

R4 Preserved

Notes

This call scans a disc or image file to find which object occupies a particular disc address. This is useful in trying to recover files, or map out a defect.

The buffer contains object name (may be corrupted if not found).

[RISC OS Open source page](#)

OS_FSControl 53

SWI &29 · OS_FSControl

The purpose of this call is to set a directory to a given path without checking if the path is valid.

Before calling

R0 53 (&35)

R1 Pointer to new path

R2 Directory to set (use number)

R3 Pointer to name of filing system

R6 Pointer to special field (or 0 for none)

On return

All registers preserved

[RISC OS Open source page](#)

OS_FSControl 54

SWI &29 · OS_FSControl

The purpose of this call is to read the path of the specified directory.

Before calling

R0 54 (&36)

R1 Pointer to buffer to hold pathname

R2 Directory to read (use number)

R3 Pointer to name of filing system

R5 Size of buffer (or 0 to read required size)

On return

R1 Pointer to rest of path, or 0 if directory is not set

R5 (buffer length) minus (length of string (with terminator))

R6 Pointer to special field, terminated with null or “,” (or 0 for none)

Notes

Like other OS_FSControl calls, use this call twice: after the first call (r5 = 0), and on exit r5 = -(bytes needed).

The buffer is filled with pathname.

[RISC OS Open source page](#)

OS_FSControl 55

SWI &29 · OS_FSControl

The purpose of this call is to reads the free space on a large disc or image file that holds a specified object.

Before calling

R0 55 (&37)

R1 Pointer to name of object

On return

R0 Least significant word of free space

R1 Most significant word of free space

R2 Size of largest object that can be created

R3 Least significant word of disc size

R4 Most significant word of disc size

Notes

This call is only implemented in RISC OS 3.60 and later – if an error is returned try OS_FSControl 49 for the prior implementation.

'Large' disc needs to be defined.

[RISC OS Open source page](#)

OS_FSControl 56

SWI &29 · OS_FSControl

The purpose of this call is to read the defects list for a large disc.

Before calling

R0 56 (&38)

R1 Pointer to name of object

R2 Pointer to buffer

R3 Length of buffer

On return

R0 Preserved

R1 Number of defects placed in the buffer

Notes

The buffer consists of 64 bit values (LSW first). If an error is returned try OS_FSControl 41 which is used to return information for smaller discs.

[RISC OS Open source page](#)

OS_FSControl 57

SWI &29 · OS_FSControl

The purpose of this call is to map out a defect from a large disc or image.

Before calling

R0 57 (&39)

R1 Pointer to name of object

R2 Least significant word of defect address

R3 Most significant word of defect address

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

This call uses 64 bit values, and thus can support large discs.

If an error is returned try OS_FSControl 42 which is used for smaller discs or images.

[RISC OS Open source page](#)

OS_FSControl 58

SWI &29 · OS_FSControl

This call enumerates open handles and returns the same details as OS_Args 254, however it avoids the need to make assumptions about the total number of handles available in the system.

Before calling

R0 58 (&3A)

R1 Previous handle, or -1 to start

On return

R0 Stream status word

R1 File handle, or -1 for no more

R2 Filing system information word

Notes

Registers R0 and R2 are preserved if R1 returns 'no more'. Beware that handles could be opened between calls to this SWI.

Added in FileSwitch 2.74 in RISC OS 5.17.

[RISC OS Open source page](#)

OS_FSControl 6

SWI &29 · OS_FSControl

The purpose of this call is to examine a directory.

Before calling

R0 6

R1 Pointer to (wildcarded) filename

On return

R0 Preserved

R1 Preserved

Notes

This command is equivalent of issuing a *Ex command.

If a pointer to "" is used, the current directory is examined.

[RISC OS Open source page](#)

OS_FSControl 7

SWI &29 · OS_FSControl

The purpose of this call is to catalogue a library.

Before calling

R0 7

R1 Pointer to (wildcarded) filename

On return

R0 Preserved

R1 Preserved

Notes

This command is equivalent of issuing a *LCat command.

If a pointer to "" is used, the current library is catalogued.

[RISC OS Open source page](#)

OS_FSControl 8

SWI &29 · OS_FSControl

The purpose of this call is to examine a library.

Before calling

R0 8

R1 Pointer to (wildcarded) filename

On return

R0 Preserved

R1 Preserved

Notes

This command is equivalent of issuing a *LEx command.

If a pointer to "" is used, the current library is examined.

[RISC OS Open source page](#)

OS_FSControl 9

SWI &29 · OS_FSControl

The purpose of this call is to return information on all objects within a directory that match the object in **R1**.

Before calling

R0 9

R1 Pointer to (wildcarded) filename

On return

R0 Preserved

R1 Preserved

Notes

This command is equivalent of issuing a *Info command.

An error is returned if the pathname pointed to is null.

[RISC OS Open source page](#)

OS_GBPB

SWI &0C

The purpose of this call is to read/write a block from/to an open file.

Before calling

R0 Reason code

Other registers dependent on reason code

On return

R0 Preserved

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

Reason Codes

- 1 – Hex #: &01 · Action: Write bytes to given pointer
- 2 – Hex #: &02 · Action: Write bytes to current pointer
- 3 – Hex #: &03 · Action: Read bytes from given pointer
- 4 – Hex #: &04 · Action: Read bytes from current pointer
- 5 – Hex #: &05 · Action: Read name & boot options
- 6 – Hex #: &06 · Action: Read current dir and privilege byte
- 7 – Hex #: &07 · Action: Read library dir and privilege byte
- 8 – Hex #: &08 · Action: Read entries from current dir
- 9 – Hex #: &09 · Action: Read entries from given dir
- 10 – Hex #: &0A · Action: Read entries and info from given dir
- 11 – Hex #: &0B · Action: Read entries and full info from given dir
- 12 – Hex #: &0C · Action: Read entries and full info with filetype from given dir

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 1

SWI &0C · OS_GBPB

The purpose of this call is to write bytes to an open file at a given pointer.

Before calling

R0 1

R1 File handle

R2 Buffer

R3 Bytes to write

R4 File pointer

On return

R0 Preserved

R1 Preserved

R2 Pointer after end of data

R3 Number of bytes not transferred

R4 New file pointer

C flag is clear

Notes

The 'EOF-error-on-next-read' flag is cleared.

Error is generated if invalid file handle, directory, or no write access.

OS_GBPB 2 writes bytes to an open file, but only at the current file pointer.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 10

SWI &0C · OS_GBPB

The purpose of this call is to read entries and file information from a specified directory, and return data in a buffer.

Before calling

R0 10

R1 Pointer to directory name

R2 Pointer to buffer

R3 Number of objects to read

R4 Where to start, 0 for first time

R5 Length of buffer

R6 Wildcarded name to match (if 0 "*" is used)

On return

R0 Preserved

R1 Preserved

R2 Buffer

R3 Number of objects read

R4 Where to continue, or -1 for no more

C flag clear if R3 = 0, else set

Notes

The correct code structure to call this SWI is:

```
repeat  
{
```

```
call OS_GBPB 10
if (R3 <> 0)
{
    process buffer
}
} until (R4 = -1)
```

as it is legal to return R3 = 0 before the end of the directory. Also, do not assume anything about the value in R4 – Filer_Action assumes that it increases by one for each entry read and comes horribly unstuck.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 11

SWI &0C · OS_GBPB

The purpose of this call is to read entries and file information from a specified directory, and return data in a buffer (extended version).

Before calling

R0 11

R1 Pointer to directory name

R2 Pointer to buffer

R3 Number of objects to read

R4 Where to start, 0 for first time

R5 Length of buffer

R6 Wildcarded name to match (if 0 "*" is used)

On return

R0 Preserved

R1 Preserved

R2 Buffer

R3 Number of objects read

R4 Where to continue, or -1 for no more

C flag clear if R3 = 0, else set

Notes

The correct code structure to call this SWI is:

```
repeat
{
    call OS_GBPB 11
    if (R3 <> 0)
    {
        process buffer
    }
} until (R4 = -1)
```

as it is legal to return R3 = 0 before the end of the directory. Also, do not assume anything about the value in R4.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 12

SWI &0C · OS_GBPB

The purpose of this call is to read entries and file information (including filetype info) from a specified directory, and return data in a buffer.

Before calling

R0 12

R1 Pointer to directory name

R2 Pointer to buffer

R3 Number of objects to read

R4 Where to start, 0 for first time

R5 Length of buffer

R6 Wildcarded name to match (if 0 "*" is used)

On return

R0 Preserved

R1 Preserved

R2 Buffer

R3 Number of objects read

R4 Where to continue, or -1 for no more

C flag clear if R3 = 0, else set

Notes

The correct code structure to call this SWI is:

```
repeat
{
call OS_GBPB 12
if (R3 <> 0)
{
process buffer
}
} until (R4 = -1)
```

as it is legal to return R3 = 0 before the end of the directory. Also, do not assume anything about the value in R4.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 2

SWI &0C · OS_GBPB

The purpose of this call is to write bytes to an open file at the current file pointer.

Before calling

R0 2

R1 File handle

R2 Buffer

R3 Bytes to write

On return

R0 Preserved

R1 Preserved

R2 Pointer after end of data

R3 Number of bytes not transferred

R4 New file pointer

C flag is clear

Notes

The 'EOF-error-on-next-read' flag is cleared.

Error is generated if invalid file handle, directory, or no write access.

OS_GBPB 1 writes bytes to an open file, and allows to specify the file pointer.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 3

SWI &0C · OS_GBPB

The purpose of this call is to read bytes from an open file at a given file pointer.

Before calling

R0 3

R1 File handle

R2 Pointer to buffer

R3 Bytes to read

R4 File pointer

On return

R0 Preserved

R1 Preserved

R2 Pointer after end of data

R3 Number of bytes not transferred

R4 New file pointer

C flag is clear if R3=0, else it is set

Notes

The 'EOF-error-on-next-read' flag is cleared.

Error is generated if invalid file handle, directory, or no read access.

OS_GBPB 4 read bytes from an open file at the current file pointer.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 4

SWI &0C · OS_GBPB

The purpose of this call is to read bytes from an open file at the current file pointer.

Before calling

R0 4

R1 File handle

R2 Pointer to buffer

R3 Bytes to read

On return

R0 Preserved

R1 Preserved

R2 Pointer after end of data

R3 Number of bytes not transferred

R4 New file pointer

C flag is clear if R3=0, else it is set

Notes

The 'EOF-error-on-next-read' flag is cleared.

Error is generated if invalid file handle, directory, or no read access.

OS_GBPB 3 read bytes from an open file at a given file pointer.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 5

SWI &0C · OS_GBPB

The purpose of this call is to read the name of the disc which contains the current directory, and its boot option.

Before calling

R0 5

R2 Buffer

On return

R0 Preserved

R2 Preserved

C flag corrupted

Notes

The data returned in the buffer is in the following form:

<name length byte><disc name><boot option byte> of disc containing the current directory.

The names are stored without termination byte.

OS_GBPB 6 reads the Currently Selected Directory (CSD), and privilege status in relation to that directory.

OS_GBPB 7 reads the name of the library directory, and privilege status in relation to that directory.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 6

SWI &0C · OS_GBPB

The purpose of this call is to read the Currently Selected Directory (CSD), and privilege status in relation to that directory.

Before calling

R0 6

R2 Buffer

On return

R0 Preserved

R2 Preserved

C flag corrupted

Notes

The data returned in the buffer is in the following format.

<0><name length byte><current directory name><privileged byte> of the currently selected directory.

The privileged byte is either:

- &00 indicates 'owner' status, and allows to create or delete objects in the directory
- &FF indicates 'public' status, and prohibits the creating or deleting of objects

ADFS and other FileCore based systems, the privileged status value returned is always &00 i.e owner status.

The names are stored without termination byte.

OS_GBPB 5 reads the name of the disc which contains the current directory, and its boot option.

OS_GBPB 7 reads the name of the library directory, and privilege status in relation to that directory.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 7

SWI &0C · OS_GBPB

The purpose of this call is to read the name of the library directory, and privilege status in relation to that directory.

Before calling

R0 7

R2 Buffer

On return

R0 Preserved

R2 Preserved

C flag corrupted

Notes

The data returned in the buffer is in the following format.

<0><name length byte><library directory name><privileged byte> of the library directory.

The names are stored without termination byte.

OS_GBPB 5 reads the name of the disc which contains the current directory, and its boot option.

OS_GBPB 6 reads the Currently Selected Directory (CSD), and privilege status in relation to that directory.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 8

SWI &0C · OS_GBPB

The purpose of this call is to read entries from the current directory on the temporary filing system.

Before calling

R0 8

R2 Buffer address

R3 Number of object names to read from directory

R4 Offset of first item to be read in directory, or 0 to start

On return

R0 Preserved

R2 Preserved

R3 Number of objects asked for but not read

R4 Next offset into directory

C flag clear if R3 = 0, else set

Notes

File names are returned in the area of memory specified in R2. The format of the returned data is:

Size / Description

1 byte – Description: Length of first object name

As specified – Description: First object name in ASCII

– – Description: ... repeated as specified in R3

On exit R4 contains the value which should be used on the next call. There is no guarantee that the number of objects you asked for will be read. This is because of the external constraints some filing systems may impose. To ensure reading all the entries you want to, this call should be repeated until R4 = -1.

This call is only provided for backward compatibility, and should not be used in any new software. This is because the length of the buffer is not communicated, so a buffer overflow could easily occur if the folder contains filenames which are longer than your code expects. Instead of using this SWI, you can use OS_GBPB 9 with a directory string of "@".

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 9

SWI &0C · OS_GBPB

The purpose of this call is to read entries from a specified directory, and return a buffer filled with a list of null-terminated strings consisting of the matched names.

Before calling

R0 9

R1 Pointer to directory name

R2 Pointer to buffer

R3 Number of objects to read
R4 Where to start, 0 for first time
R5 Length of buffer
R6 Wildcarded name to match (if 0 "*" is used)

On return

R0 Preserved
R1 Preserved
R2 Preserved
R3 Number of objects read
R4 Where to continue, or -1 for no more
C flag clear if R3 = 0, else set

Notes

This SWI reads filename entries from the specified directory, returning a group of 0-terminated strings in the buffer. The correct code structure to call this SWI is:

```
repeat
{
call OS_GBPB 9
if (R3 <> 0)
{
process buffer
}
} until (R4 = -1)
```

as it is legal to return R3 = 0 before the end of the directory. Also, do not assume anything about the value in R4.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GenerateError

SWI &2B

The purpose of this call is to generate an error and invoke the error handler.

Before calling

R0 Pointer to error block

On return

R0 Only returns (with V flag on) if X was set

Notes

The X form may appear to be completely useless as it is effectively a no-op!

However, it has at least 3 uses

- Terminated string conversion in BASIC
- Detecting whether OS_CallASWI is available by calling OS_GenerateError via OS_CallASWI and seeing what comes back in R0

- Force a callback to occur

[RISC OS Open source page](#)

OS_GenerateEvent

SWI &22

The purpose of this call is to generate an event.

Before calling

R0 Event number

R1 Event parameters

On return

R0 Preserved

R1 Preserved

Notes

The event vector will only be called if the event number given in R0 has previously been enabled using OS_Byte 14.

[RISC OS Open source page](#)

OS_GetEnv

SWI &10

The purpose of this call is to read environment parameters.

Before calling

--

On return

R0 Address of *command (read only)

R1 RAM limit

R2 Pointer to 5-byte time of program start

Notes

The string pointed to by R0 is a copy of the command line used to start the program and can be passed to OS_ReadArgs to read the parameters.

The values in R0-R2 are only guaranteed to persist until the next program is run and so this call should be used at the start of a program and always before the first Wimp_Poll.

The values returned by this SWI in R0 and R2 can be set with OS_WriteEnv, the RAM limit is set with OS_ChangeEnvironment with a handle number of zero.

[RISC OS Open source page](#)

OS_GSInit

SWI &25

The purpose of this call is to initialise registers for use by OS_GSRead.

Before calling

R0 String to translate

R2 Flags

On return

R0 Value for OS_GSRead

R1 1st non-blank character

R2 Value for OS_GSRead

Notes

OS_GSInit is one of the string routines which are used by the operating system CLI to process text strings.

This SWI is followed by repeated calls to OS_Read.

One of the advantages of these routines is that they enable the use the character '|' to introduce control characters which would otherwise be difficult to enter directly from the keyboard.

OS_GSInit also returns the first non-blank character in the string. However, this is not necessarily the same as the output from the first OS_GSRead, since OS_GSInit doesn't perform any expansion.

[RISC OS Open source page](#)

OS_GSRead

SWI &26

The purpose of this call is to return a character from a string which has been initialised by OS_GSInit.

Before calling

R0 From last OS_GSInit or OS_GSRead

R2 From last OS_GSInit or OS_GSRead

On return

R0 Updated for next call to OS_GSRead

R1 Translated character

R2 Updated for next call to OS_GSRead

C flag set if end-of-string

Notes

This SWI reads and translates a string, one character at a time. Before first call, OS_GSInit must be called. To do whole string, use OS_GSTrans.

[RISC OS Open source page](#)

OS_GSTrans

SWI &27

The purpose of this call is to provide the same functionality as a OS_GSInit call, followed by repeated calls to OS_GSRead.

Before calling

R0 Pointer to string, terminated by 10, 13, or 0

R1 Pointer to destination buffer

R2 Buffer size + flags

On return

R0 Pointer to terminator+1

R1 Pointer to buffer, or 0

R2 Characters converted

C flag set if buffer overflow

Notes

It reads and translates a whole string.

[RISC OS Open source page](#)

OS_Hardware

SWI &7A · Hardware Abstraction Layer

This SWI provides access to the hardware layer. Conceptually, it is similar to accessing the hardware registers directly in earlier versions of RISC OS - wherever possible OS routines should be used in preference. This call is primarily designed for the use of device drivers - for example the PS2Driver module makes PS2 hardware calls using this interface.

Before calling

R8 Reason code (bits 0-7) and flags (bits 8-31)

- Other register meanings dependent on R8

On return

- Depends on R8

In practice

Making hardware calls to devices normally managed by the Kernel is liable to cause the same problems as poking the hardware. However, making hardware calls is of course preferable to actually accessing the hardware directly. Use this interface with caution.

Reason Codes

/ Hex / Meaning

0 – Hex: &00 · Meaning: Call HAL routine

1 – Hex: &01 · Meaning: Get address of HAL routine

- 2 – Hex: &02 · Meaning: Add HAL device
- 3 – Hex: &03 · Meaning: Remove HAL device
- 4 – Hex: &04 · Meaning: Enumerate HAL devices, reverse order
- 5 – Hex: &05 · Meaning: Enumerate HAL devices, chronological order

[RISC OS Open source page](#)

OS_Hardware 0

SWI &7A · Hardware Abstraction Layer

This SWI calls a HAL routine. HAL routines internally are ATPCS, so R0-R3 are passed in as a1-a4, and R4-R7 are pushed on to the stack. The a1-a4 on exit from the routine are passed back in R0-R3.

Before calling

R0-R7 Parameters for hardware routine

R8 0

R9 Hardware call number

On return

R0-R3 Updated by call

R4-R9 Preserved

In practice

If the HAL routine is not available, an error is returned. Actual HAL routines do not return RISC OS errors – any possible failure will be indicated in a call-specific manner.

Notes

Error &6C1 (Hardware call not available) is returned if the requested call is unknown by this kernel/HAL version, or is a null entry (Null offset in the HAL entry table). Entries which are present but do nothing (i.e. implemented as 'MOV pc,lr') do not return an error.

[RISC OS Open source page](#)

OS_Hardware 1

SWI &7A · Hardware Abstraction Layer

This call looks up the address of a HAL routine. If it does not exist, an error is returned. Otherwise, the address of the routine is returned in R0. Calls made to the routine should be in a privileged mode, with R9 (sb) set to the static base value returned by this SWI. Refer to the HAL calling standards documentation for more details of calling conditions.

Before calling

R8 1

R9 Hardware call number

On return

R0 Routine address

R1 Static base value for routine

Notes

Error &6C1 (Hardware call not available) is returned if the requested call is unknown by this kernel/HAL version, or is a null entry (Null offset in the HAL entry table). Entries which are present but do nothing (i.e. implemented as 'MOV pc,lr') do not return an error.

[RISC OS Open source page](#)

OS_Hardware 2

SWI &7A · Hardware Abstraction Layer

Declare a new device to the system. If a device with that address is already registered, an attempt will be made to first remove it.

Before calling

R0 Pointer to device descriptor

R8 2

On return

- All registers preserved

Notes

To remove a device, call OS_Hardware 3.

Instead of calling SWIs, HAL code should use RISCOS_AddDevice to register devices.

Once the device has been added to the system, Service_Hardware 0 will be issued to notify other programs.

[RISC OS Open source page](#)

OS_Hardware 3

SWI &7A · Hardware Abstraction Layer

Removes a device from the system.

Before calling

R0 Pointer to device descriptor

R8 3

On return

- All registers preserved

Notes

Before the device is removed, the kernel will issue Service_Hardware 1 to check if any programs object to the removal (e.g. device is still in use).

To add a device, call OS_Hardware 2.

[RISC OS Open source page](#)

OS_Hardware 4

SWI &7A · Hardware Abstraction Layer

Enumerates the device list, searching for devices of a particular type.

Before calling

R0 Bits 0-15 = type to match

Bits 16-31 = maximum major version number to match

R1 0 to start an enumeration, else preserved from last call

R8 4

On return

R1 -1 if there are no (more) devices of this type

R2 Pointer to device descriptor (undefined if **R1** = -1)

- Other registers preserved

Notes

Devices will be returned in the opposite order to the order they were registered. Use OS_Hardware 5 if you want the devices to be returned in the same order as the order they were registered.

Prior to RISC OS 5.19, this SWI has a bug which can lead to a crash if it encounters a HAL device of the correct type but with a version number newer than that indicated in **R0**.

[RISC OS Open source page](#)

OS_Hardware 5

SWI &7A · Hardware Abstraction Layer

Enumerates the device list, searching for devices of a particular type.

Before calling

R0 Bits 0-15 = type to match

Bits 16-31 = maximum major version number to match

R1 0 to start an enumeration, else preserved from last call

R8 5

On return

R1 -1 if there are no (more) devices of this type

R2 Pointer to device descriptor (undefined if **R1** = -1)

- Other registers preserved

Notes

Devices will be returned in the same order as the order they were registered. Use OS_Hardware 4 if you want the devices to be returned in the opposite order.

This call first became available in RISC OS 5.19.

[RISC OS Open source page](#)

OS_Heap

SWI &1D

The purpose of this call is to perform various operations on the heap.

Before calling

R0 Reason code

R1 Pointer to heap

All other registers are dependent on reason code

On return

R0 Preserved

R1 Preserved

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0. It is not yet possible to compact the heap using this call.

Reason Codes

0 – Hex #: &00 · Action: Initialise heap

1 – Hex #: &01 · Action: Describe heap

2 – Hex #: &02 · Action: Claim block

3 – Hex #: &03 · Action: Release block

4 – Hex #: &04 · Action: Change size of block

5 – Hex #: &05 · Action: Change size of heap

6 – Hex #: &06 · Action: Read block size

7 – Hex #: &07 · Action: Get area aligned

8 – Hex #: &08 · Action: Get area skew-aligned

[RISC OS Open source page](#)

OS_Heap 0

SWI &1D · OS_Heap

The purpose of this call is to initialise the heap, and set it to a specific size.

Before calling

R0 0

R1 Pointer to heap (word aligned)

R3 Size of heap (multiple of 4)

On return

R0 Preserved

R1 Preserved

R3 Preserved

[RISC OS Open source page](#)

OS_Heap 1

SWI &1D · OS_Heap

The purpose of this call is to read the current status of a heap block.

Before calling

R0 1

R1 Pointer to heap

On return

R0 Preserved

R1 Preserved

R2 Largest free space

R3 Total free space

[RISC OS Open source page](#)

OS_Heap 2

SWI &1D · OS_Heap

The purpose of this call is to claim a section of the heap.

Before calling

R0 2

R1 Pointer to heap

R3 Size to claim

On return

R0 Preserved

R1 Preserved

R2 Pointer to block, or 0 if failed

R3 Preserved

[RISC OS Open source page](#)

OS_Heap 3

SWI &1D · OS_Heap

The purpose of this call is to release a previously claimed section of the heap.

Before calling

R0 3

R1 Pointer to heap

R2 Pointer to block

On return**R0** Preserved**R1** Preserved**R2** Preserved[RISC OS Open source page](#)

OS_Heap 4

SWI &1D · OS_Heap

The purpose of this call is to change the size of a claimed section of the heap.

Before calling**R0** 4**R1** Pointer to heap**R2** Pointer to block**R3** Size increase, or -ve for decrease**On return****R0** Preserved**R1** Preserved**R2** Pointer to new block**R3** Preserved[RISC OS Open source page](#)

OS_Heap 5

SWI &1D · OS_Heap

The purpose of this call is to change the size of the heap.

Before calling**R0** 5**R1** Pointer to heap**R3** Size increase, or -ve for decrease**On return****R0** Preserved**R1** Preserved**R3** Actual change in size**Notes**

If the heap cannot be shrunk by the required amount an error will be returned.

[RISC OS Open source page](#)

OS_Heap 6

SWI &1D · OS_Heap

The purpose of this call is to return the size of a claimed block.

Before calling

R0 6

R1 Pointer to heap

R2 Pointer to block

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Size of block (including header)

[RISC OS Open source page](#)

OS_Heap 7

SWI &1D · OS_Heap

The purpose of this call is to claim an aligned section of the heap. Use the alignment value to guarantee the minimum alignment of the returned pointer. Use the boundary value to guarantee that the allocation doesn't cross from one section of memory to another. For example, if you were to allocate a block of memory with an alignment of 32 and boundary of 4096, then the returned pointer is guaranteed to be at least 32 byte aligned, and all the data will reside within the same (4KB) page.

Before calling

R0 7

R1 Pointer to heap

R2 Required alignment, 0 for none

R3 Size to claim

R4 Required boundary, 0 for none

On return

R0 Preserved

R1 Preserved

R2 Pointer to block, or 0 if failed

R3 Preserved

R4 Preserved

In practice

Alignment and boundary values must be powers of two. The boundary value, if specified, must be greater than or equal to the block size.

If a block is resized using OS_Heap 4, there is no guarantee that the updated block will satisfy the original alignment and boundary constraints.

Notes

This reason code first became available in RISC OS 5.17

[RISC OS Open source page](#)

OS_Heap 8

SWI &1D · OS_Heap

The purpose of this call is to claim an aligned section of the heap. Its operation is the same as OS_Heap 7, with the exception that the alignment and boundary calculations are performed as if the heap base address was R5, not R1. E.g. the returned pointer will satisfy the rule that $(R2 - R1 + R5) \text{ MOD alignment} = 0$. Calling with $R5 = R1$ will result in the same behaviour as OS_Heap 7.

Before calling

R0 8

R1 Pointer to heap

R2 Required alignment, 0 for none

R3 Size to claim

R4 Required boundary, 0 for none

R5 Address to align relative to

On return

R0 Preserved

R1 Preserved

R2 Pointer to block, or 0 if failed

R3 Preserved

R4 Preserved

R5 Preserved

In practice

Alignment and boundary values must be powers of two. The boundary value, if specified, must be greater than or equal to the block size.

If a block is resized using OS_Heap 4, there is no guarantee that the updated block will satisfy the original alignment and boundary constraints.

Notes

This reason code first became available in RISC OS 5.27

[RISC OS Open source page](#)

OS_HeapSort

SWI &4F

This SWI is deprecated, because the use of flag bits in address values makes it unsafe to use on systems with 32bit logical memory maps. Please use OS_HeapSort32 instead.

Before calling

R0 Number of elements

R1 Pointer to word-array + flags

R2 Sort type or comparison routine

Optional:

R3 Workspace pointer

R4 Array of actual objects

R5 Size of object in R4

R6 Workspace of R5 bytes (if R5>16k)

On return

All registers preserved

In practice

The purpose of this call is to perform a fast and memory efficient sort on a list of objects.

[RISC OS Open source page](#)

OS_HeapSort32

SWI &7F

The purpose of this call is to perform a fast and memory efficient sort on a list of objects.

Before calling

R0 Number of elements

R1 Pointer to word-array

R2 Sort type or comparison routine

R7 Flags

Optional:

R3 Workspace pointer

R4 Array of actual objects

R5 Size of object in R4

R6 Workspace of R5 bytes (if R5>16k)

On return

All registers preserved

Notes

The CallASWI module allows this SWI to be used with earlier versions of RISC OS (before 5.00). The earlier version of this SWI OS_HeapSort is also available.

[RISC OS Open source page](#)

OS_IICOp

SWI &7B

The purpose of this call is to allow software to perform transfers on the IIC buses that are present in the system. See the `iic_transfer` documentation for details on the transfer list format and behaviour.

Before calling

R0 Pointer to list of `iic_transfer` structures

R1 Bits 0-23: Number of `iic_transfer` structures

Bits 24-31: IIC bus number

On return

- All registers preserved
- Transfer list updated, if applicable

Notes

To determine how many IIC buses are present on the system, use `OS_ReadSysInfo 14`.

`OS_IICOp` is re-entrant. Transfers are queued on a per-bus basis. Thus if `OS_IICOp` is re-entered then it will first wait for all pending transfers on the requested bus to complete before starting the set of transfers requested by the new call.

`OS_IICOp` will enable interrupts for the duration of the transfer.

This SWI is an improved, more powerful version of the older `IIC_Control` SWI.

[RISC OS Open source page](#)

OS_InstallKeyHandler

SWI &3E

This call allows you to get or set values relating to the current keyboard handler.

Before calling

R0 0 = read address of current keyboard handler

1 = read current Keyboard ID

>1 = set address of new keyboard handler

On return

R0 Address of new/old keyboard handler, or Keyboard ID

[RISC OS Open source page](#)

OS_IntOff

SWI &14

The purpose of this call is to disable interrupts.

Before calling

--

On return

--

Notes

After disabling interrupts, it returns to the caller with processor mode and registers unchanged. It does this by switching the corresponding bits in the PSR off.

Interrupts are enabled by calling OS_IntOn.

[RISC OS Open source page](#)

OS_IntOn

SWI &13

The purpose of this call is to enable interrupts.

Before calling

--

On return

--

Notes

After enabling interrupts, it returns to the caller with processor mode and registers unchanged. It does this by switching the corresponding bits in the PSR on.

Interrupts are disabled by calling OS_IntOff.

[RISC OS Open source page](#)

OS_LeaveOS

SWI &7C

The purpose of this call is to set the processor to User mode.

Before calling

--

On return

--

Notes

Interrupt and Fast Interrupt status is unaltered.

The advantage of this call is that it will cause callbacks to occur which can massively improve applications where only a quick bit of work is required in supervisor mode, yet it may be some time before any callbacks are triggered.

This SWI was introduced with RISC OS 5.00. For older machines, the CallASWI module (version 0.10) provides an implementation of the SWI. See also OS_EnterOS.

[RISC OS Open source page](#)

OS_Memory

SWI &68

The purpose of this call is to perform various operations for memory management.

Before calling

R0 Reason code (bits 0 – 7),

Flags (bits 8 – 31) which are reason code specific

All other registers are dependent on reason code

On return

R0 Preserved

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

The flags (passed by R0 on entry) are specific to each reason code.

Reason Codes

- 0 – Hex #: &00 · Action: General page block operations (32 bit)
- 1 – Hex #: &01 · Action: Reserved for system use
- 2 – Hex #: &02 · Action: Reserved for system use
- 3 – Hex #: &03 · Action: Reserved for system use
- 4 – Hex #: &04 · Action: Reserved for system use
- 5 – Hex #: &05 · Action: Reserved for system use
- 6 – Hex #: &06 · Action: Read the size of the physical memory arrangement table
- 7 – Hex #: &07 · Action: Read the physical memory arrangement table
- 8 – Hex #: &08 · Action: Read the amount of a specified sort of memory available
- 9 – Hex #: &09 · Action: Read controller presence and base logical address
- 10 – Hex #: &0A · Action: Free pool lock
- 11 – Hex #: &0B · Action: Reserved for system use
- 12 – Hex #: &0C · Action: Recommend page
- 13 – Hex #: &0D · Action: Map in IO permanent
- 14 – Hex #: &0E · Action: Map in IO temporary
- 15 – Hex #: &0F · Action: Map out temporary IO
- 16 – Hex #: &10 · Action: Memory areas
- 17 – Hex #: &11 · Action: Memory access privileges
- 18 – Hex #: &12 · Action: Find access privilege
- 19 – Hex #: &13 · Action: Prepare for DMA

- 20 – Hex #: &14 · Action: Compatibility settings
- 21 – Hex #: &15 · Action: Map in IO permanent from 64 bit space
- 22 – Hex #: &16 · Action: Map in IO temporary from 64 bit space
- 23 – Hex #: &17 · Action: Reserve RAM pages
- 24 – Hex #: &18 · Action: Check memory access
- 25 – Hex #: &19 · Action: Read controller presence and base physical address
- 64 – Hex #: &40 · Action: General page block operations (64 bit)
- 65 – Hex #: &41 · Action: Logical to physical address translation

[RISC OS Open source page](#)

OS_Memory 0

SWI &68 · OS_Memory

The purpose of this call is to convert between different memory spaces used to specify addresses in a page block, and alter their cache-ability.

Before calling

R0 0, and flags,

R1 Pointer to page block list

R2 Number of entries in page block list

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

The addresses must be in RAM; specifically, they must all correspond to pages which the OS has assigned a valid physical page number to. E.g. I/O memory which has been mapped by OS_Memory 13 will not be recognised.

Addresses need not be page aligned. If a page is made uncacheable, the relevant portion of the CPU cache will be flushed. If there is an error processing any page in the block, the call has no effect on any pages.

Because the page block format is restricted to 32 bit addresses, attempting to look up the physical address of a page which has a physical address of 2^{32} or higher will cause the call to fail with an error. To avoid this, OS_Memory 64 should be used instead.

[RISC OS Open source page](#)

OS_Memory 10

SWI &68 · OS_Memory

This call is for internal use by the operating system and must not be used by user software.

Before calling

R0 10, and flags

R1 0 = Resume background operations that may alter free pool mapping

1 = Suspend background operations that may alter free pool mapping

On return**R0** Preserved**R1** Previous state[RISC OS Open source page](#)

OS_Memory 12

SWI &68 · OS_Memory

This call can be used by dynamic area handlers which require blocks of physically contiguous RAM. If the call fails to find an area of the requested size/alignment, an error will be returned.

Before calling**R0** 12, and flags**R1** Size of physically contiguous RAM region required (bytes)**R2** log2 of required alignment of base of region (eg. 12 = 4k, 20 = 1M)**R4-R5** Lowest acceptable physical address (inclusive) (if bit 9 of R0 set)**R6-R7** Highest acceptable physical address (inclusive) (if bit 9 of R0 set)**On return****R0-R2** Preserved**R3** Page number of first page of recommended region that could be grown as specific pages by dynamic area handler (only guaranteed if grow is next page claiming operation)**In practice**

On success, the N consecutive physical page numbers starting with R3 are guaranteed to point to N pages with consecutive physical addresses which meet the requirements (where N is R1 rounded up to a page count).

Notes

Memory which has been claimed for exclusive use (e.g. via Dynamic Area PreGrow handlers, OS_DynamicArea 21 "lock" flag, or OS_Memory 23) will not be recommended.

RISC OS 5.29+ allows a physical address range to be specified in R4-R7 to restrict the search to certain ranges of physical memory. The high address in R6-R7 restricts the end of the memory block; e.g. when allocating a buffer for use by a DMA controller it should be set to the maximum permissible physical address that the controller can access. If R4-R7 is not specified, the default address range is for the first 4GB of the address space (i.e. R4,R5,R7=0 and R6=&FFFFFFFF). Note that older OS versions ignore this flag (acting as if the default "first 4GB" restriction was in place), so care must be taken if you want to restrict the recommendation to a subset of the first 4GB.

[RISC OS Open source page](#)

OS_Memory 13

SWI &68 · OS_Memory

The purpose of this call is to permanently map in a region of IO memory at run-time.

Before calling

R0 13 (&0D), and flags

R1 Physical address to map in

R2 Size to map in

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Logical address assigned

In practice

For access to 64-bit physical addresses, see OS_Memory 21.

[RISC OS Open source page](#)

OS_Memory 14

SWI &68 · OS_Memory

The purpose of this call is to temporarily map in one megabyte of IO memory at run-time. The megabyte that gets mapped in will be megabyte-aligned in both the logical and physical address spaces; i.e. by accessing (R2 AND &FFF00000) you will be accessing physical address (R1 AND &FFF00000). Although the base of the mapping is on a megabyte boundary, the value returned in R2 will have had an offset applied to it so that it corresponds to the exact address requested in R1.

Before calling

R0 14 (&0E), and flag

R1 Physical address to map in

On return

R0 Preserved

R1 Preserved

R2 Logical address corresponding to R1

R3 Reference key (required when releasing)

In practice

Because of the megabyte alignment of the mapping, if you are reading multiple bytes from R2 you must be careful not to cross a megabyte boundary.

Notes

All calls to OS_Memory 14 and OS_Memory 22 will manipulate a single shared region of the logical address map. Therefore, only one region can be mapped in at once – it is impossible to use this call to concurrently access regions of physical memory which lie in different megabytes.

This also means that this call must not be used for any long-life mappings – i.e. it is not acceptable to map in a region during a module's initialisation and continue to access that region throughout the life of the module. If two such modules using the above method were left running at the same time then they would inadvertently access each other's memory, leading to crashes, data corruption, or other improper behaviour. Instead, you must map in the region each time your code gains control of the system.

Because there is only one single megabyte region available, it is also vitally important to use OS_Memory 15 to restore the state of the region once you have finished using it. This makes sure that interrupt handlers, callbacks, etc. don't corrupt the mapping of any foreground process.

If you want to access multiple different regions in sequence, then it is only necessary to call OS_Memory 15 once, with the value of R3 that was returned to you the first time you called OS_Memory 14. This will restore the region to the state it was before your first call.

Depending on the OS version and host hardware, the size of the mapping created may be more than 1MB. However this is an implementation detail and your code should not rely on that behaviour.

For access to 64-bit physical addresses, OS_Memory 22 can be used.

[RISC OS Open source page](#)

OS_Memory 15

SWI &68 · OS_Memory

The purpose of this call is to free an area of IO memory previously mapped in at run-time.

Before calling

R0 15 (&0F), all flag bits reserved

R1 Reference key (assigned when claimed)

On return

R0 Preserved

R1 Preserved

Notes

The reference key is assigned when temporarily mapping in a region using OS_Memory 14 or OS_Memory 22.

[RISC OS Open source page](#)

OS_Memory 16

SWI &68 · OS_Memory

The purpose of this call is to return information on various areas of the memory map which are not dynamic areas, so that, for example, the task manager can show the correct amount of memory allocated to the system areas.

Before calling

R0 16 (&10), and flags

On return

R0 Preserved

R1 Base address of area

R2 Address space allocated for area (bytes, multiple of page size)

R3 Actual memory used by area (bytes, multiple of page size)

In practice

Returned values are rounded to page boundaries. If an area is not present, or is contained wholly within another area covered by this call, then zero will be returned for R1-R3. An error will be returned for any area not recognised by the OS version.

[RISC OS Open source page](#)

OS_Memory 17

SWI &68 · OS_Memory

This call allows you to enumerate the available access privileges and read their permissions.

Before calling

R0 17 (reason code)

R1 Access privilege to start search from (0 to begin enumeration)

On return

R0 Preserved

R1 Access privilege number (-1 if at end of enumeration)

R2 Permissions: (0 if at end of enumeration)

Bit 0: Executable in user mode

Bit 1: Writable in user mode

Bit 2: Readable in user mode

Bit 3: Executable in privileged modes

Bit 4: Writable in privileged modes

Bit 5: Readable in privileged modes

Bits 6+: Reserved

In practice

On exit, R1 will either be -1 or a valid AP number. If it is an AP number then R2 will contain the permission flags associated with that AP. Note that the AP number returned may be different to the AP number that was provided on entry (e.g. if the AP number provided on entry was not valid)

Notes

To enumerate all available AP values, you must increment the returned R1 value by one on each call, e.g.:

```
ap% = -1
```

```
REPEAT
```

```
ap% += 1
```

```
SYS "OS_Memory",17,ap% TO ,ap%,flags%
```

```
IF ap% <> -1 THEN PRINT ap%," ";flags%
```

UNTIL ap% = -1

This call was introduced in RISC OS 5.23. For the permissions supported by earlier RISC OS versions, see Memory Map Page Access.

OS_Memory 18 can be used to find an AP value that most closely matches a given set of permissions.

[RISC OS Open source page](#)

OS_Memory 18

SWI &68 · OS_Memory

This call allows you to find the access privilege number that provides the closest match to the permissions you specify.

Before calling

R0 18 (reason code)

R1 Desired permission flag values:

Bit 0: Executable in user mode

Bit 1: Writable in user mode

Bit 2: Readable in user mode

Bit 3: Executable in privileged modes

Bit 4: Writable in privileged modes

Bit 5: Readable in privileged modes

Bits 6+: Reserved

R2 Mask of which bits in R1 are specified

On return

R0 Number of the closest access privilege, or error

R2 Permission flags of that access privilege. Preserved on error.

In practice

The searches for an AP which satisfies the constraint of $((\text{permissions AND R2}) == \text{R1})$, and which grants the least extra permissions over those that were requested. E.g. to only allow AP values where user mode code has no access, bits 0-2 of R1 must be clear and bits 0-2 of R2 must be set.

If the system is unable to find an AP which satisfies the constraints of R1 and R2 then an error will be returned.

Notes

R1 must be a subset of R2, i.e. you must not have a bit set in R1 if the same bit is not set in R2.

Extra permissions are weighted as follows (least acceptable first):

- User write
- User execute
- User read
- Privileged write
- Privileged execute
- Privileged read

This call was introduced in RISC OS 5.23. For the permissions supported by earlier RISC OS versions, see Memory Map Page Access.

OS_Memory 17 can be used to enumerate the available access privileges and read their permissions.

[RISC OS Open source page](#)

OS_Memory 19

SWI &68 · OS_Memory

This call performs address translation and cache maintenance operations, as necessary to allow for DMA to be performed to/from cacheable memory. It's intended to be used as a replacement for the OS_Memory 0/OS_Memory 64 "Enable/disable caching" feature, since there are many issues with that approach on modern systems.

Before calling

R0 19, and flags:

Bit 8: Input function provides physical addresses (=1), else logical (=0)

Bit 9: DMA is writing to RAM

Bit 10: DMA is complete

Bit 11: Use 64 bit physical addresses (RISC OS 5.29+)

Bits 12+: Reserved (set to 0)

R1 R12 value to provide to called functions

R2 Initial R9 value to provide to input function

R3 Pointer to input function

R4 Initial R9 value to provide to output function

R5 Pointer to output function (if bit 10 of R0 clear)

On return

R2 Updated to match last value returned by input function

R4 Updated to match last value returned by output function

- All other registers preserved

In practice

As with OS_Memory 0/64, OS_Memory 19 must be called twice per DMA operation: Once at the start of the operation (with bit 10 of R0 clear), and once at the end of the operation (with bit 10 set). The same list of input addresses ranges must be provided each time.

Key differences from OS_Memory 0 are as follows:

- The cacheability of the memory pages involved in the transfer will not be modified. In some cases this can result in a performance boost, but the main benefits are that it avoids the problems that OS_Memory 0/64 suffers from.
- OS_Memory 19 will attempt to coalesce blocks (while retaining their ordering) so that the output list is as short as possible. Blocks will be coalesced if they are logically and physically contiguous, and have the same flags (i.e. bounce buffer usage). This is particularly useful for multi-megabyte transfers, where hundreds or thousands of pages may be involved in the transfer.
- Instead of requiring the caller to prepare a page list array, a callback function is used to obtain details of regions on-demand. This is a performance/memory optimisation, to avoid the need to make a copy of your address lists in a format which OS_Memory can understand.
- Similarly, address translation results are returned via a callback function rather than by filling in a pre-allocated array.

Operation

OS_Memory 19 operates under the following principles:

- DMA which reads from cacheable memory will require the corresponding cache lines to be cleaned (flushed) beforehand, so that data is written back to memory prior to the DMA reading it.
- DMA which writes to cacheable memory will require the corresponding cache lines to be cleaned beforehand, so that any pending write-back data doesn't overwrite data the DMA is writing. At the end of the operation, the cache will be invalidated again, so that any data that was prefetched (or explicitly loaded) by the CPU will be discarded and the fresh data written by the DMA will be visible. Technically the start of the transfer only needs to perform an Invalidate operation, since if DMA is going to completely overwrite the data it doesn't matter whether the data in the cache gets written back or discarded. However by cleaning the cache, we ensure that data written by the CPU won't get lost if the DMA transfer terminates early. It also provides support for read-modify-write operations.
- If DMA is writing to a cacheable area which isn't cache-line aligned, it will be forced to use a bounce buffer. This is because OS_Memory 19 doesn't assume that you have exclusive ownership of the memory. E.g. if the memory is in the RMA, a program which writes to a heap block which is located in the non-DMA part of the cache line will cause that entire cache line to be written out to memory, potentially overwriting the part of the cache line which had been updated by the DMA.

Under this strategy, it's safe for programs (or other DMA controllers) to read from memory that is being accessed by a DMA read operation. Accessing cacheable memory which is being a DMA write operation is unsafe and should not be avoided (unless you're careful to perform the necessary cache maintenance yourself).

Notes

For the 32 bit version of the call (flag bit 11 is zero), if a logical address maps to a physical address which is larger than 32 bits then that section of the transfer will be forced via a bounce buffer and the returned physical address will be undefined. To avoid this, the 64 bit version of the call should be used (flag bit 11 set). Although there's no direct way of detecting if the 64 bit version of the call is supported, OS_PlatformFeatures 0 bit 21 can be used as an indicator that it's supported / necessary.

Service_PagesUnsafe / Service_PagesUnsafe64 is issued by the OS whenever the physical page associated with a logical address is being "transparently" replaced with another physical page. This can occur as the result of the original physical page being requested by a Dynamic Area PreGrow handler. Programs which perform DMA need to listen out for this service call so that they can pause DMA while the pages are being moved, and then resume DMA (using the new logical physical mapping) once the page move is complete.

Because OS_Memory 0/64 don't perform any operations which might cause pages to be moved in this manner, programs which uses OS_Memory 0/64 only need to start paying attention to Service_PagesUnsafe/PagesUnsafe64 once they've received the results from the OS. But for OS_Memory 19, results are returned to your code in a piecemeal fashion. If your input or output function performs operations which may trigger page movement (e.g. allocating pages from the RMA), extra care is needed in order to deal with Service_PagesUnsafe/PagesUnsafe64 correctly.

If a call to Service_PagesUnsafe / Service_PagesUnsafe64 occurs during the call to OS_Memory, and it affects the pages which are involved in the OS_Memory call:

- If the input function provides physical addresses, the entire OS_Memory operation should be aborted and retried.
- If the input function provides logical addresses, you must either update the information for the segments which have been passed to your output function, or you must abort and restart the entire operation.

If code is written to memory by a DMA operation then you must still call OS_SynchroniseCodeAreas afterwards to ensure the instruction cache has been invalidated.

[RISC OS Open source page](#)

OS_Memory 20

SWI &68 · OS_Memory

This call can be used to examine or control various memory management compatibility settings. Currently the only option that can be controlled relates to the page zero “compatibility page”.

Before calling

R0 Bits 0-7: 20 (reason code)

Bits 8-31: Flags/sub-reason code (reserved, should be zero)

R1 Action:

0 = Disable compatibility page

1 = Enable compatibility page

-1 = Read status of compatibility page

On return

R0 Preserved, or error

R1 New status of compatibility page:

0 = Compatibility page is disabled

1 = Compatibility page is enabled

-1 = Compatibility page is not supported/required

Notes

Historically RISC OS has placed the CPU processor vectors and a certain amount of kernel workspace (“zero page”) at logical address &0. This means that applications or modules which accidentally write to null pointers can corrupt the workspace, crashing the system.

In RISC OS 3.8 the zero page workspace was made read-only in user mode. This prevented most buggy applications from crashing the system by writing to zero page, but provides no protection against buggy modules. It also failed to solve the problem of programs malfunctioning after accidentally reading from null pointers.

In RISC OS 5.23 the OS gained the ability to relocate the processor vectors and zero page workspace to &FFFF0000. This provides full protection against null pointer dereferences, but also revealed many bugs in both the operating system and software, most often relating to “harmless” reads from null pointers. On old systems the code would run without there being any obvious indications of problems, but on new systems the code would crash as soon as it attempted the null pointer dereference, most likely preventing the software from being used at all.

To allow this buggy software to continue to function, the option of a “compatibility page” was introduced. This is a single 4K page of memory located at &0 which is read-only in all processor modes, and contains values that are intended to allow “harmless” null pointer dereferences to work OK, similar to if the real zero page workspace/processor vectors had been accessed. If interpreted as a string, the page also contains a message warning the user that a null pointer dereference has occurred.

On systems with high processor vectors / zero page relocation, OS_Memory 20 allows the user or software to enable/disable the compatibility page, as they desire. It's recommended that software authors disable the compatibility page when testing their software.

On systems with low processor vectors, R1 will be -1 on exit, indicating that the compatibility page is not supported/required (the processor vectors will be occupying the space at &0).

OS_PlatformFeatures 0 and OS_PlatformFeatures 32 can be used to determine the location of the processor vectors.

OS_Memory 16 can also be used to determine the size and state of the compatibility page.

[RISC OS Open source page](#)

OS_Memory 21

SWI &68 · OS_Memory

The purpose of this call is to permanently map in a region of IO memory at run-time.

Before calling

R0 21 (&15), and flags

R1 Physical address to map in (low 32 bits)

R2 Physical address to map in (high 32 bits)

R3 Size to map in

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Logical address assigned

In practice

This call was introduced in RISC OS 5.27, to improve upon the existing OS_Memory 13 call (which can only deal with 32-bit physical addresses). For compatibility with older OS versions, it's recommended to use OS_Memory 13 whenever 32-bit physical addresses are sufficient.

[RISC OS Open source page](#)

OS_Memory 22

SWI &68 · OS_Memory

The purpose of this call is to temporarily map in one megabyte of IO memory at run-time. The megabyte that gets mapped in will be megabyte-aligned in both the logical and physical address spaces; i.e. by accessing (R2 AND &FFF00000) you will be accessing physical address ((R2<<32) + (R1 AND &FFF00000)). Although the base of the mapping is on a megabyte boundary, the value returned in R2 will have had an offset applied to it so that it corresponds to the exact address requested in R1,R2.

Before calling

R0 22 (&16), and flag

R1 Physical address to map in (low 32 bits)

R2 Physical address to map in (high 32 bits)

On return

R0 Preserved

R1 Preserved

R2 Logical address corresponding to R1,R2

R3 Reference key (required when releasing)

In practice

Because of the megabyte alignment of the mapping, if you are reading multiple bytes from R2 you must be careful not to cross a megabyte boundary.

Notes

All calls to OS_Memory 14 and OS_Memory 22 will manipulate a single shared region of the logical address map. Therefore, only one region can be mapped in at once – it is impossible to use this call to concurrently access regions of physical memory which lie in different megabytes.

This also means that this call must not be used for any long-life mappings – i.e. it is not acceptable to map in a region during a module's initialisation and continue to access that region throughout the life of the module. If two such modules using the above method were left running at the same time then they would inadvertently access each other's memory, leading to crashes, data corruption, or other improper behaviour. Instead, you must map in the region each time your code gains control of the system.

Because there is only one single megabyte region available, it is also vitally important to use OS_Memory 15 to restore the state of the region once you have finished using it. This makes sure that interrupt handlers, callbacks, etc. don't corrupt the mapping of any foreground process.

If you want to access multiple different regions in sequence, then it is only necessary to call OS_Memory 15 once, with the value of R3 that was returned to you the first time you called OS_Memory 14/22. This will restore the region to the state it was before your first call.

Depending on the OS version and host hardware, the size of the mapping created may be more than 1MB. However this is an implementation detail and your code should not rely on that behaviour.

This call was introduced in RISC OS 5.27, to improve upon the existing OS_Memory 14 call (which can only deal with 32-bit physical addresses). For compatibility with older OS versions, it's recommended to use OS_Memory 14 whenever 32-bit physical addresses are sufficient.

[RISC OS Open source page](#)

OS_Memory 23

SWI &68 · OS_Memory

The purpose of this call is to reserve a consecutive number of RAM pages, or to release the reservation of a consecutive number of RAM pages. This is the only call which can alter bit 7 of the Memory Page Access Flags.

Before calling

R0 23, and flags:

Bit 8: 0 to reserve, 1 to release reservation

Bits 9+: Reserved (set to 0)

R1 Base page number

R2 Number of pages

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

When reserving a range of pages, if any of the pages are already reserved the operation will fail with an error without modifying the reservation state of any of the pages.

Behaviour of reserved pages

Reserved pages differ from un-reserved pages in the following ways:

- OS_Memory 12 will not recommend a range containing reserved pages
- Dynamic Area PreGrow handlers will not be allowed to add the page to a dynamic area, unless the special "RESV" return value is used
- OS_DynamicArea 21 will be prevented from locking the page into the dynamic area, unless the entry in the page block also sets bit 7 to 1. If bit 7 is not set in the page block, but the page number is set to -2 (indicating the kernel can pick a page), the kernel will attempt to swap the reserved page for an un-reserved page

Reserved pages are intended to be used in situations where a program knows that it needs a certain set of physical pages (e.g. for future use by another bus master), but it doesn't need all of the pages right now. For example, the PCI heap (used for PCI_RAMAlloc & PCI_RAMFree) and the legacy screen dynamic area are both dynamic areas which grow and shrink on demand, but also require physically contiguous memory to operate correctly. It's always been the case that pages which have been explicitly requested by PreGrow handlers won't be recommended by OS_Memory 12, and can't be requested by other dynamic areas. This protects the memory which the PCI heap & screen are currently using. But OS_Memory 12 has no knowledge about what memory they may want to use in the future. So if a third dynamic area comes along which wants physically contiguous pages, OS_Memory 12 may recommend that it uses pages which the PCI module was intending to use. When the third dynamic area makes use of that memory, it will be blocking the PCI heap from growing above a certain size, even if there's plenty of other free memory spare in the system. OS_Memory 23 solves this problem by allowing the dynamic area owners to signify their intent to use the memory (in addition to providing some protection against dynamic areas accidentally using pages which have been reserved by other systems).

[RISC OS Open source page](#)

OS_Memory 24

SWI &68 · OS_Memory

This call examines a given logical address range and returns a set of flags indicating the access rights that programs have when accessing the block of memory. It is essentially a more verbose version of OS_ValidateAddress. However, unlike OS_ValidateAddress, it does not issue Service_ValidateAddress when given an unknown region.

Before calling

R0 24 (&18)

R1 Low address (inclusive)

R2 High address (exclusive)

On return

R0 Preserved

R1 Access flags

R2 Preserved

In practice

This call is supported by RISC OS Select and RISC OS 5.21.

Access flags

- 0 – Meaning: Range is completely readable in user mode
 - 1 – Meaning: Range is completely writable in user mode
 - 2 – Meaning: Range is completely readable in privileged modes
 - 3 – Meaning: Range is completely writable in privileged modes
 - 4 – Meaning: Range is partially readable in user mode
 - 5 – Meaning: Range is partially writable in user mode
 - 6 – Meaning: Range is partially readable in privileged modes
 - 7 – Meaning: Range is partially writable in privileged modes
 - 8 – Meaning: Range is completely physically mapped
 - 9 – Meaning: Range is completely abortable (aborts handled by code) 2
 - 10 – Meaning: Range is completely non-executable in user mode 1
 - 11 – Meaning: Range is completely non-executable in privileged modes 1
 - 12 – Meaning: Range is partially physically mapped
 - 13 – Meaning: Range is partially abortable (aborts handled by code) 2
 - 14 – Meaning: Range is partially non-executable in user mode 1
 - 15 – Meaning: Range is partially non-executable in privileged modes 1
- All other bits are reserved for future expansion.

1 Executability information

RISC OS 5.23 adds support for reporting whether areas are non-executable. This ties in with the introduction of the ability for software to mark areas as non-executable in the page tables via the use of new access privileges.

For an area to be executable, the readable bit must be set and the non-executable bit must be clear. This provides an interpretation of the flags which is backwards-compatible with old OS versions. However it is important to remember that just because an area is reported as being executable, it does not guarantee that there is valid code at that location.

Note that because the flag indicates that areas are non-executable, the flag will be set for any unmapped areas of memory. In order to allow for future flags to be added in a similar manner it is important that software ignores the values of any flags which it does not understand (e.g. do not assume that unmapped memory will return with R1 equal to zero).

2 Under RISC OS 5, the following areas will be flagged as abortable:

- Allocated regions of application space, when lazy task swapping is in use
- The full extent of Abortable Dynamic Areas
- Memory which uses access privilege 1, when the OS is using the “long descriptor” page table format

[RISC OS Open source page](#)

OS_Memory 25

SWI &68 · OS_Memory

The purpose of this call is to check for the presence of a given controller and return its base physical address if present.

Before calling

R0 25, all flag bits reserved

R1 Controller number

On return

R0 Preserved

R1 Controller base physical address

In practice

This call was created by RISCOS Ltd. to complement OS_Memory 9. The two calls use identical controller numbers. There is currently no support for this call under RISC OS 5.

[RISC OS Open source page](#)

OS_Memory 6

SWI &68 · OS_Memory

The purpose of this call is to read the size of the physical memory arrangement table, as returned by OS_Memory 7.

Before calling

R0 6, and all flags bits reserved

On return

R0 Preserved

R1 Table size (in bytes)

R2 Page size (in bytes)

[RISC OS Open source page](#)

OS_Memory 64

SWI &68 · OS_Memory

The purpose of this call is to convert between different memory spaces used to specify addresses in a page block, and alter their cache-ability.

Before calling

R0 64, and flags,

R1 Pointer to 64 bit page block list

R2 Number of entries in page block list

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

This call was introduced in RISC OS 5.29, and is an extended form of OS_Memory 0, to allow full interaction with pages which have logical or physical addresses which can't be represented in the 32 bit page block format. I.e. pages with physical addresses $\geq 2^{32}$ (possible on ARMv7+ machines which support the Large Physical Address Extension), or pages with logical addresses $\geq 2^{32}$ (for future AArch64 versions of RISC OS). For AArch32 versions of RISC OS, your code can decide whether to use OS_Memory 0 or OS_Memory 64 by checking bit 21 of OS_PlatformFeatures 0.

The addresses must be in RAM; specifically, they must all correspond to pages which the OS has assigned a valid physical page number to. E.g. I/O memory which has been mapped by OS_Memory 13 will not be recognised.

Addresses need not be page aligned. If a page is made uncacheable, the relevant portion of the CPU cache will be flushed. If there is an error processing any page in the block, the call has no effect on any pages.

[RISC OS Open source page](#)

OS_Memory 65

SWI &68 · OS_Memory

This call allows software to perform logical to physical address translation. Unlike other calls such as OS_Memory 0, this call works on all types of memory / page table entry (regular 4KB RAM pages, 1MB I/O mappings, etc.)

Before calling

R0 65

R1 Logical address

On return

R0,R1 Physical address

R2 Size/alignment of mapping

Notes

Addresses need not be page aligned. The returned address will include the appropriate sub-page offset such that it is addressing the same byte as the input address.

If there is no mapping for the provided address, a “bad address” error (error no. &FC) will be returned. However, R2 will still be updated to contain the size/alignment of the mapping, to allow code which walks over an address range to skip ahead to the next page.

This call was introduced in RISC OS 5.29.

[RISC OS Open source page](#)

OS_Memory 7

SWI &68 · OS_Memory

The purpose of this call is to read the physical memory arrangement table, into the block pointed to by R1.

Before calling

R0 7, and all flags bits reserved

R1 Pointer to table to be filled in

On return

R0 Preserved

R1 Preserved

In practice

The table is organised with one entry for each page of physical memory, where each entry is 4 bits. Each byte thus contains two entries, the lower-order 4 bits relating to the address before the higher-order bits (ie little endian).

Each nibble indicates:

0-2 – Meaning: Type of memory:

– – Meaning: 0 not present

– – Meaning: 1 DRAM

– – Meaning: 2 VRAM

– – Meaning: 3 ROM

– – Meaning: 4 I/O

– – Meaning: 5-7 undefined

3 – Meaning: 0=page available for allocation, 1=not available

Notes

To read the size of the table, use OS_Memory 6.

If a RAM page is not available for allocation, this means it's allocated for something that cannot be replaced with a different RAM page (eg OS' page tables or screen memory).

For a dynamic area with particular physical address requirements (eg contiguous physical RAM), issue this call inside the DA's PreGrow handler and then choose which pages to ask for based on the result. If you make this call before creating the area, the page availability may then change.

This entry based on RISC OS 3.6 PRM

[RISC OS Open source page](#)

OS_Memory 8

SWI &68 · OS_Memory

The purpose of this call is to read the amount of a specified type of memory available in the computer.

Before calling

R0 8, and flags

On return

R0 Preserved

R1 Number of pages of specified type of memory

R2 Page size (in bytes)

Notes

The flags are specific to this reason code.

[RISC OS Open source page](#)

OS_Memory 9

SWI &68 · OS_Memory

The purpose of this call is to check for the presence of a given controller and return its base logical address if present. The behaviour of this SWI differs slightly between RISC OS 5 and RISC OS Select/Adjust. Under RISC OS 5 the SWI will return an error if the controller number is unrecognised, or 0 if the controller isn't present. Under RISC OS Select/Adjust the SWI will return an error under both situations. Also under RISC OS Select/Adjust, even if the controller exists there is no guarantee that the

memory will have been mapped in by the kernel. Therefore programs should also use OS_Memory 25 where appropriate, in order to allow them to map in the memory themselves.

Before calling

R0 9, all flag bits reserved

R1 Controller number

On return

R0 Preserved

R1 Controller base logical address, or 0 if not present

[RISC OS Open source page](#)

OS_MMUControl

SWI &6B

The purpose of this call is to read/modify the ARM system control register (formerly known as the MMU control register), and to allow cache/TLB operations to be performed.

Before calling

R0 Reason code (bits 0 -7),

Flags (bits 8 - 31) which are reason code specific

All other registers are dependent on reason code

On return

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

The flags (passed by R0 on entry) are specific to each reason code.

Reason Codes

/ Hex # / Action

0 – Hex #: &00 · Action: Modify system control register

1 – Hex #: &01 · Action: Cache flush request

2 – Hex #: &02 · Action: Get ARMop

[RISC OS Open source page](#)

OS_MMUControl 0

SWI &6B · OS_MMUControl

The purpose of this call is to allow the value of the ARM system control register to be read or modified.

Before calling

R0 0 (reason code)

R1 EOR mask

R2 AND mask

On return

R0 Preserved

R1 Old value

R2 New value

Notes

The bits in R1 and R2 correspond directly to those in the system control register; therefore the exact assignment will differ between ARM versions.

Use of this call is discouraged, except for the purposes of enabling and disabling the caches – for which this SWI will perform the correct cache clean/invalidation sequences as required by the architecture. On all ARMv3+ ARMs to date, bit 2 controls the data/unified cache and bit 12 controls the instruction cache.

The OS may restrict the values of some of the flags in order to ensure correct operation of the system. However there are many potentially unsafe values which are not checked for, so care must be taken not to rely on the OS restricting the settings automatically.

[RISC OS Open source page](#)

OS_MMUControl 1

SWI &6B · OS_MMUControl

The purpose of this call is to flush the cache, TLB or write buffer.

Before calling

R0 1, and flags

R1 Logical address, if bit 29 set

On return

--

Notes

Only use this where it is unavoidable.

[RISC OS Open source page](#)

OS_MMUControl 2

SWI &6B · OS_MMUControl

The purpose of this call is to read the address of one of the kernels "ARMop" routines. The ARMops are the low-level cache/TLB maintenance routines that the OS uses, and calling them directly can result in higher performance than calling equivalent SWIs such as OS_MMUControl 1.

Before calling

R0 Bits 0-7: 2 (reason code)

Bits 8-15: ARMop index

Other bits: Reserved (should be zero)

On return

R0 ARMop function pointer

Notes

The available ARMops are as follows:

- 0 – : Cache_CleanInvalidateAll
- 1 – : Cache_CleanAll
- 2 – : Cache_InvalidateAll
- 3 – : Cache_RangeThreshold
- 4 – : TLB_InvalidateAll
- 5 – : TLB_InvalidateEntry
- 6 – : DSB_ReadWrite
- 7 – : IMB_Full
- 8 – : IMB_Range
- 9 – : IMB_List
- 10 – : MMU_Changing
- 11 – : MMU_ChangingEntry
- 12 – : MMU_ChangingUncached
- 13 – : MMU_ChangingUncachedEntry
- 14 – : MMU_ChangingEntries
- 15 – : MMU_ChangingUncachedEntries
- 16 – : DSB_Write
- 17 – : DSB_Read
- 18 – : DMB_ReadWrite
- 19 – : DMB_Write
- 20 – : DMB_Read
- 21 – : Cache_CleanInvalidateRange

Requesting an unknown ARMop will result in an error. Unused ARMops (e.g. IMB on CPU with unified cache) will return a pointer to a dummy routine which does nothing – this can be detected as it will be a single MOV pc,lr instruction.

The general rules for register usage and preservation in calling these ARMops are:

- Any parameters are passed in R0,R1 etc. as required
- SP must be a valid stack pointer, with at least 16 words available
- Return address must be in LR
- The CPU must be in a privileged mode
- ARMops are allowed to corrupt R0, LR, and the PSR status flags. All other registers (unless used for return values) are preserved.

All ARMops are re-entrant, and it is preferred to call them with interrupts enabled (cache clean operations can be lengthy). Note that where register values are given as logical addresses, these are RISC OS logical addresses. The equivalent ARM terminology is virtual address (VA), or modified virtual address (MVA) for architectures with the fast context switch extension. Note also that where cache invalidation is required, it is implicit that any associated operations for a particular ARM will be performed also. The most obvious example is for an ARM with branch prediction, where it may be necessary to invalidate a branch cache anywhere where instruction cache invalidation is to be performed.

This call was first introduced with RISC OS 5.23

[RISC OS Open source page](#)

OS_Module

SWI &1E

The purpose of this call is to perform a module operation.

Before calling

R0 Reason code

On return

R0 Preserved

All other registers dependent on reason code

All other registers are dependent on reason code

Notes

The action performed depends on the reason code value in R0.

Reason Codes

- 00** – Hex #: &00 · Action: Run
- 01** – Hex #: &01 · Action: Load
- 02** – Hex #: &02 · Action: Enter
- 03** – Hex #: &03 · Action: Relnit
- 04** – Hex #: &04 · Action: Delete
- 05** – Hex #: &05 · Action: Describe RMA
- 06** – Hex #: &06 · Action: Claim
- 07** – Hex #: &07 · Action: Free
- 08** – Hex #: &08 · Action: Tidy
- 09** – Hex #: &09 · Action: Clear
- 10** – Hex #: &0A · Action: Insert module from memory
- 11** – Hex #: &0B · Action: Insert module from memory and move into RMA
- 12** – Hex #: &0C · Action: Extract module information
- 13** – Hex #: &0D · Action: Extend block
- 14** – Hex #: &0E · Action: Create new instantiation
- 15** – Hex #: &0F · Action: Rename instantiation
- 16** – Hex #: &10 · Action: Make preferred instantiation
- 17** – Hex #: &11 · Action: Add expansion card module
- 18** – Hex #: &12 · Action: Lookup module name
- 19** – Hex #: &13 · Action: Enumerate ROM modules
- 20** – Hex #: &14 · Action: Enumerate ROM modules with version
- 21** – Hex #: &15 · Action: Find end of ROM module chain

22 – Hex #: &16 · Action: Enumerate modules with private word pointer

23 – Hex #: &17 · Action: Unplug or insert modules

24 – Hex #: &18 · Action: Claim aligned block

[RISC OS Open source page](#)

OS_Module 0

SWI &1E · OS_Module

The purpose of this call is to run a module, calling its start address.

Before calling

R0 0

R1 Pointer to filename (and optional parameters)

On return

Does not return unless error occurs.

Notes

It is equivalent to the *RMRun command.

Use OS_Exit or OS_ExitAndDie to leave.

[RISC OS Open source page](#)

OS_Module 1

SWI &1E · OS_Module

The purpose of this call is to load a module.

Before calling

R0 1

R1 Pointer to filename (and optional parameters)

On return

R0 Preserved

R1 Preserved

Notes

It is equivalent to the *RMLoad command.

[RISC OS Open source page](#)

OS_Module 10

SWI &1E · OS_Module

The purpose of this call is to insert a module from memory.

Before calling

R0 10 (&0A)

R1 Pointer to start of module

On return

R0 Preserved

R1 Preserved

Notes

This takes a pointer to a block of memory and links it into the module chain, without moving it. Header fields are checked for validity. All duplicate modules are killed. If it's successful, then the module is called at its initialisation entry. The word immediately before the module start (ie at address R1-4) must contain the length of the module in bytes.

[RISC OS Open source page](#)

OS_Module 11

SWI &1E · OS_Module

The purpose of this call is to insert a module from memory and move into RMA.

Before calling

R0 11 (&0B)

R1 Pointer to start of module

R2 Length of module (in bytes)

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

This takes a pointer to a block of memory, and checks its header field for validity. It then kills any duplicate module, copies the block into the RMA, initialises it and links it to the modules chain.

[RISC OS Open source page](#)

OS_Module 12

SWI &1E · OS_Module

The purpose of this call is to return information about a module, given its number. This call returns the contents of the private word; if you instead want a pointer to the private word, use OS_Module 22.

Before calling

R0 12 (&0C)

R1 Module number

R2 Instance number

On return

R0 Preserved

R1 Updated for next module/instance

R2 Updated for next module/instance

R3 Pointer to module?

R4 Private word (usually pointer to workspace)

R5 Pointer to postfix string (the instantiation's name)

Notes

If a module is not multiply instantiated, R5 will be returned pointing to the string "Base".

This is equivalent to the *Modules command.

[RISC OS Open source page](#)

OS_Module 13

SWI &1E · OS_Module

The purpose of this call is to extend/reduce a currently claimed block of RMA.

Before calling

R0 13 (&0D)

R2 Pointer to block

R3 Size increase

On return

R0 Preserved

R2 New block pointer

R3 Preserved

Notes

If required, it also moves the contents of the area.

[RISC OS Open source page](#)

OS_Module 14

SWI &1E · OS_Module

The purpose of this call is to create a new instantiation of existing modules.

Before calling

R0 14 (&0E)

R1 Pointer to a full name of new instantiation and any parameters for initialisation

On return

R0 Preserved

R1 Preserved

Notes

The syntax used is 'module_title%instantiation'.

[RISC OS Open source page](#)

OS_Module 15

SWI &1E · OS_Module

The purpose of this call is to rename an existing instantiation of a module.

Before calling

R0 15 (&0F)

R1 Pointer to current module%instantiation name

R2 Pointer to new postfix string

On return

R0 Preserved

R1 Preserved

R2 Preserved

[RISC OS Open source page](#)

OS_Module 16

SWI &1E · OS_Module

The purpose of this call is to select the preferred instantiation of a particular module.

Before calling

R0 16 (&10)

R1 Pointer to full module%instantiation name

On return

R0 Preserved

R1 Preserved

[RISC OS Open source page](#)

OS_Module 17

SWI &1E · OS_Module

The purpose of this call is to add an expansion card module.

Before calling

R0 17 (&11)

R1 Pointer to full environment string

R2 Chunk number

R3 ROM section

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

This allows expansion card, and extension ROM modules to be added to the module list.

[RISC OS Open source page](#)

OS_Module 18

SWI &1E · OS_Module

The purpose of this call is to return information about a named module.

Before calling

R0 18 (&12)

R1 Pointer to module_title%instantiation name

On return

R0 Preserved (if Module exists, error pointer if it doesn't)

R1 Module number

R2 Instantiation number

R3 Module code?

R4 Private word

R5 Postfix string

Notes

This is equivalent to the *Modules command.

[RISC OS Open source page](#)

OS_Module 19

SWI &1E · OS_Module

The purpose of this call is to enumerate ROM modules.

Before calling

R0 19 (&13)

R1 Module number, or 0 to start

R2 ROM section

On return

R0 Preserved

R1 Module number of found module + 1

R2 ROM section

R3 Pointer to module name

R4 Status

R5 Chunk number of expansion card module

Notes

This call returns information on one module that is currently in ROM along with its status. The module found is the given number of modules on from start of the given ROM section. If there are insufficient modules in the ROM section then the search continues with the next section; so the fifth module in a four module section would in fact be the first module of the next section. The ROM sections are scanned in order. The values in R0-R2 are the correct ones to use this call to enumerate the next module; hence repeated calls will give a full enumeration of all ROM modules.

The PRM states that when the end of the list an 'No more modules' will be returned with error number &107. This is incorrect; error number &108 will be returned. (&107 relates to the module prefix).

OS_Module 20 provides the same functionality, but in addition returns the version information (in R6).

[RISC OS Open source page](#)

OS_Module 2

SWI &1E · OS_Module

The purpose of this call is to enter a module.

Before calling

R0 2

R1 Pointer to module name

R2 Pointer to parameters

On return

Does not return unless error occurs.

Notes

Doesn't return unless error occurs. If the module has got a start address, it resets the supervisor stack, sets user mode and makes it the current application. Any specified instantiation will become the preferred instantiation.

[RISC OS Open source page](#)

OS_Module 20

SWI &1E · OS_Module

The purpose of this call is to enumerate ROM modules with version.

Before calling

R0 20 (&14)

R1 Module number, or 0 to start

R2 ROM section

On return

R0 Preserved

R1 Module number of found module +1

R2 ROM section

R3 Pointer to module name

R4 Status

R5 Chunk number of expansion card module

R6 BCD version number

Notes

This call returns information on one module that is currently in ROM along with its status. The module found is the given number of modules on from start of the given ROM section. If there are insufficient modules in the ROM section then the search continues with the next section; so the fifth module in a four module section would in fact be the first module of the next section. The ROM sections are scanned in order. The values in R0-R2 are the correct ones to use this call to enumerate the next module; hence repeated calls will give a full enumeration of all ROM modules.

OS_Module 19 provides the same functionality, with the exception that it does not return the version information (in R6).

[RISC OS Open source page](#)

OS_Module 21

SWI &1E · OS_Module

The purpose of this call is to locate the end of the specified ROM module chain.

Before calling

R0 21 (&15)

R1 ROM section

On return

R0 Preserved

R1 Preserved

R2 Pointer to first word after ROM module chain

Notes

Currently, only ROM section -1 (main system ROM) is supported.

This call is currently only supported by RISC OS 5.

[RISC OS Open source page](#)

OS_Module 22

SWI &1E · OS_Module

The purpose of this call is to enumerate ROM modules. Unlike OS_Module 12, this call returns a pointer to the private word, instead of the value of the word.

Before calling

R0 22 (&16)

R1 Module number, or 0 to start

R2 Instance number

On return

R0 Preserved

R1 Updated for next module/instance

R2 Updated for next module/instance

R3 Pointer to module?

R4 Pointer to private word, or 0 if module was not correctly unlinked

R5 Pointer to postfix string (the instantiation's name)

Notes

If a module is not multiply instantiated, R5 will be returned pointing to the string "Base".

This call was introduced in RISC OS 5.27.

[RISC OS Open source page](#)

OS_Module 23

SWI &1E · OS_Module

This call replicates the functionality of the *Unplug and *RMInsert commands, but without requiring the invocation of the command line interpreter. This allows the SWI to safely be used from situations in which use of OS_CLI is ill-advised, for example during service calls.

Before calling

R0 23 (&17)

R1 Pointer to module name

R2 1 to insert, 0 to unplug

R3 Module number, or &80000000 for 'all versions'

On return

- All registers preserved

Notes

Currently this SWI is unsupported by RISC OS 5 and is only available in versions of RISC OS Ltd's OS.

[RISC OS Open source page](#)

OS_Module 24

SWI &1E · OS_Module

The purpose of this call is to return an aligned area of free RMA claimed for use.

Before calling

R0 24

R3 Size of area

R4 Alignment (power of two)

On return

R2 Block

- Other registers preserved

Notes

The block address returned in R2 will satisfy $(R2 \text{ MOD } R4)=0$. This is in contrast to OS_Module 6, which always returns an address $(R2 \text{ MOD } 16)=4$.

If an aligned block is resized via OS_Module 13 then it is not guaranteed to remain aligned.

This call was introduced in RISC OS 5.27.

[RISC OS Open source page](#)

OS_Module 3

SWI &1E · OS_Module

The purpose of this call is to reinitialise a module.

Before calling

R0 3

R1 Pointer to module name plus any parameters for initialisation

On return

R0 Preserved

R1 Preserved

Notes

A Module is reinitialised by calling its Finalise address, releasing any workspace and then calling its Initialisation address.

If the module refuses to reinitialise, it is removed from RMA.

[RISC OS Open source page](#)

OS_Module 4

SWI &1E · OS_Module

The purpose of this call is to kill a module.

Before calling

R0 4

R1 Module name

On return

R0 Preserved

R1 Preserved

Notes

This is equivalent to the *RMKill command.

[RISC OS Open source page](#)

OS_Module 5

SWI &1E · OS_Module

The purpose of this call is to read information on the RMA.

Before calling

R0 5

On return

R0 Preserved

R2 Size of largest block available (in bytes)

R3 Total amount free in RMA (in bytes)

Notes

This call uses OS_Heap 1 with the appropriate descriptor to achieve this operation.

[RISC OS Open source page](#)

OS_Module 6

SWI &1E · OS_Module

The purpose of this call is to return an area of free RMA claimed for use.

Before calling

R0 6

R3 Size of area

On return

R0 Preserved

R2 Block

R3 Preserved

Notes

Starting with RISC OS 2.01, memory allocated by this call will have a start address of $(n \text{ MOD } 16)=4$. This can be used to help to improve performance by allowing programmers to easily align their code/data to cache-line boundaries (assuming a 16 byte cache line size). In RISC OS 5.00 the alignment was changed to $(n \text{ MOD } 32)=20$, which can be useful for machines with 32 byte cache lines.

OS_Module 24 can be used to allocate a “naturally aligned” block at an arbitrary power of two alignment (i.e. $(n \text{ MOD } x)=0$).

[RISC OS Open source page](#)

OS_Module 7

SWI &1E · OS_Module

The purpose of this call is to free a previously claimed area of RMA.

Before calling

R0 7

R2 Block

On return

R0 Preserved

R2 Preserved

[RISC OS Open source page](#)

OS_Module 8

SWI &1E · OS_Module

The purpose of this call is to tidy the area of memory currently used by modules.

Before calling

R0 8

On return

R0 Preserved

Notes

This gives each instantiation of all modules in turn, from the end of the module list and working backwards, a non-fatal finalisation call.

Instantiations of a particular module are killed in the order they appear on the current instantiation list.

If it succeeds, it collects the RMA together into one large unfragmented block and reinitialises the modules again. Any private words containing pointer to workspace blocks in RMA are relocated.

Note: This is NOT recommended. Note2: This reason code does nothing on RISC OS 3.5 onwards.

[RISC OS Open source page](#)

OS_Module 9

SWI &1E · OS_Module

The purpose of this call is to kill all modules which are not ROM modules.

Before calling

R0 9

On return

R0 Preserved

Notes

This is NOT recommended.

[RISC OS Open source page](#)

OS_Mouse

SWI &1C

The purpose of this call is to return the current mouse status.

Before calling

--

On return

R0 X co-ordinate

R1 Y co-ordinate

R2 Button state

R3 Time of button change

Notes

The data is returned from the mouse buffer, or the current mouse position if the buffer is empty.

The time value returned using the monotonic timer.

[RISC OS Open source page](#)

OS_NewLine

SWI &03

The purpose of this call is to send a line feed followed by a carriage return to all active output streams.

Before calling

--

On return

--

[RISC OS Open source page](#)

OS_NVMemory

SWI &76

The purpose of this call is to provide an interface that allows the full capacity of a machines NVRAM (aka CMOS RAM) to be accessed. Other calls (OS_Byte 161, OS_Byte 162) are limited in that they only provide access to the first 240 bytes of NVRAM.

Before calling

R0 Reason code

On return

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

Reason Codes

0 – Hex #: &00 · Action: Read NVMRAM size

1 – Hex #: &01 · Action: Read a byte

2 – Hex #: &02 · Action: Write a byte

3 – Hex #: &03 · Action: Read bytes

- 4 – Hex #: &04 · Action: Write bytes
 - 5 – Hex #: &05 · Action: Reserved
 - 6 – Hex #: &06 · Action: Read default value used on CMOS reset
 - 7 – Hex #: &07 · Action: Set station
- [RISC OS Open source page](#)
-

OS_NVMemory 0

SWI &76 · OS_NVMemory

The purpose of this call is to allow the size of NVRAM to be read.

Before calling

R0 0

On return

R1 NV memory size in bytes

All other registers preserved

[RISC OS Open source page](#)

OS_NVMemory 1

SWI &76 · OS_NVMemory

The purpose of this call is to allow individual bytes to be read from NVRAM.

Before calling

R0 1

R1 NVRAM address

On return

R2 Value

All other registers preserved

Notes

Use OS_NVMemory 3 if you want to read a number of bytes from sequential addresses.

This call will enable interrupts for the duration of the NVRAM operation.

[RISC OS Open source page](#)

OS_NVMemory 2

SWI &76 · OS_NVMemory

The purpose of this call is to allow individual bytes to be written to NVRAM.

Before calling

R0 2

R1 NVRAM address

R2 Value

On return

All registers preserved

Notes

Use OS_NVMemory 4 if you want to write a number of bytes to sequential addresses.

This call will enable interrupts for the duration of the NVRAM operation.

[RISC OS Open source page](#)

OS_NVMemory 3

SWI &76 · OS_NVMemory

The purpose of this call is to read a series of bytes from NVRAM and into the buffer pointed to by R1.

Before calling

R0 3

R1 NVRAM address

R2 Pointer to buffer

R3 Number of bytes

On return

All registers preserved

Notes

If you only need to read one byte, you can use OS_NVMemory 1 instead.

This call will enable interrupts for the duration of the NVRAM operation.

[RISC OS Open source page](#)

OS_NVMemory 4

SWI &76 · OS_NVMemory

The purpose of this call is to write a series of bytes to NVRAM from the buffer pointed to by R1.

Before calling

R0 4

R1 NVRAM address

R2 Pointer to buffer

R3 Number of bytes

On return

All registers preserved

Notes

If you only need to write one byte, you can use OS_NVMemory 2 instead.

This call will enable interrupts for the duration of the NVRAM operation.

[RISC OS Open source page](#)

OS_NVMemory 6

SWI &76 · OS_NVMemory

The purpose of this call is query what byte would be written by the OS when it applies its default values, for example when the NVRAM is cleared by the user holding down delete at power on.

Before calling

R0 6

R1 NVRAM address

On return

R2 Value

All other registers preserved

In practice

A return value of -1 means the value is undetermined or not managed by RISC OS.

Notes

Introduced in RISC OS 5.19.

[RISC OS Open source page](#)

OS_NVMemory 7

SWI &76 · OS_NVMemory

The purpose of this call is to allow the network manager to write the location holding the Econet station ID, which is protected from accidental writes.

Before calling

R0 7

R1 Pointer to passphrase

R2 Value to write

On return

All registers preserved

In practice

The value can always be read.

Notes

Do not use this call directly. Use the SetStation utility instead.

[RISC OS Open source page](#)

OS_PlatformFeatures

SWI &6D

The purpose of this call is to return information relating to the hardware platform currently running the Operating System.

Before calling

R0 Reason code, + Flags which are reason code specific

All other registers are dependent on reason code

On return

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

Reason Codes

/ Hex # / Action

0 – Hex #: &00 · Action: Read code features

1 – Hex #: &01 · Action: Read MMU features (RISC OS Select)

32 – Hex #: &20 · Action: Read processor vectors location

33 – Hex #: &21 · Action: Read cache information

34 – Hex #: &22 · Action: Read CPU features

35 – Hex #: &23 · Action: Read clear exclusive monitor function

64 – Hex #: &40 · Action: Describe CPU architecture and registers? (see RISC OS Pyromaniac documentation for more details)

Detecting support for non-zero reason codes

The original OS_PlatformFeatures implementation contained a bug whereby requesting an unknown reason code would always result in the error handler being called, even when the X form of the SWI was used. This bug makes it difficult for software to determine the availability of reason codes by calling the X form of the SWI and checking to see if an error is returned – instead of returning to your code, the error handler will be invoked. Note that this bug also affected the handling of any flags in R0 – with RISC OS 3.7, the only valid value for R0 when calling OS_PlatformFeatures was zero, any other value would trigger the bug.

This bug was fixed for RISC OS 3.8 (and by extension RISC OS 4 and 6), but unfortunately the fix was not merged back into the branch of the kernel that RISC OS 5 is based around. Consequently the bug remained unfixed in RISC OS 5 until it was re-discovered and fixed in RISC OS 5.23. Additionally the buggy code was used as the basis for the OS_PlatformFeatures implementation provided by the CallASWI module, and so there are several versions of CallASWI which provide buggy OS_PlatformFeatures implementations for OS versions prior to RISC OS 3.7.

OS_PlatformFeatures 0 bit 31 has now been allocated for the purpose of providing software an easy means to check if the bug has been fixed.

- If this bit is set, then it means that the error reporting bug has been fixed, and software can safely poll for the extra reason codes or use non-zero flags in R0. This flag also acts as a hint that the extra reason codes introduced by RISC OS 5 (reason codes 32+) may be present (but their presence is not guaranteed, e.g. CallASWI may not implement all the reason codes)
- If this bit is not set, then it means that the bug may be present. As stated, RISC OS 4 and 6 do not suffer from the bug. However because all versions of RISC OS 5 which implement additional reason codes will have the flag set, the only extra reason code which you might possibly encounter in this state is OS_PlatformFeatures 1, which was introduced in RISC OS Select.

[RISC OS Open source page](#)

OS_PlatformFeatures 0

SWI &6D · OS_PlatformFeatures

The purpose of this call is to return what is required to run code on the current hardware platform. The returned flags are influenced by both the properties of the CPU and the properties of the OS.

Before calling

R0 0

On return

R0 Flags

R1 Pointer to interrupt trigger routine, or 0 if no routine is necessary

Notes

New flags are added to the return value in R0 in a backwards-compatible manner. Therefore, on machines where this SWI isn't implemented (versions of RISC OS prior to 3.7) your code should act as if all the flags are set to zero. This can be done, for example, by checking if the SWI returns an unknown SWI error (error number &1E6). However care is still required with some flags, such as bit 11 (see the list of flags for more details).

The routine returned in R1 is used to help trigger interrupts on CPUs where pending interrupts won't be triggered as soon as the I bit is cleared. Enable interrupts and then call the routine to ensure that any pending interrupts have been processed. The routine is suitable for calling from any CPU mode, and preserves all flags and registers, and is reentrant.

[RISC OS Open source page](#)

OS_PlatformFeatures 32

SWI &6D · OS_PlatformFeatures

The purpose of this call is to return information about where the processor vectors are located, and the amount of space available for FIQ handler code.

Before calling

R0 32 + flags (reserved)

On return

R0 Base address of processor vectors

R1 Size of processor vectors

Notes

The size returned in R1 is the total amount of space reserved for the processor vectors, in bytes. FIQ handlers are able to use the area from R0+0x1C to R0+R1 for storing their code/data.

Currently 256 bytes are set aside for the processor vectors, to match older versions of RISC OS. However in the future more space may be made available.

Care must be taken when calling this reason code; see the main OS_PlatformFeatures page.

[RISC OS Open source page](#)

OS_PlatformFeatures 33

SWI &6D · OS_PlatformFeatures

The purpose of this call is to return information about a given cache level within the processor.

Before calling

R0 33 + flags (reserved)

R1 Cache level (zero-based)

On return

R0 Flags:

Bits 0-2: Cache type:

- 0 No cache present
- 1 Instruction cache present
- 2 Data cache present
- 3 Split I+D cache present
- 4 Unified cache present
- 5+ Reserved

Bits 3-31: Reserved (zero)

R1 D cache line length (bytes)

R2 D cache size (bytes)

R3 I cache line length (bytes)

R4 I cache size (bytes)

Notes

If R1 is out of range then zero will be returned for R0-R4. Therefore to iterate through all present cache levels, start with R1 at 0 and increase it by one on each call until zero is returned in R0.

For cache types 1 and 2, zero will be returned for the registers corresponding to the unimplemented cache. For other cache types, all registers will contain data (e.g. a unified cache will return identical values for R1-R2 and R3-R4).

Note that this call is only guaranteed to cover caches which are known to or need managing by the ARM CPU. For example, the system-level L2 cache present in the Raspberry Pi 1 will not be covered by this call, but the ARM-local L2 cache in the Raspberry Pi 2 will.

This call was introduced in RISC OS 5.23. Care must be taken when calling this reason code; see the main OS_PlatformFeatures page.

[RISC OS Open source page](#)

OS_PlatformFeatures 34

SWI &6D · OS_PlatformFeatures

This SWI allows you to query the system to determine whether arbitrary CPU features are supported by the host machine. Mainly this information relates to the availability of different instructions or instruction groups.

Before calling

R0 34 + flags (reserved)

R1 ≥ 0 : Flag number to read

< 0 : Return bit masks for flags page number NOT R1

On return

R0 0 Flag is clear

1 Flag is set

-1 Flag is unknown

R0 Word 0 of flag values

R1 Word 1 of flag values

R2 Word 2 of flag values

R3 Word 3 of flag values

R4 Word 0 validity mask

R5 Word 1 validity mask

R6 Word 2 validity mask

R7 Word 3 validity mask

OS_PlatformFeatures 34

When reading an individual flag:

When reading a flags page:

In practice

The call has two forms; passing a positive or zero value in R1 allows for a single feature to be queried, while passing a negative value allows for an entire 'page' of features to be queried, with each page containing 128 feature flags. E.g. calling with R1 = -1 will return the information for page 0, which will cover flag bits 0-127.

Notes

When querying by page, the bits in R0 represent bits 0-31 of the page, R1 represents bits 32-63, etc. The bits in R4-R7 indicate which bits in R0-R3 are valid; if a bit is clear then it indicates the OS version does not understand or contain support for querying that particular feature.

If an unknown feature page is requested then all of R0-R7 will be returned as zero.

Care must be taken when calling this reason code; see the main OS_PlatformFeatures page.

[RISC OS Open source page](#)

OS_PlatformFeatures 35

SWI &6D · OS_PlatformFeatures

The purpose of this call is to return a function pointer that can be called to clear the exclusive monitor lock in a way that is independent of the CPU.

Before calling

R0 35

On return

R1 Pointer to clear exclusive routine, or 0 if no routine is necessary

Notes

The routine must be called with a valid, full descending stack pointer in R13 and a return address in LR. The routine will preserve all registers, including the CPSR.

The very simplest implementation will be a CLREX instruction, though other CPU variants and silicon implementations may require more complex implementations. The OS will take these into account.

[RISC OS Open source page](#)

OS_Plot

SWI &45

The purpose of this call is to perform a direct VDU call.

Before calling

R0 Plot command code,

R1 X co-ordinate

R2 Y co-ordinate

On return

R0 Corrupted

R1 Corrupted

R2 Corrupted

Plot Command Codes

0 – Description: Solid line including both end points · Sequence of moves: Line

- 8 – Description: Solid line excluding the final end point · Sequence of moves: Line
- 16 – Description: Dotted line including both end points (pattern restarted) · Sequence of moves: Line
- 24 – Description: Dotted line excluding the final end point (pattern restarted) · Sequence of moves: Line
- 32 – Description: Solid line excluding 1st point · Sequence of moves: Line
- 40 – Description: Solid line excluding both end points · Sequence of moves: Line
- 48 – Description: Dotted line excluding 1st point (pattern continued) · Sequence of moves: Line
- 56 – Description: Dotted line excluding both end points (pattern continued) · Sequence of moves: Line
- 64 – Description: Point plot
- 72 – Description: Horizontal line fill (left and right) to non-background
- 80 – Description: Triangle fill · Sequence of moves: Triangle
- 88 – Description: Horizontal line fill (right only) to background
- 96 – Description: Rectangle fill · Sequence of moves: Rectangle
- 104 – Description: Horizontal line fill (left and right) to foreground
- 112 – Description: Parallelogram fill · Sequence of moves: Parallelogram
- 120 – Description: Horizontal line fill (right only) to non-foreground
- 128 – Description: Flood to non-background
- 136 – Description: Floor to foreground
- 144 – Description: Circle outline · Sequence of moves: Circle
- 152 – Description: Circle fill · Sequence of moves: Circle
- 160 – Description: Circular arc · Sequence of moves: Arc
- 168 – Description: Segment · Sequence of moves: Segment
- 176 – Description: Sector · Sequence of moves: Sector
- 184 – Description: Block copy/move · Sequence of moves: Block copy/move
- 192 – Description: Ellipse outline · Sequence of moves: Ellipse
- 200 – Description: Ellipse fill · Sequence of moves: Ellipse
- 208 – Description: Font printing (do not use)
- 216 – Description: Reserved
- 224 – Description: Reserved
- 232 – Description: Sprite Plot
- 240 – Description: Reserved
- 248 – Description: Reserved

Sequence of Moves

Some shapes need to be defined by a sequence of points. The VDU drivers use the previous positions of the graphics cursor to define these. These can be specified by a sequence of absolute or relative moves (commands 64 or 68).

Arc – Sequence of moves: To plot an arc move to centre, move to start, then plot arc to a point on the line joining centre to end. The arc will be plotted anticlockwise

Block copy/move – Sequence of moves: First define the source rectangle: move to one corner, then move to diagonally opposite corner. Then plot bottom left corner of destination

Circle – Sequence of moves: To plot a circle, move to centre, then plot circle to a point on the circumference

Ellipse – Sequence of moves: To plot an ellipse, move to centre, move to point on circumference with same Y co-ordinate as centre, then plot ellipse to the lowest or highest point on the circumference

Line – Sequence of moves: To plot a line, move to one endpoint, then plot line to other endpoint

Parallelogram – Sequence of moves: To plot a parallelogram, move to vertex, move to next vertex, then plot parallelogram to third vertex. Fourth vertex will be opposite second

Rectangle – Sequence of moves: To plot a rectangle, move to one corner, then plot rectangle to diagonally opposite corner

Sector – Sequence of moves: To plot a sector, move to centre, move to start, then plot sector to a point on the line joining centre to end. The sector will be plotted anticlockwise

Segment – Sequence of moves: To plot a segment, move to centre, move to start, then plot segment to a point on the line joining centre to end. The segment will be plotted anticlockwise

Triangle – Sequence of moves: To plot a triangle, move to vertex, move to next vertex, then plot triangle to third vertex

Modifiers

The plot commands use the top 5 bits. The bottom 3 bits are used to specify the 'modifier'. Most plot command codes use the standard modifier codes, however, a special set of modifiers are available for block copy/move commands.

- 0 – Standard modifier type: Move relative
- 1 – Standard modifier type: Plot relative, foreground
- 2 – Standard modifier type: Plot relative, inverse
- 3 – Standard modifier type: Plot relative, background
- 4 – Standard modifier type: Move absolute
- 5 – Standard modifier type: Plot absolute, foreground
- 6 – Standard modifier type: Plot absolute, inverse
- 7 – Standard modifier type: Plot absolute, background
- 0 – Block copy/move modifier type: Move relative
- 1 – Block copy/move modifier type: Relative rectangle move
- 2 – Block copy/move modifier type: Relative rectangle copy
- 3 – Block copy/move modifier type: Relative rectangle
- 4 – Block copy/move modifier type: Move absolute
- 5 – Block copy/move modifier type: Absolute rectangle move
- 6 – Block copy/move modifier type: Absolute rectangle copy
- 7 – Block copy/move modifier type: Absolute rectangle copy

Notes

The action performed depends on the plot command code value in R0.

Unknown plot codes (216-231,240-255) are sent to the unknown plot vector, which by default does nothing.

The dash pattern is set with VDU 23,6 and can be read with OS_Word 10,6. Its length is set and read with OS_Byte 163.

[RISC OS Open source page](#)

OS_Pointer

SWI &64

The purpose of this call is to perform a pointer(mouse) operation.

Before calling

R0 Reason code

All other registers are dependent on reason code

On return

Registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

Many versions of RISC OS contain a bug which will cause OS_Pointer to malfunction or crash if called with an unsupported reason code. Therefore, care must be taken when attempting to make use of reason codes 2 and above.

Internally this call makes use of the PointerV software vector to communicate with the pointer driver(s).

Reason Codes

/ Hex # / Action

- 0 – Hex #: &00 · Action: Get pointer type

1 – Hex #: &01 · Action: Set pointer type
2 – Hex #: &02 · Action: Read alternate position (desktop RISC OS)
Read pointer presence (NCOS only)
[RISC OS Open source page](#)

OS_Pointer 0

SWI &64 · OS_Pointer

The purpose of this call is to return the currently selected pointer device type.

Before calling

R0 0

On return

R0 Pointer device type

Notes

To set the currently selected pointer device type, use OS_Pointer 1.
[RISC OS Open source page](#)

OS_Pointer 1

SWI &64 · OS_Pointer

The purpose of this call is to set the currently selected pointer device type.

Before calling

R0 1

R1 Pointer device type

On return

R0 Preserved

Notes

To return the currently set pointer device type, use OS_Pointer 0.
The pointer device type can be permanently set using *Configure MouseType <n>.
Internally, this will issue PointerV 2 in order to inform the pointer driver(s) of the change.
[RISC OS Open source page](#)

OS_PPointer 2

SWI &64 · OS_PPointer

This call returns the position of the “alternate positioning device”, i.e. scroll wheel.

Before calling

R0 2

On return

R0 Signed X position of alternate device (positive = right)

R1 Signed Y position of alternate device (positive = “up”/away from user)

Notes

The returned values are an accumulation of all the delta position updates received from the pointer device driver. However if the values overflow their signed 32bit range then they will be reset to zero. Software which attempts to compute wheel movement deltas by subtracting a new accumulator value from an old accumulator value must be careful to take this behaviour into account, otherwise it will mistakenly see a very large jump in wheel position.

Many versions of RISC OS contain a bug which will cause OS_PPointer to malfunction or crash if called with an unsupported reason code. Therefore, care must be taken when attempting to make use of reason codes 2 and above.

In addition to polling this SWI, software can use Event 21,4 to detect when the alternate positioning device moves.

[RISC OS Open source page](#)

OS_PrettyPrint

SWI &44

The purpose of this call is to write an indirect string, with formatting, to all of the active output streams.

Before calling

R0 Pointer to NUL terminated string to write

R1 Pointer to dictionary (0 = RISC OS dictionary)

R2 Pointer to NUL terminated special string

On return

R0 Preserved

R1 Preserved (but see RO6 note below)

R2 Preserved

Notes

Due to a serious bug in every version of PrettyPrint since RISC OS 2.00, it is important to not end the string with a tab character – tabs must be followed by at least one visible character. Starting with a tab is fine.

Compacted text uses an escape character in the print string to indicate a dictionary entry. It is followed immediately by a byte which is the dictionary entry number. If this byte is in the range 1-255, then the appropriate string in the dictionary is substituted. If it's 0, then the special string pointed to by R2 is substituted. (This is used in particular by the *Help command.)

The format of a dictionary is a linear list of entries, which can recursively refer to other dictionary entries; each entry is a length byte (counting itself) followed by a 0 terminated string, so an empty string has a length byte of 2. A length byte of zero terminates the dictionary. A length byte of 1 is not allowed.

This call acts like OS_Write0 with several differences:

- It will break a line at a space if the next word will not fit on the line, except at hard spaces
- Compacted text is handled
- CR causes a newline to be generated
- TAB causes tabulation to the next multiple of eight columns
- ASCII 31 acts as a hard space for formatting, but is output as 32
- ESC (ASCII 27) indicates that a dictionary entry should be substituted

The RISC OS 6 version of PrettyPrint will return the address of the default dictionary in R1 if R0, R1 and R2 are zero on entry.

[RISC OS Open source page](#)

OS_PrintChar

SWI &5D

The purpose of this call is to send a character to the printer.

Before calling

R0 Character to print

On return

R0 Preserved

Notes

OS_Byte 3 and OS_Byte 5 control whether there is a printer selected and which device it is. Note that the printer ignore character is not used by this call.

[RISC OS Open source page](#)

OS_ReadArgs

SWI &49

The purpose of this call is to scan the input string, and places values in output buffer according to the syntax string.

Before calling

R0 Pointer to syntax string

R1 Pointer to input string

R2 Pointer to output buffer

R3 Size of output buffer

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Bytes left in output buffer

In practice

It is generally used to extract parameters from the command line following a call to `OS_GetEnv` and using the value of `R0` as the input string. As such this call should be used at the start of a program before the first `Wimp_Poll`.

This call will return error 490 "Parameters not recognised" if:

- the input string has a keyword that is not specified in the syntax string
- the syntax string has a keyword with qualifier `"/A"` that is not in the input string
- the input string has items which have not been matched to the syntax string

This call will return error 491 "Argument repeated" if a parameter is repeated in the input string.

This call will also call `OS_EvaluateExpression` and `OS_GSTrans` if the qualifiers `"/E"` and `"/G"`, respectively, are specified in the syntax string, and any errors from them will also be returned.

In the event of an error, this call does not give any indication as to which item in the input string has caused the problem.

[RISC OS Open source page](#)

OS_ReadC

SWI &04

The purpose of this call is to read a character from the input stream.

Before calling

--

On return

R0 ASCII (if C flag is 0), or

Error number (if C flag is 1), or

&1B = escape

Notes

It waits to get the next character, i.e. you will only get the control back when, for example, a key is pressed.

The source for the input stream can be read with `OS_Byte 177`.

If cursor editing is enabled (see `OS_Byte 4`), then pressing a cursor keys will start cursor editing.

[RISC OS Open source page](#)

OS_ReadDefaultHandler

SWI &55

The purpose of this call is to return the address of the default handler.

Before calling

R0 Handler number

On return

R0 Preserved

R1 Address of default handler

R2 Workspace address

R3 Buffer address

Notes

This enables you to re-install the handler which is set up after a reset or power on.

[RISC OS Open source page](#)

OS_ReadDynamicArea

SWI &5C

The purpose of this call is to read the size of an area.

Before calling

R0 Area to read

For area numbers 0-127, set bit 7 in order to read maximum size to R2

On return

R0 Pointer to start address of area

R1 Current number of bytes in area

R2 Maximum size of area (if input R0 outside range 0-127), else preserved

Notes

If this SWI is used on an area that has virtual memory active this SWI reads the logical size of the area, otherwise the behaviour is the same as before. In practice this means that this call can be used by the application without worrying whether virtual memory is active.

For doubly-mapped dynamic areas, the start address corresponds to the start of the first copy (i.e. the lowest valid address in the area), unlike OS_DynamicArea 2 and OS_DynamicArea 24 which return the start address of the second copy.

For Sparse Dynamic Areas, the current size is calculated from the number of pages that are currently mapped in. Therefore it is not guaranteed to describe the logical extent of the area.

For Physical Memory Pool DAs, the values in R1 and R2 will correspond to the physical properties of the dynamic area, not the logical properties. Therefore the values are completely unrelated to the base address returned in R0 (and if an area has a maximum logical size of zero, the base address will be zero). If necessary the values will be clamped at 2GB-PageSize in order to try and avoid any issues relating to signed number overflow. To be able to query a PMP without this clamping, or to be able to query the logical size values, OS_DynamicArea 24 should be used.

[RISC OS Open source page](#)

OS_ReadEscapeState

SWI &2C

The purpose of this call is to check whether an escape condition has occurred

Before calling

--

On return

C flag is set if an escape condition has occurred

Notes

Once an escape condition has been detected, (either through this or for example, with `OS_ReadC`), it should be acknowledged with `OS_Byte 126` or cleared using `OS_Byte 124`.

This call is useful if a program is executing in a loop which the user may want to escape from, but isn't performing any input operations which would let it know about the escape.

Note that this may be called from an interrupt routine. However, `OS_Byte 126` may not be, so if an escape is detected under interrupts, the interrupt routine must set a flag which is checked by the foreground task rather than try to acknowledge the escape itself.

[RISC OS Open source page](#)

OS_ReadLine

SWI &0E

This SWI is deprecated, because the use of flag bits in address values makes it unsafe to use on systems with 32bit logical memory maps. Please use `OS_ReadLine32` instead.

Before calling

R0 Buffer + flags (flags ignored in RISC OS 5)

R1 Size of buffer -1 (maximum number of bytes to return, excluding CR)

R2 Lowest char allowed

R3 Highest char allowed

R4 Character to output (if R0 bit 30 set) (ignored in RISC OS 5)

On return

R0 "Corrupted" (all implementations return the buffer address, minus flags)

R1 Bytes read, excluding Return

R2 "Corrupted" (all implementations preserve R2)

R3 "Corrupted" (all implementations preserve R3)

R4 Preserved

C flag is set if input is terminated by an escape condition

In practice

This SWI does not work correctly under RISC OS 5

The purpose of this call is to read a line from the current input stream.

Notes

Input is terminated with Return, Ctrl-J or Escape. Delete and Backspace deletes last char entered. Ctrl-U empties buffer.

When the buffer is full, each successive attempt to enter a character results in a Ctrl-G 'beep'.

Other than RISC OS 5, `ReadLineV` uses this API. RISC OS 5's `ReadLineV` uses the `OS_ReadLine32` API instead.

RISC OS Six introduced a flag in the top bit of R1 to select the `OS_ReadLine32` API instead. This identifies the API in use on `ReadLineV` in 26bit processor modes.

[RISC OS Open source page](#)

OS_ReadLine32

SWI &7D

The purpose of this call is to read a line from the current input stream.

Before calling

R0 Buffer

R1 Size of buffer -1 (maximum number of bytes to return excluding CR)

R2 Lowest char allowed

R3 Highest char allowed

R4 Flags and character to output in bottom byte (if bit 30 set)

On return

R0 "Corrupted" (all implementations have preserved R0)

R1 Bytes read, excluding Return

R2 "Corrupted" (all implementations have preserved R2)

R3 "Corrupted" (all implementations have preserved R3)

R4 Preserved

C flag is set if input is terminated by an escape condition

Notes

Input is terminated with Return, Ctrl-J or Escape. Delete and Backspace deletes last char entered. Ctrl-U empties buffer.

When the buffer is full, each successive attempt to enter a character results in a Ctrl-G 'beep'.

The CallASWI module (version 0.03) allows this SWI to be used with RISC OS pre 5.00.

[RISC OS Open source page](#)

OS_ReadMemMapEntries

SWI &52

The purpose of this call is to return the logical page number in which the memory pages specified reside. i.e. page address

Before calling

R0 Pointer to request block

On return

R0 Block at R0 updated

[RISC OS Open source page](#)

OS_ReadMemMapInfo

SWI &51

The purpose of this call is to read the page size used by the memory controller and the number of pages in use.

Before calling

--

On return

R0 Page size in bytes

R1 Number of pages

Notes

Valid page numbers are 0 to R1-1, and the total memory size is R0*R1. It's recommended to calculate this as a 64 bit value, as modern systems may have over 4GB of RAM.

[RISC OS Open source page](#)

OS_ReadModeVariable

SWI &35

The purpose of this call is to read information about a particular screen mode without having to change into that mode.

Before calling

R0 Screen mode, or -1 for current

R1 Variable number

On return

R0 Preserved

R1 Preserved

R2 Value of variable

C flag is set if variable or mode were invalid

In practice

The screen mode passed via R0 can be a mode number, a pointer to a Mode Selector Block, or a Sprite Mode Word.

Notes

This call does not check whether the video hardware can support the specified mode. For example, on a machine that only supports colour depths of 8bpp or greater, it is still possible to query the details of 1bpp modes, whether that's via a mode number, mode selector block, or sprite mode word. This is necessary in order for sprites to work correctly. To check whether a mode is supported by the video hardware, use `OS_CheckModeValid`.

Because this call does not check the capabilities of the video hardware, the information returned isn't always an accurate reflection of the mode variable values that will be used once the mode has been selected. For example, the `LineLength` and `ScreenSize` mode variables may be inaccurate if the video driver has restrictions on the framebuffer row stride. Or if a numbered mode is requested which can't be directly supported by the hardware, a substitute mode will typically be used

instead (e.g. the 256-colour mode 28 may be used instead of the 16-colour mode 27) - `OS_CheckModeValid` will indicate that this substitution will take place, but `OS_ReadModeVariable` will not.

For these reasons, it's also important to make sure that when you're interested in the details for the current mode, you either specify -1 as the mode number or you use another API such as `OS_ReadVduVariables`. Manually specifying the current mode number or selector block may result in inaccurate results.

[RISC OS Open source page](#)

OS_ReadMonotonicTime

SWI &42

The purpose of this call is to return the number of centiseconds since the last hard reset.

Before calling

--

On return

R0 Number of centiseconds

[RISC OS Open source page](#)

OS_ReadPalette

SWI &2F

The purpose of this call is to read the palette setting of a particular colour that is sent to the hardware.

R1 selects whether the normal colour, border colour or pointer colour is read as follows:

Before calling

R0 Logical colour

R1 Type of colour

On return

R0 Preserved

R1 Preserved

R2 Setting of colour (or first flashing colour if **R2**!=**R3** on exit)

R3 Setting of second flashing colour

/ Type

16 – Type: Normal colour

24 – Type: Border colour

25 – Type: Pointer colour

Notes

If **R2**=**R3** on exit the colour requested is a non-flashing colour. The values are to interpret as follows:

Bits / Meaning

0-6 – Meaning: Value showing how colour was programmed

7 – Meaning: Supremacy bit

8-15 – Meaning: Amount of red

16-23 – Meaning: Amount of green

24-31 – Meaning: Amount of blue

The bottom byte returns the second parameter to the VDU 19 command. For example:

Value / Meaning

0-15 – Meaning: Actual colour (BBC compatible)

16 – Meaning: Defined by giving amounts of R, G and B

17-18 – Meaning: Flashing colour defined by giving amounts of red, green and blue

[RISC OS Open source page](#)

OS_ReadPoint

SWI &32

The purpose of this call is to return the colour and tint of a point.

Before calling

R0 X co-ordinate

R1 Y co-ordinate

On return

R0 Preserved

R1 Preserved

R2 Colour

R3 Tint

R4 -1 if point was off screen, else 0

Notes

If in a 16 or 32 bpp mode, this will return the 15 or 24 bit colour number in R2 and tint will be 0.

[RISC OS Open source page](#)

OS_ReadRAMFsLimits

SWI &4A

The purpose of this call is to reads where in memory the RAM disc resides.

Before calling

--

On return

R0 Start of RAM disc

R1 End of RAM disc

Notes

This is not really all that useful as for the majority of the time the RAM disc is set as inaccessible to user programs.

[RISC OS Open source page](#)

OS_ReadSysInfo

SWI &58

The purpose of this call is to read and return information about the system.

Before calling

R0 Reason code

On return

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

Reason Codes

- 0 – Hex #: &00 · Action: Read configured screen size
- 1 – Hex #: &01 · Action: Read configured screen mode and monitor type
- 2 – Hex #: &02 · Action: Machine ID and IO chip presence
- 3 – Hex #: &03 · Action: 82C71x features
- 4 – Hex #: &04 · Action: Machine Ethernet address
- 5 – Hex #: &05 · Action: Machine ID raw
- 6 – Hex #: &06 · Action: Kernel value
- 7 – Hex #: &07 · Action: Information about last unexpected abort (data or prefetch)
- 8 – Hex #: &08 · Action: Platform class
- 9 – Hex #: &09 · Action: Version information
- 10 – Hex #: &0A · Action: Operating system version
- 11 – Hex #: &0B · Action: Debug information
- 12 – Hex #: &0C · Action: Extended Machine ID
- 13 – Hex #: &0D · Action: Validate key handler/read supported flags
- 14 – Hex #: &0E · Action: IIC bus count
- 15 – Hex #: &0F · Action: Enumerate extended ROM footer entries

[RISC OS Open source page](#)

OS_ReadSysInfo 0

SWI &58 · OS_ReadSysInfo

The purpose of this call is to read the configured size of screen memory.

Before calling

R0 0

On return

R0 Screen size (in bytes)

Notes

The amount of memory allocated for the screen can be configured using *Configure ScreenSize.

[RISC OS Open source page](#)

OS_ReadSysInfo 1

SWI &58 · OS_ReadSysInfo

The purpose of this call is to read the configured Mode/WimpMode, MonitorType, and Sync settings.

Before calling

R0 1

On return

R0 Screen mode (mode number or Mode Selector Block)

R1 Monitor type

R2 Sync:

0 = Vertical

1 = Composite

Notes

If any of the settings have been configured to 'Auto', then the appropriate value for the attached monitor is returned.

Automatic selection of settings is a three-stage process:

- First, Service_MonitorLeadTranslation is issued. If the service call is claimed then the settings returned by that call will be used.
- If the service call failed, the kernel will use its builtin translation table to select the settings.
- Additionally, from RISC OS 5.21 onwards, if the service call fails then the mode number selected from the kernel's builtin table may also be overridden by the current GraphicsV driver via the Select startup mode call.

In all cases, only the settings which were set to 'Auto' in CMOS will be overridden by the automatic selection logic.

[RISC OS Open source page](#)

OS_ReadSysInfo 10

SWI &58 · OS_ReadSysInfo

The purpose of this call is to read the version number of the Operating System and kernel, along with additional information about the SystemInit component.

Before calling

R0 10

On return

R0 OS version number * 100 ·

R1 Kernel version number * 100 ·

R2 SystemInit version number and system type ·

R3 Reserved (0) ·

Notes

This reason code was introduced in RISC OS Select, and is currently unsupported by RISC OS 5. For more details about the call please see the corresponding RISC OS Select documentation

[RISC OS Open source page](#)

OS_ReadSysInfo 11

SWI &58 · OS_ReadSysInfo

The purpose of this call is to obtain the addresses of functions that can be used to send and receive bytes to an external debugger. The two calls operate as follows:

Before calling

R0 11

R0 Character to transmit

--

On return

R0 Pointer to debug character output function

R1 Pointer to debug character input function

R0-R3, PSR flags Corrupted

R0 Character received, or -1 if none available

R1-R3, PSR flags Corrupted

Debug character input

The functions are expected to be called from a privileged mode, with a valid stack pointer.

Notes

On RISC OS 5 this call returns pointers to code that wraps the HAL_DebugTX and HAL_DebugRX HAL calls. If either of the calls are unavailable OS_ReadSysInfo will return with an error.

On RISC OS Select, the call will return pointers to the debug input/output code managed by the SystemInit system.

For RISC OS 5, this call first became available in RISC OS 5.17.

[RISC OS Open source page](#)

OS_ReadSysInfo 12

SWI &58 · OS_ReadSysInfo

For machines which provide machine IDs that differ from Acorn's traditional ID format, this call allows you to read the full, unmodified ID.

Before calling

R0 12

R1 Pointer to word-aligned buffer, or 0 to read required buffer size

On return

R0 Size of extended ID, or 0 if extended ID not present/corrupt

R1 Preserved. If the buffer pointer was supplied, the extended ID will have been copied into it.

Notes

For compatability with old software, if a machine's HAL provides an extended ID then the kernel will use that extended ID to synthesise an Acorn-style machine ID. This synthesised ID will be then be returned by any calls to `OS_ReadSysInfo 2/OS_ReadSysInfo 5`.

Because of this behaviour it is recommended that new software should first attempt to read the extended ID via `OS_ReadSysInfo 12` before falling back on the ID returned by `OS_ReadSysInfo 2/5`. This will help to ensure the uniqueness of the ID that the program uses.

This call first became available in RISC OS 5.17.

[RISC OS Open source page](#)

OS_ReadSysInfo 13

SWI &58 · `OS_ReadSysInfo`

The purpose of this call is to allow the owner of a key handler to validate it before installing it via `OS_InstallKeyHandler`, or to read which key handler flags this OS version supports.

Before calling

R0 13

R1 Pointer to key handler, or 0 to just return valid flags

On return

R0 Mask of valid key handler flags

Notes

If a key handler has been supplied and is invalid, error &1F4 ("Bad key handler") will be returned.

If no key handler has been supplied and the SWI generates an error (e.g. &1EC, "Unknown `OS_ReadSysInfo` call"), then take it as an indication that the kernel doesn't support the new key handler format at all.

[RISC OS Open source page](#)

OS_ReadSysInfo 14

SWI &58 · `OS_ReadSysInfo`

The purpose of this call is to allow code to determine the number of IIC buses that are present in the system and are supported by `OS_IICOp`.

Before calling

R0 14

On return

R0 Number of IIC buses present in system

Notes

This call was introduced in RISC OS 5.21. For older versions of RISC OS 5 which do not support this call, it is permissible to determine the IIC bus count by using `OS_Hardware` to call `HAL_IICBuses` (`HAL_IICBuses` is normally for internal use only,

therefore code which needs to know the bus count should prefer to use this SWI instead wherever possible).

[RISC OS Open source page](#)

OS_ReadSysInfo 15

SWI &58 · OS_ReadSysInfo

This call can be used to enumerate the data items which are present in the “extended ROM footer” which is present in most modern versions of RISC OS 5.

Before calling

R0 15

R1 0 to start enumeration

Else value from previous call to continue enumeration

On return

R1 Pointer to entry data, or 0 if no more entries found

R2 Entry tag ID (corrupt if R1 == 0)

R3 Entry size (corrupt if R1 == 0)

All other registers preserved

Notes

The contents of the ROM footer is generally only of interest to low-level system components. No guarantees are made as to what tags will be available on any given system. To allow for the possibility of tag IDs or tag content to be changed in future versions of the OS, the use of this SWI by user software is discouraged.

This call was introduced in RISC OS 5.23.

[RISC OS Open source page](#)

OS_ReadSysInfo 2

SWI &58 · OS_ReadSysInfo

The purpose of this call is to check for the presence of various chips, returning flags.

Before calling

R0 2

On return

R0 Hardware configuration word 0

R1 Hardware configuration word 1

R2 Hardware configuration word 2

R3 Low half of machine ID

R4 High half of machine ID

Hardware configuration words

0 · 0-7 – Use: Special functions chip type · Values: 0 = none/unspecified

1 = IOEB

0 · 8-15 – Use: I/O controller chip type · Values: 0 = IOC

1 = IOMD

255 = unspecified

0 · 16-23 – Use: Memory controller chip type · Values: 0 = MEMC1/MEMC1a

1 = IOMD

255 = unspecified

0 · 24-31 – Use: Video controller chip type · Values: 0 = VIDC1a

1 = VIDC20

255 = unspecified

1 · 0-7 – Use: I/O chip type · Values: 0 = none/unspecified

1 = 82C710 or similar

1 · 8-31 – Use: Reserved · Values: 0

2 · 0-7 – Use: LCD controller type · Values: 0 = none

1 = type 1 (e.g. A4 portable)

2 = type 2 (e.g. Stork portable)

2 · 8-15 – Use: IOMD variant · Values: 0 = IOMD

1 = IOMDL (i.e. ARM7500)

2 · 16-23 – Use: VIDC20 variant · Values: 0 = VIDC20

1 = VIDC2L (i.e. ARM7500)

2 · 24 – Use: IIC bus speed · Values: 0 = slow (100kHz)

1 = fast (400kHz)

2 · 25 – Use: Clock state during idle · Values: 0 = Keep I/O clocks running during idle

1 = Stop I/O clocks during idle

2 · 26-31 – Use: Reserved · Values: 0

In practice

It also reads the unique machine ID if a suitable chip is fitted to the computer; if none is, an ID of zero is returned.

Notes

The content of hardware information word 2 is generally only correct if the system is IOC or IOMD based. Modern HAL based systems typically return 0 for the whole word.

When using this call to read the machine ID, it is recommended that programs first check with OS_ReadSysInfo 12 to see if an extended machine ID is available.

If you only wish to read the old-style 64 bit machine ID, then OS_ReadSysInfo 5 can be used instead.

[RISC OS Open source page](#)

OS_ReadSysInfo 3

SWI &58 · OS_ReadSysInfo

The purpose of this call is to read features available in the 82C71x chip.

Before calling

R0 3

On return

R0 82C710/82C711 basic features

R1 82C710/82C711 extra features

R2 Extended features 1 (reserved, 0)

R3 Extended features 2 (reserved, 0)

R4 Extended features 3 (reserved, 0)

[RISC OS Open source page](#)

OS_ReadSysInfo 4

SWI &58 · OS_ReadSysInfo

The purpose of this call is to read the device's Ethernet address.

Before calling

R0 4

On return

R0 Bottom 32 bits of Ethernet address (0 for none)

R1 Top 16 bits of Ethernet address (0 for none)

Notes

0 is returned if no address exists.

[RISC OS Open source page](#)

OS_ReadSysInfo 5

SWI &58 · OS_ReadSysInfo

The purpose of this call is to read the machine's unique ID raw.

Before calling

R0 5

On return

R0 Bottom 32 bits of machine unique ID (raw)

R1 Top 32 bits of machine's unique ID (raw)

Notes

When reading the machine ID, it is recommended that programs first check with OS_ReadSysInfo 12 to see if an extended machine ID is available.

OS_ReadSysInfo 2 can also be used to read the old-style 64 bit machine ID.

[RISC OS Open source page](#)

OS_ReadSysInfo 6

SWI &58 · OS_ReadSysInfo

This call is for internal use by the operating system and certain utilities (e.g. SoftLoad, ROMPatch). It should not be used by user programs. The purpose of the call is to return information about the Kernel's internal variables.

Before calling

R0 6

R1 0, or pointer to item list terminated by -1

R2 If R1=0 then Single item to read, else pointer to result buffer

On return

R0 Preserved

R1 Preserved

R2 If R1=0 on entry, value of item

If R1<>0 on entry, block pointed to by R2 is updated

In practice

When an item number is unrecognised or the item is no longer used, 0 will be returned as the value of that item.

[RISC OS Open source page](#)

OS_ReadSysInfo 7

SWI &58 · OS_ReadSysInfo

The purpose of this call is to return information on previous unexpected data or prefetch abort, i.e. the last abort that raised an error.

Before calling

R0 7

On return

R0 Preserved

R1 32-bit PC for last abort

R2 32-bit PSR for last abort

R3 Fault address for last abort

Notes

Prefetch abort details were not tracked by this call for RISC OS 5 versions prior to 5.29.

For data aborts, the fault address is the value of the DFAR register.

For prefetch aborts, the fault address is either the same as R1 (i.e. fault address plus four), or (if the IFAR register is implemented), the value of the IFAR register plus four.

[RISC OS Open source page](#)

OS_ReadSysInfo 8

SWI &58 · OS_ReadSysInfo

The purpose of this call is to read information about the hardware platform the Operating System is running on.

Before calling

R0 8

On return

R0 Platform class

R1 Flags

R2 Mask of which bits in R1 are valid on this class

[RISC OS Open source page](#)

OS_ReadSysInfo 9

SWI &58 · OS_ReadSysInfo

The purpose of this call is to return information about the installed version of the Operating System. There is no strict format for any of the returned strings – the main purpose of this call is to provide information for debug/logging purposes. It is not to be used to programmatically infer hardware or OS capabilities.

Before calling

R0 9

R1 Reason code

On return

R0 Pointer to string of the requested information, or 0 if not available

Notes

All strings are NULL terminated.

Reason Codes

0 – OS information to return: OS name

1 – OS information to return: OS part number

2 – OS information to return: OS build date

3 – OS information to return: Dealer name

4 – OS information to return: Registered user name

5 – OS information to return: Registered user address

6 – OS information to return: Printable OS description

7 – OS information to return: Hardware platform name

[RISC OS Open source page](#)

OS_ReadUnsigned

SWI &21

The purpose of this call is to convert a string into an integer.

Before calling

R0 Base (2-36 Default 10) + flags

R1 Pointer to string to convert

R2 Maximum value, if bit 29 set

R3 Most significant word of maximum value, if bits 28 & 29 set¹

R4 "WIDE" API selector (see notes)

On return

R0 Preserved

R1 Pointer to terminator char

R2 Value converted

R3 Most significant word of value if bit 28 set¹ else preserved

R4 Bitmask of R0 flags understood by current kernel¹ else preserved

In practice

¹ Only applicable if "WIDE" API is in use.

Notes

If the string starts with &, base 16 is assumed. If the string starts with a decimal number followed by an underscore (e.g. '8_'), the base specified by the number is used. If no base is given in the string, the base in R0 is used, or 10 if that base is invalid (i.e. outside the range 2-36).

Number conversion will terminate once the first non-number character is encountered. E.g. with the string "63fz", if the base was 10 then conversion would terminate once the 'f' is reached and a result of 63 would be returned. If the base was 16 then conversion would terminate once the 'z' is reached, with a result of 1599 (&63f), etc.

"WIDE" API

In RISC OS 5.17 the SWI interface was expanded to allow reading of 64 bit numbers. In order to use this new interface, the SWI must be called with R4 set to &45444957 ("WIDE"). Note that in OS versions prior to 5.17, bits 0-28 of R0 are treated as the number base, while in 5.17 and above only bits 0-7 are treated as the number base. This means care must be taken when writing code that makes use of the new WIDE flags, as old OS versions may interpret the flags as part of the base, and then reset the base to the default value of 10 because the base number looks nonsensical.

[RISC OS Open source page](#)

OS_ReadVarVal

SWI &23

The purpose of this call is to read a variable's value

Before calling

- R0** Pointer to wildcarded name
- R1** Pointer to buffer
- R2** Length of buffer, or -ve to check existence/read length
- R3** Name pointer, or 0 for 1st call
- R4** 3 to expand macros and numbers to strings

On return

- R0** Corrupt (pointer to error - see note on `_swix()` below)
- R1** Preserved
- R2** NOT (length), or 0 if variable does not exist
- R3** New name pointer
- R4** Variable type
- R0** Preserved
- R1** Preserved
- R2** bytes read
- R3** new name pointer
- R4** Variable type

Notes

This SWI returns type and value of a variable. To check if it exists or find its length, call with `R2 < 0`. If the variable does not exist `R2` will be 0. Otherwise the length is given by `NOT(R2)`, although for anything other than a string variable (type 0) this is the unexpanded length. For either case an error will be returned.

RISC OS 2 does not let you read the length; it merely returns `R2` negative to indicate the variable exists. On RISC OS 2 variables are limited to 256 bytes. Returned strings are not zero terminated (and may contain embedded zero bytes). Code variables seem to be indistinguishable from string variables (type 0). Numeric variables are written as 4 bytes in the buffer if expansion is not requested.

This SWI is marked as being re-entrant, which would allow it to re-enter the same code variable's read code. No note is made about this in `OS_SetVarVal` and the code variable example in the PRM calls `OS_Byte 135` which has undefined re-entrancy.

This SWI is incompatible with `_swix()` when `r2` is negative (to read the length of the variable) as it always returns a 'buffer overflow' error which should be ignored. `_swix()` optimises the error-case so doesn't return any values. `_kernel_swi()` could be used instead, though `_swix()` can be used for reading the variable once you have a suitably sized buffer.

[RISC OS Open source page](#)

OS_ReadVduVariables

SWI &31

The purpose of this call is to read a series of VDU variables.

Before calling

R0 Pointer to input block

R1 Pointer to output block (can be the same as R0)

On return

R0 Preserved

R1 Preserved

Notes

The input block is a list of VDU variable numbers (words) terminated by -1. Each variable is read, and its value is written as a word into the output block.

Note: You can also read the mode variables with this call.

[RISC OS Open source page](#)

OS_Release

SWI &20

The purpose of this call is to remove a routine from the list of those that claim a vector.

Before calling

R0 Vector number

R1 Address of releasing routine

R2 Value given in R2 when claimed

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

This 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 claim is made with the same parameters, only the first one to be called is removed.

[RISC OS Open source page](#)

OS_ReleaseDeviceVector

SWI &4C

The purpose of this call is to remove a driver from the list of claimants of a device vector.

Before calling

R0 Device number

R1 Address of device driver routine

R2 Value to be passed in R12 when driver is called

R3 Address of interrupt status if R0 = module "IRQ" or "FIQ as IRQ" on entry

R4 Interrupt mask to use, if R0 = module "IRQ" or "FIQ as IRQ" on entry

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

Notes

The device vector is identified by the contents of the register on entry; R0-R2 (R0-R4 if R0 relates to one of the module device numbers (see `Module_ReadInfo?`) when registered) must be the same as when the vector was claimed.

The previous vector owner is re-installed at the head of the chain. If there is no previous owner, then the IRQs from the corresponding device are disabled.

You must not attempt to disable a device's IRQs within the IO controller when you release its vector. For expansion card IRQs, you should prevent your device from interrupting again by programming the hardware locally on your expansion card.

[RISC OS Open source page](#)

OS_RelinkApplication

SWI &4E

The purpose of this call is to remove any vectors that an application is using.

Before calling

R0 Pointer to buffer

On return

R0 Preserved

Notes

When an application is going to be swapped in, all vectors that it uses must be restored.

R0 points to a buffer, which has previously been created using `OS_DelinkApplication`.

[RISC OS Open source page](#)

OS_RemoveCallback

SWI &5F

The purpose of this call is to remove a Callback event from the list.

Before calling

R0 Pointer to code called

R1 Value of R12 called with

On return

R0 Preserved

R1 Preserved

Notes

It removes a Callback event, such as OS_AddCallback.

[RISC OS Open source page](#)

OS_RemoveCursors

SWI &36

The purpose of this call is to remove the cursors from the screen.

Before calling

--

On return

--

Notes

This call removes the cursors (output and copy, if active) from the screen, saving the old state on an internal stack for further recovering. This call must be balanced by calling OS_RestoreCursors later however, the exit handler provides this.

[RISC OS Open source page](#)

OS_RemoveTickerEvent

SWI &3D

The purpose of this call is to remove a ticker timer event.

Before calling

R0 Pointer to code called

R1 Value of R12 called with

On return

R0 Preserved

R1 Preserved

Notes

Removes a ticker timer event, such as `OS_CallEvery` or `OS_CallAfter`. All events which match the specified parameters will be removed.

However, note that it is not possible to remove an `OS_CallEvery` event which is currently in the middle of execution.

[RISC OS Open source page](#)

OS_Reset

SWI &6A

The purpose of this call is to perform a hard reset.

Before calling

--

On return

--

Notes

This SWI performs the equivalent of pressing the Reset button on the hardware (if present).

Use with extreme caution.

If `OS_ReadSysInfo` 8 states that the machine supports software power control, then R0 with a value of "&OFF" (&46464F26) will turn the power off too.

Available from RISC OS 3.50 onwards.

[RISC OS Open source page](#)

OS_RestoreCursors

SWI &37

The purpose of this call is to restore the screen cursors.

Before calling

--

On return

--

Notes

This restores the cursor state previously saved on the internal stack using `OS_RemoveCursors`.

[RISC OS Open source page](#)

OS_ResyncTime

SWI &6C

The purpose of this call is to resync the software copy of the time to the value contained in the RTC chip.

Before calling

--

On return

--

[RISC OS Open source page](#)

OS_ScreenMode

SWI &65

The purpose of this call is to perform miscellaneous operations for screen mode handling.

Before calling

R0 Reason code

All other registers dependent on reason code

On return

R0 Preserved

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

Reason codes marked with ¶ are used by RISC OS Select's video driver interface and are unimplemented in RISC OS 5.

Reason Codes

- 0 – Hex #: &00 · Action: Select screen mode
- 1 – Hex #: &01 · Action: Return current screen mode
- 2 – Hex #: &02 · Action: Enumerate screen modes
- 3 – Hex #: &03 · Action: Change monitor type (internal use only)
- 4 – Hex #: &04 · Action: Configure acceleration
- 5 – Hex #: &05 · Action: Flush screen memory from cache
- 6 – Hex #: &06 · Action: Force flush screen memory from cache
- 7 – Hex #: &07 · Action: Return number of screen banks ¶
- 8 – Hex #: &08 · Action: Select display bank ¶
- 9 – Hex #: &09 · Action: Select VDU bank ¶
- 10 – Hex #: &0A · Action: Copy screen bank ¶
- 11 – Hex #: &0B · Action: Select VideoV Display or GraphicsV Device
- 12 – Hex #: &0C · Action: Read details about VideoV Display device ¶
- 13 – Hex #: &0D · Action: Convert mode string to mode selector block
- 14 – Hex #: &0E · Action: Convert mode selector block to mode string

- 15 – Hex #: &0F · Action: Select screen mode by string
- 64 – Hex #: &40 · Action: Register GraphicsV Device driver
- 65 – Hex #: &41 · Action: Start GraphicsV Device driver
- 66 – Hex #: &42 · Action: Stop GraphicsV Device driver
- 67 – Hex #: &43 · Action: Deregister GraphicsV Device driver
- 68 – Hex #: &44 · Action: Enumerate GraphicsV Device drivers
- 253 – Hex #: &FD · Action: Read maximum VideoV Display driver ¶
- 254 – Hex #: &FE · Action: Deregister VideoV Display driver ¶
- 255 – Hex #: &FF · Action: Register VideoV Display driver ¶

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 0

SWI &65 · OS_ScreenMode

The purpose of this call is to select a given screen mode.

Before calling

R0 0

R1 Mode Specifier

On return

R0 Preserved

R1 Preserved

Notes

An error will be returned if the mode selected is invalid.

See also OS_ScreenMode 15 to set the screen mode using a mode string.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 1

SWI &65 · OS_ScreenMode

The purpose of this call is to return information about the current screen mode.

Before calling

R0 1

On return

R0 Preserved

R1 Mode Specifier

Notes

It is also possible to return information about the current screen mode using OS_Byte 135.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 10

SWI &65 · OS_ScreenMode

This call copies the contents of one bank to another.

Before calling

R0 10 (RISC OS 6 only)

R1 Bank number to copy from, 1+

R2 Bank number to copy to, 1+

Notes

RISC OS 6 only – calls VideoV?.

[RISC OS Open source page](#)

OS_ScreenMode 11

SWI &65 · OS_ScreenMode

The purpose of this call is to select the active video driver, or to return the current driver.

Before calling

R0 11

R1 New GraphicsV Device / VideoV Display number, or -1 to read current

On return

R1 Previous (or current, if reading) driver number

All other registers preserved

Notes

An error will be returned if the selected Device or Display does not exist, or if the driver refuses to activate.

On RISC OS 5 this selects which GraphicsV Device driver is in use, while on RISC OS Select this selects which VideoV? Display driver is in use.

On RISC OS 5 the value -2 will be returned in R1 if no driver is active. -2 can also be passed in to deactivate the video system, however this is inadvisable as the OS does not fully cope with no driver being active.

Under both RISC OS 5 and RISC OS Select, Service_DisplayChanged will be issued as appropriate in order to notify the rest of the system that the current driver is changing.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 13

SWI &65 · OS_ScreenMode

The purpose of this call is to convert a mode string to a mode selector block.

Before calling

R0 13

R1 Pointer to Mode String

R2 Pointer to Mode Selector Block to fill in

R3 Size of specifier buffer (bytes)

On return

R0 Error, or preserved

- All other registers preserved

Notes

It's not possible to read required size of the selector block using this call. Instead the caller must make an educated guess as to the required buffer size. It's reasonable to expect up to four mode variables to be returned (NColour, mode flags, X & Y eigen factors), but future extensions or other implementations may return more.

See also OS_ScreenMode 14 to convert a mode specifier to a mode string.

This call is available in RISC OS Select and RISC OS 5.21+

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 14

SWI &65 · OS_ScreenMode

The purpose of this call is to convert a mode selector block to a mode string.

Before calling

R0 14

R1 Pointer to Mode Selector Block

R2 Pointer to Mode String to fill in

R3 Size of string buffer (bytes)

On return

R0 Error, or preserved

R3 Less than zero: Negative size of buffer required

Zero: String has been filled in

- All other registers preserved

In practice

To determine the size of the required buffer, call once with R3 set to zero, and then allocate -R3 bytes to store the string.

Notes

Note that converting a mode selector block to a string and then back to a selector may result in the loss of some information; mode strings are not capable of representing the full set of values that can be specified in mode selectors.

See also OS_ScreenMode 13 to convert a mode string to a mode selector block.

This call is available in RISC OS Select and RISC OS 5.21+

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 15

SWI &65 · OS_ScreenMode

The purpose of this call is to select a given screen mode as specified by a mode string.

Before calling

R0 15

R1 Mode String

On return

R0 Preserved

R1 Preserved

Notes

An error will be returned if the mode selected is invalid.

See also OS_ScreenMode 0 to set the screen mode using a selector block or mode number.

This call is available in RISC OS Select and RISC OS 5.21+

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 2

SWI &65 · OS_ScreenMode

The purpose of this call is to enumerate the available screen modes.

Before calling

R0 2

R2 Value of R2 to pass to Service_EnumerateScreenModes

R6 Value of R6

R7 Value of R7

On return

R0 Preserved

R1 Value of R2 to returned by Service_EnumerateScreenModes

R2 Value of R2 returned

R6 Value of R6 returned

R7 Value of R7 returned

Notes

This SWI provides a front end to Service_EnumerateScreenModes.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 3

SWI &65 · OS_ScreenMode

The purpose of this call is to alter the monitor type of a running system.

Before calling

R0 3

R1 Monitor type, or -1 to reset to the configured type

On return

R0 Preserved

R1 Preserved

Notes

This call is for internal use only.

It allows ScreenModes to set the current monitor type to 7 when mode definitions are loaded from an MDF or EDID, thereby disabling the kernel's builtin mode definitions.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 4

SWI &65 · OS_ScreenMode

This SWI is used to configure the acceleration in use by the current display driver.

Before calling

R0 4

R1 flags. , or -1 to read current value

bit 0 = set to suspend cached screen until mode change

bit 1 = set to suspend screen cleaning

bit 2 = set to disable hardware acceleration

Other = reserved, must be 0

R2 number of VSynCs between automatic screen cleaning (1-3), or -1 to read current value

On return

R0 Preserved

R1 new flag state (value actually invoked given any restrictions for current platform), or current flag state if R1 = -1 on entry

R2 new number of VSynCs between automatic screen cleaning, or current number of VSynCs if R2 = -1 on entry

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 5

SWI &65 · OS_ScreenMode

This SWI is used to clean the cache for the screen and only works if screen caching hasn't been previously disabled via OS_ScreenMode 4.

Before calling

R0 5

On return

R0 Preserved

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 6

SWI &65 · OS_ScreenMode

This SWI will forcibly clean the cache for the screen whether disabled or not via OS_ScreenMode 4.

Before calling

R0 6

On return

R0 Preserved

Other languages documentation

[Français](#)

[RISC OS Open source page](#)

OS_ScreenMode 64

SWI &65 · OS_ScreenMode

The purpose of this call is to register a new GraphicsV Device driver with the system. This call should be made before the driver installs itself on the GraphicsV vector, so that the driver knows which driver ID to respond to.

Before calling

R0 64

R1 Flags (reserved, should be zero)

R2 Pointer to GraphicsV Device driver name

On return

R0 Allocated driver number

All other registers preserved

In practice

Once the driver has installed itself on GraphicsV and completed its hardware initialisation, it should call OS_ScreenMode 65 to indicate that it is ready for use.

Notes

The driver name pointed to by R2 must be null-terminated and must remain valid for the lifetime of the driver (i.e. until OS_ScreenMode 67 is called). The name is intended to be used by the OS and applications as a unique identifier for the device; the following conventions should be used:

- The name must not contain spaces or other non-printable characters. This makes it easy to use the driver name on the command line, in system variables, configuration files, etc.
- In most cases the driver module name is an acceptable driver identifier.
- If the same driver is able to support multiple devices, use a name of the format '`<driver>#<id>`' where `<driver>` is the driver name and `<id>` is a unique identifier for the device instance
- Where possible the device id suffix must uniquely identify the same device on the same system across system reboots. For modules, the module number should be sufficient. For PCI devices, the PCI device handle should be sufficient. For USB

devices, using the usb device number is not recommended as USB devices may be connected/disconnected frequently or the number may be unstable across reboots due to device enumeration order. Instead some combination of the vendor & product ID or device name are recommended.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 65

SWI &65 · OS_ScreenMode

GraphicsV Device drivers should make this call once they have completed their initialisation and are ready for use by the OS. Once this call has been made the driver should expect to receive requests on the GraphicsV vector.

Before calling

R0 65

R1 GraphicsV Device driver number

On return

All registers preserved

Notes

Prior to making this call, OS_ScreenMode 64 should be used to request a driver ID from the system, and the driver must install itself on the GraphicsV vector.

When the driver is ready to shut down, it must call OS_ScreenMode 66, remove itself from GraphicsV, and then call OS_ScreenMode 67 to release the driver ID.

Service_DisplayStatus will be issued by this call in order to notify the rest of the system that the new driver is ready for use. The kernel may also attempt to start using the driver if there is currently no other driver active.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 66

SWI &65 · OS_ScreenMode

GraphicsV Device drivers should make this call when they wish to deregister themselves from the system. Once the call returns, the driver should no longer be receiving calls on GraphicsV and can safely remove itself from the vector by calling OS_Release. It can also stop generating VSync calls.

Before calling

R0 66

R1 GraphicsV Device driver number

On return

All registers preserved

Notes

An error will be returned if the driver ID is invalid or the driver has not started itself using OS_ScreenMode 65.

Once the driver has released its claim on GraphicsV and stopped generating VSync calls, it should call OS_ScreenMode 67 to release the driver ID.

Service_DisplayStatus will be issued by this call in order to notify the rest of the system that the driver is shutting down.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 67

SWI &65 · OS_ScreenMode

GraphicsV Device drivers should make this call in order to release a driver ID that was previously allocated by OS_ScreenMode 64.

Before calling

R0 67

R1 GraphicsV Device driver number

On return

All registers preserved

Notes

An error will be returned if the driver ID is invalid or the driver is still in the 'started' state (i.e. OS_ScreenMode 66 has not been called).

Before making this call the driver must make sure it has released its claim on GraphicsV and has stopped generating VSync calls.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 68

SWI &65 · OS_ScreenMode

This call is used to enumerate the list of registered GraphicsV Device drivers and to read basic details about them.

Before calling

R0 68

R1 GraphicsV Device driver number to return details of, or -2 to return the ID of the first valid driver

R2 Flags (reserved, should be zero)

On return

R1 ID of next driver, or -2 if no more to enumerate

R2 Driver flags on registration (R1 of OS_ScreenMode 64)

R3 Driver name on registration (R2 of OS_ScreenMode 64)

R4 Driver state:

0 = Driver is invalid/unallocated (R2-R3 invalid)

1 = Driver is initialising or shutting down and cannot be used

2 = Driver is ready for use

All other registers preserved

Notes

When calling GraphicsV manually, it is only safe to interact with drivers which are in state 2. This state corresponds to a driver which has called both OS_ScreenMode 64 and OS_ScreenMode 65.

To determine the current driver, pass -1 to OS_ScreenMode 11.

Other languages documentation

Français

[RISC OS Open source page](#)

OS_ScreenMode 7

SWI &65 · OS_ScreenMode

The purpose of this call is to read the number of display banks available to be selected by OS_Byte 112 and OS_Byte 113.

Before calling

R0 7 (RISC OS 6 only)

On return

R1 Number of display banks available

All other registers preserved

Notes

RISC OS 6 only – calls VideoV?.

[RISC OS Open source page](#)

OS_ScreenMode 8

SWI &65 · OS_ScreenMode

This call is used in RISC OS 6 only to implement OS_Byte 113.

Before calling

R0 8 (RISC OS 6 only)

R1 Bank number to display, 1+

Notes

Calls VideoV?.

[RISC OS Open source page](#)

OS_ScreenMode 9

SWI &65 · OS_ScreenMode

This call is used in RISC OS 6 only to implement OS_Byte 112.

Before calling

R0 9 (RISC OS 6 only)

R1 Bank number for VDU code, 1+

Notes

Calls VideoV?.

[RISC OS Open source page](#)

OS_SerialOp

SWI &57

The purpose of this call is to access the serial device driver directly.

Before calling

R0 Reason code

All other registers dependent on reason code

On return

R0 Preserved

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

OS_SerialOp is vectored via SerialV.

Reason Codes

- 0 – Hex #: &00 · Action: Read/write serial states
- 1 – Hex #: &01 · Action: Read/write data format
- 2 – Hex #: &02 · Action: Send break
- 3 – Hex #: &03 · Action: Send byte
- 4 – Hex #: &04 · Action: Get byte
- 5 – Hex #: &05 · Action: Read/write receive baud rate
- 6 – Hex #: &06 · Action: Read/write transmit baud rate
- 7 – Hex #: &07 · Action: Internal use only
- 8 – Hex #: &08 · Action: Read/write serial input buffer threshold value
- 9 – Hex #: &09 · Action: Enumerate available serial port speeds
- 10 – Hex #: &0A · Action: Get device name

[RISC OS Open source page](#)

OS_SerialOp 0

SWI &57 · OS_SerialOp

The purpose of this call is to read/write the serial state.

Before calling

R0 0

R1 EOR mask

R2 AND mask

On return

R0 Preserved

R1 Old value of state

R2 New value of state

Notes

The structure of this call is very similar to that of OS_Byte's between &A6 and &FF the new state is determined by:

New State (see Serial Status Word) = (Old state AND R2) EOR R1

[RISC OS Open source page](#)

OS_SerialOp 1

SWI &57 · OS_SerialOp

The purpose of this call is to set the encoding of characters when sent and received on the serial line.

Before calling

R0 1

R1 -1 to read, or new format value

On return

R0 Preserved

R1 Old format value
[RISC OS Open source page](#)

OS_SerialOp 10

SWI &57 · OS_SerialOp

The purpose of this call is to determine the name of the serial device within DeviceFS. E.g. a returned name of 'Serial' corresponds to 'Devices:\$.Serial'.

Before calling

R0 10

On return

R0 Preserved

R1 Pointer to device name

Notes

This call first became available in RISC OS 5.19. If the call is unavailable or returns an error, your code should use the default device name of 'Serial'.

[RISC OS Open source page](#)

OS_SerialOp 2

SWI &57 · OS_SerialOp

The purpose of this call is to send a break down the serial device line.

Before calling

R0 2

R1 Length of break in centiseconds

On return

R0 Preserved

R1 Preserved

Notes

This call sets the ACIA to transmit a break, then waits R1 centiseconds before resetting it to normal. Any character being transmitted at the time the call is made may be garbled. After sending the break the transmit process is either awakened if the buffer is not empty, or made dormant if the buffer is empty.

[RISC OS Open source page](#)

OS_SerialOp 3

SWI &57 · OS_SerialOp

The purpose of this call is to send a byte into the serial output buffer.

Before calling

R0 3

R1 Character to be sent

On return

R0 Preserved

R1 Preserved

C flag clear if character was sent, or set if character was not sent (ie the buffer was full)

Notes

This call puts a character in the serial output buffer, and re-enables the transmit interrupt if it had been disabled by RISC OS. If the serial output buffer is full, the call returns immediately with the C flag set.

[RISC OS Open source page](#)

OS_SerialOp 4

SWI &57 · OS_SerialOp

The purpose of this call is to remove a character from the serial input buffer if one is present.

Before calling

R0 4

On return

R0 Preserved

R1 Character received (if C flag cleared), or
preserved (if C flag set) i.e. no character available in buffer to read)

Notes

If removing a character leaves the input buffer with more free space than are specified by OS_Byte 203, then the transmitting device is re-enabled in the way specified by the serial port state (as set by OS_SerialOp 0).

[RISC OS Open source page](#)

OS_SerialOp 5

SWI &57 · OS_SerialOp

The purpose of this call is to read or set the receive baud rate of the serial device.

Before calling

R0 5

R1 -1 to read, or 0-19 to set to a baud rate

On return

R0 Preserved

R1 Old receive baud rate

Notes

The transmit baud rate can be read or set using OS_SerialOp 6.

[RISC OS Open source page](#)

OS_SerialOp 6

SWI &57 · OS_SerialOp

The purpose of this call is to read or set the receive baud rate of the serial device.

Before calling

R0 6

R1 -1 to read, or 0-19 to set to a baud rate

On return

R0 Preserved

R1 Old transmit baud rate

Notes

The receive baud rate can be read or set using OS_SerialOp 5.

[RISC OS Open source page](#)

OS_SerialOp 8

SWI &57 · OS_SerialOp

The purpose of this call is to read or write the input buffer threshold value.

Before calling

R0 8

R1 -1 to read, or new threshold

On return

R0 Preserved

R1 Old threshold

Notes

OS_SerialOp 0 can be used to examine or change the handshaking method.

The default value on RISC OS 3.5 is 17 characters, but this is subject to change and should not be relied upon.

The serial input routine attempts to halt input when the amount of free space left in the input buffer falls below a certain level. This call allows the value at which input is halted to be read or changed.

[RISC OS Open source page](#)

OS_SerialOp 9

SWI &57 · OS_SerialOp

The purpose of this call is to read or enumerates the available serial port speeds.

Before calling

R0 9

On return

R0 Preserved

R1 Pointer to word aligned table of supported baud rates

R2 Number of entries in table

Notes

Each entry in the table is in units of 0.5 bit/sec.

The index into the table (starting at 1) can be used in OS_SerialOp 5 and OS_SerialOp 6 calls to set the corresponding baud rate.

As the hardware on a given machine may not support certain baud rate combinations the table may have 'holes' in it, represented by a zero word - this way the index into the table can still be used with OS_SerialOp 5.

[RISC OS Open source page](#)

OS_ServiceCall

SWI &30

The purpose of this call is to issue a service call to a module.

Before calling

R1 Service number

All other registers dependent on reason code

On return

R1 0 if service call was claimed

Other registers may be altered if the service was claimed

Notes

This call should be used in instead of OS_Byte 143, as it is now deprecated.

[RISC OS Open source page](#)

OS_SetCallBack

SWI &1B

The purpose of this call is to cause a call to the CallBack handler.

Before calling

--

On return

--

Notes

This sets the CallBack flag and so causes entry to the CallBack handler when the system next exits to user mode code with interrupts enabled (apart, of course, from the exit from this SWI).

This SWI may be used if the code linked into the system (via vector or as a SWI handler, etc) is required to do things on exit from the system. The callback handler is set with OS_ChangeEnvironment, and is distinct from the "transient" CallBacks set with OS_AddCallBack.

[RISC OS Open source page](#)

OS_SetColour

SWI &61

The purpose of this call is to set or read the text and graphics colours.

Before calling

R0 Flag word

R1 Pointer to ECF pattern (if R0 bit5 is set), or
Colour number (if R0 bit5 clear)

On return

--

Notes

When writing colours, graphics colours may be set using a pattern block, text colours must be specified as a colour number. When reading the colours, text colours are returned as a colour number in R1, but you must supply a pattern block to read the graphics colours.

[RISC OS Open source page](#)

OS_SetECFOrgin

SWI &56

The purpose of this call is to set the origin of the ECF pattern.

Before calling

R0 X co-ordinate

R1 Y co-ordinate

On return

R0 Preserved

R1 Preserved

Notes

This SWI sets the point with which to align the bottom left of ECF patterns with.

This call is the equivalent of VDU 23,17,6.

[RISC OS Open source page](#)

OS_SetEnv

SWI &12

The purpose of this call is to set environment parameters.

Before calling

R0 Address of the handler for OS_Exit, 0 to read

R1 Address of the memory limit for OS_GetEnv, 0 to read

R4 Address for undefined instructions, 0 to read

R5 Address for prefetch abort, 0 to read

R6 Address for handler for data abort, 0 to read

R7 Address for handler for address exceptions, 0 to read

On return

R0 Address of the previous "exit handler"

R1 Address of the previous "memory limit handler"

R4 Address of the previous "undefined instruction handler"

R5 Address of the previous "prefetch abort handler"

R6 Address of the previous "data abort handler"

R7 Address of the previous "address exception handler"

Notes

This call is now deprecated. Please use OS_ChangeEnvironment instead.

[RISC OS Open source page](#)

OS_SetMemMapEntries

SWI &53

The purpose of this call is to set the logical to physical mapping of the memory pages.

Before calling

R0 Pointer to request block

On return

R0 Block at R0 updated

Notes

Use of this call is discouraged. OS_Memory 0 and the facilities offered by user-created dynamic areas should be considered as alternatives to using this call.

If you are using this call, then you can only use OS_ChangeDynamicArea or other memory management SWIs if the kernel's limits are maintained, and all appropriate areas contain continuous memory.

Prior to RISC OS 5, only the lower two bits of the page flags were used - the other bits were forced to zero.

[RISC OS Open source page](#)

OS_SetVarVal

SWI &24

The purpose of this call is to write a variable value.

Before calling

R0 Pointer to name

R1 Pointer to value to set to

R2 Length, or -1 to delete

R3 Name pointer (0 for 1st call)

R4 Variable type

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 New name pointer

R4 Variable type

Notes

This SWI sets a variable's value to that specified, or deletes the variable. The name may be wildcarded for deletion and update (using '*' and '#'). Code variables will not be deleted unless R4 = 16. Literal strings do not need to be null terminated, as R2 is used for the length.

OS_ReadVarVal is marked as being re-entrant, which would allow it to re-enter the same code variable's read code. No note is made about this in the PRM and the code variable example in the PRM calls OS_Byte 135 which has undefined re-entrancy!

[RISC OS Open source page](#)

OS_SpriteOp

SWI &2E

The purpose of this call is to perform different sprite related actions.

Before calling

R0 Reason code + Area value

All other registers dependent on reason code

On return

R0 Preserved

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

Area Value

The area value can be added to the reason code in R0 to specify how to select a sprite. R1 and R2 are often used to select a sprite.

Value / R1 / R2

+0 – R1: System Sprite area1 · R2: Sprite name

+256 – R1: Sprite area · R2: Sprite name

+512 – R1: Sprite area · R2: Sprite pointer

1 It is strongly recommended not to use the system sprite area.

When using a sprite name, the string can be terminated with a space or control character (i.e. any character ≤ 32). Sprite name matching is case-insensitive, and only the first 12 bytes of the supplied name will be examined.

This SWI is indirected via SpriteV.

Reason Codes

2 – Hex #: &02 · Action: Screen save 2

3 – Hex #: &03 · Action: Screen load

8 – Hex #: &08 · Action: Read area control block

9 – Hex #: &09 · Action: Initialise sprite area 2

10 – Hex #: &0A · Action: Load sprite file 2

11 – Hex #: &0B · Action: Merge sprite file

12 – Hex #: &0C · Action: Save sprite file

13 – Hex #: &0D · Action: Return name

14 – Hex #: &0E · Action: Get sprite from cursor 2

15 – Hex #: &0F · Action: Create sprite 2

16 – Hex #: &10 · Action: Get sprite from user coords 2

17 – Hex #: &11 · Action: Verify sprite area

23 – Hex #: &27 · Action: Create/read/remove area metadata (Cerilica)

24 – Hex #: &18 · Action: Select sprite/Get address

25 – Hex #: &19 · Action: Delete sprite 2

26 – Hex #: &1A · Action: Rename sprite 2

27 – Hex #: &1B · Action: Copy sprite 2

- 28 – Hex #: &1C · Action: Put sprite to screen
- 29 – Hex #: &1D · Action: Create mask 2
- 30 – Hex #: &1E · Action: Remove mask 2
- 31 – Hex #: &1F · Action: Insert row 2 3
- 32 – Hex #: &20 · Action: Delete row 2 3
- 33 – Hex #: &21 · Action: Flip about x axis 2
- 34 – Hex #: &22 · Action: Put sprite at user coords
- 35 – Hex #: &23 · Action: Append sprite 1 2
- 36 – Hex #: &24 · Action: Set pointer shape 1
- 37 – Hex #: &25 · Action: Create/read/remove palette 1
- 38 – Hex #: &26 · Action: Create/remove alpha 1 2
- 39 – Hex #: &27 · Action: Create/read/remove sprite metadata (Cerilica)
- 40 – Hex #: &28 · Action: Read sprite info
- 41 – Hex #: &29 · Action: Read pixel colour
- 42 – Hex #: &2A · Action: Write pixel colour 2
- 43 – Hex #: &2B · Action: Read pixel mask
- 44 – Hex #: &2C · Action: Write pixel mask 2
- 45 – Hex #: &2D · Action: Insert column 2 3
- 46 – Hex #: &2E · Action: Delete column 2 3
- 47 – Hex #: &2F · Action: Flip about y axis 2
- 48 – Hex #: &30 · Action: Plot sprite mask
- 49 – Hex #: &31 · Action: Plot mask at user coords
- 50 – Hex #: &32 · Action: Plot mask scaled 1
- 51 – Hex #: &33 · Action: Paint char scaled 1
- 52 – Hex #: &34 · Action: Paint sprite scaled 1
- 53 – Hex #: &35 · Action: Put sprite grey scaled 1
- 54 – Hex #: &36 · Action: Remove left-hand wastage 2
- 55 – Hex #: &37 · Action: Plot mask transformed 1
- 56 – Hex #: &38 · Action: Plot sprite transformed 1
- 57 – Hex #: &39 · Action: Insert/delete rows 1 2
- 58 – Hex #: &3A · Action: Insert/delete columns 1 2
- 60 – Hex #: &3C · Action: Switch output to sprite 2
- 61 – Hex #: &3D · Action: Switch output to mask 2
- 62 – Hex #: &3E · Action: Read save area size
- 65 – Hex #: &41 · Action: Tile sprite scaled 1

1 Requires SpriteExtend module to be loaded.

2 Not available via Wimp_SpriteOp

3 Originally these calls were implemented directly in the kernel, but as of RISC OS 3.5 they require SpriteExtend to be loaded.

[RISC OS Open source page](#)

OS_SpriteOp 10

SWI &2E · OS_SpriteOp

This call will erase the contents of sprite area R1 and replace it with the contents of sprite file R2.

Before calling

R0 10 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite file name

On return

All registers preserved

Notes

When using your own sprite area, make sure you initialise it first using OS_SpriteOp 9.

See also OS_SpriteOp 11 if you want to merge a file into an area instead of completely overwriting the destination, and OS_SpriteOp 12 if you want to save an area out to disc.

[RISC OS Open source page](#)

OS_SpriteOp 11

SWI &2E · OS_SpriteOp

This call will load the sprite file identified by R2 and merge it into the sprite area identified by R1.

Before calling

R0 11 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite file name

On return

All registers preserved

Notes

Sprites present in R2 take priority over sprites present in R1; i.e. if both areas contain a sprite with the same name then the sprite in R1 will be discarded and replaced with the sprite from R2.

You can use OS_SpriteOp 10 if you want to completely replace the contents of a sprite area with the contents of a sprite file.

[RISC OS Open source page](#)

OS_SpriteOp 12

SWI &2E · OS_SpriteOp

This call will save the contents of sprite area R1 to disc, using the filename given in R2.

Before calling

R0 12 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite file name

On return

All registers preserved

Notes

You can use OS_SpriteOp 10 to load the resulting file back into memory.

[RISC OS Open source page](#)

OS_SpriteOp 13

SWI &2E · OS_SpriteOp

This call will return the name of a given sprite.

Before calling

R0 13 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Pointer to buffer to store name

R3 Length of buffer

R4 Sprite number

On return

R3 Length of returned string

All other registers preserved

In practice

The first sprite in the area has number 1, therefore valid values of R4 are $1 \leq R4 \leq \text{number of sprites}$.

The returned name will always be null terminated. Therefore to be able to read the longest sprite name (12 characters), a buffer length of 13 bytes or more is required.

On entry R3 should be the total length of the buffer, i.e. 13. On exit it will be the number of characters in the name, i.e. 1 to 12.

Notes

Once you have the sprite name, you can use OS_SpriteOp 24 to get its address. Using the sprite address to interact with sprites can be significantly faster than using the sprite name, especially with sprite areas which contain lots of sprites.

[RISC OS Open source page](#)

OS_SpriteOp 14

SWI &2E · OS_SpriteOp

This call creates a new sprite with the given name, by copying a rectangle of pixels from the screen.

Before calling

R0 14 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name

R3 1 to create palette, 0 to exclude palette

On return

All registers preserved

In practice

The behaviour of this call is identical to OS_SpriteOp 16, except that the sprite bounds are defined by the graphics cursor coordinates, i.e. as set by the two most recent OS_Plot 'move' calls.

Notes

See OS_SpriteOp 16 for more detailed information about this call.

[RISC OS Open source page](#)

OS_SpriteOp 15

SWI &2E · OS_SpriteOp

This call creates a new sprite with the given name.

Before calling

R0 15 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name

R3 1 to create palette, 0 to exclude palette

R4 Width in pixels

R5 Height in pixels

R6 Mode Specifier

On return

All registers preserved

Notes

If a sprite with the given name already exists, it will first be deleted before the new sprite is created.

All of the sprite pixels will be initialised to zero.

If R6 is a valid Sprite Mode Word (i.e. a mode number, or a RISC OS 3.5 or RISC OS 5 style sprite mode word) then it will be used as-is to initialise the mode word of the new sprite. Otherwise it must be a pointer to a Mode Selector Block. The block will be converted to a RISC OS 3.5 or RISC OS 5 sprite mode word, favouring the most backwards compatible format.

[RISC OS Open source page](#)

OS_SpriteOp 16

SWI &2E · OS_SpriteOp

This call creates a new sprite with the given name, by copying a rectangle of pixels from the screen.

Before calling

R0 16 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name

R3 1 to create palette, 0 to exclude palette

R4 X0 coordinate

R5 Y0 coordinate

R6 X1 coordinate

R7 Y1 coordinate

On return

All registers preserved

In practice

The Sprite Mode Word will be chosen such that the underlying pixel format of the sprite and the screen are identical. This allows for easy creation of sprites which can be blitted directly to screen memory via the fast OS_SpriteOp 34 call.

If a palette is requested then the palette will be initialised to a copy of the current screen palette. For 256 colour modes with a full 256 entry palette this will result in the sprite having the full 256 entries.

Notes

If a sprite with the given name already exists, it will first be deleted before the new sprite is created.

The coordinates must be specified in OS units. There is no defined order for the coordinates (i.e. X0 can either be the left or right edge of the sprite). Coordinates are inclusive.

After the sprite has been created, it will be set as the active sprite, as if OS_SpriteOp 24 had been called.

See also OS_SpriteOp 14, to create a sprite from the area defined by the graphics cursor position

[RISC OS Open source page](#)

OS_SpriteOp 17

SWI &2E · OS_SpriteOp

This call verifies a sprite area by checking that the values in the sprite area header are sensible, that all sprites are within the bounds of the area, and that all sprites have sensible properties.

Before calling

R0 17 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

On return

All registers preserved

In practice

If the area fails any of the checks, error &71E ("Unrecognised sprite data") will be returned.

Notes

Old format sprites (where the Sprite Mode Word is just a mode number) have fewer checks performed than new format sprites; this call cannot be used to check if the OS recognises the mode used by an old format sprite.

New format sprites which have an unknown or unsupported sprite type will result in the verification check failing. This allows this call to be used to check if certain sprite types or features (e.g. alpha masks) are supported by the OS.

[RISC OS Open source page](#)

OS_SpriteOp 2

SWI &2E · OS_SpriteOp

This call saves the contents of the current graphics window as a sprite file on disc.

Before calling

R0 2 (Reason code)

R2 File name

R3 0 => Exclude palette

1 => Include palette

On return

All registers preserved

Notes

The sprite will be named 'screendump'

An error will be returned if the current screen mode is not a graphics mode.

[RISC OS Open source page](#)

OS_SpriteOp 23

SWI &2E · OS_SpriteOp

This call allows MetaData to be added or removed from the sprite area's Extension Area, or for the size and location of MetaData to be read.

Before calling

R0 23 (Reason code) + Area value

R1 Sprite area (as defined by area value in R0)

R3 -1 Read MetaData

else Length of MetaData

R4 -1 Remove MetaData

0 zero-initialise

else uPtr to MetaData to write

R5 FourCC identifier

or 1-based index

On return

If R3 = -1 on entry:

R3 Size of MetaData (bytes), 0 if none

R4 aPtr to MetaData, or 0 if none

R5 FourCC identifier

All other registers preserved

Notes

This SpriteOp is provided by Cerilica's MetaSprite module.

[RISC OS Open source page](#)

OS_SpriteOp 24

SWI &2E · OS_SpriteOp

This call has two uses, depending on the area value:

Before calling

R0 24 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

On return

R2 Pointer to sprite (if not system sprite area), else preserved

All other registers preserved

In practice

- If the system sprite area is used, this call will select which sprite should be used by the OS_Plot sprite operations (&E8-&EF). R2 will not be updated.
- If a user sprite area is used, this call will return in R2 the address of the given sprite. This makes the call useful as a method of finding the address of a sprite when you only have its name.

[RISC OS Open source page](#)

OS_SpriteOp 25

SWI &2E · OS_SpriteOp

This call is used to delete a sprite.

Before calling

R0 25 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

On return

All registers preserved

Notes

An error will be returned if the indicated sprite does not exist.

[RISC OS Open source page](#)

OS_SpriteOp 26

SWI &2E · OS_SpriteOp

This call is used to rename a sprite.

Before calling

R0 26 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 New name

On return

All registers preserved

Notes

The sprite name given in R3 must be terminated by a space or control character (i.e. any character ≤ 32). The name will be automatically converted to lower case and truncated to 12 characters by the OS.

[RISC OS Open source page](#)

OS_SpriteOp 27

SWI &2E · OS_SpriteOp

This call is used to create a copy of a sprite within the same sprite area.

Before calling

R0 27 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 Name of copy

On return

All registers preserved

Notes

If a sprite with the target name already exists, an error will be returned.

[RISC OS Open source page](#)

OS_SpriteOp 28

SWI &2E · OS_SpriteOp

This call is identical to OS_SpriteOp 34 - Put sprite at user coords, with the exception that the current graphics coordinates are used instead of user-supplied coordinates.

Before calling

R0 28 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R5 Bits 0-2: GCOL action

Bit 3: Use sprite mask

On return

All registers preserved

Notes

Please see OS_SpriteOp 34 for information about the behaviour of this call.

[RISC OS Open source page](#)

OS_SpriteOp 29

SWI &2E · OS_SpriteOp

This call allows a mask to be added to a sprite. If the sprite mode word has the top bit set (the wide mask flag) then an 8bpp alpha mask will be created, otherwise a standard binary mask will be created.

Before calling

R0 29 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

On return

All other registers preserved

In practice

The new mask will be filled with a default value such that all pixels are 100% opaque (i.e. all mask pixels set to 255 for alpha masks, or 1 for binary masks)

Notes

If the sprite already has a mask, its contents will be erased, just as if a fresh mask had been created.

If you wish to convert between different mask formats, or to transfer data to/from an alpha channel, use OS_SpriteOp 38.

[RISC OS Open source page](#)

OS_SpriteOp 3

SWI &2E · OS_SpriteOp

This call loads the contents of a sprite file into the current graphics window.

Before calling

R0 3 (Reason code)

R2 File name

On return

All registers preserved

Notes

The first sprite within the sprite file will be used; the name is irrelevant.

If the sprite has a palette, it will replace the screen's current palette.

If the sprite was created in a different screen mode to the current screen mode, an attempt will be made to switch to the same screen mode the sprite was created in. This will reset the graphics window to default. If loading a new-style (i.e. RISC OS 3.5 or RISC OS 5) sprite then the sprite width and height will be used to select the mode width and height.

[RISC OS Open source page](#)

OS_SpriteOp 30

SWI &2E · OS_SpriteOp

This call allows a mask to be removed from a sprite.

Before calling

R0 30 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

On return

All registers preserved

Notes

When used on a sprite with an alpha mask, the 'wide mask' flag will remain set in the sprite mode word. Therefore any subsequent call to OS_SpriteOp 29 will create a new alpha mask. If this is not intended then you must manually remove the flag after removing the mask.

If the sprite has no mask, this call has no effect.

See also OS_SpriteOp 38, which can be used to convert between different mask formats, or to transfer data to/from an alpha channel.

[RISC OS Open source page](#)

OS_SpriteOp 31

SWI &2E · OS_SpriteOp

This call inserts a blank row into the given sprite.

Before calling

R0 31 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 Row number to insert below

On return

All other registers preserved

Notes

Sprite rows are numbered from the bottom of the sprite. Therefore specifying 0 in R3 will cause a new row to be added to the bottom of the sprite, and specifying the sprite height will cause the new row to be added to the top.

See also OS_SpriteOp 32 to delete a row, or OS_SpriteOp 57 to insert/delete multiple rows.

[RISC OS Open source page](#)

OS_SpriteOp 32

SWI &2E · OS_SpriteOp

This call deletes a row from the given sprite.

Before calling

R0 32 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 Row number to delete

On return

All other registers preserved

Notes

Sprite rows are numbered from the bottom of the sprite, starting from zero.

See also OS_SpriteOp 31 to insert a row, or OS_SpriteOp 57 to insert/delete multiple rows.

[RISC OS Open source page](#)

OS_SpriteOp 33

SWI &2E · OS_SpriteOp

This call flips the given sprite about its X axis, i.e. it performs a vertical flip.

Before calling

R0 33 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

On return

All other registers preserved

Notes

To flip about the Y axis (horizontal flip), use OS_SpriteOp 47

[RISC OS Open source page](#)

OS_SpriteOp 34

SWI &2E · OS_SpriteOp

This call is used to plot a sprite to the screen.

Before calling

R0 34 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 X coordinate

R4 Y coordinate

R5 Bits 0-2: GCOL action

Bit 3: Use sprite mask

On return

All registers preserved

In practice

This call does not attempt to translate between different palettes, pixel formats or colour depths. Unless you know the source and destination are in compatible modes, it's recommended to use OS_SpriteOp 52 - Paint Sprite Scaled instead. However due to the overheads involved in OS_SpriteOp 52, using OS_SpriteOp 34 where possible can result in significant performance gains.

Notes

Coordinates are in OS units, and specify the bottom-left corner of the sprite.

No scaling is performed; source pixels will map directly to destination pixels, regardless of the sprite DPI or screen eigen values.

This call does not support alpha blending. Specifically, if an attempt is made to perform a masked plot of a sprite with an alpha mask, an error will be returned.

For sprites with an alpha channel, the mask flag is ignored, and the contents of the alpha channel will be copied directly to the screen alpha channel (assuming the sprite and screen have matching pixel formats).

[RISC OS Open source page](#)

OS_SpriteOp 35

SWI &2E · OS_SpriteOp

This call is used to merge two sprites together, either horizontally or vertically. The second sprite will be merged into the first, either on the right hand side of the first sprite (for a horizontal merge), or on the bottom side (for a vertical merge). The second sprite will be deleted.

Before calling

R0 35 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0) for first sprite

R3 Sprite name/pointer (as defined by area value in R0) for second sprite

R4 Mode:

0 Merge horizontally

Otherwise Merge vertically

On return

All registers preserved

Notes

Both sprites must have identical mode words. If the sprites have palettes, no remapping of colours will be performed.

Usually this call will not attempt to grow the sprite area - the merge will be performed in-place. However if one sprite has a mask and the other doesn't, the sprite which is missing a mask will have a default mask created, which will by necessity cause the sprite area to grow.

As this call moves and deletes sprites, you must discard any cached sprite pointers following the operation.

This operation will use Scratch Space for temporary storage.

[RISC OS Open source page](#)

OS_SpriteOp 36

SWI &2E · OS_SpriteOp

This call is used to configure a pointer shape and the pointer palette based around the contents of an arbitrary sprite. The exact operation of the call can be controlled by bits 4-6 of R3; if these are all set the call is effectively a no-op.

Before calling

R0 36 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 Flags:

Bits 0-3: Pointer shape (1-4)

Bit 4 set: Don't update pointer image

Bit 5 set: Don't update pointer palette

Bit 6 set: Don't set the active pointer afterwards

R4 X coordinate of active point

R5 Y coordinate of active point

R6 Pointer to scaling factors, or 0 to use mode-dependent default

R7 Pointer to pixel translation table, or 0 for no translation

On return

All registers preserved

In practice

This call is a more flexible and user-friendly alternative to using OS_Word 21,0 to define pointer shapes.

Notes

This call does not support the use of true colour sprites; the sprite must be 256 colours or less.

The active point coordinates are in pixels, relative to the top-left corner of the sprite.

If the sprite has a mask and no left-hand wastage, the supplied active point coordinates will be ignored and the active point will instead be selected automatically based around the mask data. Starting from the topmost row and working down, each mask row will be examined in turn and the active point will be set to the first pixel (from the left of the image) which is set.

If bit 5 of R3 is clear then the sprites palette will be used to set the pointer palette. The translation table (if supplied) will be used for a reverse lookup into the sprite palette, in order to determine which colours should be used for pointer colours 1, 2 and 3. In the absence of a translation table, sprite palette entries 1-3 will be used directly.

If no scaling factors are supplied, the DPI and eigen values of the source and destination are used to calculate appropriate scale factors.

As the OS limits the maximum pointer image size to 32×32 pixels, any image larger than this will be cropped, with only the top-left portion visible.

The translation table supplied in R7 should be a standard pixel translation table suitable for converting the image to a 2bpp mode, with the exception that colour 0 should normally be avoided, as the OS always treats pointer colour 0 as transparent. Unless the sprite is 2bpp, a translation table must be supplied.

[RISC OS Open source page](#)

OS_SpriteOp 37

SWI &2E · OS_SpriteOp

This call allows a palette to be added or removed from a sprite, or for the size and location of a palette to be read.

Before calling

R0 37 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 Mode:

-1 Read palette size

0 Remove palette
Otherwise Create palette
If bit 31 set, create full 256 entry palette

On return

If R3 = -1 on entry:

R3 Size of palette (number of entries), 0 if none

R4 Pointer to palette, or 0 if none

R5 Sprite Mode Word

All other registers preserved

In practice

For 256 colour sprites, bit 31 of R3 controls whether a 64 or 256 entry palette is created. For other colour depths this bit is ignored.

Notes

If an attempt is made to add a palette to a true-colour sprite, an error will be returned.

[RISC OS Open source page](#)

OS_SpriteOp 38

SWI &2E · OS_SpriteOp

This call allows the mask or alpha data of a sprite to be converted between the four different types:

Before calling

R0 38 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 Flags:

Bits 0-7: Cutoff value when reducing to 1bpp

Bit 8: Remove mask completely if possible

Bits 30-31: Operation:

0 = Convert to 1bpp mask

1 = Reserved

2 = Convert to 8bpp alpha mask

3 = Convert to alpha channel

On return

R3 If the call failed due to lack of space in the sprite area, R3 is the amount of extra space required

In all other cases, R3 is zero

All other registers preserved

In practice

- No mask/alpha
- 1bpp mask

- 8bpp alpha mask
- Alpha channel

When a sprite is being converted to a 1bpp alpha mask/alpha channel, bits 0-7 of R3 are used as a cutoff value to decide whether mask pixels should be 0 or 1. If the source mask/alpha value (once expanded to 8 bits) is greater than or equal to the cutoff value, a value of 1 will be written, else a value of 0.

When the number of bits in the mask/alpha is being reduced (8bpp to 4bpp, 8bpp to 1bpp, or 4bpp to 1bpp), setting bits 0-7 of R3 to zero will prevent the conversion from being performed if it will result in the loss of data. For a 1bpp destination this means that the conversion will only occur if the source mask/alpha is a simple binary mask (i.e. 8bpp mask pixels are only 0 or 255). For a 4bpp destination this means that the conversion will only occur if the top and bottom nibbles of the 8bpp source data match (i.e. there are only 16 levels of transparency &00, &11, &22, &33, etc.)

If bit 8 of R3 is set then the mask/alpha channel will be removed if it is redundant; i.e. if all pixels have an alpha of 255. This redundancy checking is performed with respect to the destination alpha/mask format. This means that, for example, when converting an 8bpp mask to 1bpp, if all pixels are above the cutoff value and bit 8 is set then the mask will be discarded.

Note that for compatibility with RISC OS Select, bits 0-8 are ignored for operation 2. For bits 0-7 this doesn't matter (as the mask is never losing precision), but for bit 8 it means there's no way of removing a redundant alpha mask. See the examples below for a workaround 1

If the call fails due to lack of space in the sprite area, it will exit with V set, R0 pointing at an error block (error &85, "SpriteExtend: Not enough memory for sprite operation"), and with R3 containing the amount the sprite area must be grown by in order for the call to succeed. If the call completes successfully, or fails with any other error, R3 will be zero.

Examples

To convert a sprite to a 1bpp mask, but only if the existing mask/alpha data is binary, use a flags word of &100

To convert a sprite to a 1bpp mask, with a cutoff value of 50%, and to discard the mask where possible, use a flags word of &180

To convert a sprite to a RISC OS Select-compatible 8bpp alpha mask, or to create an empty alpha mask, use a flags word of &80000000

To convert a sprite to a RISC OS 5-compatible alpha channel, with a cutoff value of 50% when converting to 1bpp alpha from an 8bpp alpha mask, use a flags word of &c0000080

1 Discarding redundant alpha masks

If you use a flags word of &C0000100 then an attempt will be made to convert the mask to an alpha channel, without losing any precision, and discarding it if it is irrelevant. The determination of which alpha format to use is made after an initial check for redundancy, so this call will successfully remove redundant alpha masks even when used on sprites where an alpha channel is impossible (e.g. palettised sprites).

Once the conversion is complete, use OS_ReadModeVariable to read the Mode Flags of the Sprite Mode Word. If the flags indicate RGB colour space with the alpha flag set, call OS_SpriteOp 38 again, but with a flags word of &80000000. This will convert the alpha channel back to an alpha mask. For other mode flag combinations no action should be necessary, as it either means the mask was redundant (and removed) or non-redundant (but impossible to remove due to an alpha channel being impossible to add).

Notes

This call only supports sprites which have a RISC OS 3.5 or RISC OS 5 style sprite mode word. Attempting to use it on a sprite with an old-style mode word (i.e. where the mode word is just a mode number) will result in an error.

Where possible, the sprite mode word will be transformed into its most backwards-compatible form. This includes removing the alpha mask flag if the sprite doesn't have an alpha mask, and attempting to convert any RISC OS 5 sprite mode word to its equivalent RISC OS 3.5 form. This mode word conversion will take place even if the mask/alpha data is not being modified by a particular call.

Conversion to an alpha channel is only possible for sprite types 5 (32K colours), 6 (16M colours at 32bpp), and 16 (4K colours).

[RISC OS Open source page](#)

OS_SpriteOp 39

SWI &2E · OS_SpriteOp

This call allows MetaData to be added or removed from a sprite, or for the size and location of MetaData to be read.

Before calling

R0 39 (Reason code) + Area value

R1 Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 -1 Read MetaData

else Length of MetaData

R4 -1 Remove MetaData

0 zero-initialise

else uPtr to MetaData to write

R5 FourCC identifier

or 1-based index

On return

If R3 = -1 on entry:

R3 Size of MetaData (bytes), 0 if none

R4 aPtr to MetaData, or 0 if none

R5 FourCC identifier

All other registers preserved

Notes

This SpriteOp is provided by Cerilica's MetaSprite module.

[RISC OS Open source page](#)

OS_SpriteOp 40

SWI &2E · OS_SpriteOp

This call is used to read the details of a sprite.

Before calling

R0 40 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

On return

R3 Sprite width in pixels

R4 Sprite height in pixels

R5 0/1 for whether a mask is present

R6 Sprite Mode Word

All other registers preserved

Notes

This call will return an error if the kernel does not understand the format of the sprite mode word, or if the sprite uses an unrecognised numbered mode.

[RISC OS Open source page](#)

OS_SpriteOp 41

SWI &2E · OS_SpriteOp

This call allows the colour of a sprite pixel to be read.

Before calling

R0 41 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 X coordinate

R4 Y coordinate

On return

R5 Pixel colour

R6 Pixel tint

All other registers preserved

Notes

Coordinates are specified in pixels, with (0,0) being the bottom-left corner of the sprite.

For 256 colour sprites which lack a full palette, R5 will contain the pixel colour/palette index (0-63) and R6 the tint value (0, 64, 128 or 192).

For all other sprites, R5 will be the pixel colour/palette index and R6 will be zero.

[RISC OS Open source page](#)

OS_SpriteOp 42

SWI &2E · OS_SpriteOp

This call allows the colour of a sprite pixel to be written.

Before calling

R0 42 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 X coordinate

R4 Y coordinate

R5 Pixel colour

R6 Pixel tint

On return

All registers preserved

Notes

Coordinates are specified in pixels, with (0,0) being the bottom-left corner of the sprite.

For 256 colour sprites which lack a full palette, R5 should contain the pixel colour/palette index (0-63) and R6 the tint value (0, 64, 128 or 192).

For all other sprites, R5 should be the pixel colour/palette index and R6 will be ignored.

[RISC OS Open source page](#)

OS_SpriteOp 43

SWI &2E · OS_SpriteOp

This call allows the value of a sprite mask pixel to be read.

Before calling

R0 43 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 X coordinate

R4 Y coordinate

On return

R5 Mask value

All other registers preserved

Notes

Coordinates are specified in pixels, with (0,0) being the bottom-left corner of the sprite.

For sprites with the wide mask flag set in the Sprite Mode Word, the returned value is an alpha value between 0 and 255, with 255 representing a fully opaque pixel and 0 representing a fully transparent pixel.

For other sprites, the returned value is either 0 or 1, representing transparent and opaque respectively.

If the sprite lacks a mask, the value returned is the appropriate value for if the pixel was opaque (i.e. 255 or 1).

[RISC OS Open source page](#)

OS_SpriteOp 44

SWI &2E · OS_SpriteOp

This call allows the value of a sprite mask pixel to be written.

Before calling

R0 44 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 X coordinate

R4 Y coordinate

R5 Mask value

On return

All registers preserved

Notes

Coordinates are specified in pixels, with (0,0) being the bottom-left corner of the sprite.

For sprites with the wide mask flag set in the Sprite Mode Word, the value in R5 should be an alpha value between 0 and 255, with 255 representing a fully opaque pixel and 0 representing a fully transparent pixel.

For other sprites, if R5 is zero the mask pixel will be set to zero (i.e. transparent), and if R5 is non-zero the mask pixel will be set to one (i.e. opaque).

An error will be returned if an attempt is made to use this operation on a sprite without a mask.

[RISC OS Open source page](#)

OS_SpriteOp 45

SWI &2E · OS_SpriteOp

This call inserts a blank column into the given sprite.

Before calling

R0 45 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 Column number to insert to the left of

On return

All other registers preserved

Notes

Sprite columns are numbered from the left of the sprite. Therefore specifying 0 in R3 will cause a new column to be added to the left of the sprite, and specifying the sprite width will cause the new column to be added to the right.

See also OS_SpriteOp 46 to delete a column, or OS_SpriteOp 58 to insert/delete multiple columns.

[RISC OS Open source page](#)

OS_SpriteOp 46

SWI &2E · OS_SpriteOp

This call deletes a column from the given sprite.

Before calling

R0 46 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 Column number to delete

On return

All other registers preserved

Notes

Sprite columns are numbered from the left of the sprite, starting from zero.

See also OS_SpriteOp 44 to insert a column, or OS_SpriteOp 58 to insert/delete multiple columns.

[RISC OS Open source page](#)

OS_SpriteOp 47

SWI &2E · OS_SpriteOp

This call flips the given sprite about its Y axis, i.e. it performs a horizontal flip.

Before calling

R0 47 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

On return

All other registers preserved

Notes

To flip about the X axis (vertical flip), use OS_SpriteOp 33

[RISC OS Open source page](#)

OS_SpriteOp 48

SWI &2E · OS_SpriteOp

This call is identical to OS_SpriteOp 49 - Plot mask at user coords, with the exception that the current graphics coordinates are used instead of user-supplied coordinates.

Before calling

R0 48 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

On return

All registers preserved

Notes

Please see OS_SpriteOp 49 for information about the behaviour of this call.

[RISC OS Open source page](#)

OS_SpriteOp 49

SWI &2E · OS_SpriteOp

This call paints a sprite mask to the screen. For sprite pixels where the mask is non-zero, the corresponding screen pixel will be set to the currently selected background colour. If the mask is zero the screen pixel will not be updated.

Before calling

R0 49 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 X coordinate

R4 Y coordinate

On return

All registers preserved

In practice

This call does not attempt to translate between different colour depths. Unless you know the source and destination are in compatible modes, it's recommended to use OS_SpriteOp 50 - Plot Mask Scaled instead. However due to the overheads involved in OS_SpriteOp 50, using OS_SpriteOp 49 where possible can result in significant performance gains.

Notes

Coordinates are in OS units, and specify the bottom-left corner of the sprite.

No scaling is performed; source pixels will map directly to destination pixels, regardless of the sprite DPI or screen eigen values.

When used on a sprite without a mask, a filled rectangle the size of the scaled sprite will be drawn.

This call does not support plotting of alpha masks.

[RISC OS Open source page](#)

OS_SpriteOp 50

SWI &2E · OS_SpriteOp

This call paints a sprite mask to the screen. For sprite pixels where the mask is non-zero, the corresponding screen pixel will be set to the currently selected background colour. If the mask is zero the screen pixel will not be updated.

Before calling

R0 50 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 X coordinate

R4 Y coordinate

R6 Pointer to scaling factors, or 0 for 1:1 scaling

On return

All registers preserved

Notes

Coordinates are in OS units, and specify the bottom-left corner of the sprite.

1:1 scaling maps directly from source pixels to destination pixels; the DPI and eigen values of the source and destination are ignored.

When used on a sprite without a mask, a filled rectangle the size of the scaled sprite will be drawn.

OS_SpriteOp 49 - Plot mask at user coords is a less sophisticated sprite plotting interface which can be used in certain situations. If you have large numbers of sprites to render and are concerned about performance, consider using that call where possible. Per-pixel performance will be identical to OS_SpriteOp 50, but there will be lower overheads per sprite rendered.

[RISC OS Open source page](#)

OS_SpriteOp 51

SWI &2E · OS_SpriteOp

This call plots the specified character at the x and y co-ordinates using the desired scaling.

Before calling

R0 51 (Reason code) + Area value

R1 Character Code

R2 Unused

R3 X coordinate

R4 Y coordinate

R5 Unused

R6 Pointer to scaling factors, or 0 for 1:1 scaling

On return

All registers preserved

Notes

Coordinates are in OS units, and specify the bottom-left corner of the sprite.

The foreground colour is used to paint the desired colour.

[RISC OS Open source page](#)

OS_SpriteOp 52

SWI &2E · OS_SpriteOp

This call paints a scaled sprite to the screen. When supplied with the right translation table, it is able to paint a sprite created in any source mode to any destination mode. If a translation table is not supplied when one is required, an error will be generated. Therefore when plotting arbitrary sprites it is recommended to always call `ColourTrans_SelectTable` or `ColourTrans_GenerateTable` before the plot in order to generate a suitable table.

Before calling

R0 52 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 X coordinate

R4 Y coordinate

R5 Flags

R6 Pointer to scaling factors, or 0 for 1:1 scaling

R7 Pointer to pixel translation table / Colour Mapping Descriptor, or 0 for no translation

On return

All registers preserved

Notes

Coordinates are in OS units, and specify the bottom-left corner of the sprite.

1:1 scaling maps directly from source pixels to destination pixels; the DPI and eigen values of the source and destination are ignored.

See `OS_SpriteOp Scaled/Transformed Plot Flags` for details on how the flags affect the plot operation and dictate some of the input parameters.

`OS_SpriteOp 34 - Put sprite at user coords` is a less sophisticated sprite plotting interface which can be used in certain situations. If you have large numbers of sprites to render and are concerned about performance, consider using that call where possible. Per-pixel performance will be identical to `OS_SpriteOp 52`, but there will be lower overheads per sprite rendered.

[RISC OS Open source page](#)

OS_SpriteOp 53

SWI &2E · OS_SpriteOp

This call anti-aliases a sprite as it scales it.

Before calling

R0 53 (Reason code) + Area value

R1 Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 X coordinate

R4 Y coordinate

R5 0

R6 Pointer to scaling factors, or 0 for 1:1 scaling

R7 Pointer to pixel translation table / Colour Mapping Descriptor, or 0 for no translation

On return

All registers preserved

Notes

This reason code is not available on SpriteExtend version 0.99 and later.

The sprite must have been defined in a 16-colour mode whose pixels reflect a linear grey scale. This is considerably slower than OS_SpriteOp 52

Coordinates are in OS units, and specify the bottom-left corner of the sprite.

1:1 scaling maps directly from source pixels to destination pixels; the DPI and eigen values of the source and destination are ignored.

[RISC OS Open source page](#)

OS_SpriteOp 54

SWI &2E · OS_SpriteOp

This call removes any columns from the left hand edge of a sprite which exist in the sprite data but have been hidden by a non-zero value in word 6 of the sprite header.

Before calling

R0 54 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

On return

All registers preserved

In practice

This can help to reduce memory usage, and is required if screen output is to be redirected to the sprite. However the kernel will automatically perform this operation itself when required, so apart from saving some memory there is little reason to use this call yourself.

Notes

This call is effectively a no-op when used on new format sprites (i.e. RISC OS 3.5 or RISC OS 5 style mode word), as new format sprites are not allowed any left-hand wastage.

[RISC OS Open source page](#)

OS_SpriteOp 55

SWI &2E · OS_SpriteOp

This call paints a transformed sprite mask to the screen. For sprite pixels where the mask is non-zero, the corresponding screen pixel will be set to the currently selected background colour. If the mask is zero the screen pixel will not be updated.

Before calling

R0 55 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 Flags:

Bit 0: 1 R6 is destination coordinate block. 0 R6 is a matrix.

Bit 1: 1 R4 is source coordinate block in sprite

R4 Source coordinate block in sprite

R6 Coordinate block/matrix

On return

All registers preserved

In practice

If R3 bit 0 is clear, R6 is treated as a pointer to a Draw module Transformation Matrix describing the transform to apply to the mask. If bit 0 is set then R6 is treated as a pointer to a block giving the screen coordinates of a parallelogram to transform the mask into:

0 – Value: X0 coordinate, in 1/256th OS units

1 – Value: Y0 coordinate, in 1/256th OS units

2 – Value: X1 coordinate, in 1/256th OS units

3 – Value: Y1 coordinate, in 1/256th OS units

4 – Value: X2 coordinate, in 1/256th OS units

5 – Value: Y2 coordinate, in 1/256th OS units

6 – Value: X3 coordinate, in 1/256th OS units

7 – Value: Y3 coordinate, in 1/256th OS units

If R3 bit 1 is set then R4 is treated as a pointer to a coordinate block specifying a subrectangle of the mask to render from, with coordinates given in pixels:

Word / Value

0 – Value: x0 coordinate

1 – Value: y0 coordinate

2 – Value: x1 coordinate

3 – Value: y1 coordinate

When specifying a destination parallelogram, the source rectangle is mapped onto the destination as follows:

x0,y0 -> X0,Y0

x1,y0 -> X1,Y1

x1,y1 -> X2,Y2

x0,y1 -> X3,Y3

Notes

See OS_SpriteOp Scaled/Transformed Plot Flags for details on how the flags affect the plot operation and dictate some of the input parameters.

When used on a sprite without a mask, all pixels within the destination parallelogram will be filled with the background colour.

[RISC OS Open source page](#)

OS_SpriteOp 56

SWI &2E · OS_SpriteOp

This call paints a transformed sprite to the screen. When supplied with the right translation table, it is able to paint a sprite created in any source mode to any destination mode. If a translation table is not supplied when one is required, an error will be generated. Therefore when plotting arbitrary sprites it is recommended to always call ColourTrans_SelectTable or ColourTrans_GenerateTable before the plot in order to generate a suitable table.

Before calling

R0 56 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 Flags:

Bit 0: 1 R6 is destination coordinate block. 0 R6 is a matrix.

Bit 1: 1 R4 is source coordinate block in sprite

R4 Source coordinate block in sprite

R5 Flags

R6 Coordinate block/matrix

R7 Pointer to pixel translation table / Colour Mapping Descriptor, or 0 for no translation

On return

All registers preserved

In practice

If R3 bit 0 is clear, R6 is treated as a pointer to a Draw module Transformation Matrix describing the transform to apply to the sprite. If bit 0 is set then R6 is treated as a pointer to a block giving the screen coordinates of a parallelogram to transform the sprite into:

0 – Value: X0 coordinate, in 1/256th OS units

1 – Value: Y0 coordinate, in 1/256th OS units

2 – Value: X1 coordinate, in 1/256th OS units

3 – Value: Y1 coordinate, in 1/256th OS units

4 – Value: X2 coordinate, in 1/256th OS units

5 – Value: Y2 coordinate, in 1/256th OS units

6 – Value: X3 coordinate, in 1/256th OS units

7 – Value: Y3 coordinate, in 1/256th OS units

If R3 bit 1 is set then R4 is treated as a pointer to a coordinate block specifying a subrectangle of the sprite to render from, with coordinates given in pixels:

Word / Value

0 – Value: x0 coordinate

1 – Value: y0 coordinate

2 – Value: x1 coordinate

3 – Value: y1 coordinate

When specifying a destination parallelogram, the source rectangle is mapped onto the destination as follows:

x0,y0 -> X0,Y0

x1,y0 -> X1,Y1

x1,y1 -> X2,Y2

x0,y1 -> X3,Y3

Notes

See OS_SpriteOp Scaled/Transformed Plot Flags for details on how the flags affect the plot operation and dictate some of the input parameters.

[RISC OS Open source page](#)

OS_SpriteOp 57

SWI &2E · OS_SpriteOp

This call inserts or deletes rows from the given sprite.

Before calling

R0 57 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 Base row number

R4 Row count

On return

All registers preserved

In practice

To insert rows, set R3 to the row number to insert the new row(s) underneath, and set R4 to the row count.

To delete rows, set R3 to the index of the bottommost row to delete, and set R4 to -(row count).

Notes

Sprite rows are numbered from the bottom of the sprite, starting at zero.

See also OS_SpriteOp 31 to insert a single row, or OS_SpriteOp 32 to delete a single row.

[RISC OS Open source page](#)

OS_SpriteOp 58

SWI &2E · OS_SpriteOp

This call inserts or deletes columns from the given sprite.

Before calling

R0 58 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 Base column number

R4 Column count

On return

All registers preserved

In practice

To insert columns, set R3 to the column number to insert the new column(s) to the left of, and set R4 to the column count.

To delete columns, set R3 to the index of the leftmost column to delete, and set R4 to -(column count).

Notes

Sprite columns are numbered from the left of the sprite, starting at zero.

See also OS_SpriteOp 45 to insert a single column, or OS_SpriteOp 46 to delete a single column.

[RISC OS Open source page](#)

OS_SpriteOp 60

SWI &2E · OS_SpriteOp

This call can be used to redirect screen output to a sprite image, or to return output back to the screen.

Before calling

R0 60 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

Or 0 to switch output to screen

R3 VDU Save Area:

0: Use no save area (discouraged)

1: Use kernel save area

Otherwise: Pointer to user save area

On return

R0-R3 Required values to restore previous context

All other registers preserved

In practice

While output is redirected to a sprite, all VDU operations including text output, OS_Plot operations, and sprite rendering will act on the sprite and not the screen. The VDU Variables and Mode Variables will also reflect the change.

If the sprite has a palette, ColourTrans will use that palette for any colour translation or palette lookup operations. If the sprite has no palette, the current screen palette will be used instead. This may result in unpredictable behaviour if the current screen mode is a true colour mode.

If no save area is supplied in R3 or the save area is not yet initialised, the VDU state will be reset to the default for the sprite/screen mode. See VDU Save Area for more details on the state that save areas store.

Notes

To redirect screen output to a sprite mask, see OS_SpriteOp 61.

You must be careful to restore output back to the screen should an error occur. It's recommended to install environment handlers so that unexpected errors (e.g. data aborts) can be fully accounted for.

To restore output back to its previous location, issue OS_SpriteOp with R0-R3 containing the values returned from this call.

The low-level PaletteV API only deals with the screen palette, and so does not adapt itself in any way when screen output is redirected to a sprite. Therefore any PaletteV calls made while output is redirected will manipulate the screen palette and not the sprite palette. Additionally, ColourTrans_WritePalette will return an error if called when output is redirected.

Service_SwitchingOutputToSprite will be issued by this call.

If a call is made to change the screen mode or switch video drivers while screen output is redirected, the output will be restored to the screen.

Redirecting output to a sprite without using a save area is strongly discouraged. The reason being that it can cause issues if you either directly or indirectly call a SWI which wants to use redirection internally (e.g. the font manager makes use of redirection to build the font cache). Save areas are small (currently under 512 bytes), so there is little reason to not use them.

[RISC OS Open source page](#)

OS_SpriteOp 61

SWI &2E · OS_SpriteOp

This call can be used to redirect screen output to a sprite mask, or to return output back to the screen.

Before calling

R0 61 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

Or 0 to switch output to screen

R3 VDU Save Area:

0: Use no save area (discouraged)

1: Use kernel save area

Otherwise: Pointer to user save area

On return

R0-R3 Required values to restore previous context

All other registers preserved

In practice

While output is redirected to a sprite, all VDU operations including text output, OS_Plot operations, and sprite rendering will act on the sprite and not the screen. The VDU Variables and Mode Variables will also reflect the change.

When output is redirected to an alpha mask, ColourTrans will report a default greyscale palette (black to white gradient). For other types of mask the same logic as OS_SpriteOp 60 is used. In this situation using colour translation functions to determine the pixel values/colours to write to the mask may result in invalid values being stored.

If no save area is supplied in R3 or the save area is not yet initialised, the VDU state will be reset to the default for the sprite/screen mode. See VDU Save Area for more details on the state that save areas store.

Notes

To redirect screen output to a sprite image, see OS_SpriteOp 60.

You must be careful to restore output back to the screen should an error occur. It's recommended to install environment handlers so that unexpected errors (e.g. data aborts) can be fully accounted for.

To restore output back to its previous location, issue OS_SpriteOp with R0-R3 containing the values returned from this call.

The low-level PaletteV API only deals with the screen palette, and so does not adapt itself in any way when screen output is redirected to a sprite. Therefore any PaletteV calls made while output is redirected will manipulate the screen palette and not the sprite palette. Additionally, ColourTrans_WritePalette will return an error if called when output is redirected.

Service_SwitchingOutputToSprite will be issued by this call.

If a call is made to change the screen mode or switch video drivers while screen output is redirected, the output will be restored to the screen.

Redirecting output to a sprite without using a save area is strongly discouraged. The reason being that it can cause issues if you either directly or indirectly call a SWI which wants to use redirection internally (e.g. the font manager makes use of redirection to build the font cache). Save areas are small (currently under 512 bytes), so there is little reason to not use them.

[RISC OS Open source page](#)

OS_SpriteOp 62

SWI &2E · OS_SpriteOp

This call is used to measure the required size of the VDU Save Area that is to be created for use with OS_SpriteOp 60 or OS_SpriteOp 61.

Before calling

R0 62 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

Or 0 for the screen

On return

R3 Size of save area

All other registers preserved

In practice

Redirecting output to a sprite without using a save area is strongly discouraged. The reason being that it can cause issues if you either directly or indirectly call a SWI which wants to use redirection internally (e.g. the font manager makes use of redirection to build the font cache). Save areas are small (currently under 512 bytes), so there is little reason to not use them.

Notes

After allocating the memory for the save area, it must be initialised by storing the value 0 in the first word.

At present the save area size and format is fixed for any given OS version, however software should not rely on this and should instead use this call to measure the required size for each sprite they wish to use a save area with.

[RISC OS Open source page](#)

OS_SpriteOp 65

SWI &2E · OS_SpriteOp

This call paints multiple copies of a scaled sprite to the screen, tiling them in order to completely fill the current graphics window.

Before calling

R0 65 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

R2 Sprite name/pointer (as defined by area value in R0)

R3 X coordinate

R4 Y coordinate

R5 Flags

R6 Pointer to scaling factors, or 0 for 1:1 scaling

R7 Pointer to pixel translation table / Colour Mapping Descriptor, or 0 for no translation

On return

All registers preserved

In practice

Compared to calling OS_SpriteOp 52 or OS_SpriteOp 34 in a loop, this call has greater scope for hardware acceleration to be used, and is therefore the preferred method for drawing tiled sprites.

Notes

Coordinates are in OS units, and specify the bottom-left corner of one of the tiles. Whether the coordinates specify a point inside or outside the graphics window is irrelevant, as the tiled area is effectively infinite, with the graphics window cropping it to a finite area.

This call is a tiled version of OS_SpriteOp 52, and therefore supports all the features of that call.

[RISC OS Open source page](#)

OS_SpriteOp 8

SWI &2E · OS_SpriteOp

This call reads the contents of the sprite area header

Before calling

R0 8 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

On return

R2 Size of sprite area

R3 Number of sprites in area

R4 Offset to first sprite

R5 Offset to first free word in area

All other registers preserved

[RISC OS Open source page](#)

OS_SpriteOp 9

SWI &2E · OS_SpriteOp

This call erases all sprites from the indicated sprite area.

Before calling

R0 9 (Reason code) + Area value

R1 Unused/Sprite area (as defined by area value in R0)

On return

All registers preserved

Notes

When used on the system sprite area, this also causes the OS to forget the currently selected sprite.

[RISC OS Open source page](#)

OS_SubstituteArgs

SWI &43

This SWI is deprecated, because the use of flag bits in address values makes it unsafe to use on systems with 32bit logical memory maps. Please use OS_SubstituteArgs32 instead.

Before calling

R0 Pointer to argument list, and flag in top bit

R1 Pointer to buffer for resulting string

R2 Length of buffer

R3 Pointer to template string

R4 Length of template string

On return

R0 Preserved

R1 Preserved

R2 Length of result string (including terminator)

R3 Preserved

R4 Preserved

In practice

The purpose of this call is to substitute command line arguments

Notes

This call performs the hard work involved in substituting a list of arguments into a 'template' string. Its main use is in the processing of command Alias\$... variables by the system. It's also useful in other situations; it has been made available to users. For example, FileSwitch uses it in the processing of Alias\$LoadType_TTT variables.

The argument list is a string consisting of space separated items which will be substituted into the template string. Spaces within double quotation marks are not counted as argument separators. Typically, the argument string will just be the tail of a *Command. Its control character terminated. The result of substituting the arguments into the template string is placed in the buffer. The length of the buffer is given so that the call can check for buffer overflow. The template string is copied into the result buffer char for char. However, when a '%' appears in the template string (even within quotation marks), it marks where an argument should be placed into the output buffer. The '%' is followed by a single digit from 0 - 9. 0 stands for the first argument in the argument list and so on. % means a single . Anything else following the " is not treated specially, i.e. both the % and the character are copied over. The template string doesn't have a terminator; instead its length is given. At the end of substitution, any arguments after the highest one mentioned in the template string are appended to the result string. To prevent this set the top bit of R0 on entry. If a non-existing argument is specified in the template string, then a null string is substituted, no error is given.

[RISC OS Open source page](#)

OS_SubstituteArgs32

SWI &7E

The purpose of this call is to substitute command line arguments

Before calling

R0 Pointer to argument list

R1 Pointer to buffer for resulting string

R2 Length of buffer

R3 Pointer to template string

R4 Length of template string

R5 Flag bit in bit 31, (bits 0-30 reserved)

On return

R0 Preserved

R1 Preserved

R2 Length of result string (including terminator)

R3 Preserved

R4 Preserved

R5 Preserved

Notes

This call performs the hard work involved in substituting a list of arguments into a 'template' string. Its main use is in the processing of command Alias\$... variables by the system. It's also useful in other situations; it has been made available to users. For example, FileSwitch uses it in the processing of Alias\$LoadType_TTT variables.

The argument list is a string consisting of space separated items which will be substituted into the template string. Spaces within double quotation marks are not counted as argument separators. Typically, the argument string will just be the tail of a *Command. Its control character terminated. The result of substituting the arguments into the template string is placed in the buffer. The length of the buffer is given so that the call can check for buffer overflow. The template string is copied into the result buffer char for char. However, when a '%' appears in the template string (even within quotation marks), it marks

where an argument should be placed into the output buffer. The '%' is followed by a single digit from 0 – 9. 0 stands for the first argument in the argument list and so on. % means a single . Anything else following the " is not treated specially, i.e. both the % and the character are copied over. The template string doesn't have a terminator; instead its length is given. At the end of substitution, any arguments after the highest one mentioned in the template string are appended to the result string. To prevent this set the top bit of R5 on entry. If a non-existing argument is specified in the template string, then a null string is substituted, no error is given.

The CallASWI module (version 0.03) allows this SWI to be used with RISC OS pre 5.00.

[RISC OS Open source page](#)

OS_SWINumberFromString

SWI &39

The purpose of this call is to convert a SWI name string to a SWI number.

Before calling

R1 Pointer to name (control terminated)

On return

R0 SWI number

R1 Preserved

Notes

An error is given if the SWI name is not recognized.

The conversion is as follows:

- A leading X is stripped. If present, it will cause &20000 to be added to the number returned (bit 17)
- System names are checked for. Note that the conversion of SWIs isn't very bidirectional: the name OS_Writel+ " " can be produced, but only OS_Writel is recognised
- Modules are scanned. If the module prefix matches the one given, and the suffix to the name is a number, then that number is added to the module's SWI 'chunk' base, and the sum returned. For example, Wimp_&23 returns &400E3, as the Wimp's chunk number is &400C0
- If the suffix is a name, and this can be matched by the module, the appropriate number is returned. For example, Wimp_Poll returns &400C7

[RISC OS Open source page](#)

OS_SWINumberToString

SWI &38

The purpose of this call is to convert a SWI number to a string containing its SWI name.

Before calling

R0 SWI number

R1 Pointer to buffer

R2 Buffer length

On return

R0 Preserved

R1 Preserved

R2 Length of string in buffer including terminator

Notes

The returned string is null-terminated, and starts with an X if the SWI number has bit 17 set.

SWI numbers < &200 have an 'OS_' prefix to the main part, and a SWI-dependent end section (which is 'Undefined' for unknown OS SWIs).

SWI numbers in the range &100 to &1FF are converted in the form OS_Writel+"A", or OS_Writel+23 if the character is not a printable one.

SWI numbers >= &200 are looked for in modules. If a suitable name is found, it is given in the form module_name or module_number, eg. Wimp_Initialise, Wimp_32. If no name is found in the modules, the string 'User' is returned.

The PRM incorrectly states that this SWI returns the length of the string; it actually returns the length of the string and the terminator.

[RISC OS Open source page](#)

OS_SynchroniseCodeAreas

SWI &6E

On a system with split instruction + data caches, if the CPU was to write new instructions to memory then there is no guarantee that the instruction fetch unit will be able to see those instructions. The new instructions may be sat in the data cache, waiting for writeback to main memory, or the instruction cache may already have the relevant area cached - in which case it will not request the new instructions from main memory. This SWI is used to ensure that the relevant area of the data cache has been flushed, and that the relevant area of the instruction cache has been invalidated, ensuring that any instructions which have been written to memory prior to this call can be executed by the CPU.

Before calling

R0 Flags

R1 Low address, word aligned (if bit 0 of R0 set)

R2 High address, word aligned inclusive (if bit 0 of R0 set)

On return

- All registers preserved

Notes

This SWI is typically used with programs that contain self-modifying code, or after loading a program from disc (e.g. OS_File 16 will synchronise the caches if bit 31 of R3 is set).

Synchronising the caches can be a slow operation.

- Avoid synchronising areas which you know haven't changed or don't contain code
- Avoid synchronising areas more often than required
- Avoid performing global synchronisation (bit 0 of R0 clear)

Note that this SWI only guarantees that the data cache is synchronised with the instruction cache, to the point where the CPU is able to execute code that has been written by the CPU. It does not guarantee that cached data has been written out to

main memory, nor that data written by an external bus master (e.g. DMA) will be visible to the CPU.

R1 & R2 on entry are only used if bit 0 of R0 is set.

This SWI was introduced in RISC OS 3.7, but is made available for earlier OS versions by the CallASWI module. OS_PlatformFeatures 0 can also be used to determine whether use of OS_SynchroniseCodeAreas is required.

[RISC OS Open source page](#)

OS_UnusedSWI

SWI &19

The purpose of this call is to set up the handler for unused SWIs.

Before calling

R0 Address of the Unused SWI handler?, or 0 to read

On return

R0 Address of previous Unused SWI handler?

Notes

This call is deprecated. Please use OS_ChangeEnvironment instead.

[RISC OS Open source page](#)

OS_UpCall

SWI &33 · Controlling sources: Hdr:UpCall managed by the allocations manager

The purpose of this call is to call the vector UpCallV, to warn of particular conditions.

Before calling

R0 Reason code

All other registers dependent on reason code

On return

R0 Preserved

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

This SWI calls the vector UpCallV. To use UpCalls, you must either claim the vector and install a routine on it, or install an UpCall handler.

They are called UpCalls because they are calls that RISC OS makes up to an application, rather than calls that an application makes down to RISC OS. They generally occur in the foreground, and are hence different to Events, which occur in the background.

Routines that handle UpCalls must only call the error-returning SWI calls. That is SWI calls that have their X bit set. i.e. either add the value &20000 to the SWI number, or add the letter 'X' in front of the SWI name.

If a routine installed on the vector deals with the situation, it should intercept the call to the vector, as there is no longer any point informing any other routines or the UpCall handler of the situation. If it can't deal with the situation, it must pass the call on, as another one may be able to do so.

Reason Codes

- 1 – Hex #: &01 · Action: Media not present
- 2 – Hex #: &02 · Action: Media not known
- 3 – Hex #: &03 · Action: Miscellaneous file changed information
- 4 – Hex #: &04 · Action: Media search end
- 5 – Hex #: &05 · Action: Password?
- 6 – Hex #: &06 · Action: Taskwindow sleep
- 7 – Hex #: &07 · Action: PipeFS sleep no more
- 8 – Hex #: &08 · Action: Buffer filling
- 9 – Hex #: &09 · Action: Buffer emptying
- 10 – Hex #: &0A · Action: DeviceFS stream created
- 11 – Hex #: &0B · Action: DeviceFS stream closed
- 12 – Hex #: &0C · Action: Freeway
- 13 – Hex #: &0D · Action: Media Set-Top Box online?
- 14 – Hex #: &0E · Action: Media Set-Top Box standby?
- 15 – Hex #: &0F · Action: Device receive data present
- 16 – Hex #: &10 · Action: Device threshold above
- 17 – Hex #: &11 · Action: Device threshold below
- 18 – Hex #: &12 · Action: Progress UI?
- 19 – Hex #: &13 · Action: Media search end with message
- 20 – Hex #: &14 · Action: Keyboard status change?
- 21 – Hex #: &15 · Action: MPEG?
- 22 – Hex #: &16 · Action: PMonitor?
- 23 – Hex #: &17 · Action: PDumper action?
- 24 – Hex #: &18 · Action: Drive added
- 25 – Hex #: &19 · Action: Drive removed
- 26 – Hex #: &1A · Action: Certificate confirm?
- 27 – Hex #: &1B · Action: Certificate confirm end?
- 256 – Hex #: &100 · Action: New application
- 257 – Hex #: &101 · Action: Move memory request

[RISC OS Open source page](#)

OS_UpCall 1

SWI &33 · OS_UpCall

The purpose of this call is to warn applications that a filing medium is not present.

Before calling

R0 1

R1 Filing system number

R2 Pointer to media name

R3 Device number

R4 Iteration count

R5 Timeout delay

R6 Pointer to media type

On return

R0 0 if medium change, or -1 if medium no longer required, else preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

R5 Preserved

R6 Preserved

Notes

This Upcall is called when a media change (e.g. another disc) is required.

The following information applies to both OS_UpCall 1 & OS_UpCall 2.

Any routines on this vector should:

- Prompt you to supply the medium with a string built up using:
- The medium type string (passed in R6) #the filing system name (obtained by calling XOS_FSCtrl 33 acting on the value of R1)
- The medium name (passed in R2), for example: 'Please insert disc ADFS:XYZ and press Space (Escape to abort)'
- Give you a way of indicating that you have either supplied the medium, or wish to cancel the operation
- Intercept the vector with R0 = -1 if you wish to cancel the operation
- Intercept the vector with R0 = 0 if the time limit is reached, or if you say you have supplied the medium

When you intercept the call to the vector, control passes back to the filing system routine that called OS_UpCall...

- If R0 = -1, then the routine calls OS_UpCall 4; it then returns an error to say that the medium wasn't found.
- If R0 = 0, then the routine checks for you that the medium has been changed and the correct one supplied. If so, it calls OS_UpCall 4; otherwise it just calls OS_UpCall 1 or OS_UpCall 2 again, after incrementing R4

The timeout period in R5 is set to a small value for media that can detect when the medium has been changed (such a floppy disc drives) and to a large value (typically &FFFFFFF) for other media. In the former case, this means that RISC OS will automatically detect that a new medium has been supplied, and check that it is the correct one.

If R1 is unchanged on exit, nobody handled the request. In general, this should be treated by filing systems as the media not present error.

The most common use of OS_UpCall 1 and 2 are to request that a floppy disc is inserted.

[RISC OS Open source page](#)

OS_UpCall 10

SWI &33 · OS_UpCall

The purpose of this call is to inform your application that a stream has been created.

Before calling

R0 10 (&0A)

R1 Device driver's handle

R2 0 if created for reception (else created for transmission)

R3 File handle for stream

R4 DeviceFS stream handle, as passed to device driver on initialisation

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

Notes

DeviceFS issues this call when a stream is created. It serves as a broadcast, and all registers should be preserved.

This call is never issued under RISC OS 2.

[RISC OS Open source page](#)

OS_UpCall 11

SWI &33 · OS_UpCall

The purpose of this call is to inform your application that a stream has been closed.

Before calling

R0 11 (&0B)

R1 Device driver's handle

R2 0 if closed for reception (else code for transmission)

R3 File handle for stream

R4 DeviceFS stream handle, as passed to device driver on initialisation

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

Notes

DeviceFS issued this call when a stream is closed. It serves as a broadcast, and all registers should be preserved.

This call is never issued under RISC OS 2.

[RISC OS Open source page](#)

OS_UpCall 12

SWI &33 · OS_UpCall

The purpose of this call is to warn that an object has been added or changed, removed (by the holder), or deleted (by Freeway).

Before calling

R0 12 (&0C)

R1 Freeway reason code

R2 Object type number

R3 Pointer to buffer for object name

R4 Length of buffer for object descriptor

R5 Pointer to object descriptor

R6 IP address of computer which holds the object

On return

--

Reason Codes

Value / Meaning

0 – Meaning: Object added

1 – Meaning: Object removed

2 – Meaning: Object attributes have changed

3 – Meaning: Local object deleted by Freeway

Notes

This call warns controlling applications that the named object has been added, removed or changed. It is also issued when Freeway has detected that the information held about a locally held object is unreliable, (as a result of possible temporary name duplication, for example) and has removed it unilaterally.

It is issued on callback, and the contents of the supplied name and descriptor buffers are guaranteed not to change provided they are read from within the application's UpCall handler.

[RISC OS Open source page](#)

OS_UpCall 15

SWI &33 · OS_UpCall

The purpose of this call is to inform your application when there is data entering a previously empty buffer.

Before calling

R0 15 (&0F)

R1 FileSwitch stream handle

On return

R0 Preserved

R1 Preserved

Notes

This service is issued when there is data entering a previously empty buffer. Only devices opened for read will receive this UpCall.

[RISC OS Open source page](#)

OS_UpCall 16

SWI &33 · OS_UpCall

The purpose of this call is to inform your application when the amount of data in a stream exceeds the threshold.

Before calling

R0 16 (&10)

R1 FileSwitch stream handle

On return

R0 Preserved

R1 Preserved

Notes

This service is issued when the amount of data in a stream exceeds the threshold.

[RISC OS Open source page](#)

OS_UpCall 17

SWI &33 · OS_UpCall

The purpose of this call is to inform your application when the amount of data in a stream falls below the threshold.

Before calling

R0 17 (&11)

R1 FileSwitch stream handle

On return

R0 Preserved

R1 Preserved

Notes

This service is issued when the amount of data in a stream falls below the threshold.

[RISC OS Open source page](#)

OS_UpCall 19

SWI &33 · OS_UpCall

The purpose of this call is to inform your application that a missing filing medium has been supplied, or that an operation involving one has been cancelled.

Before calling

R0 19 (&13)

R1 pointer to message string

On return

R0 Preserved

Notes

This call is identical to OS_UpCall 4 except that the message pointed to by R1 is displayed. This is useful where there may be some time before another screen update, for example during printing.

[RISC OS Open source page](#)

OS_UpCall 2

SWI &33 · OS_UpCall

The purpose of this call is to warn applications that a filing medium is not known.

Before calling

R0 2

R1 Filing system number

R2 Pointer to media name

R3 Device number

R4 Iteration count

R5 Timeout delay

R6 Pointer to media type

On return

R0 0 if medium change, or -1 if medium no longer required, else preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

R5 Preserved

R6 Preserved

Notes

This Upcall is called when a media change (e.g. another disc) is required.

The following information applies to both OS_UpCall 1 & OS_UpCall 2.

Any routines on this vector should:

- Prompt you to supply the medium with a string built up using:
- The medium type string (passed in R6) #the filing system name (obtained by calling XOS_FSCControl 33 acting on the value of R1)
- The medium name (passed in R2), for example: 'Please insert disc ADFS:XYZ and press Space (Escape to abort)'
- Give you a way of indicating that you have either supplied the medium, or wish to cancel the operation
- Intercept the vector with R0 = -1 if you wish to cancel the operation
- Intercept the vector with R0 = 0 if the time limit is reached, or if you say you have supplied the medium

When you intercept the call to the vector, control passes back to the filing system routine that called OS_UpCall...

- If R0 = -1, then the routine calls OS_UpCall 4; it then returns an error to say that the medium wasn't found.
- If R0 = 0, then the routine checks for you that the medium has been changed and the correct one supplied. If so, it calls OS_UpCall 4; otherwise it just calls OS_UpCall 1 or OS_UpCall 2 again, after incrementing R4

The timeout period in R5 is set to a small value for media that can detect when the medium has been changed (such a floppy disc drives) and to a large value (typically &FFFFFFF) for other media. In the former case, this means that RISC OS will automatically detect that a new medium has been supplied, and check that it is the correct one.

If R1 is unchanged on exit, nobody handled the request. In general, this should be treated by filing systems as the media not present error.

The most common use of OS_UpCall 1 and 2 are to request that a floppy disc is inserted.

[RISC OS Open source page](#)

OS_UpCall 24

SWI &33 · OS_UpCall

The purpose of this call is to inform applications that a new drive has been added.

Before calling

R0 24 (&13)

R1 Pointer to disc name (e.g. "SCSI:4")

On return

R0, R1 Preserved

Notes

Filesystem Filer Modules can act on this UpCall to add an appropriate Filer icon to the taskbar, if it is from their own Filesystem.

[RISC OS Open source page](#)

OS_UpCall 25

SWI &33 · OS_UpCall

The purpose of this call is to inform applications that an existing drive has been removed.

Before calling

R0 25 (&13)

R1 Pointer to disc name (e.g. "SCSI:4")

On return

R0, R1 Preserved

Notes

Filesystem Filer Modules can act on this UpCall to remove the appropriate Filer icon from the taskbar, if it is from their own Filesystem.

[RISC OS Open source page](#)

OS_UpCall 256

SWI &33 · OS_UpCall

The purpose of this call is to inform your application that a new application is going to be started.

Before calling

R0 256 (&100)

R2 Currently active object Pointer

On return

R0 0 to stop application, else R0 is preserved

R2 Preserved?

Notes

This call is made just before a new application is going to be started in the current application space – for example, due to a *Run or module command. It calls the UpCall vector.

One reason to use this call is so that an application can tidy up itself before a new one is started, e.g. removing routines from vector.

Another reason to use this UpCall is to prevent an application from starting. If you don't want the application to start, your routine should set R0 to 0, and intercept the call to the vector. This will cause the error 'Unable to start application' to be given. Otherwise, you must pass the call on with all registers preserved.

[RISC OS Open source page](#)

OS_UpCall 257

SWI &33 · OS_UpCall

The purpose of this call is to inform your application that the Operating System would like to move memory

Before calling

R0 257 (&101)

R1 Amount that application space is going to change by

On return

R0 0 to permit memory moved, else preserved

R1 Preserved

Notes

This call is made just before OS_ChangeDynamicArea tries to move memory. The call is only made if the currently active object is in the application space. It calls the UpCall vector. By default, (if you don't claim the vector) the memory isn't moved.

Your routine must shuffle your application's workspace so that the memory can go ahead. It must then set R0 = 0, and pass on the call to the vector.

[RISC OS Open source page](#)

OS_UpCall 3

SWI &33 · OS_UpCall

The purpose of this call is to warn applications that a file is being modified.

Before calling

R0 3

R1 Pointer to filename

R2 Load address

R3 Execution address

R4 Pointer to start of buffer

R5 Pointer to end of buffer

R6 Pointer to special field (or 0)

R8 Filing system information word

R9 0 (reason code)

R0 3

R1 Pointer to filename

R2 Load address

R3 Execution address

R5 Attributes

R6 Pointer to special field (or 0)

R8 Filing system information word

R9 1 (reason code)
R0 3
R1 Pointer to filename
R2 Load address
R6 Pointer to special field (or 0)
R8 Filing system information word
R9 2 (reason code)
R0 3
R1 Pointer to filename
R3 Execution address
R6 Pointer to special field (or 0)
R8 Filing system information word
R9 3 (reason code)
R0 3
R1 Pointer to object name
R5 Attributes
R6 Pointer to special field (or 0)
R8 Filing system information word
R9 4 (reason code)
R0 3
R1 Pointer to object name
R6 Pointer to special field (or 0)
R8 Filing system information word
R9 6 (reason code)
R0 3
R1 Pointer to filename
R2 Load address
R3 Execution address
R4 Start address
R5 End address
R6 Pointer to special field (or 0)
R8 Filing system information word
R9 7 (reason code)
R0 3
R1 Pointer to directory name
R2 Load address (to be used as timestamp)
R3 Execution address (to be used as timestamp)
R4 Number of entries (0 for default)
R6 Pointer to special field (or 0)
R8 Filing system information word
R9 8 (reason code)
R0 3
R1 Pointer to filename
R2 External handle that file will be given (if successfully opened)
R6 Pointer to special field (or 0)

R8 Filing system information word
R9 257 (reason code)
R0 3
R1 Pointer to filename
R2 External handle that file will be given (if successfully opened)
R6 Pointer to special field (or 0)
R8 Filing system information word
R9 258 (reason code)
R0 3
R1 External file handle
R8 Filing system information word
R9 259 (reason code)
R0 3
R1 External file handle
R2 Size to ensure
R8 Filing system information word
R9 512 (reason code)
R0 3
R1 Pointer to current object name
R2 Pointer to desired object name
R6 Pointer to current special field
R7 Pointer to desire special field
R8 Filing system information word
R9 520 (reason code)
R0 3
R1 Pointer to current object name
R2 Pointer to attribute string (control character terminated)
R8 Filing system information word
R9 521 (reason code)

On return

R0 Preserved
All registers preserved
All registers preserved
All registers preserved
All registers preserved
All registers preserved
All registers preserved
All registers preserved
All registers preserved
All registers preserved
All registers preserved
All registers preserved
All registers preserved
All registers preserved
All registers preserved

All registers preserved

In practice

The action performed depends on the reason code value in R9.

Reason Codes

- 1 – &00: &01 · Saving memory to file: Writing catalogue information
- 2 – &00: &02 · Saving memory to file: Writing load address only
- 3 – &00: &03 · Saving memory to file: Writing execution address only
- 4 – &00: &04 · Saving memory to file: Writing attributes only
- 6 – &00: &06 · Saving memory to file: Deleting file
- 7 – &00: &07 · Saving memory to file: Creating empty file
- 8 – &00: &08 · Saving memory to file: Creating directory
- 257 – &00: &101 · Saving memory to file: Creating and opening for update
- 258 – &00: &102 · Saving memory to file: Opening for update
- 259 – &00: &103 · Saving memory to file: Closing file
- 512 – &00: &200 · Saving memory to file: Ensure file size
- 520 – &00: &208 · Saving memory to file: Renaming file
- 521 – &00: &209 · Saving memory to file: Setting attributes

Notes

It is made when a program calls one of several SWI provided by FileSwitch module:

- Reason codes 0-8 are caused by calls to OS_File
- Reason codes 257-9 are caused by calls to OS_Find
- Reason codes 520 & 521 are caused by calls to OS_FSControl

The following general points apply:

- All strings are 0-terminated except where specified
- All object names will already have been expanded by FileSwitch, checked for basic validity, and had filing system prefixes stripped
- Object names will also be canonicalised, except under RISC OS 2.

This UpCall is made before the operation, which may subsequently fail. For example, you may receive a rename UpCall for a locked file, which will subsequently fail to rename (because it's locked). If a file name is invalid for a given operation (eg you try to create a file with a wildcarded leafname) FileSwitch will generate an error, and no UpCall will be generated.

The call is used by the desktop filer to maintain its directory displays. It's provided for information only; if you wish to use this UpCall, you must not intercept it, nor must you alter the contents of any of these registers used to pass parameters.

%2Systems#special_fields

[RISC OS Open source page](#)

OS_UpCall 4

SWI &33 · OS_UpCall

The purpose of this call is to inform your application that a missing filing medium has been supplied, or that an operation involving one has been cancelled.

Before calling

R0 4

On return

R0 Preserved

Notes

It is always preceded by call(s) of OS_UpCall 1 or OS_UpCall 2. This call is typically used to remove error messages displayed when OS_UpCall 1 or 2 was first generated.

[RISC OS Open source page](#)

OS_UpCall 6

SWI &33 · OS_UpCall

The purpose of this call is to inform the TaskWindow module that a task wants to sleep until some termination condition is met.

Before calling

R0 6

R1 Pointer to poll word (in a global memory area, eg the RMA)

On return

R0 0 if UpCall claimed

R1 Preserved?

Notes

This call is made by a task that wants to sleep until some termination condition is met, signalled by the contents of the poll word becoming non-zero. It is not available in RISC OS 2.

The poll word must be in non-paged memory (e.g. the RMA or another dynamic area), and must be readable from user mode (TaskWindow 0.65+ will read it from user mode).

Control may return to the task before the poll word becomes non-zero. While the task is sleeping, other tasks will continue to be polled by the Wimp.

If the termination condition can be recognised externally (i.e. in another Wimp task or under interrupt) hence causing the poll word to be set non-zero, the calling task should set the poll word to zero on entry. Otherwise, the poll word must be non-zero on entry, so that the control will return to the calling task after each Wimp_Poll.

A task must not use this UpCall if it's not re-entrant.

The calling task must be running in a task window. The task window module intercepts this UpCall; you should not do so yourself. These two restrictions may be removed in future versions of RISC OS.

[RISC OS Open source page](#)

OS_UpCall 7

SWI &33 · OS_UpCall

The purpose of this call is to inform the TaskWindow module that an open pipe has been closed or deleted

Before calling

R0 7

R1 Pointer to poll word (in a global memory area, e.g. the RMA)

On return

R0 Preserved (if V flag clear)

Pointer to error block (if V flag set)

Notes

It is made by PipeFS if an open pipe is closed or deleted. It's not available in RISC OS 2.

The Task Window module then traps this and objects if any of its tasks are currently waiting for the poll word related to that pipe to become non-zero, by returning an error.

This prevents a *Shut command from deleting the workspace which is being accessed by the Task Window, which could potentially cause address exceptions.

[RISC OS Open source page](#)

OS_UpCall 8

SWI &33 · OS_UpCall

The purpose of this call is to inform your application that a buffer's free space has become less than its specified threshold.

Before calling

R0 8

R1 Buffer handle

R2 0

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

The Buffer Manager issues this call when data is inserted into the specified buffer, and the free space becomes less than its current threshold.

This call is never issued under RISC OS 2.

[RISC OS Open source page](#)

OS_UpCall 9

SWI &33 · OS_UpCall

The purpose of this call is to inform your application that a buffer's free space has become greater than or equal to its specified threshold.

Before calling

R0 9

R1 Buffer handle

R2 -1

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

The Buffer Manager issues this call when data is removed from the specified buffer, and the free space becomes greater than or equal to its current threshold.

This call is never issued under RISC OS 2.

[RISC OS Open source page](#)

OS_UpdateMEMC

SWI &1A

The purpose of this call is to read or write the contents of the MEMC control register.

Before calling

R0 New bits

R1 Mask

On return

R0 Old bits value

R1 Old mask value

Notes

This SWI programs individual bits in the MEMC control register, on machines still supporting Acorn's custom chipsets. Since the MEMC is write-only, this is done via a software copy, maintained by the OS.

Reg = (RegCopy AND NOT R1) OR (R0 AND R1)

To read: R0,R1 = 0

[RISC OS Open source page](#)

OS_ValidateAddress

SWI &3A

The purpose of this call is to check that a given address range is valid, i.e. that it maps to some kind of RAM.

Before calling

R0 Minimum address

R1 Maximum address

On return

R0 Preserved

R1 Preserved

C flag is clear if the range is OK, else set

Notes

This call does not check that the memory actually exists, only that it ought to exist. If you have used `OS_SetMemMapEntries`, then this call will not work as expected. This call also does not report any access information – no guarantee is made that you will be able to read or write to the memory in your current CPU mode.

The behaviour of `OS_ValidateAddress` has changed significantly between different OS versions.

Archimedes

Prior to RISC OS 3.5, the following areas were considered valid:

- Any logical address (0-32MB) which has RAM mapped to it
- The second mapping of screen memory at the start of the physical mapping of RAM
- (In RISC OS 3) anything reported as valid by `Service_ValidateAddress`

RISC OS 3.5

From RISC OS 3.5, the following areas were considered valid:

- Any dynamic area
- Several special kernel regions – zero page workspace, processor stacks, etc.
- Screen memory
- Anything reported as valid by `Service_ValidateAddress`

However note that this version of `Service_ValidateAddress` does not correctly detect valid regions of Sparse Dynamic Areas. Additionally, not all special regions are reported – e.g. the CAM, page tables, kernel buffers used with long command lines, RISC OS 5 HAL workspace, etc.

RISC OS Select

Under RISC OS Select the call was reimplemented as a call to `OS_Memory 24`. Physically mapped regions will not be reported as valid, however `Service_ValidateAddress` can be used to override this. The `LegacyScreen` module uses this technique to mark screen memory as valid. All other regions reported by `OS_Memory 24` are considered valid, e.g. system dynamic areas such as the ROM dynamic area. Special code also exists to mark zero page as valid.

RISC OS 5.21

In RISC OS 5.21 the call was also changed to be implemented via `OS_Memory 24`. The OS considers the following locations as valid:

- Any non-physically mapped memory which is completely user or privileged readable or writable
- Any completely physically mapped memory which is also completely user readable & writable (i.e. screen memory on some systems)
- Anything reported as valid by `Service_ValidateAddress`.

Under RISC OS 5 no special code is required to mark zero page as valid, as zero page is one of the areas which `OS_Memory 24` will report on.

[RISC OS Open source page](#)

OS_Word

SWI &07 · Controlling sources: `Hdr:OsWords` managed by the allocations manager

The purpose of this call is to provide a number of general operations.

Before calling

R0 Reason code

All other registers dependent on reason code

On return

R0 Preserved

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in **R0**.

Reason Codes

- 0** – Hex #: &00 · Action: Read line from input stream
- 1** – Hex #: &01 · Action: Read system clock
- 2** – Hex #: &02 · Action: Write system clock
- 3** – Hex #: &03 · Action: Read interval timer
- 4** – Hex #: &04 · Action: Write interval timer
- 7** – Hex #: &07 · Action: Make a SOUND
- 9** – Hex #: &09 · Action: Read pixel logical colour
- 10** – Hex #: &0A · Action: Read a character definition
- 11** – Hex #: &0B · Action: Read palette
- 12** – Hex #: &0C · Action: Write palette
- 13** – Hex #: &0D · Action: Read current and last graphics cursors
- 14** – Hex #: &0E · Action: Read CMOS clock
- 15** – Hex #: &0F · Action: Write CMOS clock
- 21** – Hex #: &15 · Action: Define pointer and mouse parameters
- 22** – Hex #: &16 · Action: Write screen base address

[RISC OS Open source page](#)

OS_Word 0

SWI &07 · OS_Word

The purpose of this call is to read a line from the input stream.

Before calling

R0 0

R1 Pointer to 5 byte Buffer

On return

R0 Preserved

R1 Buffer updated

Notes

This call is deprecated. Please use OS_ReadLine instead.

[RISC OS Open source page](#)

OS_Word 1

SWI &07 · OS_Word

The purpose of this call is to read the system clock.

Before calling

R0 1

R1 Pointer to 5 byte Buffer

On return

R0 Preserved

R1 Preserved

Notes

This SWI reads the system clock. On exit, the buffer contains five byte system clock, with LSB first.

The clock is incremented 100 times a second, and reset on a hard break.

To write to system clock, use OS_Word 2.

[RISC OS Open source page](#)

OS_Word 10

SWI &07 · OS_Word

The purpose of this call is to read a character definition.

Before calling

R0 10

R1 9 byte block:

+0 Character code

On return

R0 Preserved

R1 Buffer containing:

+1 Top row of definition

+. ...

+. ...

+8 Bottom row of definition

Notes

Valid values are 32-126 and 128-256.

Values 2-5 returns ECF pattern 1-4, while value 6 returns dot-dash pattern.

[RISC OS Open source page](#)

OS_Word 11

SWI &07 · OS_Word

The purpose of this call is to read the palette.

Before calling

R0 11 (&0B)

R1 Buffer:

+0 Logical colour to read

On return

R0 Preserved

R1 Preserved

+1 Physical colour associated with the logical colour

+2 Red component

+3 Green component

+4 Blue component

Notes

This call is deprecated. Please use OS_ReadPalette instead.

[RISC OS Open source page](#)

OS_Word 12

SWI &07 · OS_Word

The purpose of this call is to write the palette.

Before calling

R0 12 (&0C)

R1 Buffer:

+0 Logical colour to change

+1 New physical colour

+2 Red component

+3 Green component

+4 Blue component

On return

R0 Preserved

R1 Preserved

Notes

This call is equivalent to VDU 19, but is faster and may be used in interrupt routines.

To read the palette use OS_ReadPalette.

[RISC OS Open source page](#)

OS_Word 13

SWI &07 · OS_Word

The purpose of this call is to read the current and last graphics cursors.

Before calling

R0 13 (&0D)

R1 Pointer to 8 byte buffer

On return

R0 Preserved

R1 Buffer updated

Notes

The buffer in R1 gets updated with the current and previous graphics cursors co-ordinates.

This call is deprecated. Please use OS_ReadVduVariables instead.

[RISC OS Open source page](#)

OS_Word 14

SWI &07 · OS_Word

The purpose of this call is to provide a number of read operations relating to the CMOS clock.

In practice

The action performed depends on the reason code value at offset +0 in R1.

Reason Codes

/ Hex # / Action

0 – Hex #: &00 · Action: Read soft-copy of CMOS clock as string

1 – Hex #: &01 · Action: Read CMOS clock as BCD

2 – Hex #: &02 · Action: Convert BCD to string

3 – Hex #: &03 · Action: Read soft-copy of CMOS clock as 5-byte integer

[RISC OS Open source page](#)

OS_Word 15

SWI &07 · OS_Word

The purpose of this call is to provide a number of write operations relating to CMOS clock.

In practice

The action performed depends on the reason code value at offset +0 in R1.

Reason Codes

/ Hex # / Action

5 – Hex #: &05 · Action: Write CMOS clock and soft-copy time from 5 byte centisecond value

8 – Hex #: &08 · Action: Write CMOS clock and soft-copy time from string

15 – Hex #: &0F · Action: Write CMOS clock and soft-copy date from string

24 – Hex #: &18 · Action: Write CMOS clock and soft-copy time and date from string

[RISC OS Open source page](#)

OS_Word 2

SWI &07 · OS_Word

The purpose of this call is to write the system clock.

Before calling

R0 2

R1 Pointer to 5 byte block containing new system clock value, LSB first

On return

R0 Preserved

R1 Preserved

Notes

The clock is incremented 100 times a second, and reset on a hard break.

The 5 byte block returned, has the LSB first.

To read the system clock, use OS_Word 1.

[RISC OS Open source page](#)

OS_Word 21

SWI &07 · OS_Word

The purpose of this call is to provide a number of operations relating to the mouse and pointer.

In practice

The action performed depends on the reason code value at offset +0 in R1.

Reason Codes

- 0 – Hex #: &00 · Action: Define pointer shape
- 1 – Hex #: &01 · Action: Define the mouse co-ordinate bounding box
- 2 – Hex #: &02 · Action: Set mouse multipliers
- 3 – Hex #: &03 · Action: Set mouse position
- 4 – Hex #: &04 · Action: Read unbuffered mouse position
- 5 – Hex #: &05 · Action: Set pointer position
- 6 – Hex #: &06 · Action: Read pointer position
- 7 – Hex #: &07 · Action: Read the mouse bounding box (RISCOS Ltd Select only, not implemented on RISC OS 5)

[RISC OS Open source page](#)

OS_Word 22

SWI &07 · OS_Word

The purpose of this call is to setup a new screen base address.

Before calling

R0 22 (&16)

R1 Pointer to 5-byte buffer, containing:

+0 Type

+1 Unaligned word containing new screen offset

On return

R0 Preserved

R1 Preserved

Notes

It is given as the offset from the base of the screen. The offset must be divisible by 16.

[RISC OS Open source page](#)

OS_Word 3

SWI &07 · OS_Word

The purpose of this call is to read the interval timer.

Before calling

R0 3

R1 Pointer to 5 byte block

On return

R0 Preserved

R1 Preserved

Notes

On exit, the buffer contains five byte interval timer, with LSB first.

The timer is incremented 100 times a second, and generates a interval timer event (event number 5) when it reaches 0. (You must enable the event with OS_Byte 14).

To generate repeating events you would have to reload the timer on each event. The timer is obsolescent – use OS_CallAfter or OS_CallEvery instead.

To write the interval timer, use OS_Word 4.

[RISC OS Open source page](#)

OS_Word 4

SWI &07 · OS_Word

The purpose of this call is to write the interval timer.

Before calling

R0 4

R1 Pointer to 5 byte block

On return

R0 Preserved

R1 Preserved

Notes

On exit, the buffer contains five byte interval timer, with LSB first.

The timer is incremented 100 times a second, and generates a interval timer event (event number 5) when it reaches 0. (You must enable the event with OS_Byte 14).

To generate repeating events you would have to reload the timer on each event. The timer is obsolescent – use OS_CallAfter or OS_CallEvery instead.

To read the interval timer, use OS_Word 3.

[RISC OS Open source page](#)

OS_Word 7

SWI &07 · OS_Word

This call makes a sound, equivalent to the SOUND keyword in BBC BASIC.

Before calling

R0 7

R1 8 byte block:

+0 Channel

+2 Amplitude

+4 Pitch

+6 Duration

On return

R0 Preserved

R1 Preserved

Notes

Internally this reason code calls Sound_ControlPacked and is only provided for compatibility with the BBC Microcomputer system call.

[RISC OS Open source page](#)

OS_Word 9

SWI &07 · OS_Word

The purpose of this call is to return the logical colour at the specified pixel.

Before calling

R0 9

R1 Pointer to 5 byte block

On return

R0 Preserved

R1 Logical colour block

Notes

This SWI returns the logical colour at the specified pixel. &FF is returned if:

- Screen is in a 256-colour mode.
- Pixel is off screen
- Mode is non-graphical

This call is deprecated. Please use OS_ReadPoint instead.

[RISC OS Open source page](#)

OS_Write0

SWI &02

The purpose of this call is to send a string pointed to by R0 to all active output streams.

Before calling

R0 Pointer to null-terminated string

On return

R0 Pointer to the byte after the null

Notes

If WrchV vector has not been intercepted with OS_Claim it will directly call the VDU driver which is faster, otherwise it calls OS_WriteC to write each character.

An output stream can be specified by using OS_Byte 3.

[RISC OS Open source page](#)

OS_WriteC

SWI &00

The purpose of this call is to send a "character" to all of the active output streams.

Before calling

R0 "Character" to write

On return

R0 Preserved

Parameters

The meaning of "character" is highly context-dependent. If there is a VDU queue (OS_Byte 218) then it's a data byte. Otherwise its width is defined by the current Alphabet (eg for #117 UTF16 it is 16 bit) - see Service_International 332.

If it's between 0 and 31 it will usually be a control code (though during OS_Plot 208-215 that's a FontManager control code).

For multicode Alphabets like UTF-8 and Shift-JIS, it may require multiple codes to specify a single character code.

Notes

This SWI sends R0 to all active output streams (see OS_Byte 3 for specifying a stream). It will call WrchV if that has claimants, it may call VDUXV, and may subsequently call UKVDU23V.

Do not assume that this SWI will never generate an error - it can and occasionally does. Consider the case where output is redirected to a file on a filing system which has no space. This is the most common example.

A less common, but possible case is where an application displays a string in its Exit handler. Under a TaskWindow, if you close the window before the task has exited, its Exit handler is called. This should tidy up and exit. If it prints strings using non-X versions of the SWIs an error will be generated. This will call the Error handler, which will either display a message or call the Exit handler. In the former case, if X SWIs are used the Exit handler is usually called.

The outcome of this is that a TaskWindow can hog 99% of the processor time as it switches in and out of SVC mode incredibly quickly in an infinite sequence of errors. Avoid this by using X SWIs.

[RISC OS Open source page](#)

OS_WriteEnv

SWI &48

The purpose of this call is to set the environment string as read by OS_GetEnv.

Before calling

R0 Pointer to environment string

R1 Pointer to real time the program was started (5 bytes)

On return

R0 Preserved

R1 Preserved

[RISC OS Open source page](#)

OS_Writel

SWI &100 - &1FF

The purpose of this call is to write an immediate byte to all of the active output streams.

Before calling

--

On return

--

Notes

Writes the character contained in the bottom byte of the SWI, using OS_WriteC.

[RISC OS Open source page](#)

OS_WriteN

SWI &46

The purpose of this call is to write a counted string to the VDU.

Before calling

R0 Pointer to string

R1 Number of bytes to write

On return

R0 Preserved

R1 Preserved

Notes

This SWI sends the R1 bytes in R0 to all active output streams (specified by OS_Byte 3).

If WrchV has not been intercepted with OS_Claim it will directly call the VDU driver which is faster, otherwise it calls OS_WriteC to write each character.

[RISC OS Open source page](#)

OS_WriteS

SWI &01

The purpose of this call is to write the following string to all of the active output streams.

Before calling

--

On return

--

Notes

This SWI writes the null terminated string following this SWI in memory using OS_WriteC. It uses the value in R14 to find the string, and updates R14 to point after string.

If WrchV has not been intercepted with OS_Claim it will directly call the VDU driver which is faster, otherwise it calls OS_WriteC to write each character.

This SWI can not be conditionally executed [unless the string you wish to write out also happens to be valid ARM code for the condition failed case :-)].

[RISC OS Open source page](#)

Parallel_HardwareAddress

SWI &42EC0 · ParallelDeviceDriver

This call is for internal use only.

Before calling

--

On return

R0 Address of parallel controller in memory map, or 0 if no hardware present

R1 Address of routine to be called in SVC mode to do: Parallel control register = (Old value AND R1) EOR R0 or preserved if R0 was returned as 0

Notes

Do not use this SWI Call. Use Parallel_Op instead.

[RISC OS Open source page](#)

Parallel_Op

SWI &42EC1 · ParallelDeviceDriver

The purpose of this call is to perform direct operations to the parallel hardware.

Before calling

R0 Reason code

Reason Codes

Value / Meaning

0 – Meaning: Read data and status registers

1 – Meaning: Write data registers

2 – Meaning: Read/write control register

Notes

Reason Codes shown are in decimal.

[RISC OS Open source page](#)

Parallel_Op 0

SWI &42EC1 · ParallelDeviceDriver

The purpose of this call is to read the data and status registers taken from the printer port.

Before calling

R0 0 (Reason code)

On return

R0 Preserved

R1 Data register contents

R2 Status register contents

Status Register Contents

0 · Reserved –

1 · Reserved –

2 · Reserved –

3 · Error – Port: 15

4 · Select – Port: 13

5 · Paper error – Port: 12

6 · Acknowledge – Port: 10

7 · Not Busy – Port: 11

Notes

The data register is either the current value on the port, or the last value written if output is enabled.

[RISC OS Open source page](#)

Parallel_Op 1

SWI &42EC1 · ParallelDeviceDriver

The purpose of this call is to write the data and status registers taken from the printer port.

Before calling

R0 1 (Reason code)

R1 Data to write

On return

R0 Preserved

R1 Preserved

Notes

A copy of this data will be returned on the next read via Parallel_Op 0.

[RISC OS Open source page](#)

Parallel_Op 2

SWI &42EC1 · ParallelDeviceDriver

The purpose of this call is to read/write the parallel control register.

Before calling

R0 2 (Reason code)

R1 EOR mask

R2 AND mask

On return

R0 Preserved

R1 Old contents of register

R2 New contents of register

Status Register Contents

0 · Not Strobe – Port: 1

1 · Not Autofeed – Port: 14

2 · Init – Port: 16

3 · Not SelectIn – Port: 17

4 · IRQ enable –

5 · Direction –

6 · Reserved –

7 · Reserved –

Notes

This SWI writes or reads the the control register sent to the printer port.

[RISC OS Open source page](#)

PCI_ConfigurationRead

SWI &50388 · PCI

The purpose of this call is to perform a Configuration Read cycle of the denoted size.

Before calling

R0 Configuration space address (0-255) or bus/dev/fn/address

R2 Access size (1, 2 or 4 for byte/half word/word)

R3 Function handle, or 0

On return

R0 Preserved ?

R1 Value read

R2 Preserved ?

R3 Preserved ?

Notes

PCI_ConfigurationWrite is used to perform a Configuration Write cycle of the denoted size.

[RISC OS Open source page](#)

PCI_ConfigurationWrite

SWI &50389 · PCI

The purpose of this call is to perform a Configuration Write cycle of the denoted size.

Before calling

R0 Configuration space address (0-255) or bus/dev/fn/address

R1 Value to write

R2 Access size (1, 2 or 4 for byte/half word/word)

R3 Function handle, or 0

On return

R0 Preserved ?

R1 Preserved ?

R2 Preserved ?

R3 Preserved ?

Notes

PCI_ConfigurationRead is used to perform a Configuration Read cycle of the denoted size.

[RISC OS Open source page](#)

PCI_EnumerateFunctions

SWI &50383 · PCI

The purpose of this call is to iteratively read through each of the PCI devices on each call, until an error is returned.

Before calling

R3 0 to start enumeration, or last value from PCI_EnumerateFunctions

On return

R0 Vendor ID (bits 0 – 15)

Device ID (bits 16 – 31)

R1 Subsystem Vendor ID (bits 0 – 15)

Subsystem ID (bits 16 – 31)

R2 Class code

R3 Function handle

[RISC OS Open source page](#)

PCI_FindByClass

SWI &5038F · PCI

The purpose of this call is to find the function handle based on the class passed.

Before calling

R0 Class code (24 bit)

R1 Mask - matches if (function_class AND R1) = R0

R3 0 to start, or R3 value last found

On return

R0 Preserved ?

R1 Preserved ?

R3 Function handle

Notes

Only the bits in R1 of the class code will be checked so a mask of -1 will search for an exact match.

[RISC OS Open source page](#)

PCI_FindByID

SWI &5038E · PCI

The purpose of this call is to find the function handle based on wildcarded information.

Before calling

R0 Vendor ID, or -1 for wildcard

R1 Device ID, or -1 for wildcard

R3 0 to start, or R3 value last found

R4 Subsystem ID, or -1 for wildcard

On return

R0 Preserved ?

R1 Preserved ?

R3 Function handle

R4 Preserved ?

Notes

A value of -1 for R0, R1, R2 or R4 is used to specify a wildcard search.

[RISC OS Open source page](#)

PCI_FindByLocation

SWI &5038D · PCI

The purpose of this call is to find the function handle based on the topology passed in R0 and R1.

Before calling

R0 Bus

R1 Dev/fn

On return

R0 Preserved ?

R1 Preserved ?

R3 Function handle

Notes

This topology will be platform specific and could vary if backplanes with bridges are inserted into the system.

[RISC OS Open source page](#)

PCI_HardwareAddress

SWI &5038A · PCI

The purpose of this call is to define how a PCI address maps into the logical memory map.

Before calling

R0 Input flags

R1 Address index

R3 Function handle

On return

R0 Output flags

R1 PCI address

R2 Area size

R3 Logical address (if R0 bit 31 is clear)

[RISC OS Open source page](#)

PCI_IORead

SWI &5038A · PCI

The purpose of this call is to perform an IO read cycle of the denoted size.

Before calling

R0 PCI address

R2 Access size (1, 2 or 4 for byte/half word/word)

On return

R0 Preserved ?

R1 Value read

R2 Preserved ?

Notes

The PCI address used by R0 on entry can be found by calling PCI_HardwareAddress.

An IO write operation is performed by calling PCI_IOWrite.

[RISC OS Open source page](#)

PCI_IOWrite

SWI &50385 · PCI

The purpose of this call is to perform an IO write cycle of the denoted size.

Before calling

R0 PCI address

R1 Access size (1, 2 or 4 for byte/half word/word)

R2 Value to write

On return

R0 Preserved ?

R1 Preserved ?

R2 Preserved ?

Notes

The PCI address used by R0 on entry can be found by calling `PCI_HardwareAddress`.

An IO read operation is performed by calling `PCI_IORead`.

[RISC OS Open source page](#)

PCI_LogicalAddress

SWI &50392 · PCI

The purpose of this call is to map-in the given PCI address into the logical memory map.

Before calling

R0 Input flags

R1 PCI address

R2 Length

On return

R0 Preserved ?

R1 Preserved ?

R2 Preserved ?

R4 Logical address

Notes

The PCI address in R0 can be found by calling `PCI_HardwareAddress`.

[RISC OS Open source page](#)

PCI_MemoryRead

SWI &50386 · PCI

The purpose of this call is to perform a Memory Read cycle of the denoted size.

Before calling

R0 PCI address

R1 Length of transfer (whole number of words)

R2 Pointer to buffer

R4 Flags

On return

R0 Preserved ?

R1 Buffer updated

R2 Preserved ?

R4 Preserved ?

Notes

The PCI address in R0 can be found by calling `PCI_HardwareAddress`.

[RISC OS Open source page](#)

PCI_MemoryWrite

SWI &50387 · PCI

The purpose of this call is to perform a Memory Write cycle of the denoted size.

Before calling

R0 PCI address

R1 Length of transfer (whole number of words)

R2 Pointer to buffer

R4 Flags

On return

R0 Preserved ?

R1 Buffer updated

R2 Preserved ?

R4 Preserved ?

Notes

The PCI address in R0 can be found by calling `PCI_HardwareAddress`.

[RISC OS Open source page](#)

PCI_RAMAlloc

SWI &50390 · PCI

The purpose of this call is to allocate memory from a fixed, contiguous memory pool, suitable for access by other bus masters.

Before calling

R0 Size required

R1 Alignment required (e.g. &1000 it must be 4K aligned – 0 if none)

R2 Boundary limitation (e.g. &10000 it mustn't cross 64K boundary – 0 if none)

On return

R0 Logical address

R1 Physical address

R2 Preserved

Notes

This is provided for the convenience of drivers that only need a few simple data structures for communication with their PCI device, and don't want the complexity of dealing with cache coherency, memory fragmentation and physical page moving (Service_PagesUnsafe et al).

Memory will be uncachable but bufferable – you must ensure that any writes you perform have taken place before another bus master reads the memory, e.g. by using the barrier routines available through OS_MMUControl 2, or by calling OS_MMUControl 1 with bit 28 of R0 set. The memory is not accessible from user mode.

[RISC OS Open source page](#)

PCI_RAMFree

SWI &50391 · PCI

The purpose of this call is to return the block of memory pointed to by R0 (claimed earlier with PCI_RAMAlloc) to the PCI memory pool for others to use.

Before calling

R0 Logical address (returned by PCI_RAMAlloc)

On return

--

[RISC OS Open source page](#)

PCI_ReadHeader

SWI &50381 · PCI

The purpose of this call is to read as much of the header as requested, rather than the limited information returned by PCI_ReadID.

Before calling

R1 Pointer to buffer

R2 Number of bytes to read (4-256, must be multiple of 4)

R3 Function handle

On return

R1 Buffer updated

R2 Preserved ?

R3 Preserved ?

[RISC OS Open source page](#)

PCI_ReadID

SWI &50380 · PCI

The purpose of this call is to returns the device and vendor information for the given PCI device.

Before calling

R3 Function handle

On return

R0 Vendor ID (bits 0 – 15)

Device ID (bits 16 – 31)

R1 Subsystem Vendor ID (bits 0 – 15)

Subsystem ID (bits 16 – 31)

R3 Preserved ?

[RISC OS Open source page](#)

PCI_ReadInfo

SWI &5038B · PCI

The purpose of this call is to return the device and vendor information for the given PCI device.

Before calling

R0 Information request

R1 Word aligned buffer to receive results (4 bytes each)

R2 Length of buffer

R3 Function handle

On return

R0 Preserved ?

R1 Preserved ?

R2 Length of results written

R3 Preserved ?

Notes

The bit settings passed in R0 specifies the information that must be returned.

The string pointers returned point to temporary buffers in the PCI Manager's workspace so you should take a local copy of the strings as they will be invalidated by the next call.

This call is analogous to Podule_ReadInfo?, for a given R3.

[RISC OS Open source page](#)

PCI_ReturnNumber

SWI &50382 · PCI

The purpose of this call is to return the number of function handles that have been used.

Before calling

--

On return

R0 Number of Function handles that have been used

Notes

Device drivers may search handles 1 to R0 inclusive to find a device.

[RISC OS Open source page](#)

PCI_SpecialCycle

SWI &5038C · PCI

The purpose of this call is to issue a special cycle on the PCI bus.

Before calling

R0 Bus

R1 Message/data

On return

R0 Preserved ?

R1 Preserved ?

Notes

The value in R1 will be call specific.

[RISC OS Open source page](#)

Portable_CommandBMU

SWI &42FC4 · Portable

The purpose of this call is to issue a command to the Battery Management Unit (BMU).

Before calling

R0 Reason code:

- 1 = Remove Power
- 2 = Reserved
- 3 = Reserved
- 4 = Set autostart delay to R1
- 5 = Turn off AC power
- 6 = Turn on AC power
- 7 = Disable BMU event generation
- 8 = Enable BMU event generation

R1 Autostart delay in minutes

On return

R0 Preserved

R1 Preserved

Notes

The value of BMU variables may not immediately be updated.

Not all commands are available in all variants of the module; see the Portable Feature Matrix.

[RISC OS Open source page](#)

Portable_Control

SWI &42FC1 · Portable

The purpose of this call is to control the power settings and features.

Before calling

R0 EOR mask

R1 AND mask

On return

R0 Old control value

R1 New control value

Notes

New value = (old value AND R1) EOR R0

[RISC OS Open source page](#)

Portable_EnumerateBMU

SWI &42FD0 · Portable

This call allows you to enumerate the available BMUs in the system.

Before calling

R0 Previous BMU number

Or -1 to start enumeration

Or 0 to skip the legacy BMU

R1 Flags (Reserved, should be zero)

On return

R0 This BMU index, or -1 for end

R1 BMU device flags (only valid if R0 <> -1)

R2 Mask of supported BMU status flags (only valid if R0 <> -1)

Notes

BMU 0 is a legacy BMU provided for compatibility with software designed to work with the A4, Stork and A9 versions of the Portable module.

- The legacy BMU may be the same as one of the other BMUs
- The legacy BMU may be a composite of several BMUs (e.g. in a system with two standard BMUs, each managing a removable battery, the 'battery present' flag will be set as long as at least one battery is present)

Since the BMU interface now allows for communication with BMUs which don't power the main system (e.g. the BMU for an RTC or NVRAM backup battery), the Portable module may elect not to create a legacy BMU if none of the BMUs present in the system are able to report on the power state of the main battery/power supply.

The current legacy BMU implementation returns values in the same units as the other PortableHAL BMUs. However in future this may be changed to return values compatible with the A4 version of the Portable module.

[RISC OS Open source page](#)

Portable_Idle

SWI &42FC6 · Portable

The purpose of this call is to put the CPU into a low-power idle state until an IRQ or FIQ interrupt is pending. It implements functionality equivalent to the ARMv7 'WFI' instruction.

Before calling

-

On return

-

Notes

SWI Portable_ReadFeatures can be used to determine if this call is available.

It's important to realise that Portable_Idle will function as intended even if IRQs/FIQs are disabled on entry. Therefore, to wait for a specific interrupt (or interrupt driven event), it's recommended to use the following procedure:

- Disable interrupts
- Check if the event has occurred. If so, re-enable interrupts and exit
- Call `Portable_Idle`
- Re-enable interrupts
- Call the interrupt delay routine returned by `OS_PlatformFeatures 0` to ensure the pending interrupt has been handled
- Go to 1

Keeping IRQs disabled for steps 2 and 3 avoids a race condition where the event may fire after you've checked to see if it's fired, but before the CPU is able to go to sleep. If IRQs were enabled and the race condition occurred then the CPU would go to sleep for no reason. At best this will reduce the responsiveness of your code, at worst it will deadlock the system (e.g. if there are no other regular IRQs)

[RISC OS Open source page](#)

Portable_ReadBMUVariable

SWI &42FC2 · Portable

The purpose of this call is to read the Battery Management Unit (BMU) variables.

Before calling

R0 Bits 0-23: BMU variable number

Bits 24-31: BMU number

On return

R0 Preserved

R1 Value

Notes

If your code wants to read multiple variables, `Portable_ReadBMUVariables` is recommended.

The BMU settings can be set by using `Portable_WriteBMUVariable`.

On the A4 and Stork versions of the Portable module, the variable with was 8 bits. The charge estimate (variable 6) would be returned as 255 if its value is unknown.

On the HAL version of the Portable module, the variable width has been increased to 32 bits, and -1 is returned for any unknown values. The measurement units used by some variables has also been changed.

[RISC OS Open source page](#)

Portable_ReadBMUVariables

SWI &42FD1 · Portable

The purpose of this call is to read the Battery Management Unit (BMU) variables.

Before calling

R0 BMU number

R1 Number of variables to read

R2 Pointer to variable list containing R1 entries

On return

- All registers preserved

Variable list updated

In practice

This is an improved version of `Portable_ReadBMUVariable`, which allows multiple variables to be read at once. Using this call when reading multiple variables may allow the BMU driver to perform a more efficient request.

Rather than returning an error state for the entire operation, a status value is returned individually for each variable.

Notes

The BMU settings can be set by using `Portable_WriteBMUVariable`.

[RISC OS Open source page](#)

Portable_ReadFeatures

SWI &42FC5 · Portable

This call allows the user to determine what Portable features are available on the current hardware.

Before calling

-

On return

R1 Available features flags word:

Bit 0: SWI `Portable_Speed` supported

Bit 4: SWI `Portable_Idle` supported

Bit 5: SWI `Portable_Stop` supported

Bit 6: SWI `Portable_SMBusOp?` supported

Other bits: Reserved

Notes

For an overview of which features are available on which hardware platforms and Portable implementations, see the [Portable Feature Matrix](#).

Note that this SWI was introduced after the release of the A4 portable. Therefore if calling this SWI returns an unknown SWI error, but the Portable module is present, your code should assume that an A4 level of functionality is available (i.e. `Portable_Speed` is supported). See the feature matrix for more details.

[RISC OS Open source page](#)

Portable_ReadSensor

SWI &42FD2 · Portable

This call allows you to query the value of certain hardware monitoring sensors which may be present in the system.

Before calling

R0 Sensor type:

0 => temperature (scaled to 0.1K per LSB)

others => reserved for future use

R1 Unit type:

0 => CPU die

1 => BMU

2 => harddisc

3 => GPU die

4 => DRAM

others => reserved for future use

R2 Unit index:

0-N => as appropriate for unit N

On return

R0 Sensor value

[RISC OS Open source page](#)

Portable_SleepTime

SWI &42FCC · Portable

This call allows programs to specify how long the machine should be left idle for before it enters suspend mode (via `Portable_Stop`).

Before calling

R0 Sleep time in centiseconds, or 0 to disable

On return

- All registers preserved

Notes

SWI `Portable_ReadFeatures` can be used to determine if suspend functionality is available.

The machine will only automatically enter suspend mode if the screen saver is active; therefore if the suspend time specified here is lower than the screen saver delay time, the suspend will be postponed until the screensaver starts.

See also `*FreezeTime` which allows the delay (in units of seconds) to be specified via a `*Command`.

[RISC OS Open source page](#)

Portable_Speed

SWI &42FC0 · Portable

The purpose of this call is to set the processor speed.

Before calling

R0 EOR mask

R1 AND mask

On return

R0 Old speed value

R1 New speed value

Notes

The new speed is set as:

New value = (old speed AND R1) EOR R0

Currently supported speed settings are:

0 / Fast

1 – Fast: Slow

Some hardware allows programs to exert more control over the CPU speed by SWI Portable_Speed2 to specify what clock rates the 'slow' and 'fast' speed settings equate to.

[RISC OS Open source page](#)

Portable_Speed2

SWI &42FCE · Portable

The purpose of this call is to provide extra functionality not available through Portable_Speed.

Before calling

R0 Reason code

All other registers are dependent on reason code

On return

All registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

Reason Codes

00 – Hex #: &00 · Action: Report current CPU speed in MHz

01 – Hex #: &01 · Action: Report min & max CPU speed in MHz

02 – Hex #: &02 · Action: Set 'slow' and 'fast' speeds in MHz

03 – Hex #: &03 · Action: Report current 'slow' and 'fast' speeds in MHz

04 – Hex #: &04 · Action: Report number of CPU speeds available

05 – Hex #: &05 · Action: Convert speed index to MHz

- 06 – Hex #: &06 · Action: Convert MHz to index
- 07 – Hex #: &07 · Action: Report current CPU speed as index
- 08 – Hex #: &08 · Action: Set 'slow' and 'fast' speeds using indices
- 09 – Hex #: &09 · Action: Report current 'slow' and 'fast' speed indices

[RISC OS Open source page](#)

Portable_Speed2 0

SWI &42FCE · Portable

The purpose of this call is to read the current CPU speed, in MHz. To instead read the CPU speed index, use Portable_Speed2 7.

Before calling

R0 0 (Reason code)

On return

R0 Current CPU speed, in MHz

Notes

There is no direct method available to set the CPU speed to a desired value. Instead programs must use Portable_Speed2 2 (or Portable_Speed2 8) to set the desired 'slow' and 'fast' speeds, and then Portable_Speed to switch to the desired speed.

[RISC OS Open source page](#)

Portable_Speed2 1

SWI &42FCE · Portable

The purpose of this call is to return the minimum and maximum CPU speeds that the system supports.

Before calling

R0 1 (Reason code)

On return

R0 Minimum supported CPU speed, in MHz

R1 Maximum supported CPU speed, in MHz

Notes

To find out more information about the available CPU speeds, see Portable_Speed2 4 and Portable_Speed2 5. To find the current CPU speed, see Portable_Speed2 0.

To set the CPU speed, see Portable_Speed2 2 or Portable_Speed2 8 (to set the desired 'slow' and 'fast' speeds), and Portable_Speed (to switch between the configured 'slow' and 'fast' speeds).

[RISC OS Open source page](#)

Portable_Speed2 2

SWI &42FCE · Portable

The purpose of this call is to allow the user to select the clock speeds that are associated with Portable_Speeds 'slow' and 'fast' settings. The values the user supplies in R1 and R2 will be matched to the set of speeds that the hardware supports, and the updated values, representing the new speeds that will actually be used, will be returned to the caller.

Before calling

R0 2 (Reason code)

R1 Desired 'slow' speed, in MHz

R2 Desired 'fast' speed, in MHz

On return

R0 Preserved

R1 New 'slow' speed, in MHz

R2 New 'fast' speed, in MHz

Notes

Portable_Speed2 8 provides similar functionality to this call, but the speeds are expressed as indices into the list of available speeds, not as raw MHz values.

To read the currently configured 'slow' and 'fast' speeds, use Portable_Speed2 3.

To find out more information about the available CPU speeds, see Portable_Speed2 4 and Portable_Speed2 5. To find the current CPU speed, see Portable_Speed2 0.

[RISC OS Open source page](#)

Portable_Speed2 3

SWI &42FCE · Portable

The purpose of this call is to read the currently configured 'slow' and 'fast' speeds. These are the speeds that Portable_Speed will use when asked to switch the CPU to either the 'slow' or 'fast' speed.

Before calling

R0 3 (Reason code)

On return

R0 Preserved

R1 Currently configured 'slow' speed, in MHz

R2 Currently configured 'fast' speed, in MHz

Notes

Portable_Speed2 9 provides similar functionality to this call, but the speeds will be expressed as indices into the list of supported CPU speeds, not as raw MHz values.

To find out more information about the available CPU speeds, see Portable_Speed2 4 and Portable_Speed2 5. To find the current CPU speed, see Portable_Speed2 0.

To set the CPU speed, see `Portable_Speed2 2` or `Portable_Speed2 8` (to set the desired 'slow' and 'fast' speeds), and `Portable_Speed` (to switch between the configured 'slow' and 'fast' speeds).

[RISC OS Open source page](#)

Portable_Speed2 4

SWI &42FCE · Portable

The purpose of this call is to return the number of different CPU speeds that are available.

Before calling

R0 4 (Reason code)

On return

R0 Number of available CPU speeds

Notes

Due to hardware restrictions or for performance reasons, most systems will only support a limited number of CPU clock rates. This call, along with `Portable_Speed2 5` and `Portable_Speed2 6` allow the user to enumerate the list of available clock rates. Index 0 is always the slowest available speed, and `R0-1` is always the highest available speed.

To find the current CPU speed, see `Portable_Speed2 0`.

To set the CPU speed, see `Portable_Speed2 2` or `Portable_Speed2 8` (to set the desired 'slow' and 'fast' speeds), and `Portable_Speed` (to switch between the configured 'slow' and 'fast' speeds).

[RISC OS Open source page](#)

Portable_Speed2 5

SWI &42FCE · Portable

The purpose of this call is to convert a speed index to a MHz clock rate.

Before calling

R0 5 (Reason code)

R1 Speed index

On return

R0 Corresponding MHz value

Notes

Due to hardware restrictions or for performance reasons, most systems will only support a limited number of CPU clock rates. This call, along with `Portable_Speed2 4` and `Portable_Speed2 6` allow the user to enumerate the list of available clock rates. Index 0 is always the slowest available speed, and (return value of `Portable_Speed2 4`)-1 is always the highest available speed.

To set the CPU speed, see `Portable_Speed2 2` or `Portable_Speed2 8` (to set the desired 'slow' and 'fast' speeds), and `Portable_Speed` (to switch between the configured 'slow' and 'fast' speeds).

[RISC OS Open source page](#)

Portable_Speed2 6

SWI &42FCE · Portable

The purpose of this call is to convert a CPU clock rate from a MHz value to a speed index. If the input value of R1 does not match an available speed, the index of the next lowest available speed will be returned. If R1 is lower than the lowest available speed, 0 (the slowest possible speed) will be returned.

Before calling

R0 6 (Reason code)

R1 CPU speed in MHz

On return

R0 Corresponding speed index

Notes

Due to hardware restrictions or for performance reasons, most systems will only support a limited number of CPU clock rates. This call, along with Portable_Speed2 4 and Portable_Speed2 5 allow the user to enumerate the list of available clock rates. Index 0 is always the slowest available speed, and (return value of Portable_Speed2 4)-1 is always the highest available speed.

To set the CPU speed, see Portable_Speed2 2 or Portable_Speed2 8 (to set the desired 'slow' and 'fast' speeds), and Portable_Speed (to switch between the configured 'slow' and 'fast' speeds).

[RISC OS Open source page](#)

Portable_Speed2 7

SWI &42FCE · Portable

The purpose of this call is to read the current CPU speed, as an index into the list of available speeds. To instead read the CPU speed as a MHz value, use Portable_Speed2 0.

Before calling

R0 7 (Reason code)

On return

R0 Current CPU speed, as an index

Notes

Due to hardware restrictions or for performance reasons, most systems will only support a limited number of CPU clock rates. Portable_Speed2 4, Portable_Speed2 5, and Portable_Speed2 6 allow the user to enumerate the list of available clock rates. Index 0 is always the slowest available speed, and (return value of Portable_Speed2 4)-1 is always the highest available speed.

There is no direct method available to set the CPU speed to a desired value. Instead programs must use Portable_Speed2 2 (or Portable_Speed2 8) to set the desired 'slow' and 'fast' speeds, and then Portable_Speed to switch to the desired speed.

[RISC OS Open source page](#)

Portable_Speed2 8

SWI &42FCE · Portable

The purpose of this call is to allow the user to select the clock speeds that are associated with Portable_Speeds 'slow' and 'fast' settings. The values the user supplies in R1 and R2 will clamped to lie between the minimum and maximum speed indices, and the updated values, representing the new speeds that will actually be used, will be returned to the caller.

Before calling

R0 8 (Reason code)

R1 Desired 'slow' speed, as index

R2 Desired 'fast' speed, as index

On return

R0 Preserved

R1 New 'slow' speed, as index

R2 New 'fast' speed, as index

Notes

Portable_Speed2 2 provides similar functionality to this call, but the speeds are expressed as MHz values, not as indices into the list of available speeds.

Due to hardware restrictions or for performance reasons, most systems will only support a limited number of CPU clock rates. Portable_Speed2 4, Portable_Speed2 5, and Portable_Speed2 6 allow the user to enumerate the list of available clock rates. Index 0 is always the slowest available speed, and (return value of Portable_Speed2 4)-1 is always the highest available speed.

To read the currently configured 'slow' and 'fast' speeds, use Portable_Speed2 9.

To read the current CPU speed, see Portable_Speed2 7.

[RISC OS Open source page](#)

Portable_Speed2 9

SWI &42FCE · Portable

The purpose of this call is to read the currently configured 'slow' and 'fast' speeds. These are the speeds that Portable_Speed will use when asked to switch the CPU to either the 'slow' or 'fast' speed.

Before calling

R0 9 (Reason code)

On return

R0 Preserved

R1 Currently configured 'slow' speed, as index

R2 Currently configured 'fast' speed, as index

Notes

Portable_Speed2 3 provides similar functionality to this call, but the speeds will be expressed as MHz values instead of speed indices.

Due to hardware restrictions or for performance reasons, most systems will only support a limited number of CPU clock rates. Portable_Speed2 4, Portable_Speed2 5, and Portable_Speed2 6 allow the user to enumerate the list of available clock rates. Index 0 is always the slowest available speed, and (return value of Portable_Speed2 4)-1 is always the highest available speed.

To set the CPU speed, see Portable_Speed2 2 or Portable_Speed2 8 (to set the desired 'slow' and 'fast' speeds), and Portable_Speed (to switch between the configured 'slow' and 'fast' speeds).

[RISC OS Open source page](#)

Portable_Status

SWI &42FC8 · Portable

The purpose of this call is to return the status of connected devices.

Before calling

-

On return

R0 Status flags:

Bit 0: Docking station connected

Bit 1: External floppy connected

Bit 2: Internal trackball connected

Bit 3: Internal (LCD) display in use

Other bits: Reserved

Notes

This call was introduced with the Stork version of the module.

[RISC OS Open source page](#)

Portable_Stop

SWI &42FC7 · Portable

The purpose of this call is to perform what's commonly referred to as a "suspend to RAM" operation - i.e. the system is placed into a low power state of suspension, where only a handful of external events (keyboard, RTC alarm, etc.) can be used to wake the machine. The RAM is left powered, so that when the machine is woken the operating system and all software will be in the same state as it was prior to suspension.

Before calling

-

On return

R0 Restart code:

1 = Keyboard interrupt
2 = RTC alarm interrupt
Other values: Reserved

In practice

If the system cannot be placed into a state of suspension (e.g. because a module objects to the freeze request service call) and error will be returned. Otherwise, upon return from the SWI, R0 will indicate the reason for the machine to have been woken.

Notes

SWI Portable_ReadFeatures can be used to determine if this call is available.

Prior to suspending the system, the Portable module will issue the Portable service call with the freeze request reason code (5). If nothing objects to the request the module will then issue the freeze service call (reason code 3), to indicate that the freeze is taking place. Upon return from the low power state the unfreeze service call (reason code 4) will be issued in order to allow software to recover (e.g. to adapt to the sudden jump in RTC time).

[RISC OS Open source page](#)

Portable_WriteBMUVariable

SWI &42FC3 · Portable

The purpose of this call is to write the Battery Management Unit (BMU) variables.

Before calling

R0 Bits 0-23: BMU variable number
Bits 24-31: BMU number
R1 Value

On return

R0 Preserved
R1 Preserved

Notes

The BMU settings can be read, using Portable_ReadBMUVariable.

Variables other than the charge estimate (variable 6) should not be written.

[RISC OS Open source page](#)

RamFS_DescribeDisc

SWI &402785 · RamFS

The purpose of this call is to describe the Structure of the filing system disc. It uses the same entry and exit structure as FileCore_DescribeDisc SWI Call.

Before calling

See FileCore_DescribeDisc

On return

See FileCore_DescribeDisc

Notes

This call utilises the FileCore_DescribeDisc SWI Call after first passing a pointer to the instance of the RamFS in R8.

[RISC OS Open source page](#)

RamFS_DiscOp

SWI &40780 · RamFS

The purpose of this call is to provide a number of low-level disc operations. It uses the same entry and exit structure as FileCore_DiscOp SWI Call.

Before calling

See FileCore_DiscOp

On return

See FileCore_DiscOp

Notes

This call utilises the FileCore_DiscOp SWI Call after first passing a pointer to the instance of RamFS in R8.

[RISC OS Open source page](#)

RamFS_DiscOp64

SWI &40785 · RamFS

The purpose of this call is to provide a number of low-level disc operations using the more advanced Disc Operation DiscOp64.

Before calling

See FileCore_DiscOp64

On return

See FileCore_DiscOp64

Notes

This call utilises the FileCore_DiscOp64 SWI Call after first passing a pointer to the instance of RamFS in R8.

[RISC OS Open source page](#)

RamFS_Drives

SWI &40782 · RamFS

The purpose of this call is to return information on the filing system's drives.

Before calling

See FileCore_Drives

On return

See FileCore_Drives

Notes

This call utilises the FileCore_Drives SWI Call after first passing a pointer to the instance of RamFS in R8.

[RISC OS Open source page](#)

RamFS_FreeSpace

SWI &40783 · RamFS

The purpose of this call is to return the total free space on given disc, and the largest object that can be created on it.

Before calling

See FileCore_FreeSpace

On return

See FileCore_FreeSpace

Notes

This call utilises the FileCore_FreeSpace SWI Call after first passing a pointer to the instance of RamFS in R8.

[RISC OS Open source page](#)

ResourceFS_DeregisterFiles

SWI &41B41 · ResourceFS

The purpose of this call is to remove files from the ResourceFS structure before the memory containing the files is deallocated (e.g. it should be called before a module is killed).

Before calling

R0 Pointer to Resource File block

On return

R0 Corrupted

Notes

ResourceFS delinks the files and then issues `Service_ResourceFSStarted` to announce to programs that the ResourceFS structure has changed. You do not need to deregister files explicitly with this SWI upon receipt of `Service_ResourceFSDying`.
[RISC OS Open source page](#)

ResourceFS_RegisterFiles

SWI &41B40 · ResourceFS

The purpose of this call is to allow a module to add files to the ResourceFS structure.

Before calling

R0 Pointer to Resource File block

On return

R0 Corrupted

Notes

ResourceFS links the files into Resources;, and then issues `Service_ResourceFSStarting` to announce to programs that the ResourceFS structure has changed.

[RISC OS Open source page](#)

RT_ChangePriority

SWI &575C4 · RTSupport

This SWI allows the priority setting of a routine to be changed on the fly, for example if process has become more time-critical, or if it is blocking a routine of a higher priority. It may cause a thread switch if a different thread is promoted above the priority of the current thread, or if the current thread is demoted below other unblocked threads.

Before calling

R0 flags (reserved, should be zero)

R1 routine handle returned from `RT_Register`, or 0 to change the priority of the foreground process

R2 new priority (as for `RT_Register`)

On return

R0 previous priority level (as a number from 0-255)

- All other registers are preserved

In practice

Change the priority of an existing real-time routine

Interrupts

- Interrupts are disabled
- Fast interrupts are enabled

Processor Mode

- Processor is in SVC mode

Re-entrancy

- SWI is re-entrant

The priority level of the foreground process may be changed, but this is only intended for short-term use and not when the foreground is in USR mode (otherwise system callbacks will also be promoted above some real-time threads, which is undesirable behaviour).

[RISC OS Open source page](#)

RT_Deregister

SWI &575C1 · RTSupport

This SWI can be used to remove a routine from the list held by the RTSupport module. If the routine in question is currently executing then this call does not return.

Before calling

R0 flags (reserved, should be zero)

R1 routine handle returned from RT_Register

On return

- All registers are preserved

In practice

Remove a routine previously registered with the RTSupport module

Interrupts

- Interrupts are disabled
- Fast interrupts are enabled

Processor Mode

- Processor is in SVC mode

Re-entrancy

- SWI is re-entrant

[RISC OS Open source page](#)

RT_ReadInfo

SWI &575C5 · RTSupport

This SWI enables you to read current state variables. Reason codes currently defined are:

Before calling

R0 reason code

On return

R0 value

- All other registers are preserved
- V flag is set if the reason code was not recognised

In practice

Read status information

Interrupts

- Interrupt state is unchanged
- Fast interrupts are enabled

Processor Mode

- Processor is in SVC mode

Re-entrancy

- SWI is re-entrant

Reason / Meaning

0 – Meaning: RTSupport handle for current routine, or

0 if in foreground, or

-1 if in interrupt context

1 – Meaning: Priority setting (integer value) of current routine, or -1 if in interrupt context

2 – Meaning: Base address of SVC stack for real-time routines

[RISC OS Open source page](#)

RT_Register

SWI &575C0 · RTSupport

This SWI adds a new routine to the list of routines to potentially be executed each time the interrupt dispatcher is unthreaded (but before transient and non-transient callbacks are executed, if the dispatcher is returning to USR mode). Each routine remains registered until removed by a call to `RT_Deregister`, but it may sleep for a time by use of a pollword. It is recommended that if it is known that a routine will sleep for an extended time, that a `Deregister/Register` pair should be used instead, to reduce cache thrashing during interrupt handling (when the RTSupport module has to check all possible pollwords).

Before calling

R0 flags (reserved, should be zero)

R1 pointer to default entry point for routine

R2 handle to pass in R0 to routine

R3 handle to pass in R12 to routine

R4 default pointer to pollword for routine

R5 initial value of R10 for routine

R6 initial value of R13_sys for routine

R7 initial priority for routine: integer between 1-255, or a pointer to a string

R0 contents of R2 when registered

R10 first time, this is contents of R5 when registered; subsequently, it is preserved from the previous call

R11 0 (to support C code)

R12 contents of R3 when registered

R13_sys first time, this is contents of R6 when registered; subsequently, it is preserved from the previous call

R13_svc first time, this is the top of an 8K stack based at a megabyte boundary (the same logical address is used for all real-time routine SVC stacks, but it differs from the SVC stack address used by the foreground process); subsequently, it is preserved from the previous call

R14_sys return address (only needed if using the callback model)

SYS mode

First time, IRQs and FIQs enabled;

subsequently IRQ disable state is preserved

Other registers undefined

Static relocation offsets at base of SVC stack are undefined

On return

R0 RTSupport module's handle for this routine

- All other registers are preserved

R0 flags:

bit 0 set => rescan all higher priority routines (for example, if this routine unblocked a semaphore that a higher priority routine may be sleeping on) otherwise rescan all equal priority routines, including this routine, before descending

bit 1 set => R1 contains new pollword pointer (otherwise the pollword given at registration is used)

bit 2 set => R2 contains a monotonic time after which control should be returned even if the pollword is not set

bit 3 set => R14_sys contains the address to enter next time (otherwise the address used at registration is re-entered)

other bits are reserved and must be zero

R1 pollword pointer (if R0 bit 1 is set)

R2 timeout monotonic time (if R0 bit 2 is set)

R10 value to use at next entry

R13_sys value to use at next entry

R13_svc value to use at next entry

R14_sys next entry point (if R0 bit 3 is set)

SYS mode, IRQs enabled or disabled, FIQs enabled

SVC stack may be non-empty

Processor flags, R1-R9 (except where used to return parameters), R11, R12 and R14_svc may be corrupted

In practice

Register a routine with the RTSupport module

Interrupts

- Interrupts are disabled
- Fast interrupts are enabled

Processor Mode

- Processor is in SVC mode

Re-entrancy

- SWI is re-entrant

Each time the RTSupport module gains control of the CPU, it scans its list of routines in decreasing priority order for routines that are unblocked (ie where the pollword is non-zero). If a real-time routine was interrupted, then routines of a higher priority are executed, followed by the interrupted routine, and then equal and lower priority routines are examined.

(However, note that if and only if routines have their priorities changed dynamically, it is possible that while a routine is executing, a second routine at the same priority level may be in a pre-empted state.) If the RTSupport module encounters two unblocked routines at the same priority level, then the calling order is undefined.

The routines are called as follows:

So for example the routine may be implemented as

- Assembler, with the static data referenced using R12 as is conventional in RISC OS.
- C code interfaced directly, as long as R10 and R13_sys are set up to describe a standard stack chunk with the reserved words at its base set up to enable static data relocation, for example by copying them from the base of the SVC stack prior to the RT_Register call:

```
_kernel_stack_chunk *chunk = calloc(1, CHUNK_SIZE);
chunk->sc_mark = 0xF60690FF;
chunk->sc_size = CHUNK_SIZE;
memcpy(chunk + 1, _kernel_current_stack_chunk() + 1, 28);
_kernel_swi_regs r = { /* ... */ .r[5] = ((int) chunk) + 560,
.r[6] = ((int) chunk) + CHUNK_SIZE, /* ... */ };
_kernel_swi(RT_Register, &r, &r);
```

Here is an example callback-model routine written in C that repeatedly sleeps for a second at a time (assuming the default pollword is never set):

```
typedef struct {
    unsigned int flags;
    unsigned int *pollword;
    unsigned int timeout;
} routine_result_t;
__value_in_regs routine_result_t MyRoutine(void *r0) {
    unsigned int time;
    _swix(OS_ReadMonotonicTime, _OUT(0), &time);
    return (routine_result_t) { 1 << 2, NULL, time + 100 };
}
```

- C code interfaced via a veneer: if you want to be able to call _kernel_raise_error() and functions which rely upon it like exit(), abort() and assert(), which assume the outermost stack frame was called as a function returning a _kernel_oserror *, or for use as a threading library where actions need to be taken when the thread exits (eg unblocking joining threads) you will need an assembler veneer to the C code, to patch the r14_sys the C code sees on entry with a pointer to a suitable cleanup function.

The priority passed in R7 can be an integer from 1 (lowest priority) to 255 (highest priority). Priority 0 is reserved for the exclusive use of the foreground process. However, it is preferred that a string pointer be used instead: the string is used to look up a priority in the file RTSupport:Priorities. This enables easy tuning of priority levels for a given system, without the need for each real-time component to be individually configurable. For example, the file might be as follows:

```
Critical:224
AudioFill:192
VideoPaint:160
Normal:128
AudioDecode:96
VideoDecode:64
Coroutine:32
```

Notes

If the routines pollword is nonzero at the time the routine is registered, and the routine is at a higher priority than the current context, then RT_Register will attempt to yield to the routine straight away. This also opens up the possibility for the routine to deregister itself before the RT_Register call returns.

[RISC OS Open source page](#)

RT_TimedYield

SWI &575C3 · RTSupport

This SWI is similar to RT_Yield except that the sleep does not extend indefinitely.

Before calling

R1 pointer to pollword

R2 monotonic time at which to return control even if pollword is unset

On return

– All registers are preserved

In practice

Perform a time-limited sleep of the current thread and force a test of all pollwords

Interrupts

- Interrupts are disabled
- Fast interrupts are enabled

Processor Mode

- Processor is in SVC mode

Re-entrancy

- SWI is re-entrant

[RISC OS Open source page](#)

RT_Yield

SWI &575C2 · RTSupport

This SWI has two uses: firstly it enables code in real-time context to sleep, and secondly it enables the foreground process to force a test of all pollwords (for example if the foreground sets the pollword of a thread, this may be used to wake the thread rather than having to wait for the next interrupt). The pollword is usually insignificant for the foreground process, because by definition control returns to it when no pollwords are set; however, it may be useful if the foreground is currently set to a priority above that of a real-time routine.

Before calling

R1 pointer to pollword

On return

– All registers are preserved

In practice

Sleep the current thread and force a test of all pollwords

Interrupts

- Interrupts are disabled
-

- Fast interrupts are enabled

Processor Mode

- Processor is in SVC mode

Re-entrancy

- SWI is re-entrant

It is an error to make this call from interrupt context.

While in real-time context, OS_UpCall 6 is interpreted as a call to this SWI. (OS_UpCall 7 is also intercepted if an attempt is made to remove a pollword being used by the real-time system.)

[RISC OS Open source page](#)

RTC_Adjust

SWI &594C3 · RTC

The purpose of this call is to request a correction to be applied to the RTC.

Before calling

R0 Time error low word

R1 Time error high word

R2 Time span to apply correction over (centiseconds)

R3 Priority

On return

- All registers preserved

In practice

R0 and R1 form a signed 64 bit integer giving the correction value to be applied (in units of centiseconds). R2 specifies the length of time to apply the correction over.

R3 is used to specify a priority value, with higher values indicating a higher priority.

Notes

The RTC module will only apply one correction at a time. If it is already performing a correction, and a new request is received at a lower priority level than the current correction, then the request will be rejected and an error will be returned.

The automatic synchronisation of the soft clock with the RTC is performed using a priority level of &1000000 – using a priority level lower than this is guaranteed to fail and therefore pointless. NetTime? uses a priority level of &2000000.

An error will also be returned if an attempt is made to exceed the limits of the 5 byte time format that RISC OS uses internally.

If a request is made to perform a small time correction, the module will skew the soft clock by adjusting the rate of the hardware timer that drives the 100Hz TickerV interrupt. The timer will remain close enough to its original frequency that software is unlikely to be able to determine that anything unusual is happening. Note that current versions of the module will not adjust the hardware RTC in this case.

If a request is made to perform a large time correction, the module will avoid skewing the hardware timer and will instead apply the time correction immediately, via the Territory_SetTime SWI. This will result in all software seeing a visible jump in the current time, with both the soft clock and hardware RTC updated to the new time.

[RISC OS Open source page](#)

RTC_Features

SWI &594C0 · RTC

The purpose of this call is to read the available features of the RTC module.

Before calling

- None

On return

R0 Feature flags

In practice

This SWI can also be used as a simple test for whether the module is available.

Notes

Currently no features are defined, and so R0 will be zero on exit.

[RISC OS Open source page](#)

RTC_Read

SWI &594C1 · RTC

The purpose of this call is to read the current time from the hardware RTC

Before calling

R0 Pointer to rctime structure

On return

- On success, rctime struct updated

R0 Error pointer (on failure), preserved (on success)

- All other registers preserved

Notes

An error will be returned if the read failed, or if the RTC is known to be in an invalid state (e.g. clock stopped)

To provide a full 4-digit year, the value read from the RTC may be augmented by the year stored in CMOS.

Rather than reading time from the RTC directly, most software should read the softcopy of the time that's maintained by RISC OS, e.g. via OS_Word 14

[RISC OS Open source page](#)

RTC_Write

SWI &594C2 · RTC

The purpose of this call is to write a time value to the hardware RTC

Before calling

R0 Pointer to rtctime structure

On return

R0 Error pointer (on failure), preserved (on success)

- All other registers preserved

Notes

By initialising certain members of the rtctime struct to 255, it is possible to do a partial update of the date/time. The supported combinations are:

- Full time update - all the rtctime members are valid and are expected to be written to the RTC
- Time update - The DayOfMonth, Month, YearLO and YearHI members will be set to 255. Only the time portion of the RTC will be updated.
- Date update - The Centiseconds, Seconds, Minutes and Hours members will be set to 255. Only the date portion of the RTC will be updated.

Other combinations are invalid and should not be used.

The year bytes held in CMOS may be updated by this operation.

If the RTC was previously stopped, this call will start it running.

Rather than writing time to the RTC directly, most software should use a SWI such as Territory_SetTime to update both the RTC and the soft copy.

[RISC OS Open source page](#)

ScreenBlanker_Control

SWI &43100 · ScreenBlanker

The purpose of this call is to control screen blanking.

Before calling

R0 Reason code

All other registers are dependent on reason code

On return

Registers dependent on reason code

Notes

None.

Reason Codes

0 – Hex #: &00 · Action: Blank screen immediately

1 – Hex #: &01 · Action: Unblank screen immediately

- 2 – Hex #: &02 · Action: Flash screen
- 3 – Hex #: &03 · Action: Set blanking delay
- 4 – Hex #: &04 · Action: Get blanking delay (in seconds)
- 5 – Hex #: &05 · Action: Get blanking delay (in centiseconds)
- 6 – Hex #: &06 · Action: Set dim timer
- 7 – Hex #: &07 · Action: Get dim timer

[RISC OS Open source page](#)

ScreenBlanker_Control 2

SWI &43100 · ScreenBlanker

The purpose of this call is to Flash the screen a specific number of times.

Before calling

- R0** 2 (Flash screen)
- R1** Time to turn on (in centiseconds)
- R2** Time to turn off (in centiseconds)
- R3** Maximum number of flashes
- R4** Flags (not documented yet, see Service_ScreenRestored and Service_ScreenBlanked)

On return

--

Notes

None.

[RISC OS Open source page](#)

ScreenBlanker_Control 3

SWI &43100 · ScreenBlanker

The purpose of this call is to specify the blanking delay time in centiseconds.

Before calling

- R0** 3 (set blanking delay)
- R1** Delay time (in centiseconds), or 0 to disable

On return

--

Notes

None.

[RISC OS Open source page](#)

ScreenBlanker_Control 4

SWI &43100 · ScreenBlanker

The purpose of this call is to return the blanking delay time in seconds.

Before calling

R0 4 (get blanking delay in seconds)

On return

R0 Internal blanking time

R1 Delay time (in seconds), or 0 if currently disabled

Notes

None.

[RISC OS Open source page](#)

ScreenBlanker_Control 5

SWI &43100 · ScreenBlanker

The purpose of this call is to return the blanking delay time in centiseconds.

Before calling

R0 5 (get blanking delay in centiseconds)

On return

R0 Internal blanking time

R1 Delay time (in centiseconds), or 0 if currently disabled

Notes

None.

[RISC OS Open source page](#)

ScreenBlanker_Control 6

SWI &43100 · ScreenBlanker

The purpose of this call is to specify the dim delay time in centiseconds.

Before calling

R0 6 (set dim timer)

R1 Delay time (in centiseconds), or 0 to disable

On return

--

Notes

None.

[RISC OS Open source page](#)

ScreenBlanker_Control 7

SWI &43100 · ScreenBlanker

The purpose of this call is to return the dim delay time in centiseconds.

Before calling

R0 7 (get dim timer)

On return

R0 Internal blanking time

R1 Delay time (in centiseconds), or 0 if currently disabled

Notes

None.

[RISC OS Open source page](#)

ScreenModes_EnumerateAudioFormats

SWI &487C1 · ScreenModes

This call provides a means for software to enumerate the audio formats that are supported by the connected monitor.

Before calling

R0 Flags:

Bit 0: 0 => Return data in 'raw' form

1 => Return data in 'friendly' form

Bits 1+: Reserved, should be zero

R1 Format code to start from (-1 for first)

R2 Enumeration token for format code (-1 to start at first entry)

On return

R0 Preserved

R1 Format code of this entry (-1 if no more entries)

R2 Enumeration token for this entry (-1 if no more entries)

R3-R5 Dependent on flags

All other registers preserved

R3 Max channels

R4 Audio short descriptor byte 2

R5 Audio short descriptor byte 3

R3 Max channels

R4 Sample rate, in units of 1/1024Hz

R5 For format code 1 (LPCM): Sample bit depth

For format codes 2-8: Max bit rate in Hz

In practice

For 'raw' results, R3-R5 are as follows:

For 'friendly' results, R3-R5 are as follows:

Enumeration can start at the first supported format (by specifying R1 and R2 as -1) or at a specific format (Set R1 to the required format code, and R2 to -1). On return R1 and R2 will indicate the validity of the data; for valid data they will both not be -1, and for invalid data they will both be -1. Only if valid data is indicated will R3-R5 be updated.

The format code specified in R1 corresponds to the format codes defined by CEA 861, e.g. basic LPCM audio is format code 1.

The audio formats can be enumerated in two modes: 'raw' mode and 'friendly' mode. 'raw' mode returns bytes 2 and 3 of the CEA short audio descriptor directly (byte 1 will have already been decoded to determine the values of R1 and R3). 'friendly' mode interprets the bytes and converts them to standard units – however this is currently only supported for format codes 1 through 8.

Notes

The results of the enumeration will be ordered by their format code, so if you are only interested in one format code you can simply begin enumeration with R1 set to that value and end enumeration when R1 deviates from that value.

Enumeration tokens used with 'friendly' mode are not compatible with enumeration tokens used with 'raw' mode.

For monitors with complex EDID, it is possible that the same format will be reported twice when enumerating in 'friendly' mode.

When dealing with multi-channel audio, ScreenModes_ReadInfo 2 can be used to determine which speakers are present on the monitor.

[RISC OS Open source page](#)

ScreenModes_Features

SWI &487C2 · ScreenModes

This call returns information on which features are supported by the ScreenModes module.

On return

R0 Features mask:

Bit 0: ScreenModes supports EDID

Bits 1+: Reserved (zero)

EDID support

If bit 0 is set then it indicates that ScreenModes supports EDID. This includes:

- The ability to load EDID files using *LoadModeFile
- Automatic export of EDID files to Resources:\$.Resources.ScreenMode.Monitors.EDID<n> on driver initialisation (where <n> is the GraphicsV driver number)
- Automatic use of EDID-derived modes on boot when the configured MonitorType is set to EDID

However, this does not guarantee that any particular driver / monitor supports the reading of EDID.

[RISC OS Open source page](#)

ScreenModes_ReadInfo

SWI &487C0 · ScreenModes

This call allows miscellaneous information to be read.

Before calling

R0 Reason code

On return

Dependent on reason code

Notes

The action performed depends on the reason code value in R0.

Reason Codes

/ Hex # / Action

0 – Hex #: &00 · Action: Return monitor name

1 – Hex #: &01 · Action: Return supported DPMS state

2 – Hex #: &02 · Action: Return speaker mask

[RISC OS Open source page](#)

ScreenModes_ReadInfo 0

SWI &487C0 · ScreenModes

This call allows the monitor name/model to be read, e.g. for display to the user.

Before calling

R0 0

On return

R0 Pointer to null-terminated monitor name string

All other registers preserved

Notes

The string pointer returned in R0 will be invalidated the next time a mode file is loaded, e.g. the next time Service_ModeFileChanged is issued.

[RISC OS Open source page](#)

ScreenModes_ReadInfo 1

SWI &487C0 · ScreenModes

This call provides a means for software to determine which DPMS state will be used when the screen is blanked. This will either be the DPMS state that was specified in the MDF, or the DPMS state that was extracted from the EDID.

Before calling

R0 1

On return

R0 Supported DPMS State

All other registers preserved

Notes

This information is of an informative nature; the preferred way for software to blank and unblank the screen is via the ScreenBlanker SWIs, which do not require the DPMS state to be specified (the kernel, which is ultimately in charge of calling GraphicsV 4, will remember the DPMS state that was specified in the VIDC control list that was used during the last mode change).

[RISC OS Open source page](#)

ScreenModes_ReadInfo 2

SWI &487C0 · ScreenModes

This call provides a means for software to determine which speakers may be available on a given display. Normally this information is determined by reading the displays EDID.

Before calling

R0 2

On return

R0 Speaker mask

R1 Validity mask

All other registers preserved

In practice

Because the speaker mask is an optional part of the EDID, two values are returned: the speaker mask as reported from the monitor (in R0), and a validity mask (in R1) which indicates which bits of R0 have been specified. The following table can be used as a guide to determine whether a given speaker is available or not:

Speaker mask bit value / Validity mask bit value / Interpretation

0 – Validity mask bit value: 0 · Interpretation: Monitor did not provide information – speaker may or may not be present

0 – Validity mask bit value: 1 · Interpretation: Speaker is not present

1 – Validity mask bit value: 0 · Interpretation: Reserved

1 – Validity mask bit value: 1 · Interpretation: Speaker is present

Notes

Bit allocation in the two words follows the same scheme as the CEA 861 speaker allocation data block:

0 – Interpretation: Front Left & Front Right

1 – Interpretation: LFE

2 – Interpretation: Front Center

3 – Interpretation: Rear Left & Rear Right

4 – Interpretation: Rear Center

5 – Interpretation: Front Left Center & Front Right Center

6 – Interpretation: Rear Left Center & Rear Right Center

7+ – Interpretation: Reserved

[RISC OS Open source page](#)

SDIO_ClaimDeviceVector

SWI &59006 · SDIODriver

This SWI is used to claim a Device Vector of a SDIO card.

Before calling

R0 Bits 0-7 = unit number

Bits 8-15 = slot number

Bits 16-23 = bus number

R1 Address of interrupt handler routine

R2 Value to pass in R12 to routine

On return

--

Re-entrancy

This SWI is re-entrant.

Notes

To simplify SDIO interrupt dispatch, device drivers should use this instead of calling OS_ClaimDeviceVector. Since SDIO cards are only supported on buses with one card per slot, there is no need to specify an RCA to identify the card.

[RISC OS Open source page](#)

SDIO_Control

SWI &59001 · SDIODriver

This SWI is used to control the SD Card. The exact use is specified by the Reason Code passed in R0.

Before calling

R0 Reason code

R1 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

On return

--

Reason Codes

/ Reason Code

- 0 – Reason Code: Acquire exclusive access to slot, non-blocking
- 1 – Reason Code: Acquire exclusive access to slot, blocking
- 2 – Reason Code: Acquire exclusive access to slot, blocking with sleep
- 3 – Reason Code: Release exclusive access to slot
- 4 – Reason Code: Abort all operations on slot
- 5 – Reason Code: Abort operation
- 6 – Reason Code: Poll card detection

Re-entrancy

Dependent on Reason Code used.

Notes

None.

[RISC OS Open source page](#)

SDIO_Control 0

SWI &59001 · SDIODriver

This SWI is used to acquire exclusive access to slot, non-blocking.

Before calling

R0 0 (reason code)

R1 Bits 0 – 7 = slot number

Bits 8 – 15 = bus number

On return

Error &81EE06 ("Slot in use") is returned if exclusive access cannot be acquired at the present time

Re-entrancy

This SWI is re-entrant.

Notes

This call is one of a group that must be used around every call to SWI SDIO_Op. The reason for this is that many SD commands need to be issued in groups of two or more. If another SD command is queued in the middle of such a group, it may find that the device is not in a receptive state for that command, and the operation of the command group that was interrupted may also be disrupted.

This version of the acquire-slot SWI is suitable for use from the background, but the caller must be prepared for it to fail and reschedule the command at a later time.

[RISC OS Open source page](#)

SDIO_Control 1

SWI &59001 · SDIODriver

This SWI is used to acquire exclusive access to slot, blocking.

Before calling

R0 1 (reason code)

R1 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

On return

--

Re-entrancy

This SWI is not re-entrant.

Notes

This call is similar to SDIO_Control 0, but waits (forever if necessary) for the slot to become available.

[RISC OS Open source page](#)

SDIO_Control 2

SWI &59001 · SDIODriver

This SWI is used to acquire exclusive access to slot, blocking with sleep.

Before calling

R0 2 (reason code)

R1 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

On return

--

Re-entrancy

This SWI is not re-entrant.

Notes

This call is similar to SDIO_Control 1, but calls OS_UpCall 6 while the slot is not available, in order to yield the current application.

[RISC OS Open source page](#)

SDIO_Control 3

SWI &59001 · SDIODriver

This SWI is used to release exclusive access to slot.

Before calling

R0 3 (reason code)

R1 Bits 0 – 7 = slot number

Bits 8 – 15 = bus number

On return

--

Re-entrancy

This SWI is re-entrant.

Notes

This call makes the specified slot available for others to use. It should be used routinely as soon as a group of related SD commands has been issued, to avoid locking other routines out.

[RISC OS Open source page](#)

SDIO_Control 4

SWI &59001 · SDIODriver

This SWI is used to abort all operations on slot.

Before calling

R0 4 (reason code)

R1 Bits 0 – 7 = slot number

Bits 8 – 15 = bus number

On return

--

Re-entrancy

This SWI is re-entrant.

Notes

This call cancels all outstanding operations on a slot. You should abort individual operations instead wherever possible. You should acquire exclusive access to the slot before making this call (otherwise the owner of the slot may queue new operations during this call which won't get aborted).

[RISC OS Open source page](#)

SDIO_Control 5

SWI &59001 · SDIODriver

This SWI is used to abort operation with the specified SD Card.

Before calling

R0 5 (reason code)

R1 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

R2 Operation ID as returned by SDIO_Op for background operations

On return

--

Re-entrancy

This SWI is re-entrant.

Notes

This call cancels the specified background operation.

[RISC OS Open source page](#)

SDIO_Control 6

SWI &59001 · SDIODriver

This SWI is used to poll card detection.

Before calling

R0 6 (reason code)

R1 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

On return

--

Re-entrancy

This SWI is re-entrant.

Notes

With hardware that has a functional card detect line, SDIODriver polls the hardware at regular intervals to determine whether a card is inserted. However, it is possible that if you have an operation ongoing when a card is removed that the operation will return an error before SDIODriver does its polling and has chance to issue Service_SDIOUnitDetached. Therefore this SWI gives you a mechanism to force an extra poll (and therefore trigger any outstanding service calls) so you can determine whether the error was caused by the card having been removed.

[RISC OS Open source page](#)

SDIO_ControllerFeatures

SWI &59003 · SDIODriver

This SWI is used to enable the caller to discover features of the SD controller.

Before calling

R0 Feature number:

Index meaning

0 = log2 of maximum block length supported for data transfers

R1 Bits 0-7 = slot number

Bits 8-15 = bus number

On return

R2 Value corresponding to the requested feature

Re-entrancy

This SWI is re-entrant.

Notes

An error is generated for unknown feature index values.

[RISC OS Open source page](#)

SDIO_Enumerate

SWI &59002 · SDIODriver

This SWI is used to enumerate SD card slots or units attached to the system.

Before calling

R0 Reason code

On return

R0 Preserved

Reason Codes

/ Reason Code

0 – Reason Code: Enumerate slots

1 – Reason Code: Enumerate units

Re-entrancy

Dependent on Reason Code used.

Notes

None.

[RISC OS Open source page](#)

SDIO_Enumerate 0

SWI &59002 · SDIODriver

This SWI is used to enable the caller to discover which slots are available.

Before calling

R0 0 (reason code: enumerate slots)

R1 0 on first call, or preserved on subsequent calls

On return

R0 Preserved

R1 0 if no more slots (R2 is not valid in this case), else the value to pass in R1 to next call

R2 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

Bit 16 set => slot contains only non-removable IO device(s)

Bit 17 set => slot is removable but has non-functional card detect line

Bit 18 set => slot contains non-removable memory devices(s)

Re-entrancy

This SWI is re-entrant.

Notes

None.

[RISC OS Open source page](#)

SDIO_Enumerate 1

SWI &59002 · SDIODriver

This SWI is used to enable the caller to discover the memory cards and IO functions available in a given slot.

Before calling

R0 1 (reason code: enumerate units)

R1 0 on first call, or preserved on subsequent calls

R2 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

Bits 16 - 31 ignored

On return

R0 Preserved

R1 0 if no more units (R2 and R3 are not valid in this case), else the value to pass in R1 to next call

R2 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

Bits 16 - 31 = relative card address (RCA)

R3 Bits 0 - 7 = unit number

Bits 8 - 15 = function interface code (FIC) for IO, or 0 for memory(1 = UART, 2 = Bluetooth Type-A etc.)

Bit 16 set => device is write-protected

Bit 17 set => MMC (rather than SD/SDIO) device

Bits 18 - 31 reserved, 0

R4 Low word of card capacity (for memory cards)

R5 High word of card capacity (for memory cards)

Re-entrancy

This SWI is not re-entrant.

Notes

None.

[RISC OS Open source page](#)

SDIO_Initialise

SWI &59000 · SDIODriver

This SWI is used to initialise the SD card device.

Before calling

R0 Reason code

On return

--

Reason Codes

/ Reason Code

0 – Reason Code: Reset bus

1 – Reason Code: Rescan slot

Re-entrancy

Dependent on Reason Code used.

Notes

None.

[RISC OS Open source page](#)

SDIO_Initialise 0

SWI &59000 · SDIODriver

This SWI is used to reset the specified host controller and rescans its slots.

Before calling

R0 0 (reason code: reset bus)

R1 Bits 0 – 7 = bus number

On return

--

Re-entrancy

This SWI is not re-entrant.

Notes

None.

[RISC OS Open source page](#)

SDIO_Initialise 1

SWI &59000 · SDIODriver

This SWI is used to rescan the devices in a given slot and calculates the new optimal electrical configuration for the slot.

Before calling

R0 1 (reason code: rescan slot)

R1 Bits 0 – 7 = bus number

Bits 8 – 15 = bus number

On return

--

Re-entrancy

This SWI is not re-entrant.

Notes

This call is required after locking or unlocking a card.

[RISC OS Open source page](#)

SDIO_Op

SWI &59005 · SDIODriver

This SWI is used to perform low-level operations.

Before calling

R0 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

Bit 16 set => check busy after response (use for R1B)

Bit 17 set => command CRC check enabled (use for R1, R1B, R2, R4 (MMC), R5 (MMC), R5 (SDIO), R6, R7)

Bit 18 set => command index check enabled (use for R1, R1B, R4 (MMC), R5 (MMC), R5 (SDIO), R6, R7)

Bit 19 set => test response word for R1-style errors (use for R1, R1B)

Bit 20 set => test response word for R5-style errors (use for R5 (SDIO))

Bits 24 - 25 = transfer direction (0 = no data transfer, 1 = read, 2 = write)

Bit 26 set => scatter transfer

Bit 27 set => don't poll for escape during foreground transfers

Bit 28 reserved - bits 26-29 are equivalent to bits 5-8 for FileCore DiscOps

Bit 29 set => background transfer

R1 Length of command and response block, must be 8, 12 or 23

R2 Pointer to command and response block

R3 RAM ptr for start of transfer or pointer to scatter list of address length pairs if bit 26 of R0 is set

R4 Length of transfer (in bytes) - must be 0 if no data transfer is specified in R0 bits 24-25, and must not be 0 otherwise

R5 If bit 29 of R0 is clear, this is timeout in centiseconds or 0 for no timeout

If bit 29 of R0 is set, this is the value to pass in R5 to the callback routine

R6 Address of callback routine, if bit 26 of R0 is set

R7 Value to pass in R12 to the callback routine, if bit 26 of R0 is set

On return

If error was detected,

R0 Pointer to error block

R4 Updated to number of bytes not transferred

V set

If foreground transfer completed successfully,

R0 0 (consider as a flags word with all bits reserved)

R4 0

V clear

If background transfer has started,

R0 0 (consider as a flags word with all bits reserved)

R1 operation handle, can be used to cancel the operation using SDIO_Control

V clear

Command and Response Block

0 – Meaning: Command index

1 – Meaning: Flags:

Bit 0 set => this is an application command (i.e. precede this command with a CMD55)

Bit 1 set => issue automatic CMD12 to terminate multiple-block data transfer command

Bits 4 - 7 = log2 of the data transfer block split size (block length used is the lesser of R4 and this number)

Other bits reserved, should be 0

2 – Meaning: RCA (relative card address) for application commands, otherwise 0

4 – Meaning: Bits 8 - 39 of the command argument

8 – Meaning: Buffer space filled in on exit with the response bytes (if any) excluding the index and CRC bytes

Re-entrancy

This SWI is reentrant when used to specify a background transfer, otherwise not reentrant.

Notes

The scatter list is not updated (in other words, it behaves more like the low-level SCSI Switch API than the backward-compatible SCSI Driver API). FileCore-style wrapping is accepted (ie if an address entry is between &FFFF0000 and &FFFFFFFF, and the length field is 0, then that should be treated as an instruction to push the scatter list pointer backwards by an offset of up to 64K).

The callback operates as follows:

- The call is made in privileged mode. IRQs may be enabled or disabled
- If the operation completed, V is clear & R0 is set to 0
- If an error occurred, V is set and R0 is an error pointer
- R4 holds the amount of data not transferred
- R5 and R12 are set to the values passed in
- All registers will be preserved

For most commands, the space required at R2+8 will be 4 bytes, but for R2 it will be 15 bytes, and a few commands (such as GO_IDLE_STATE) have no response at all.

[RISC OS Open source page](#)

SDIO_ReadRegister

SWI &59004 · SDIODriver

This SWI is used to read and cache certain registers as part of the process of scanning each slot.

Before calling

R0 register number:

Index meaning

0 = wired AND of memory OCR register(s) for slot (4 bytes)

1 = IO OCR register (4 bytes)

2 = CID register (15 bytes excluding CRC byte)

3 = CSD register (15 bytes excluding CRC byte)

4 = SCR register (8 bytes)

R1 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

Bits 16 - 31 RCA

R2 Pointer to buffer (assumed large enough for the register selected)

On return

R0 Preserved

R1 Preserved

R2 Preserved

Re-entrancy

This SWI is not re-entrant.

Notes

The buffer is filled in with register contents, stored in little-endian order.

This SWI makes available for the convenience of other modules. An error is generated for unknown register index values, or if the specified card does not provide the specified register.

[RISC OS Open source page](#)

SDIO_ReleaseDeviceVector

SWI &59007 · SDIODriver

This SWI is used to release a Device Vector of a SDIO card.

Before calling

R0 Bits 0-7 = unit number

Bits 8-15 = slot number

Bits 16-23 = bus number

R1 Address of interrupt handler routine

R2 Value to pass in R12 to routine

On return

--

Re-entrancy

This SWI is re-entrant.

Notes

None.

[RISC OS Open source page](#)

SharedCLibrary_LibInitAPCS_32

SWI &80683 · SharedCLibrary

On entry

R0 – Contents: Pointer to list of stub descriptions each having the following format:

Offset · Contents

+0 · Library chunk id (1 or 2)

+4 · Entry vector base

+8 · Entry vector limit

+12 · Static data base

+16 · Static data limit

R1 – Contents: Pointer to workspace start
R2 – Contents: Pointer to workspace limit
R3 – Contents: -1
R4 – Contents: 0
R5 – Contents: -1
R6 – Contents: Bits 0-15=0, Bits 16-31=Root stack size in KB

In practice

On exit

Entry vectors specified by the stubs descriptions are patched to contain branches to routines in the library.

If $R5 > R4$ on entry the users statics are copied to the bottom of the workspace specified in R1 and the Client static data offset (at byte offset +24 from the stack base) is initialised.

For each library chunk the library statics are copied either into the workspace specified in R1 if $R5 > R4$ on entry or to the static data area specified in the chunks stub description if $R5 \leq R4$.

The Library static data offset (at byte offset +20 from the stack base) is initialised.

Space for the root stack chunk is claimed from the workspace specified in R1.

Notes

The stub list pointed to in R0 is terminated by an entry with a library chunk id of -1

Register conventions

APCS-32 · R0 – APCS: a1 · Contents: Argument 1

R1 · a2 · Argument 2

R2 · a3 · Argument 3

R3 · a4 · Argument 4

R4 · v1 · Register variable 1

R5 · v2 · Register variable 2

R6 · v3 · Register variable 3

R7 · v4 · Register variable 4

R8 · v5 · Register variable 5

R9 · v6 · Register variable 6

R10 · sl · Stack Limit / Stack Chunk Handle

R11 · fp · Frame pointer

R12 · ip · Used as temporary workspace

R13 · sp · Stack pointer

R14 · lr · Link register

R15 · pc · Program counter

Library kernel functions

Chunk 1 is 192 bytes containing 48 entries Statics are 800 bytes

C Library Functions

Chunk 2 is 732 bytes containing 183 entries Statics are 2888 bytes

[RISC OS Open source page](#)

SharedCLibrary_LibInitAPCS_A

SWI &80680 · SharedCLibrary

On entry

R0 – Contents: Pointer to list of stub descriptions each having the following format:

Offset · Contents

+0 · Library chunk id (1 or 2)

+4 · Entry vector base

+8 · Entry vector limit

+12 · Static data base

+16 · Static data limit

R1 – Contents: Pointer to workspace start

R2 – Contents: Pointer to workspace limit

R3 – Contents: -1

R4 – Contents: 0

R5 – Contents: -1

In practice

On exit

Entry vectors specified by the stubs descriptions are patched to contain branches to routines in the library.

If $R5 > R4$ on entry the users statics are copied to the bottom of the workspace specified in R1 and the Client static data offset (at byte offset +24 from the stack base) is initialised.

For each library chunk the library statics are copied either into the workspace specified in R1 if $R5 > R4$ on entry or to the static data area specified in the chunks stub description if $R5 \leq R4$.

The Library static data offset (at byte offset +20 from the stack base) is initialised.

Space for the root stack chunk is claimed from the workspace specified in R1.

Notes

This is legacy from Arthur and should not be used. It's documented here for historic reasons.

The stub list pointed to in R0 is terminated by an entry with a library chunk id of -1

Register conventions

APCS-Arthur · R0 – APCS: a1 · Contents: Argument 1

R1 · a2 · Argument 2

R2 · a3 · Argument 3

R3 · a4 · Argument 4

R4 · v1 · Register variable 1

R5 · v2 · Register variable 2

R6 · v3 · Register variable 3

R7 · v4 · Register variable 4

R8 · v5 · Register variable 5

R9 · v6 · Register variable 6

R10 · fp · Frame pointer

R11 · ip · Used as temporary workspace

R12 · sp · Lower end of current stack frame

R13 · sl · Stack limit / Stack chunk handle

R14 · lr · Link register

R15 · pc · Program counter

Library kernel functions

Chunk 1 is 152 bytes containing 38 entries (the last being `_kernel_fpavailable`) Statics are 800 bytes (matching APCS-R)

C Library Functions

Chunk 2 is 672 bytes containing 168 entries (the last being `__assert`) Statics are 2888 bytes (matching APCS-R)

[RISC OS Open source page](#)

SharedCLibrary_LibInitAPCS_R

SWI &80681 · SharedCLibrary

On entry

R0 – Contents: Pointer to list of stub descriptions each having the following format:

Offset · Contents

+0 · Library chunk id (1 or 2)

+4 · Entry vector base

+8 · Entry vector limit

+12 · Static data base

+16 · Static data limit

R1 – Contents: Pointer to workspace start

R2 – Contents: Pointer to workspace limit

R3 – Contents: -1

R4 – Contents: 0

R5 – Contents: -1

R6 – Contents: Bits 0-15=0, Bits 16-31=Root stack size in KB

In practice

On exit

Entry vectors specified by the stubs descriptions are patched to contain branches to routines in the library.

If $R5 > R4$ on entry the users statics are copied to the bottom of the workspace specified in R1 and the Client static data offset (at byte offset +24 from the stack base) is initialised.

For each library chunk the library statics are copied either into the workspace specified in R1 if $R5 > R4$ on entry or to the static data area specified in the chunks stub description if $R5 \leq R4$.

The Library static data offset (at byte offset +20 from the stack base) is initialised.

Space for the root stack chunk is claimed from the workspace specified in R1.

Notes

The stub list pointed to in R0 is terminated by an entry with a library chunk id of -1

Register conventions

APCS-RISCOS · R0 – APCS: a1 · Contents: Argument 1

R1 · a2 · Argument 2

R2 · a3 · Argument 3

R3 · a4 · Argument 4

R4 · v1 · Register variable 1

R5 · v2 · Register variable 2

R6 · v3 · Register variable 3

R7 · v4 · Register variable 4

R8 · v5 · Register variable 5
R9 · v6 · Register variable 6
R10 · sl · Stack Limit / Stack Chunk Handle
R11 · fp · Frame pointer
R12 · ip · Used as temporary workspace
R13 · sp · Stack pointer
R14 · lr · Link register
R15 · pc · Program counter

Library kernel functions

Chunk 1 is 192 bytes containing 48 entries Statics are 800 bytes

C Library Functions

Chunk 2 is 732 bytes containing 183 entries Statics are 2888 bytes

[RISC OS Open source page](#)

SharedSound_HandlerInfo

SWI &4B442 · SharedSound

Will return information on an installed sound handler.

Before calling

R0 Handler number or 0 to get number of first handler

On return

R0 Number of next available handler or 0 for none

R1-R5 Corrupted

R0 Number of next available handler or 0 for none

R1 Flags

R2 Pointer to Handler name

R3 Sample Frequency

R4 Sample type

R5 Volume

[RISC OS Open source page](#)

SharedSound_HandlerVolume

SWI &4B443 · SharedSound

Will set a sound handlers Left/Right channel volume.

Before calling

R0 Handler number

R1 volume (bits 0-15 left channel, 16-31 right channel)

On return

R0 Preserved

R1 Current volume

[RISC OS Open source page](#)

SharedSound_InstallDriver

SWI &4B447 · SharedSound

Can either install a new driver or make an existing driver active.

Before calling

R0 Pointer to driver table address

or 1 = Linear

or 2 = Log

R1 Parameter to pass in R12 when driver called

R2 Flags

R3 Volume

On return

All registers preserved

[RISC OS Open source page](#)

SharedSound_InstallHandler

SWI &4B440 · SharedSound

Each handler is called when the current driver is called if the handler is an immediate handler or on call back if the handler is either a call back or process handler. The handlers in each type are called in the order in which they were installed. Process handlers are called after call back handlers.

Before calling

R0 Address of handler

R1 Parameter passed to handler

R2 Flags

R3 Address of handler name

R4 Type

On return

R0 Handler number if successful (V clear)

Flags /

Bit 0 – : if set then use type in R4 otherwise set type to default.

Type /

0 – : Immediate handler

1 – : Call back handler

2 – : Process handler

In practice

Each handler is passed a buffer to fill with the registers set as follows:

R1 – parameter: base address of buffer to fill

R2 – parameter: end of buffer

R3 – parameter: flags

Bit 0 If set then data must be mixed otherwise overwrite.

Bit 30 Set if the volume in R7 is not equal to 0xFFFFFFFF

Bit 31 Set if the volume in R7 is 0 (i.e. sound is muted)

All other bits are currently passed as 0, but it is anticipated that they may be used in the following way in future revisions of the module:

Bit 1 If set then use highest quality e.g. oversampling.

Bits 3-7 Overrun count

0 No overrun

1–30 Number of times buffer has been overrun

31 Buffer has been overrun by more than 30 times

Bit 29 Stereo. Reverse stereo if set.

R4 – parameter: sample frequency

R5 – parameter: sample period

R6 – parameter: fractional step as specified by SharedSound_SampleRate

R7 – parameter: LR volume

R8 – parameter: Pointer to table of helpful fill routines

R4, R5 and R6 are set by SharedSound to the current values as passed in from the lower level Sound Handler, thus may change between calls to the clients handler.

On exit R3 contains flags as above, modified according to the action of the handler. The important requirement on exit is that bit 0 of R3 should be set if any data has been written into the buffer. In fact, SharedSound cannot currently cope with no data being filled, so zeros should always be written.

Helpful fill routines On entry to the fill code, R8 points to a table in the following format: [R8,#0]=Flags word; currently 0. [R8,#4]=Pointer to silence fill routine. [R8,#8]=Pointer to data fill routine.

This table may be extended in future to include more fill variants; it is anticipated that the presence/absence of these routines will be indicated by bits in the flag word.

The data fill routine is called in the following way:

R1 – : Base address of buffer to be filled

R2 – : End of the buffer to be filled

R3 – : Flags (as on entry to the fill routine)

R4 – : Base of buffer to fill from

R5 – : End of buffer to fill from

R6 – : Fractional step value

R7 – : LR volume

R9 – : Fractional accumulator

On Exit /

R0 – : New flags (to be returned from fill handler in R3)

R1 – : Next byte to be filled

R4 – : Next byte to fill from

R9 – : New Fractional accumulator

– – : R2,R3,R5,R6,R7,R8,R12 preserved

– – : All other registers corrupted.

The fill code will fill/mix from the given buffer to the output buffer. It will return either when the buffer is filled, or when it runs out of data.

The silence code is identical except that R4 and R5 are not used, and no data is read; the code fills/mixes zeros.

The silence/data fill code entry points understand all the currently defined flags in R3. The intention is that if future flags are added to R3 then the supplied fill routines will be extended to understand them too. This should provide a degree of

forwards compatibility.

The overall sound fill code in a client can therefore look something like this (assuming the client is filling from a cyclic buffer):

```
fill
STMFD r13!,{r0-r2,r4-r12,r14}
MOV r12,r0
; We are filling from play_ptr to where?
LDR r4,play_ptr
LDR r5,fill_ptr
LDR r9,buffer_start
ADD r4,r4,r9
ADD r5,r5,r9
LDR r9,accumulator
; Are we wrapped, or unwrapped?
CMP r4,r5
BEQ out_of_data
BLO unwrapped
; |*-----*****|
; ^fill_ptr ^play_ptr
LDR r5,buffer_end ; Fill from fill_ptr to buffer_end
; Call the fill code
MOV R14,PC
LDR PC,[R8,#8]
; Did we run out of data?
CMP R4,R5 ;
LDRHS R4,buffer_start ; If r4 >= r5 then we used all the
; data in the buffer.
CMP R1,R2 ; Did we finish the fill?
BHS finish ; If r1 >= r2 then we finished.
LDR R5,fill_ptr ; No, so we ran out of data. Jump
CMP R5,#0 ; Check if there any data there -
BEQ out_of_data ; If not, play silence.
ADD R5,R5,R4
unwrapped
; |-----*****-----|
; ^play_ptr ^fill_ptr
; Call the fill code
MOV R14,PC
LDR PC,[R8,#8]
; Either we finished the fill, or we ran out of data.
CMP R1,R2 ; Did we finish?
BEQ finish
out_of_data
; Fill with silence
MOV R14,PC
LDR PC,[R8,#4]
finish
LDR r3,buffer_start
STR R9,accumulator
SUB r4,r4,r3
STR R4,play_ptr
```

```
MOV r3,r0
LDMFD r13!,{r0-r2,r4-r12,PC}
```

[RISC OS Open source page](#)

SharedSound_RemoveHandler

SWI &4B441 · SharedSound

Will remove a linear sound handler previously added via SharedSound_InstallHandler.

Before calling

R0 Handler number

On return

All registers preserved
[RISC OS Open source page](#)

SharedSound_SampleRate

SWI &4B446 · SharedSound

Can either set the current driver sample rate, or set the sample rate for a specific sound handler.

Before calling

R0 0 for current driver

or >0 Handler number

R1 Sample rate required in 1/1024 Hz

On return

R0 Corrupted

R1 Actual sample rate in 1/1024 Hz (the nearest available sample rate if the required sample rate is not available)

R2 Sample period in $1/2^{24}$ μ s

R3 Fractional step used to get the required rate in 8.24 fixed point

[RISC OS Open source page](#)

Shell_Create

SWI &405C0 · ShellCLI

This SWI is reserved for internal use by the ShellCLI module, and must not be used.

[RISC OS Open source page](#)

Shell_Destroy

SWI &405C1 · ShellCLI

This SWI is reserved for internal use by the ShellCLI module, and must not be used.

[RISC OS Open source page](#)

Socket_Accept

SWI &41203 · Internet

The purpose of this call is to accept an incoming connection from a socket that is currently listening.

Before calling

R0 Socket descriptor of the listening socket

R1 Pointer to an address to be filled in

R2 Pointer to an integer containing the size of the address

On return

R0 Socket descriptor of the accepted connection

Berkeley sockets equivalence

This SWI directly corresponded to the function `accept` in 4.3BSD.

Notes

This SWI is deprecated; the format of the address block in R1 was changed in Internet 5.00 and a new SWI added which accommodates the new layout. See `Socket_Accept_1` for details.

[RISC OS Open source page](#)

Socket_Accept_1

SWI &4121B · Internet

The purpose of this call is to accept an incoming connection from a socket that is currently listening.

Before calling

R0 Socket descriptor of the listening socket

R1 Pointer to an address to be filled in

R2 Pointer to an integer containing the size of the address

On return

R0 Socket descriptor of the accepted connection

Berkeley sockets equivalence

This SWI directly corresponds to the function `accept` and would ordinarily be called from C using the `accept()` function in `socklib.h`.

[RISC OS Open source page](#)

Socket_Bind

SWI &41201 · Internet

The purpose of this call is to bind a socket to a specified local address.

Before calling

R0 Socket descriptor

R1 Pointer to local address to bind this socket to

R2 Size of local address

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponds to the function `bind` and would ordinarily be called from C using the `bind()` function in `socklib.h`.

[RISC OS Open source page](#)

Socket_Close

SWI &41210 · Internet

The purpose of this call is to close a socket previously opened, shutting it down if necessary and freeing the descriptor.

Before calling

R0 Socket descriptor

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponds to the function `close` and would ordinarily be called from C using the `socketclose()` function in `socklib.h`.

Notes

The different function name is needed because in RISC OS file descriptors and socket descriptors come from two different number spaces, while under Unix they share the same number space so a single `close()` function serves to close either type.

[RISC OS Open source page](#)

Socket_Connect

SWI &41204 · Internet

The purpose of this call is to connect a socket to a specified remote address.

Before calling

R0 Socket descriptor

R1 Pointer to address on the remote host to connect to

R2 Size of remote address

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponds to the function connect and would ordinarily be called from C using the connect() function in socklib.h.

[RISC OS Open source page](#)

Socket_Creat

SWI &41200 · Internet

The purpose of this call is to create a new socket of the type given for a specified protocol. This only allocates the resources for a new socket, it does not bind or connect it to anything.

Before calling

R0 Domain

R1 Type

R2 Protocol, or 0 for default for socket type

On return

R0 Socket descriptor

Berkeley sockets equivalence

This SWI directly corresponds to the function socket and would ordinarily be called from C using the socket() function in socklib.h.

[RISC OS Open source page](#)

Socket_Getpeername

SWI &4120E · Internet

The purpose of this call is to return the remote address that the socket is connected to (if any).

Before calling

R0 Socket descriptor

R1 Pointer to an address to be filled in

R2 Pointer to an integer containing the size of the address

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponded to the function `getpeername` in 4.3BSD.

Notes

This SWI is deprecated; the format of the address block in R1 was changed in Internet 5.00 and a new SWI added which accommodates the new layout. See `Socket_Getpeername_1` for details.

[RISC OS Open source page](#)

Socket_Getpeername_1

SWI &4121F · Internet

The purpose of this call is to return the remote address that the socket is connected to (if any).

Before calling

R0 Socket descriptor

R1 Pointer to an address to be filled in

R2 Pointer to an integer containing the size of the address

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponds to the function `getpeername` and would ordinarily be called from C using the `getpeername()` function in `socklib.h`.

[RISC OS Open source page](#)

Socket_Getsockname

SWI &4120F · Internet

The purpose of this call is to return the local address that the socket is connected to (if any).

Before calling

R0 Socket descriptor

R1 Pointer to an address to be filled in

R2 Pointer to an integer containing the size of the address

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponded to the function `getsockname` in 4.3BSD.

Notes

This SWI is deprecated; the format of the address block in R1 was changed in Internet 5.00 and a new SWI added which accommodates the new layout. See `Socket_Getsockname_1` for details.

[RISC OS Open source page](#)

Socket_Getsockname_1

SWI &41220 · Internet

The purpose of this call is to return the local address that the socket is connected to (if any).

Before calling

R0 Socket descriptor

R1 Pointer to an address to be filled in

R2 Pointer to an integer containing the size of the address

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponds to the function `getsockname` and would ordinarily be called from C using the `getsockname()` function in `socklib.h`.

[RISC OS Open source page](#)

Socket_Getsockopt

SWI &4120D · Internet

The purpose of this call is to read one of the options for an open socket.

Before calling

R0 Socket descriptor

R1 Option level

R2 Option number to read

R3 Pointer to buffer to receive value into

R4 Pointer to an integer containing the size of buffer provided

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponds to the function `getsockopt` and would ordinarily be called from C using the `getsockopt()` function in `socklib.h`.

Notes

If the integer pointed to by R4 describes a buffer which is too small to receive the value, the value will be truncated to fit the buffer. If the buffer is sufficiently large, the integer pointed to by R4 will be updated with the actual number of bytes copied.

[RISC OS Open source page](#)

Socket_Gettsize

SWI &41218 · Internet

The purpose of this call is to read how many sockets the system could allocate.

Before calling

--

On return

R0 Maximum number of sockets available

Berkeley sockets equivalence

This SWI directly corresponds to the function `getstablesz` and would ordinarily be called from C using the `getstablesz()` function in `socklib.h`.

Notes

The value may be less than `FD_SETSIZE`, that is the number of sockets which can be polled by `select()` which is not the same as the maximum number of sockets that can exist.

[RISC OS Open source page](#)

Socket Ioctl

SWI &41212 · Internet

The purpose of this call is to perform miscellaneous I/O control operations on a socket.

Before calling

R0 Socket descriptor

R1 Operation to perform

R2 Pointer to buffer containing the value (writes) or to return the value into (reads)

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponds to the function `ioctl` and would ordinarily be called from C using the `socketioctl()` function in `socklib.h`.

Notes

The different function name is needed because in RISC OS file descriptors and socket descriptors come from two different number spaces, while under Unix they share the same number space so a single `ioctl()` function serves to manipulate either type.

[RISC OS Open source page](#)

Socket Listen

SWI &41202 · Internet

The purpose of this call is to listen on a socket of type stream for inbound connections.

Before calling

R0 Socket descriptor

R1 Maximum backlog of inbound connections

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponds to the function `listen` and would ordinarily be called from C using the `listen()` function in `socklib.h`.

[RISC OS Open source page](#)

Socket_Read

SWI &41213 · Internet

The purpose of this call is to read data from a socket. It is equivalent to calling `recv()` with flags of 0.

Before calling

R0 Socket descriptor

R1 Pointer to buffer to receive into

R2 Size of buffer

On return

R0 Number of bytes received

Berkeley sockets equivalence

This SWI directly corresponds to the function `read` and would ordinarily be called from C using the `socketread()` function in `socklib.h`.

Notes

The different function name is needed because in RISC OS file descriptors and socket descriptors come from two different number spaces, while under Unix they share the same number space so a single `read()` function serves to read from either type.

[RISC OS Open source page](#)

Socket_Readv

SWI &41216 · Internet

The purpose of this call is to read data from a socket from a vector of buffer descriptors. It is equivalent to calling `read()` except in the way the buffers need not be contiguous.

Before calling

R0 Socket descriptor

R1 Pointer to an array of buffer descriptors

R2 Number of buffer descriptors

On return

R0 Number of bytes received

Berkeley sockets equivalence

This SWI directly corresponds to the function `readv` and would ordinarily be called from C using the `socketreadv()` function in `socklib.h`.

Notes

The different function name is needed because in RISC OS file descriptors and socket descriptors come from two different number spaces, while under Unix they share the same number space so a single `readv()` function serves to read from either type.

[RISC OS Open source page](#)

Socket_Recv

SWI &41205 · Internet

The purpose of this call is to read data from a socket.

Before calling

R0 Socket descriptor

R1 Pointer to buffer to receive into

R2 Size of buffer

R3 Flags

On return

R0 Number of bytes received

Berkeley sockets equivalence

This SWI directly corresponds to the function `recv` and would ordinarily be called from C using the `recv()` function in `socklib.h`.

[RISC OS Open source page](#)

Socket_Recvfrom

SWI &41206 · Internet

The purpose of this call is to read data from a socket without needing to have connected the socket prior to use.

Before calling

R0 Socket descriptor

R1 Pointer to buffer to receive into

R2 Size of buffer

R3 Flags

R4 Pointer to an address to be filled in

R5 Pointer to an integer containing the size of the address

On return

R0 Number of bytes sent

Berkeley sockets equivalence

This SWI directly corresponded to the function `recvfrom` in 4.3BSD.

Notes

This SWI is deprecated; the format of the address block in R4 was changed in Internet 5.00 and a new SWI added which accommodates the new layout. See `Socket_Recvfrom_1` for details.

[RISC OS Open source page](#)

Socket_Recvfrom_1

SWI &4121C · Internet

The purpose of this call is to read data from a socket without needing to have connected the socket prior to use.

Before calling

R0 Socket descriptor

R1 Pointer to buffer to receive into

R2 Size of buffer

R3 Flags

R4 Pointer to an address to be filled in

R5 Pointer to an integer containing the size of the address

On return

R0 Number of bytes sent

Berkeley sockets equivalence

This SWI directly corresponds to the function `recvfrom` and would ordinarily be called from C using the `recvfrom()` function in `socklib.h`.

[RISC OS Open source page](#)

Socket_Recvmsg

SWI &41207 · Internet

The purpose of this call is to read data from a socket given a message descriptor which includes the address and a vector of buffers to use.

Before calling

R0 Socket descriptor

R1 Pointer to a message descriptor

R2 Flags

On return

R0 Number of bytes received

Berkeley sockets equivalence

This SWI directly corresponded to the function `recvmsg` in 4.3BSD.

Notes

This SWI is deprecated; the format of the message block in R1 was changed in Internet 5.00 and a new SWI added which accommodates the new layout. See `Socket_Recvmsg_1` for details.

[RISC OS Open source page](#)

Socket_Recvmsg_1

SWI &4121D · Internet

The purpose of this call is to read data from a socket given a message descriptor which includes the address and a vector of buffers to use.

Before calling

R0 Socket descriptor

R1 Pointer to a message descriptor

R2 Flags

On return

R0 Number of bytes received

Berkeley sockets equivalence

This SWI directly corresponds to the function `recvmsg` and would ordinarily be called from C using the `recvmsg()` function in `socklib.h`.

[RISC OS Open source page](#)

Socket_Select

SWI &41211 · Internet

The purpose of this call is to wait until one of the descriptors becomes ready based on the read/write/exception masks provided, or a timeout occurs.

Before calling

R0 Number of descriptors to consider from each set

R1 Pointer to a read set

R2 Pointer to a write set

R3 Pointer to a exception set

R4 Pointer to a 2 word timeout value

On return

R0 Number of descriptors which became ready

Berkeley sockets equivalence

This SWI directly corresponds to the function `select` and would ordinarily be called from C using the `select()` function in `socklib.h`.

Notes

The value on entry in **R0** would typically be the highest bit set to inspect plus one in each of the masks. For example if the socket descriptors were 3 + 4 + 5 then there are 6 descriptors to consider (0 to 5 inclusive). Setting it to a larger value than necessary just wastes processor time looking at descriptors which are of no interest.

[RISC OS Open source page](#)

Socket_Send

SWI &41208 · Internet

The purpose of this call is to write data to a socket.

Before calling

R0 Socket descriptor

R1 Pointer to buffer containing the data to send

R2 Size of buffer

R3 Flags

On return

R0 Number of bytes sent

Berkeley sockets equivalence

This SWI directly corresponds to the function `send` and would ordinarily be called from C using the `send()` function in `socklib.h`.

[RISC OS Open source page](#)

Socket_Sendmsg

SWI &4120A · Internet

The purpose of this call is to write data to a socket a socket given a message descriptor which includes the address and a vector of buffers to use.

Before calling

R0 Socket descriptor

R1 Pointer to a message descriptor

R2 Flags

On return

R0 Number of bytes sent

Berkeley sockets equivalence

This SWI directly corresponded to the function `sendmsg` in 4.3BSD.

Notes

This SWI is deprecated; the format of the message block in R1 was changed in Internet 5.00 and a new SWI added which accommodates the new layout. See `Socket_Sendmsg_1` for details.

[RISC OS Open source page](#)

Socket_Sendmsg_1

SWI &4121E · Internet

The purpose of this call is to write data to a socket a socket given a message descriptor which includes the address and a vector of buffers to use.

Before calling

R0 Socket descriptor

R1 Pointer to a message descriptor

R2 Flags

On return

R0 Number of bytes sent

Berkeley sockets equivalence

This SWI directly corresponds to the function `sendmsg` and would ordinarily be called from C using the `sendmsg()` function in `socklib.h`.

[RISC OS Open source page](#)

Socket_Sendto

SWI &41209 · Internet

The purpose of this call is to write data to a socket without needing to have connected the socket prior to use.

Before calling

R0 Socket descriptor

R1 Pointer to buffer containing the data to send

R2 Size of buffer

R3 Flags

R4 Pointer to address on the remote host to connect to

R5 Size of remote address

On return

R0 Number of bytes sent

Berkeley sockets equivalence

This SWI directly corresponds to the function `sendto` and would ordinarily be called from C using the `sendto()` function in `socklib.h`.

[RISC OS Open source page](#)

Socket_Setsockopt

SWI &4120C · Internet

The purpose of this call is to set one of the options for an open socket.

Before calling

R0 Socket descriptor

R1 Option level

R2 Option number to write

R3 Pointer to buffer containing the new option value

R4 The size of buffer provided

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponds to the function `setsockopt` and would ordinarily be called from C using the `setsockopt()` function in `socklib.h`.

[RISC OS Open source page](#)

Socket_Shutdown

SWI &4120B · Internet

The purpose of this call is to shut down an existing send and receive operations on a stream socket.

Before calling

R0 Socket descriptor

R1 Direction of shut down

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponds to the function `shutdown` and would ordinarily be called from C using the `shutdown()` function in `socklib.h`.

Notes

The socket descriptor is not freed by this and could be subsequently reused for another connection.

[RISC OS Open source page](#)

Socket_Stat

SWI &41215 · Internet

The purpose of this call is to read status information about a socket descriptor.

Before calling

R0 Socket descriptor

R0 Pointer to a status structure buffer

On return

R0 Corrupted

Berkeley sockets equivalence

This SWI directly corresponds to the function `fstat` and would ordinarily be called from C using the `socketstat()` function in `socklib.h`.

Notes

The different function name is needed because in RISC OS file descriptors and socket descriptors come from two different number spaces, while under Unix they share the same number space so a single `fstat()` function serves to query either type.

There aren't direct parallels in RISC OS to all of the members of the status descriptor that might be returned under Unix, therefore some values may be dummy placeholders. For example the inode number has no meaning.

[RISC OS Open source page](#)

Socket_Sysctl

SWI &4121A · Internet

The purpose of this call is to perform low level kernel variable operations.

Before calling

R0 Pointer to name of a kernel value to read or write

R1 Length of the name

R2 Pointer to buffer to read old value into

R3 Pointer to an integer containing the length of buffer in R2

R4 Pointer to buffer containing the new value to be set

R5 Length of buffer in R4

On return

R0 Corrupted

Notes

In this context "kernel" means the emulation of the Unix kernel by the Internet module rather than the RISC OS kernel - and not all values are implemented.

For read operations, R4 should be NULL, and R5 set to 0.

This function is for internal use only, and the meaning of the system variables supported by any given version of the Internet module may change. Use the `*SysCtl?` command line tool instead.

[RISC OS Open source page](#)

Socket_Version

SWI &41222 · Internet

This call reports the Internet module version multiplied by 100.

Before calling

--

On return

R0 Version number of Internet module * 100

Notes

This call is only available from Internet 5.04 onwards.

[RISC OS Open source page](#)

Socket_Write

SWI &41214 · Internet

The purpose of this call is to write data to a socket. It is equivalent to calling `send()` with flags of 0.

Before calling

R0 Socket descriptor

R1 Pointer to buffer containing the data to send

R2 Size of buffer

On return

R0 Number of bytes sent

Berkeley sockets equivalence

This SWI directly corresponds to the function `write` and would ordinarily be called from C using the `socketwrite()` function in `socklib.h`.

Notes

The different function name is needed because in RISC OS file descriptors and socket descriptors come from two different number spaces, while under Unix they share the same number space so a single `write()` function serves to write to either type.

[RISC OS Open source page](#)

Socket_Writev

SWI &41217 · Internet

The purpose of this call is to write data to a socket from a vector of buffer descriptors. It is equivalent to calling `write()` except in the way the buffers need not be contiguous.

Before calling

R0 Socket descriptor

R1 Pointer to an array of buffer descriptors

R2 Number of buffer descriptors

On return

R0 Number of bytes sent

Berkeley sockets equivalence

This SWI directly corresponds to the function `writv` and would ordinarily be called from C using the `socketwritv()` function in `socklib.h`.

Notes

The different function name is needed because in RISC OS file descriptors and socket descriptors come from two different number spaces, while under Unix they share the same number space so a single `writv()` function serves to write to either type.

[RISC OS Open source page](#)

Sound_AttachNamedVoice

SWI &4018A · SoundChannels

Attaches a named voice to a channel. The previous voice is shut down and the new voice is reset.

Before calling

R0 Channel number (1 – 8)

R1 Pointer to voice name (null terminated)

On return

R0 Preserved (or 0 for fail)

R1 Preserved

In practice

- `Sound_RemoveVoice`
- `Sound_AttachVoice`

[RISC OS Open source page](#)

Sound_AttachVoice

SWI &40185 · SoundChannels

Attaches a voice on a given slot to a channel. The previous voice is shut down and the new voice is reset.

Before calling

R0 Channel number (1 – 8)

R1 Voice slot to attach (0 to detach and mute channel)

On return

R0 Preserved (or 0 if illegal channel number)

R1 Previous voice number (or 0 if not previously attached)

In practice

- Sound_RemoveVoice
- Sound_AttachNamedVoice

[RISC OS Open source page](#)

Sound_Configure

SWI &40140 · SoundDMA

Used to configure the number of sound channels, the sample period and the sample length. It can also be used by specialised applications to replace the default Channel Handler and Scheduler. All current settings may be read by using zero for all input parameters.

Before calling

R0 Channels for 8 bit sound, rounded up to 1,2,4 or 8

or 0 to preserve previous value

R1 Samples per channel (in bytes)

or 0 to preserve previous value

R2 Sample period (in microseconds per channel)

or 0 to preserve previous value

R3 Pointer to Channel Handler

or 0 to preserve previous value

R4 Pointer to Scheduler

or 0 to preserve previous value

On return

R0-R4 Previous values

Notes

R0 only controls the number of 8 bit sound channels; the 16 bit sound system always has two channels (left and right).

Although R2 can be used to specify the sample period, the preferred method to set or read the sample rate for RISC OS 3.5 and above is to use Sound_SampleRate.

If no Channel Handler and/or Scheduler are installed, the previous pointers returned in R3 / R4 will be invalid. Validity may be confirmed by checking the lowest two bits of the relevant register are zero, ie TST R3, #3

Service_Sound 8 will be issued if the sound configuration changes as a result of this call.

[RISC OS Open source page](#)

Sound_Control

SWI &40189 · SoundChannels

Plays a sound immediately on the given channel.

Before calling

R0 Channel number (1-8)

R1 Amplitude:

&FFF1 – &FFFF and 0 for BBC emulation amplitude (0 to -15)

&0001 – &000F BBC envelope not emulated

&0100 – &01FF for full amplitude/gate control:

bit 7 is:

0 for gate ON/OFF

1 for smooth update (gate not retriggered)

bits 6 – 0 are 7-bit logarithm of amplitude

R2 Pitch:

&0000 – &00FF for BBC emulation pitch

&0100 – &7FFF for enhanced pitch control:

bits 14 – 12 = 3-bit octave

bits 11 – 0 = 12-bit fractional part of octave (&4000 is Middle C)

&8000 + n for phase accumulator increment (n is 0 – &7FFF)

R3 Duration:

&0001 – &00FE for BBC emulation in 5 centisecond periods

&00FF for BBC emulation 'infinite' time (converted to &F0000000)

> &00FF for duration in 5 centisecond periods

On return

R0-R3 Preserved

[RISC OS Open source page](#)

Sound_ControllerInfo

SWI &4014A · SoundDMA

This call can be used to read the properties of a specific audio controller device.

Before calling

R0 Pointer to space/control terminated controller ID string

R1 Pointer to buffer for result, or 0 to check required length

R2 Length of buffer

R3 Property to read

On return

R0 0, or pointer to error block

R1 Buffer updated with value of property

R2 Length of data

Notes

Many versions of SoundDMA do not return errors when unknown SWIs are called. Therefore, to detect if this SWI is supported, check if R0 has been set to 0 on exit (Valid reason codes start at 1), or use Sound_ReadSysInfo 1.

If the supplied buffer is not large enough, or the pointer was 0, a buffer overflow error will be generated and no data will be copied to the buffer.

[RISC OS Open source page](#)

Sound_ControlPacked

SWI &40186 · SoundChannels

This call is identical to Sound_Control, but the parameters are packed 16-bit at a time into low R0, high R0, low R1, high R1 respectively. It is provided for BBC compatibility and for the use of the Scheduler. The Sound_Control call should be used in preference where possible.

Before calling

R0 AAAACCCC (Amplitude / Channel)

R1 DDDDPPPP (Duration / Pitch)

On return

R0, R1 Preserved

[RISC OS Open source page](#)

Sound_Enable

SWI &40141 · SoundDMA

This SWI is used to enable or disable sound output. It can also be used to read the state of the sound system.

Before calling

R0 0 No action, just return current state

1 Disable sound output

2 Enable sound output

On return

R0 Previous state (1=disabled, 2=enabled)

Notes

When sound output is disabled, sound interrupts will cease, and any processing which is tied to them (for example the sound scheduler) will effectively pause.

Service_Sound 9 or Service_Sound 10 will be issued if sound output stops or starts as a result of this call.

[RISC OS Open source page](#)

Sound_EnumerateControllers

SWI &40149 · SoundDMA

This call can be used to enumerate the audio controller devices that are available for use by the sound system.

Before calling

R0 Pointer to space/control terminated controller ID string to start from, or 0 or an empty string to start enumeration

R1 Pointer to buffer for result (can be R0), or 0 to check required length

R2 Length of buffer

On return

R0 0, or pointer to error block

R1 Buffer updated with null-terminated ID string of next controller, or empty string if no more controllers

R2 Length of data (including null terminator)

Notes

Many versions of SoundDMA do not return errors when unknown SWIs are called. Therefore, to detect if this SWI is supported, check if R0 has been set to 0 on exit (Valid reason codes start at 1), or use Sound_ReadSysInfo 1.

If the supplied buffer is not large enough, or the pointer was 0, a buffer overflow error will be generated and no data will be copied to the buffer.

[RISC OS Open source page](#)

Sound_InstallVoice

SWI &40183 · SoundChannels

This SWI can perform several different operations, determined by combinations of R0 & R1. Consult the table below for details.

R0 / R1 / Action

Pointer – R1: - · Action: Install voice generator

0 – R1: 0 · Action: Find free voice slot

0 – R1: 1-32 · Action: Read name of installed voice

1 – R1: - · Action: Install voice generator, specifying its local language name

2 – R1: - · Action: Read name of installed voice and its local language name

3 – R1: - · Action: Change local language name of voice

Notes

If an error occurs, this SWI does not set V in the usual manner. Instead R1 is zero on exit, and R0 points directly to a null-terminated error string.

[RISC OS Open source page](#)

Sound_InstallVoice 0

SWI &40183 · SoundChannels

Read the name of the voice installed in the specified slot.

Before calling

R0 0

R1 Voice slot

On return

R0 Pointer to name of installed voice

R1 Preserved

[RISC OS Open source page](#)

Sound_InstallVoice 0,0

SWI &40183 · SoundChannels

This call can be used to locate the first free voice slot.

Before calling

R0 0

R1 0

On return

R0 0, or null terminated error string if **R1** = 0

R1 Index of first free voice handle (1-32), or 0 on error

Notes

If an error occurs, this SWI does not set **V** in the usual manner. Instead **R1** is zero on exit, and **R0** points directly to a null-terminated error string.

[RISC OS Open source page](#)

Sound_InstallVoice 1

SWI &40183 · SoundChannels

Used to add a Voice Generator to the table of available voices, specifying its name in the local language.

Before calling

R0 1

R1 Voice slot (1-31) or 0 to use next free slot

R2 Pointer to Voice Generator

R3 Pointer to Voice name in local language, or 0 if no local name

On return

R0 Preserved

R1 Voice number allocated, or 0 if unable to install

R2 Pointer to previous voice, or null terminated error string if R1=0

R3 Preserved

Notes

If an error occurs, this SWI does not set V in the usual manner. Instead R1 is zero on exit, and R0 points directly to a null-terminated error string.

[RISC OS Open source page](#)

Sound_InstallVoice 2

SWI &40183 · SoundChannels

Read the name of the voice installed in the specified slot and its local name. If the slot is unused, R2, R3 are null pointers.

Before calling

R0 2

R1 Voice slot

On return

R0, R1 Preserved

R2 Pointer to name of installed voice

R3 Pointer to name of installed voice in local language

[RISC OS Open source page](#)

Sound_InstallVoice 3

SWI &40183 · SoundChannels

Changes the local name of the voice installed in the specified slot.

Before calling

R0 3

R1 Voice slot

R2 0

R3 Pointer to new voice name in local language

On return

R0-R3 Preserved

[RISC OS Open source page](#)

Sound_InstallVoice V

SWI &40183 · SoundChannels

Used to add a Voice Generator to the table of available voices.

Before calling

R0 Pointer to Voice Generator

R1 Voice slot (1-32), or 0 to install in next free slot

On return

R0 Pointer to name of previous voice, or null terminated error string if R1 = 0

R1 Voice number allocated, or 0 if unable to install

Notes

If an error occurs, this SWI does not set V in the usual manner. Instead R1 is zero on exit, and R0 points directly to a null-terminated error string.

If an attempt is made to replace an existing Voice Generator R0 will point to the previous Voice Generator name, R1=0 and the new Voice Generator will not be installed.

[RISC OS Open source page](#)

Sound_LinearHandler

SWI &40145 · SoundDMA

This SWI is used to read the details of, or replace, the 16 bit linear sound handler.

Before calling

R0 Reason code

All other registers dependent on reason code

On return

Dependent on reason code

Notes

This SWI should only be used if Sound_Mode 0 indicates that 16 bit sound output is supported.

Reason Codes

/

0 – : Read current linear handler

1 – : Install new linear handler

Other reason codes: reserved

[RISC OS Open source page](#)

Sound_LinearHandler 0

SWI &40145 · SoundDMA

This SWI is used to read the details of the current 16 bit linear sound handler.

Before calling

R0 0 (Reason code)

On return

R1 Current handler address

R2 Current handler workspace (passed to handler in R0)

Notes

This SWI should only be used if Sound_Mode 0 indicates that 16 bit sound output is supported.

If no handler is currently installed, 0 will be returned in R1 and -1 in R2.

[RISC OS Open source page](#)

Sound_LinearHandler 1

SWI &40145 · SoundDMA

This SWI is used to replace the current 16 bit linear sound handler.

Before calling

R0 1 (Reason code)

R1 New handler address (0 to uninstall handler)

R2 New handler workspace (passed to handler in R0, ignored if R1=0)

On return

R1 Old handler address

R2 Old handler workspace (passed to handler in R0)

Notes

This SWI should only be used if Sound_Mode 0 indicates that 16 bit sound output is supported.

If no handler was previously installed, 0 will be returned in R1 and -1 in R2.

[RISC OS Open source page](#)

Sound_LogScale

SWI &40182 · SoundChannels

This SWI is used to scale an 8 bit mu-law sample by the current volume (as set by Sound_Volume).

Before calling

R0 8 bit mu-law logarithmic sample

On return

R0 8 bit mu-law logarithmic sample, scaled by current volume

Notes

Voice generators should avoid calling this SWI from within their buffer fill code – the overhead of calling a SWI for every sample to write will be too much for older systems to cope with.

[RISC OS Open source page](#)

Sound_Mode

SWI &40144 · SoundDMA

This SWI is used to detect the presence of and control various features of the 16 bit sound system.

Before calling

R0 Reason code

All other registers dependent on reason code

On return

Dependent on reason code

Notes

This SWI was first introduced with RISC OS 3.5. However SoundDMA never raises an error for unknown SWIs – it simply returns with all registers preserved. Therefore care must be taken when attempting to use this SWI. The recommended behaviour is to first call this SWI with R0=0 to check if the 16 bit sound system is available. If it returns with R0=1 then 16 bit sound is available and the other reason codes should be functional. If it returns with R0=0 then either the SWI was not recognised or 16 bit sound is unavailable. But in both cases this indicates that the other reason codes are unavailable and should not be used.

Also note that calling with an unknown or unsupported reason code will not generate an error. However on a system without 16 bit sound, R0 and R1 may be corrupted.

Reason Codes

/

0 – : Read presence of 16 bit sound system

1 – : Oversampling control

2 – : Set external sound clock handler (unimplemented)

3 – : Mono output control

Other reason codes: reserved

[RISC OS Open source page](#)

Sound_Mode 0

SWI &40144 · SoundDMA

This SWI is used to detect the presence of the 16 bit sound system and to read its status and features.

Before calling

R0 0 (reason code)

On return

R0 0 only 8 bit sound is available

1 16 bit sound available (plus emulated 8 bit sound)

R1 0 if 16 bit sound unavailable. Otherwise:

Bits 0-3: Sound output format and clock control status:

1 16-bit linear output, 44k1, 22k05, 11k025 and selected original rates

2 16-bit linear output, internal clock, selected original rates only

3 16-bit linear output, external master clock source, custom rates only

Bit 4: Oversampling mode (1 = enabled)

Bit 5: Mono conversion status (1 = enabled)

Bits 6-31: Reserved for future expansion (0 returned)

Notes

If R0 is 1 on exit, it indicates that the following features are available:

- Sound_Mode with non-zero reason codes
- Sound_LinearHandler
- Sound_SampleRate

If R0 is 0 then the above features are unavailable and no attempt should be made to use them.

Use of bits 0-3 of R1 to determine the support sample rates is deprecated. Sound_SampleRate should be used instead.

Bit 4 of R1 reflects the desired oversampling state, as set by Sound_Mode 1. Note that even if bit 4 is set, it does not guarantee that oversampling is currently in use – for certain sound system configurations, hardware or software limits may prevent it from being used.

[RISC OS Open source page](#)

Sound_Mode 1

SWI &40144 · SoundDMA

This SWI is used to control whether automatic oversampling of sound output should be performed.

Before calling

R0 1 (reason code)

R1 New oversampling state:

0 disable oversampling

Non-zero use oversampling where possible

On return

R1 Previous oversampling state (0=disabled, 1=enabled)

Notes

If oversampling is enabled, and certain hardware/software requirements are met, then all sound output (8 bit and 16 bit) will be 2x oversampled via linear interpolation. Generally speaking, oversampling will be used if the following requirements are met:

- Oversampling is enabled via this call
- The desired sample rate is under 24kHz
- The hardware supports both the requested sample rate and 2x the requested sample rate
- The buffer size used for 16 bit sound is 2KB or smaller (to allow the oversampled buffer to fit in one 4KB page)

Sound_Mode 0 can be used to read the oversampling state without changing it.

[RISC OS Open source page](#)

Sound_Mode 3

SWI &40144 · SoundDMA

This SWI is used to control whether the stereo sound output should be downmixed to mono.

Before calling

R0 3 (reason code)

R1 New mono conversion state:

0 disable mono conversion

Non-zero convert to mono output

On return

R1 Previous mono conversion state (0=disabled, 1=enabled)

Notes

If mono conversion is enabled, then all sound output (8 bit and 16 bit) will be downmixed to mono. In the current implementation, this conversion is performed in-place - the hardware still receives a stereo data stream, but both left and right samples will be set to identical values.

Sound_Mode 0 can be used to read the mono conversion state without changing it.

[RISC OS Open source page](#)

Sound_Pitch

SWI &40188 · SoundChannels

Converts a 15-bit pitch to internal pitch value, suitable for a voice phase accumulator oscillator.

Before calling

R0 15-bit pitch value:

bits 14 - 12 are a 3-bit octave number

bits 11 - 0 are a 12-bit fraction of an octave (in units of 1/4096 octave)

On return

R0 32-bit phase accumulator value, or preserved if R0 ³ &8000 on entry

[RISC OS Open source page](#)

Sound_QBeat

SWI &401C6 · SoundScheduler

Sets or reads the beat counter or bar length.

Before calling

R0 0 to return current beat number

-1 to return current bar length

<-1 to disable beat counter and set bar length 0

+N to enable beat counter with bar length N (counts 0 to N-1)

On return

R0 current beat number (R0 = 0 on entry), otherwise the previous bar length

In practice

The simplest use of this call is to read either the current value of the beat counter or the current bar length. When the beat counter is disabled both it and the bar length are reset to zero. All scheduling occurs relative to the time the scheduling call is issued. When the beat counter is enabled it is reset to zero. It then increments, resetting every time it reaches the programmed bar length (N-1). Scheduling using Sound_QSchedule then occurs relative to the last bar reset; however, scheduling using *QSound is still relative to the time the command is issued.

[RISC OS Open source page](#)

Sound_QFree

SWI &401C3 · SoundScheduler

This call returns the minimum number of slots guaranteed free. The calculation assumes the worst case of data structure overheads that could occur, so it is likely that more slots can in fact be used. If this guaranteed free slot count is exceeded this call will return negative values, and the return status of Sound_QSchedule must be carefully monitored to observe when overflow occurs.

Before calling

No parameters

On return

R0 Number of guaranteed slots free

< 0 indicates over worst case limit, but may still be free slots

[RISC OS Open source page](#)

Sound_QInit

SWI &401C0 · SoundScheduler

Initialises the Scheduler's event queue.

Before calling

No parameters

On return

R0 0 if successful

In practice

This call flushes out all events currently scheduled and re-initialises the event queue. The tempo is set to the default, the beat counter is reset and disabled, and the bar length set to zero.

[RISC OS Open source page](#)

Sound_QInterface

SWI &401C7 · SoundScheduler

You must not use this in your own code.

[RISC OS Open source page](#)

Sound_QRemove

SWI &401C2 · SoundScheduler

You must not use this in your own code.

[RISC OS Open source page](#)

Sound_QSchedule

SWI &401C1 · SoundScheduler

This call schedules a sound SWI call. If the beat counter is enabled the schedule period is measured from the last start of a bar, otherwise it is measured from the time the call is made. A schedule time of -1 forces the new event to be queued for activation concurrently with the previously scheduled one.

Before calling

R0 Schedule period

-1 to synchronise with the previously scheduled event

-2 for immediate scheduling

R1 0 to schedule a Sound_ControlPacked call, or SWI to schedule (in the form &F000000 + SWI number)

R2 SWI parameter to be passed in R0

R3 SWI parameter to be passed in R1

On return

R0 0 for successfully queued

< 0 for failure (queue full)

[RISC OS Open source page](#)

Sound_QSDispatch

SWI &401C4 · SoundScheduler

You must not use this in your own code.

[RISC OS Open source page](#)

Sound_QTempo

SWI &401C5 · SoundScheduler

This command sets the tempo for the Scheduler. The default tempo is &1000, which corresponds to one beat per centisecond; doubling the value doubles the tempo (ie &2000 gives two beats per centisecond), while halving the value halves the tempo (ie &800 gives half a beat per centisecond).

Before calling

R0 0 New tempo (or 0 for no change)

On return

R0 Previous tempo value

In practice

The parameter can be thought of as a hexadecimal fractional number, where the three least significant digits are the fractional part.

[RISC OS Open source page](#)

Sound_ReadControlBlock

SWI &4018B · SoundChannels

Reads 32-bit data values from the Sound Channel Control Block (SCCB) for the designated channel. This call can be used to read parameters not catered for in the Sound_Control calls returned by Voice Generators, using an area of the SCCB reserved for the programmer.

Before calling

R0 Channel number (1 – 8)

R1 Offset to read from (0 – 255)

On return

R0 Preserved (or 0 if fail, invalid channel, or invalid read offset)

R1 Preserved

R2 32-bit word read (if R0 non-zero on exit)

[RISC OS Open source page](#)

Sound_ReadSysInfo

SWI &40147 · SoundDMA

The purpose of this call is to allow you to read various parameters of the sound system.

Before calling

R0 Reason code

On return

R0 0

All other registers are dependent on reason code

All other registers dependent on reason code

Notes

The action performed depends on the reason code value in R0.

Many versions of SoundDMA do not return errors when unknown SWIs are called. Therefore, to detect if this SWI is supported, check if R0 has been set to 0 on exit (Valid reason codes start at 1).

[RISC OS Open source page](#)

Sound_RemoveVoice

SWI &40184 · SoundChannels

Removes a voice from the system. The Channel Handler is notified that a Voice Generator is being removed.

Before calling

R1 Voice slot (1 – 32)

On return

R0 Pointer to name of previous voice (or error message)

R1 Voice number de-allocated (0 for fail)

In practice

If an error occurs, V is not set on exit instead R1 returns zero and R0 points to an error string.

- Sound_AttachVoice
- Sound_AttachNamedVoice

[RISC OS Open source page](#)

Sound_SampleRate

SWI &40146 · SoundDMA

Read or change the sound sample rate.

Before calling

R0 Reason code

All other registers dependent on reason code

On return

R0 Preserved

All other registers dependent on reason code

Notes

This SWI should only be used if Sound_Mode 0 indicates that 16 bit sound output is supported.

Reason Codes

/ Action

0 – Action: Read available sample rates

1 – Action: Read current sample index and rate

2 – Action: Convert sample index to rate

3 – Action: Set sample rate index

[RISC OS Open source page](#)

Sound_SampleRate 0

SWI &40146 · SoundDMA

This call can be used to read the number of different sample rates that are supported by the 16 bit sound system. This value can then be used to enumerate the available rates by calling Sound_SampleRate 2.

Before calling

R0 0 (Reason code)

On return

R1 Number of sample rates

All other registers preserved

Notes

This SWI should only be used if Sound_Mode 0 indicates that 16 bit sound output is supported.

[RISC OS Open source page](#)

Sound_SampleRate 1

SWI &40146 · SoundDMA

This call can be used to read the current sample rate index, and the sample rate that index corresponds to.

Before calling

R0 1 (Reason code)

On return

R1 Current sample rate index (1+)

R2 Current sample rate (1/1024Hz)

All other registers preserved

Notes

This SWI should only be used if Sound_Mode 0 indicates that 16 bit sound output is supported.

Sample rate indices are 1-based.

Sample rates are measured in units of 1/1024Hz. E.g. 22050Hz would be represented as $22050 \times 1024 = 22579200$.

Sound_SampleRate 3 can be used to set the current sample rate.

[RISC OS Open source page](#)

Sound_SampleRate 2

SWI &40146 · SoundDMA

This call can be used to read the sample rate value that corresponds to a given index.

Before calling

R0 2 (Reason code)

R1 Sample rate index (1+)

On return

R2 Corresponding sample rate value (1/1024Hz)

All other registers preserved

In practice

Internally sample rates are sorted in increasing order. This allows a binary chop or similar algorithm to be used to find the sample rate that's closest to a given desired rate without having to iterate through all the available rates.

Notes

This SWI should only be used if Sound_Mode 0 indicates that 16 bit sound output is supported.

Sample rate indices are 1-based. Sound_SampleRate 0 can be used to find the number of available rates, i.e. the maximum permissible value for R1.

Sample rates are measured in units of 1/1024Hz. E.g. 22050Hz would be represented as $22050 \times 1024 = 22579200$.

Sound_SampleRate 3 can be used to set the current sample rate.

[RISC OS Open source page](#)

Sound_SampleRate 3

SWI &40146 · SoundDMA

This call can be used to set the current sound system sample rate. Rather than specifying the sample rate value directly you must use Sound_SampleRate 0 and Sound_SampleRate 2 to find the index that's closest to your desired sample rate, and then set the rate using that index.

Before calling

R0 3 (Reason code)**R1** Sample rate index (1+)

On return

R1 Previous sample rate index (1+)**R2** Previous sample rate value (1/1024Hz)

All other registers preserved

In practice

The previously set sample rate index and value will be returned by the call.

Notes

This SWI should only be used if Sound_Mode 0 indicates that 16 bit sound output is supported.

Sample rate indices are 1-based.

Sample rates are measured in units of 1/1024Hz. E.g. 22050Hz would be represented as $22050 \times 1024 = 22579200$.

Sound_SampleRate 1 can be used to read the current sample rate setting without altering it.

It is also possible to alter the sample rate of the sound system by passing a sample period to Sound_Configure. However when the 16 bit sound system is in use this behaviour is discouraged; all that will happen is that SoundDMA will pick the closest available rate out of the list of rates supported by the hardware and then use that. In some cases this may be an exact match for the requested rate, while in other cases it may differ significantly.

[RISC OS Open source page](#)

Sound_SelectDefaultController

SWI &40148 · SoundDMA

The purpose of this call is to select the audio controller device that should be used as the default device. This is the device which the 8-bit and 16-bit sound systems will use to output their audio.

Before calling

R0 Pointer to space/control terminated controller ID string

On return

R0 0, or pointer to error block

Notes

Many versions of SoundDMA do not return errors when unknown SWIs are called. Therefore, to detect if this SWI is supported, check if R0 has been set to 0 on exit (Valid reason codes start at 1), or use Sound_ReadSysInfo 1.

The current audio device can be determined by calling Sound_ReadSysInfo 2.

*Audio can also be used to select the default controller.

[RISC OS Open source page](#)

Sound_SoundLog

SWI &40181 · SoundChannels

This SWI is used to convert a signed linear sample to the 8 bit logarithmic format that's used by the 8 bit sound system. The returned value will be scaled by the current volume (as set by Sound_Volume).

Before calling

R0 32bit signed linear sample

On return

R0 8 bit mu-law logarithmic sample

Notes

Voice generators should avoid calling this SWI from within their buffer fill code - the overhead of calling a SWI for every sample to write will be too much for older systems to cope with.

[RISC OS Open source page](#)

Sound_Speaker

SWI &40143 · SoundDMA

This SWI is used to turn the machines internal speaker on or off.

Before calling

R1 Speaker setting:

0 No change (just return previous setting)

1 Turn speaker off

2 Turn speaker on

Other values reserved

On return

R0 Previous speaker setting (1 or 2)

In practice

If a machine does not have an internal speaker, or has no control over the speaker power, this SWI will always report that the speaker is on.

On RISC OS 5 this SWI simply calls SoundCtrl_SetMix and SoundCtrl_GetMix to mute or unmute the speaker.

[RISC OS Open source page](#)

Sound_Stereo

SWI &40142 · SoundDMA

This SWI is used to set or read the stereo position of a channel within the 8 bit sound system.

Before calling

R0 Channel number (1-8)

R1 Stereo position (-127 to 127), or -128 to read

On return

R1 Previous position, or -128 if an invalid channel was specified

In practice

A position of -127 corresponds to 100% left, 0 corresponds to center (50% left, 50% right), and 127 corresponds to 100% right. Intermediate values can be used to select other positions.

[RISC OS Open source page](#)

Sound_Tuning

SWI &40187 · SoundChannels

The purpose of this call is to allow alter the tuning of the 8-bit sound system, or to enable/disable automatic tuning.

Before calling

R0 0 => Read current tuning value

1-&7FFF => Set tuning value

&4f545541 ("AUTO") => control automatic tuning

R1 Automatic tuning option (if R0="AUTO"):

0 => Read auto tuning state

1 => Disable auto tuning

2 => Enable auto tuning

On return

- If R0 <> "AUTO" on entry:

R0 Previous tuning value (1-&7fff)

- If R0 = "AUTO" on entry:

R0 New auto tuning state (1=off, 2=on)

R1 Previous auto tuning state (1=off, 2=on)

In practice

Tuning values take the form &off where 'o' is the octave and 'fff' is the fraction of the octave.

Notes

Automatic tuning is a feature introduced in SoundChannels 1.37 (RISC OS 5.19) to allow several aspects of the 8-bit sound system to automatically adjust itself to the current sound system sample rate and buffer size. The default state is for automatic tuning to be enabled, and while automatic tuning is enabled any attempt to set a manual tuning value will silently fail.

Automatic tuning currently affects the following features of the sound system:

- Tuning value
- Sound duration (when using Sound_Control)
- Tempo value (SoundScheduler 1.31; tempo can still be set via Sound_QTempo, but will be correctly adjusted such that the value is the number of beats per centisecond)

If an attempt is made to use automatic tuning on a version of the SoundChannels module which does not support the option, the parameter &4f545541 will be interpreted as a tuning value, causing the tuning to be set to &5541. The suggested way of checking if automatic tuning is supported is to first call with R1=0, to attempt to read the current automatic tuning state. If R1=0 on exit then it indicates that automatic tuning isn't understood and you should call Sound_Tuning again in order to restore the previous tuning value (simply pass in R0 = R0)

Even if the version of SoundChannels in use supports automatic tuning, it is possible that the feature will be unavailable, due to lack of support for the sound config changing service call. In this case any attempt to enable automatic tuning will fail, with R0 being 0 on exit.

Starting with SoundChannels 1.39, Service_Sound 11 will be issued when this SWI is used to turn automatic tuning on or off.

[RISC OS Open source page](#)

Sound_Volume

SWI &40180 · SoundChannels

This SWI is used to read or set the master volume of the 8 bit sound system.

Before calling

R0 New volume (1-127)

Or 0 to read current volume

On return

R0 Previous volume (1-127)

Notes

Volume is specified using a logarithmic scale, with 127 representing a 1x multiplier. For every 16 you subtract from the base volume of 127, the linear output will be halved. I.e.:

127 – Multiplier: 1

127-16 – Multiplier: 1/2

127-32 – Multiplier: 1/4

127-48 – Multiplier: 1/8

127-64 – Multiplier: 1/16

127-80 – Multiplier: 1/32

127-96 – Multiplier: 1/64

127-112 – Multiplier: 1/128

The setting made here or via *Volume influences the return values of SWIs Sound_SoundLog and Sound_LogScale, and it also controls the contents of the linear-to-log and log amplification tables that are available at offsets +8 and +12 of the sound channel handler block (available via Sound_Configure). Well-behaved voice generators should make use of one of

the above interfaces to determine the volume scale they should use when they generate their output.

[RISC OS Open source page](#)

Sound_WriteControlBlock

SWI &4018C · SoundChannels

Writes 32-bit data values to the Sound Channel Control Block (SCCB) for the designated channel. This call can be used to read parameters not catered for in the Sound_Control calls returned by Voice Generators, using an area of the SCCB reserved for the programmer.

Before calling

R0 Channel number (1 – 8)

R1 Offset to read from (0 – 255)

R2 32-bit word to write

On return

R0 Preserved (or 0 if fail, invalid channel, or invalid write offset)

R1 Preserved

R2 previous 32-bit word (if R0 non-zero on exit)

[RISC OS Open source page](#)

SoundCtrl_ExamineMixer

SWI &50000 · SoundControl

This SWI is used to read the features of a given mixer.

Before calling

R0 Mixer system

R1 Pointer to word-aligned buffer for result

R2 Buffer size (may be 0)

On return

R1 Buffer filled with as many blocks as possible

R2 Decrement by number of bytes needed for complete description

R3 Individual block size (bytes) – will be a multiple of 4

R4 Number of blocks placed in buffer

In practice

If R2 is negative on exit, then the supplied buffer was too small to contain the full description and should be resized.

Notes

R0 can either be a pointer to a null/control-terminated audio controller device ID string (as returned by Sound_EnumerateControllers or 0 (NULL) to refer to the mixer for the current default controller.

Other SoundControl APIs such as SoundCtrl_SetMix use category and index parameters to identify each mixer channel, but the blocks that are placed within the buffer only indicate the category. The index of a given channel can be determined by

looking at how many channels with the same category occur before it; for any given category, the first block in the buffer with that category is index 0, the second block with that category is index 1, etc.

[RISC OS Open source page](#)

SoundCtrl_GetMix

SWI &50002 · SoundControl

This SWI is used to read the volume settings of a specific mixer channel.

Before calling

R0 Mixer system

R1 Category

R2 Index within category

On return

R3 1 = muted, 0 = unmuted

R4 Gain in units of 1/16dB

Notes

R0 can either be a pointer to a null/control-terminated audio controller device ID string (as returned by Sound_EnumerateControllers or 0 (NULL) to refer to the mixer for the current default controller.

Gain is specified in units of 1/16dB, so for example -16 corresponds to -1dB.

[RISC OS Open source page](#)

SoundCtrl_SetMix

SWI &50001 · SoundControl

This SWI is used to alter the volume settings of a specific mixer channel.

Before calling

R0 Mixer system

R1 Category

R2 Index within category

R3 1 = mute, 0 = unmute

R4 Gain in units of 1/16dB

On return

- All registers preserved

Notes

R0 can either be a pointer to a null/control-terminated audio controller device ID string (as returned by Sound_EnumerateControllers or 0 (NULL) to refer to the mixer for the current default controller.

Gain is specified in units of 1/16dB, so for example -16 corresponds to -1dB.

[RISC OS Open source page](#)

Squash_Compress

SWI &42700 · Squash

The purpose of this call is to provide lossless compression.

Before calling

R0 Compress Flags

R1 Input size of file to be compressed

-1 >= do not return maximum output size

R0 Compress Flags

R1 Workspace pointer

R2 Input pointer

R3 Number of bytes of input available

R4 Output pointer

R5 Number of bytes of output space available

On return

R0 Required workspace size

R1 Maximum output size (if requested), or -1 if not asked or unable to determine

R0 Compress Status

R1 Preserved

R2 Pointer to first unused input byte

R3 Input bytes unused

R4 Pointer to first unused output byte

R5 Unused output space remaining

Notes

Call this SWI initially with **R0** = 8 and **R1** = input file size, using the sizes it reports to allocate workspace and output buffers as appropriate. Then call repeatedly with **R0** bit 3 clear until all the data is processed (as reported via the status bits returned in **R0**). If all the input data can be presented at once (**R0** bits 0,1 clear) and the output buffer is large enough, a faster algorithm is used.

[RISC OS Open source page](#)

Squash-Decompress

SWI &42701 · Squash

The purpose of this call is to provide lossless decompression.

Before calling

R0 Decompress Flags

R1 Input size of file to be decompressed

-1 >= do not return maximum output size

R0 Decompress Flags

R1 Workspace pointer

R2 Input pointer
R3 Number of bytes of input available
R4 Output pointer
R5 Number of bytes of output space available

On return

R0 Required workspace size
R1 Maximum output size (if requested), or -1 if not asked or unable to determine
R0 Decompress Status
R1 Preserved
R2 Pointer to first unused input byte
R3 Input bytes unused
R4 Pointer to first unused output byte
R5 Unused output space remaining

Notes

Call this SWI initially with R0 = 8 and R1 = input file size, using the sizes it reports to allocate workspace and output buffers as appropriate. Note that the current algorithm always returns R1 = -1, so you must get this size by some other means. Then call repeatedly with R0 bit 3 clear until all the data is processed (as reported via the status bits returned in R0). If the input is exhausted (R3 < 12, R0 bit 1 set) the unused input must be resupplied.

If all the input data can be presented at once (R0 bits 0,1 clear) and the output buffer is large enough, a faster algorithm is used.

[RISC OS Open source page](#)

Super_Sample45

SWI &40D81 · SuperSample

This call is for internal use only and must not be used.

Before calling

--

On return

--

[RISC OS Open source page](#)

Super_Sample90

SWI &40D80 · SuperSample

This call is for internal use only and must not be used.

Before calling

--

On return

--

[RISC OS Open source page](#)

TaskManager_EnumerateTasks

SWI &42681 · TaskManager

This call returns information about all active tasks.

Before calling

R0 0 for first call, or value from previous

R1 Pointer to buffer

R2 Size of buffer

On return

R0 Value to use for next call, or <0 for no more

R1 Pointer to end of buffer filled

R2 Bytes not used

Buffer Each active task block contains

+0 Task handle

+4 Pointer to task name (which should be copied if you use it)

+8 Slot size (in bytes)

+12 Flags (Bits 0 and 1)

Bit 0 1=> module task or 0=> application task

Bit 1 1=> slot bar can be dragged or 0=> slot bar fixed

Notes

On exit, the buffer is filled details about each active task using a 4 word block.

[RISC OS Open source page](#)

TaskManager_Shutdown

SWI &42682 · TaskManager

This call initiates a desktop shutdown. The actions performed are controlled by the following shutdown flags:

Before calling

R0 Shutdown types (see below)

On return

--

0 – Meaning when set: Exit to command prompt once shutdown is complete

1 – Meaning when set: Do not broadcast Message_PreQuit

- 2 – Meaning when set: Set flag in CMOS as portable PowerDown
- 3 – Meaning when set: Send a Message_Shutdown
- 4 – Meaning when set: Reject OS_UpCall 1 & 2 reason codes
- 5 – Meaning when set: Reboot machine once shutdown is complete¹
- 6 – Meaning when set: Power off machine once shutdown is complete²
- 7 – Meaning when set: Skip running of PostDesk scripts³
- 8-31 – Meaning when set: Must be 0

Notes

If bit 2 is set, then it is flagged in CMOS that the last shutdown was forced by a low battery, and a message stating this reason will be displayed upon next Desktop start.

1 TaskManager 1.10 (RISC OS 4.01) or TaskManager 1.36 (RISC OS 5.19) onwards

2 TaskManager 1.44 (RISC OS 4.39) or TaskManager 1.36 (RISC OS 5.19) onwards

3 TaskManager 1.37 (RISC OS 5.19) onwards

[RISC OS Open source page](#)

TaskManager_StartTask

SWI &42683 · TaskManager

The purpose of this call is to allow Wimp tasks to be started from outside the desktop or from a non-desktop context (for example, a module). When the desktop is entered or becomes free the Task Manager will execute the *Command via Wimp_StartTask.

Before calling

R0 Pointer to *Command to execute

Notes

The command is queued, so this SWI can be called a number of times.

This SWI was introduced in TaskManager 1.10.

[RISC OS Open source page](#)

TaskManager_TaskNameFromHandle

SWI &42680 · TaskManager

This call returns the description string which the task used when it called Wimp_Initialise.

Before calling

R0 Task handle

On return

R0 Pointer to task name

Notes

This is the string that is displayed in the Task Manager's window. A copy of the string should be made if you wish to keep it.

If a Message_TaskInitialise has just been received by a task, then this SWI may not return a valid name for the same handle since the TaskManager itself may not have received the message yet. As the task name is supplied in the message, this

should not be a problem.

[RISC OS Open source page](#)

TaskWindow_TaskInfo

SWI &43380 · TaskWindow

The purpose of this call is to check if the program is currently running inside a TaskWindow.

Before calling

R0 0 (other reason codes are reserved)

On return

R0 0 if not in TaskWindow, otherwise in a TaskWindow

[RISC OS Open source page](#)

Territory_Alphabet

SWI &43052 · TerritoryManager

The purpose of this call is to return the alphabet number that should be selected for the given territory.

Before calling

R0 Territory number, or -1 for current

On return

R0 Alphabet number used

Notes

Territory_SelectAlphabet must have been issued, before this call will return a given territory.

[RISC OS Open source page](#)

Territory_AlphabetIdentifier

SWI &43053 · TerritoryManager

The purpose of this call is to return an identifier string for the alphabet that should be used for the given territory.

Before calling

R0 Territory number, or -1 to use current territory

On return

R0 Pointer to identifier string for alphabet used

Notes

The identifier of each alphabet is guaranteed to be the same no matter which territory returns it, and to consist of ASCII characters only (ie 7 bit characters).

[RISC OS Open source page](#)

Territory_AlphabetNumberToName

SWI &43045 · TerritoryManager

The purpose of this call is to return the name of the alphabet in the current territory's language.

Before calling

R0 Alphabet number used

R1 Pointer to buffer to hold name of alphabet in current territory

R2 Length of buffer

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

It is not clear from the PRM how this is meant to be achieved given that this SWI does not correspond to an entry point in the territory module. It appears that the territory module has to put in place a Messages file in Resources:\$.Resources.TerrMgr.Messages and the Territory Manager uses that to look up the names.

[RISC OS Open source page](#)

Territory_CharacterPropertyTable

SWI &43056 · TerritoryManager

The purpose of this call is to return a pointer to a character property table.

Before calling

R0 Territory number, or -1 for current

R1 Code for required character property table

On return

R0 Pointer to character property table

Notes

The character property table is 256 bytes in size. Its purpose is to indicate if each character in the given territory's alphabet has a particular property.

If a bit is set, the corresponding character has that property.

[RISC OS Open source page](#)

Territory_Collate

SWI &4305D · TerritoryManager

The purpose of this call is to compare strings in the given territory's alphabet.

Before calling

R0 Territory number, or -1 for current

R1 Pointer to string 1 (null terminated)

R2 Pointer to string 2 (null terminated)

R3 Collate Flags

On return

R0 < 0 if string1 < string2

= 0 if string1 = string2

> 0 if string1 > string2

R1 Preserved

R2 Preserved

R3 Preserved

Collate Flags

Bit / Meaning when set

0 – Meaning when set: Ignore case

1 – Meaning when set: Ignore accents

2 – Meaning when set: Collate naturally (so "File10" > "File9")

3 - 31 – Meaning when set: Reserved, must be zero

[RISC OS Open source page](#)

Territory_ControlTable

SWI &43059 · TerritoryManager

The purpose of this call is to return a pointer to a control character table.

Before calling

R0 Territory number, or -1 for current

On return

R0 Pointer to control table

Notes

A control character table is a 256 byte table giving the value of each character in the given territory's alphabet if it is typed while the Ctrl key is depressed.

Characters that do not have a corresponding control character appear unchanged in the table.

[RISC OS Open source page](#)

Territory_ConvertDateAndTime

SWI &4304B · TerritoryManager

The purpose of this call is to convert a 5 byte UTC time into a string, giving the date and time.

Before calling

- R0** Territory number, or -1 for current
- R1** Pointer to UTC Time (5 bytes in size)
- R2** Pointer to output buffer
- R3** Size of buffer
- R4** Pointer to format string (null terminated)

On return

- R0** Pointer to buffer (R2 on entry)
- R1** Pointer to terminating zero in buffer
- R2** Number of free bytes in buffer
- R3** Pointer to format string
- R4** Preserved

Notes

This call returns the local time, language and alphabet.

The format string is copied directly into the result buffer, except when a '%' character appears. In this case the next two characters are treated as a special field name which is replaced by a component of the current time.

This call replaces the use of OS_ConvertDateAndTime.

[RISC OS Open source page](#)

Territory_ConvertOrdinalsToTime

SWI &43051 · TerritoryManager

The purpose of this call is to convert local time ordinals for the given territory to a 5 byte UTC time.

Before calling

- R0** Territory number, or -1 for current
- R1** Pointer to block of memory containing UTC Time (5 bytes in size)
- R2** Pointer to block of memory containing Date Time Ordinal Block

On return

- R1** Preserved
- R2** Preserved

Notes

Only the first 28 bytes of the ordinal block are used (the "day of week" and "day of year" values are ignored and can be left uninitialised)

[RISC OS Open source page](#)

Territory_ConvertStandardDate

SWI &4304D · TerritoryManager

The purpose of this call is to convert a 5 byte UTC time into a string, giving the date only.

Before calling

- R0** Territory number, or -1 for current
- R1** Pointer to UTC Time (5 bytes in size)
- R2** Pointer to output buffer
- R3** Size of buffer

On return

- R0** Pointer to buffer (R2 on entry)
- R1** Pointer to terminating zero in buffer
- R2** Number of free bytes in buffer
- R3** Preserved

Notes

This call returns the local date, time, language and alphabet.

[RISC OS Open source page](#)

Territory_ConvertStandardDateAndTime

SWI &4304C · TerritoryManager

The purpose of this call is to convert a 5 byte UTC time into a string, giving the date and time.

Before calling

- R0** Territory number, or -1 for current
- R1** Pointer to UTC Time (5 bytes in size)
- R2** Pointer to output buffer
- R3** Size of buffer

On return

- R0** Pointer to buffer (R2 on entry)
- R1** Pointer to terminating zero in buffer
- R2** Number of free bytes in buffer
- R3** Preserved

Notes

This call returns the local date, time, language and alphabet.

[RISC OS Open source page](#)

Territory_ConvertStandardTime

SWI &4304E · TerritoryManager

The purpose of this call is to convert a 5 byte UTC time into a string, giving the time only.

Before calling

- R0** Territory number, or -1 for current
- R1** Pointer to UTC Time (5 bytes in size)
- R2** Pointer to output buffer
- R3** Size of buffer

On return

- R0** Pointer to buffer (R2 on entry)
- R1** Pointer to terminating zero in buffer
- R2** Number of free bytes in buffer
- R3** Preserved

Notes

This call returns the local time for the given territory, and in the local language and alphabet.

[RISC OS Open source page](#)

Territory_ConvertTextToString

SWI &73075 · TerritoryManager

Not yet available.

Before calling

--

On return

--

Notes

This call is not yet implemented, and returns immediately to the caller, with all registers preserved.

[RISC OS Open source page](#)

Territory_ConvertTimeFormats

SWI &43078 · TerritoryManager

This SWI can be used to convert times between different formats and timezones.

Before calling

R0 Territory number, or -1 for current territory

R1 Pointer to input block

R2 Pointer to output block

R3 Conversion flags:

Bits 0-7: Input format

Bits 8-15: Output format

Bit 16: Convert as though DST was in use

Bit 17: If DST calculation unavailable (bit 18), use standard time instead of erroring

Bit 18: Calculate bit 16 automatically

Bit 19: Treat R4 as a signed time adjust in centiseconds

Bits 20-31: Reserved, should be zero

R4 Timezone within territory, or signed number of centiseconds

On return

- Buffer at R2 updated

Input/output formats

/

0 – : Local Date Time Ordinal Block

1 – : Local 5-byte time

2 – : UTC Date Time Ordinal Block

3 – : UTC 5-byte time

4+ – : Reserved for future expansion

Notes

Input ordinals only need to be 28 bytes long (day-of-week and day-of-year not used), output ordinals are 36 bytes long.

Ordinal blocks must be word aligned, while 5-byte time blocks can be byte aligned.

R0, R4, and R3 bits 16-18 are only required when converting between UTC and local time (or vice-versa); if conversion is local-local or UTC-UTC they can be omitted.

This SWI was introduced in TerritoryManager 0.47. For earlier versions, see the list below for the relevant conversion SWIs that are available.

[RISC OS Open source page](#)

Territory_ConvertTimeStringToOrdinals

SWI &43050 · TerritoryManager

The purpose of this call is to convert a time string to time ordinals.

Before calling

R0 Territory number, or -1 for current

R1 Reason code

R2 Pointer to time string

R3 Pointer to word aligned buffer to contain ordinals (as per Date Time Ordinal Block)

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Reason Codes

/ Hex # / String supplied in format

1 – Hex #: &01 · String supplied in format: %24:%MI:%SE

2 – Hex #: &02 · String supplied in format: %W3, %DY-%M3-%CE%YR

3 – Hex #: &03 · String supplied in format: %W3, %DY-%M3-%CE%YR.%24:%MI:%SE

Notes

This SWI returns the time block as a set of useful numbers. Values not passed in the string are assigned -1 in the block.

[RISC OS Open source page](#)

Territory_ConvertTimeToOrdinals

SWI &4304F · TerritoryManager

The purpose of this call is to convert a 5 byte UTC time to local time ordinals for the given territory.

Before calling

R0 Territory number, or -1 for current

R1 Pointer to block of memory containing UTC Time (5 bytes in size)

R2 Pointer to block of memory containing Date Time Ordinal Block

On return

R0 -

R1 Preserved

R2 Preserved

R3 Preserved

Notes

None.

[RISC OS Open source page](#)

Territory_ConvertTimeToUTCOrdinals

SWI &43049 · TerritoryManager

The purpose of this call is to convert a 5 byte UTC time to UTC time ordinals

Before calling

R1 Pointer to block of memory containing UTC Time (5 bytes in size)

R2 Pointer to block of memory containing Date Time Ordinal Block

On return

R1 Preserved

R2 Preserved

Notes

None.

[RISC OS Open source page](#)

Territory_DaylightRules

SWI &43063 · TerritoryManager

This call is for internal use by TerritoryManager, for communication with territory modules. Application code should use Territory_DaylightSaving instead.

Before calling

R1 Reason code

On return

- Dependent on reason code

Reason codes

/

0 – : Check supported

1 – : Get bounds

[RISC OS Open source page](#)

Territory_DaylightRules 0

SWI &43063 · TerritoryManager

This reason code is used to check whether Territory_DaylightRules is supported.

Before calling

R1 0 (reason code)

On return

R0 1 to indicate SWI is supported

Else, V set and R0 is error pointer

Notes

This call is for internal use by TerritoryManager, for communication with territory modules. Application code should use Territory_DaylightSaving instead.

[RISC OS Open source page](#)

Territory_DaylightRules 1

SWI &43063 · TerritoryManager

This reason code is used to determine the start and end times for DST within a given year, as per Territory_DaylightSaving 5.

Before calling

R1 1 (reason code)

R2 Timezone within territory

R3 Year

R4 Pointer to buffer to receive UTC 5-byte time for when DST starts

R5 Pointer to buffer to receive UTC 5-byte time for when DST ends

On return

- Buffers updated and all registers preserved if suitable DST rule was found

Else, V set and R0 is error pointer

Notes

This call is for internal use by TerritoryManager, for communication with territory modules. Application code should use Territory_DaylightSaving instead.

[RISC OS Open source page](#)

Territory_DaylightSaving

SWI &43077 · TerritoryManager

This SWI can be used to perform various operations relating to daylight saving time.

Before calling

R0 Territory number, or -1 for current territory

R1 Reason code

- Other registers dependent on reason code

On return

- Dependent on reason code

Reason codes

/ Action

0 – Action: Read current DST state

1 – Action: Write current DST state

2 – Action: Check AutoDST support

3 – Action: Apply DST rule to UTC ordinals

4 – Action: Apply DST rule to UTC 5-byte time

5 – Action: Get DST bounds for given year

Notes

This SWI was introduced in TerritoryManager 0.47

[RISC OS Open source page](#)

Territory_DaylightSaving 0

SWI &43077 · TerritoryManager

This SWI can be used to read the current DST status, as per “*Status DST”.

Before calling

R0 -1, or current territory number

R1 0 (reason code)

On return

R0 DST state flags:

Bit 0: DST on/off flag

Bit 1: Automatic DST switching on/off flag

Notes

Unlike most Territory_DaylightSaving reason codes, this call only supports reading the setting for the current territory. Thus R0 must either be -1 or the number of the current territory.

This SWI was introduced in TerritoryManager 0.47

[RISC OS Open source page](#)

Territory_DaylightSaving 1

SWI &43077 · TerritoryManager

This SWI can be used to set the current DST status. It is equivalent to the following *Configure commands:

Before calling

R0 -1, or current territory number

R1 1 (reason code)

R2 DST state flags:

Bit 0: DST on/off flag

Bit 1: Automatic DST switching on/off flag

Other bits reserved, should be zero

On return

- All registers preserved

DST flags / Command

00 – Command: *Configure NoDST

01 – Command: *Configure DST

1x – Command: *Configure AutoDST (bit 0 is ignored if bit 1 is set)

Notes

Unlike most Territory_DaylightSaving reason codes, this call only supports the current territory. Thus R0 must either be -1 or the number of the current territory.

This SWI was introduced in TerritoryManager 0.47

[RISC OS Open source page](#)

Territory_DaylightSaving 2

SWI &43077 · TerritoryManager

This SWI can be used to check whether the indicated territory supports automatic DST switching.

Before calling

R0 Territory number, or -1 for current territory

R1 2 (reason code)

On return

R0 1 if this territory supports AutoDST use

0 or an error if AutoDST unsupported

Notes

Even if automatic DST switching is supported, there is no guarantee that it is supported for all time periods. For example historic times prior to the release of the specific territory module, or far-future times, might not be supported.

Territory_DaylightSaving 5 can be used to discover which years have DST data available.

This SWI was introduced in TerritoryManager 0.47

[RISC OS Open source page](#)

Territory_DaylightSaving 3

SWI &43077 · TerritoryManager

This SWI can be used to check whether a UTC date/time, specified as ordinals, is subject to daylight saving time for a given territory + timezone.

Before calling

- R0** Territory number, or -1 for current territory
- R1** 3 (reason code)
- R2** Timezone within territory
- R3** Pointer to 28-byte UTC Date Time Ordinal Block

On return

- R0** DST state flags:
 - Bit 0: True/false for whether the given date/time lies within a DST period

Notes

The day-of-week and day-of-year members of the ordinal block are unused and do not need to be present.

This SWI was introduced in TerritoryManager 0.47

[RISC OS Open source page](#)

Territory_DaylightSaving 4

SWI &43077 · TerritoryManager

This SWI can be used to check whether a UTC date/time, specified as a 5-byte RISC OS timestamp, is subject to daylight saving time for a given territory + timezone.

Before calling

- R0** Territory number, or -1 for current territory
- R1** 3 (reason code)
- R2** Timezone within territory
- R3** Pointer to 5-byte UTC time

On return

- R0** DST state flags:
 - Bit 0: True/false for whether the given date/time lies within a DST period

Notes

This SWI was introduced in TerritoryManager 0.47

[RISC OS Open source page](#)

Territory_DaylightSaving 5

SWI &43077 · TerritoryManager

This SWI can be used to determine when DST starts and stops for a given year.

Before calling

- R0** Territory number, or -1 for current territory
- R1** 5 (reason code)
- R2** Timezone within territory
- R3** Year, or -1 for current
- R4** Pointer to buffer to receive UTC 5-byte time for when DST starts
- R5** Pointer to buffer to receive UTC 5-byte time for when DST ends

On return

- Buffers at R4 and R5 updated

Notes

The returned time values will be within the indicated year. Depending on the territory, this may result in the end time being before the start time; e.g. territories in the southern hemisphere.

An error will be returned if the territory module does not have sufficient knowledge of the DST rules for the indicated year (error &197, "No rule for given timezone")

This SWI was introduced in TerritoryManager 0.47

[RISC OS Open source page](#)

Territory_Deregister

SWI &43042 · TerritoryManager

The purpose of this call is to remove the given territory from the list of active territories.

Before calling

- R0** Territory number

On return

- R0** Preserved

Notes

A territory module must issue this call from its finalisation entry point when it is killed.

[RISC OS Open source page](#)

Territory_Exists

SWI &43044 · TerritoryManager

The purpose of this call is to check if the given territory is currently present.

Before calling

R0 Territory number

On return

R0 Preserved

Notes

The Z flag is set if the territory is loaded (i.e. is EQ).

[RISC OS Open source page](#)

Territory_IME

SWI &43062 · TerritoryManager

This call is used to determine which IME? should be used for a given territory.

Before calling

R0 Territory number, or -1 for current

On return

R0 SWI chunk of IME?, or 0 if no IME is available

Notes

The InternationalIME? module uses this SWI to determine what the default IME should be for each territory.

[RISC OS Open source page](#)

Territory_LowerCaseTable

SWI &43057 · TerritoryManager

The purpose of this call is to return a pointer to a lower case table.

Before calling

R0 Territory number, or -1 for current

On return

R0 Pointer to lower case table

Notes

This call returns a pointer to a lower case table, which is a 256 byte table giving the lower case version of each character in the given territory's alphabet. Characters that do not have a lower case version (eg numbers, punctuation) appear unchanged in the table.

[RISC OS Open source page](#)

Territory_NameToNumber

SWI &43060 · TerritoryManager

The purpose of this call is to return the number of the given territory.

Before calling

R0 Territory number, or -1 for current

R1 Pointer to territory name in alphabet of territory (null terminated)

On return

R0 Territory number, or 0 if unknown

Notes

This call accepts the name of one territory in the language of another territory (probably – but not necessarily – different). It returns the number of the named territory

[RISC OS Open source page](#)

Territory_Number

SWI &43040 · TerritoryManager

The purpose of this call is to return the territory number of the current territory.

Before calling

--

On return

R0 Current Territory number

Notes

None.

[RISC OS Open source page](#)

Territory_NumberToName

SWI &43043 · TerritoryManager

The purpose of this call is to return the name of the given territory in the current territory's language.

Before calling

R0 Territory number

R1 Pointer to buffer to contain territory name

R2 Length of buffer

On return

R1 Preserved

Notes

It is not clear from the PRM how this is meant to be achieved given that this SWI does not correspond to an entry point in the territory module. It appears that the territory module has to put in place a Messages file in `Resources:$.Resources.TerrMgr.Messages` and the Territory Manager uses that to look up the names.

[RISC OS Open source page](#)

Territory_PlainTable

SWI &4305A · TerritoryManager

The purpose of this call is to return a pointer to an unaccented character table.

Before calling

R0 Territory number, or -1 for current

On return

R0 Pointer to unaccented character table

Notes

A character property table is a 256 byte table giving the unaccented version of each character

[RISC OS Open source page](#)

Territory_ReadCalendarInformation

SWI &4305F · TerritoryManager

The purpose of this call is to return various information about the given territory's calendar.

Before calling

R0 Territory number, or -1 for current

R1 Pointer to block of memory containing UTC Time (5 bytes in size)

R2 Pointer to block of memory (12 words in size)

On return

R0 Preserved

R1 Preserved

R2 Preserved

Output Buffer

0 – Value: Number of first working day in the week

4 – Value: Number of last working day in the week

8 – Value: Number of months in the current year

12 – Value: Number of days in the current month

16 – Value: Maximum length of AM/PM string

- 20 – Value: Maximum length of WE string
- 24 – Value: Maximum length of W3 string
- 28 – Value: Maximum length of DY string
- 32 – Value: Maximum length of ST string (may be zero)
- 36 – Value: Maximum length of MO string
- 40 – Value: Maximum length of M3 string
- 44 – Value: Maximum length of TZ string

Notes

An explanation of all the different Date format strings tokens is available.

[RISC OS Open source page](#)

Territory_ReadCurrentTimeZone

SWI &43048 · TerritoryManager

The purpose of this call is to return information on the current time zone.

Before calling

--

On return

- R0 Pointer to name of the current time zone (null terminated)
- R1 Offset from UTC to current time zone in centi-seconds (signed 32-bit)

Notes

None.

[RISC OS Open source page](#)

Territory_ReadSymbols

SWI &4305E · TerritoryManager

The purpose of this call is to convert local time ordinals for the given territory to a 5 byte UTC time.

Before calling

- R0 Territory number, or -1 for current
- R1 Reason code

On return

- R0 Requested Value

Reason Codes

- 0 – Meaning: Pointer to decimal point string
- 1 – Meaning: Pointer to thousands separator
- 2 – Meaning: Pointer to byte list containing size of each group of digits in formatted non-monetary quantities (least significant first)
- 255 = no further grouping
- 0 = repeat last grouping for rest of number

other = size of current group; next byte contains size

3 – Meaning: Pointer to international currency symbol

4 – Meaning: Pointer to local currency symbol

5 – Meaning: Pointer to monetary decimal point string

6 – Meaning: Pointer to monetary thousands separator

7 – Meaning: Pointer to byte list containing size of each group of digits in formatted monetary quantities (least significant first)

255 = no further grouping

0 = repeat last grouping for rest of number

other = size of current group; next byte contains size

8 – Meaning: Pointer to positive sign used for monetary quantities

9 – Meaning: Pointer to negative sign used for monetary quantities

10 – Meaning: Number of fractional digits to be displayed in a formatted international monetary quantity

11 – Meaning: Number of fractional digits to be displayed in a formatted monetary quantity

12 – Meaning: Return for a non-negative formatted monetary quantity

0 = currency symbol precedes value

1 = currency symbol follows value

13 – Meaning: Return for a non-negative formatted monetary quantity

0 = a space separates currency symbol and value

1 = symbol and value not separated

14 – Meaning: Return for a negative formatted monetary quantity

0 = currency symbol precedes value

1 = currency symbol follows value

15 – Meaning: Return for a negative formatted monetary quantity

0 = a space separates currency symbol and value

1 = symbol and value not separated

16 – Meaning: Return for a non-negative formatted monetary quantity

0 = parentheses surround symbol and value

1 = the sign string precedes value and symbol

2 = the sign string follows value and symbol

3 = the sign string immediately precedes the symbol

4 = the sign string immediately follows the symbol

17 – Meaning: Return for a negative formatted monetary quantity

0 = parentheses surround symbol and value

1 = the sign string precedes value and symbol

2 = the sign string follows value and symbol

3 = the sign string immediately precedes the symbol

4 = the sign string immediately follows the symbol

18 – Meaning: Return pointer to list separator

Notes

None.

[RISC OS Open source page](#)

Territory_ReadTimeZones

SWI &4304A · TerritoryManager

The purpose of this call is to return information on the time zones for the given territory.

Before calling

R0 Territory number, or -1 for current

On return

R0 Pointer to name of standard time zone for territory

R1 Pointer to name of summer time

R2 Offset from UTC to standard time, in centiseconds (signed 32-bit)

R3 Offset from UTC to summer time, in centiseconds (signed 32-bit)

Notes

None.

[RISC OS Open source page](#)

Territory_Register

SWI &43041 · TerritoryManager

The purpose of this call is to add the given territory to the list of active territories.

Before calling

R0 Territory number

R1 Pointer to table of entry points

R2 Value of R12 on entry to territory

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

This call adds the given territory to the list of active territories, making it available for application programs.

A territory module must issue this call from its initialisation entry point when it is initialised, and whenever it receives the service call `Service_TerritoryManagerLoaded`.

The table pointed to by R1 should contain 43 entries, each of which is a pointer to code to handle one of the SWIs that – although in the territory manager SWI chunk – are actually handled by a territory module. The first entry corresponds to the SWI &4304A, the second to SWI &4304B, and so on through to the last entry which is for SWI &43074.

Note that SWIs &43075 to &4307F in the Territory manager do not have any corresponding entry points in the table. They are therefore only suitable for functionality which can be implemented by calling the other entry points, or for management functions like `Territory_Select`.

Some of these SWI numbers (currently from &43064 to &43074) are reserved for future expansion and so territory modules cannot implement them. You must nevertheless provide code for each of these entry points, which must return error &43040 with the text 'Unknown Territory SWI' or a suitable translation. You must not just return without error, as there would

then be no way for a future Territory manager to tell whether your territory module supports any new entry points which have come into use.

[RISC OS Open source page](#)

Territory_RepresentationTable

SWI &4305C · TerritoryManager

The purpose of this call is to return a pointer to a numeric representation table.

Before calling

R0 Territory number, or -1 for current

On return

R0 Pointer to numeric representation table

Notes

A character property table is a 16 byte table giving the 16 characters in the alphabet which should be used to represent the values 0-15.

[RISC OS Open source page](#)

Territory_Select

SWI &43076 · TerritoryManager

This SWI selects the given territory, if it exists.

Before calling

R0 Territory number

On return

- All registers preserved

Notes

Issuing this call will not automatically change any language, keyboard or alphabet settings - it simply changes the Territory Manager's record of the current territory, then issues Service_TerritoryStarted.

This call was introduced in TerritoryManager 0.34

[RISC OS Open source page](#)

Territory_SelectAlphabet

SWI &43046 · TerritoryManager

The purpose of this call is to select the correct alphabet for the given territory.

Before calling

R0 Territory number, or -1 for current

On return

R0 Preserved

Notes

None.

[RISC OS Open source page](#)

Territory_SelectKeyboardHandler

SWI &43054 · TerritoryManager

The purpose of this call is to select the keyboard handler for the given territory.

Before calling

R0 Territory number, or -1 for current

On return

--

Notes

None.

[RISC OS Open source page](#)

Territory_SetTime

SWI &43047 · TerritoryManager

The purpose of this call is to set the clock to a given 5 byte UTC time.

Before calling

R0 Pointer to block of memory containing UTC Time (5 bytes in size)

On return

--

Notes

None.

[RISC OS Open source page](#)

Territory_TransformString

SWI &43061 · TerritoryManager

The purpose of this call is to transforms a string to allow direct territory independent string comparison.

Before calling

R0 Territory number, or -1 for current

R1 Pointer to buffer to hold transformed string

R2 Pointer to source string

R3 Length of buffer, or 0 to find required length

On return

R0 Length of transformed string (excluding terminating null)

Notes

Transforms a string such that a byte-for-byte comparison of two output strings gives the same results as calling Territory_Collate on two source strings.

[RISC OS Open source page](#)

Territory_UpperCaseTable

SWI &43058 · TerritoryManager

The purpose of this call is to return a pointer to an upper case table.

Before calling

R0 Territory number, or -1 for current

On return

R0 Pointer to upper case table

Notes

A character property table is a 256 byte table with the upper case version of each character in the alphabet.

[RISC OS Open source page](#)

Territory_ValueTable

SWI &43058 · TerritoryManager

The purpose of this call is to return a pointer to a numeric value table.

Before calling

R0 Territory number, or -1 for current

On return

R0 Pointer to numeric value table

Notes

A character property table is a 256 byte table giving the numeric value of each character in the given territory's alphabet. This includes non-decimal numbers, hence F and f have a value of 15, while Z has no value.

[RISC OS Open source page](#)

Territory_WriteDirection

SWI &43055 · TerritoryManager

The purpose of this call is to return the direction of writing used in the given territory.

Before calling

R0 Territory number, or -1 for current

On return

R0 Write direction bitfield

Write Direction

Bit / Value / Meaning

- 0** – Value: 0 · Meaning: Writing goes from left to right
- 1** · Writing goes from right to left
- 1** – Value: 0 · Meaning: Writing goes from top to bottom
- 1** · Writing goes from bottom to top
- 2** – Value: 0 · Meaning: Lines of text are horizontal
- 1** · Lines of text are vertical

Notes

None.

[RISC OS Open source page](#)

URL_Deregister

SWI &83E06 · URL_Fetcher

This call deregisters the client session from the URL module, freeing up any information the URL module may have kept about the client session (e.g. proxy information). The session identifier ceases to be valid and becomes available for re-issue on a subsequent call to SWI URL_Register.

Before calling

R0 Flags:

Bits 0-31 · reserved (0)

R1 Session identifier

On return

R0 Status word (see SWI URL_Status for details)

All other registers preserved

In practice

Deregister a client session with the URL module.

Interrupts

Interrupt status undefined

Re-entrancy

SWI is not re-entrant.

In practice

When this SWI is called, the URL module invokes the Protocol_Stop SWI for the protocol module concerned, if it has not already done so (e.g. during the processing of URL_Stop).

Related SWIs

- URL_Register
- URL_Stop
- Protocol_Stop

Related vectors

None

[RISC OS Open source page](#)

URL_EnumerateProxies

SWI &83E09 · URL_Fetcher

This call is used to discover which URLs proxies are set for on a per session or global basis, or which URLs are not to be proxied. The information pointed to by R3 and R4 where applicable is a copy of that which was passed to SWI URL_SetProxy when the setting was made.

Before calling

R0 Flags:

Bit 0 => · If set, enumerate the no-proxy list

Bits 31..1 · Reserved (0)

R1 Session identifier (or zero for global proxies/no-proxies)

R2 context (0 for first call)

On return

R0 Status flags (currently unused)

R1 Preserved

R2 Context for next call (-1 if finished)

R3 If R0:0 clear: · Pointer to read-only URL to proxy (if R2 is not -1)

If R0:0 set: · Pointer to read-only URL to not proxy (if R2 is not -1)

R4 If R0:0 clear: · Pointer to read-only proxy URL information (if R2 is not -1)

If R0:0 set: · Corrupted, contains no useful information

All other registers preserved

In practice

Enumerate proxies or no-proxy URLs.

Interrupts

Interrupt state is undefined.

Re-entrancy

SWI is not re-entrant.

In practice

If R0:0 is set on entry, then R4 will be corrupted on exit and may not contain a meaningful value.

URL will not cope gracefully if the proxy list is updated between calls to this SWI (you may get duplicate entries or miss some out).

Related SWIs

- URL_SetProxy

Related vectors

None

[RISC OS Open source page](#)

URL_EnumerateSchemes

SWI &83E08 · URL_Fetcher

This call is used to discover which schemes are currently available to the URL module. It may be used, for example, to determine whether or not a client of the URL module may deal with a given URL (in combination with SWI URL_ParseURL to extract the scheme) and if not, pass it to the Acorn URI handler to see if anything else in the system can deal with it (see the Acorn URI Handler Functional Specification, 1215,215/FS).

Before calling

R0 Flags:

Bits 0-31 · All bits currently reserved (must be zero)

R1 Context (0 for first call)

On return

R0 Status flags (currently unused)

R1 Context for next call (-1 if finished)

R2 Pointer to read-only URL fetch scheme (if R1 is not -1)

R3 Pointer to read-only help string (if R1 is not -1)

R4 Protocol module SWI base (if R1 is not -1)

R5 Protocol module version (*100, if R1 is not -1)

All other registers preserved

In practice

Enumerate available fetch schemes.

Interrupts

Interrupt state is undefined

Re-entrancy

SWI is not re-entrant.

In practice

URL will not cope gracefully if the protocol module list is updated between calls to this SWI (you may get duplicate modules or miss some out).

[RISC OS Open source page](#)

URL_GetURL

SWI &83E01 · URL_Fetcher

This SWI is used to instigate a transfer of data to or from (mainly from) a resource server. When this SWI is called, the URL module checks the per-session and global proxy settings, looking for a match (see SWI URL_SetProxy for details on setting proxies and proxy conflict resolution). If no proxy is to be used, then URL looks for a protocol module which is capable of handling the URL specified by R3. If a proxy setting is found, then a pointer to the proxy URL is placed in R7, R0:31 is forced to value 1, and URL looks for a protocol module which is capable of handling the specified proxy URL. In both cases, if a suitable module cannot be located, the URL module generates an error. If a protocol module capable of handling the URL is found, then all client registers are passed onto the protocol module via the Protocol_GetData SWI call with the exceptions stated above for proxy handling. On exit, R0 holds the status code returned by the protocol module.

Before calling

R0 Flags: ·

Bit 0 => · R6 is valid ·

Bit 1 => · R5 holds length of data in R4 specified buffer, otherwise a single NUL terminated string in buffer

Bits 29..2 · reserved (0) ·

Bit 30 => · Prevent AcornHTTP fetcher from adding cookies to the request. ·

Bits 31 · reserved (0) ·

R1 Session identifier

R2 Bits 7..0 => · Method

Bits 15..8 · Method dependent

Bits 31..16 · Reserved (0)

R3 URL – The resource that we are requesting including the protocol, e.g. "http://www.bbc.co.uk"

R4 Data block: Data to send in addition to the URL. Validity is protocol and method dependent.

R5 If R0:1 is set, length of data in block pointed to by R4. If R0:0 is clear, then 0=return body only, 1=return header only, 2=return both.

R6 User Agent: Pointer to string to use as "User Agent" identifier in request header if R0:0 is set. A NULL pointer or NULL string implies use default identifier – see below.

On return

R0 Protocol status (as defined for URL_Status)

All other registers preserved

In practice

Instigate data transfer from / to a resource server.

Interrupts

Interrupt state is undefined.

Re-entrancy

SWI is not re-entrant.

In practice

The extra data pointed to by R4 on entry is method and protocol specific. For example, in HTTP, the data comprises HTTP headers and, if appropriate, an entity body. Protocol modules should use this style wherever possible. Note that these headers do not include lines such as an HTTP Request-Line (ie. the "GET / HTTP/1.1" part. For example, when posting data to an HTTP URL as the result of a form submission on a web page, the web browser would supply a Content-Type header, Content-Length header, potentially some kind of encoding header, a blank line and then the entity body.

The User Agent string pointed to by R6 if R0:0 is set, is in indication to the underlying protocol module of how the module should identify itself to remote systems. This controls the User-Agent header for the HTTP protocol module, for example. The protocol module is free to define its default identifier as it pleases, however, following the format of the HTTP User-Agent is recommended where possible and appropriate to the protocol. Modules may choose to ignore or amend any User-Agent string. For example, the AcornHTTP module will suffix the client's User-Agent with its own version number, resulting in complete identifiers such as:

where the client only specified "Acorn Browse/2.06".

FTP · HTTP and others · Comment

- 1 · RETR/LIST · GET · "Get this object" operation
- 2 · n/a · HEAD · "Get entity headers" operation
- 3 · n/a · OPTIONS · "Get server options" operation
- 4 · n/a · POST · "HTTP POST" operation
- 5 · n/a · TRACE · "HTTP TRACE" operation
- 6 · n/a · n/a · Reserved – do not use
- 7 · n/a · n/a · Reserved – do not use
- 8 · STOR · PUT · "Store this object" operation
- 9 · MKD · n/a · "Create directory" operation
- 10 · RMD · n/a · "Remove directory" operation
- 11 · RNFR/RNTO · n/a · "Rename object" operation
- 12 · DELE · DELETE · "Delete object" operation
- 13 · STOU · n/a · "Store object unique" operation

The AcornHTTP module provides a mechanism for the temporary registration of additional methods for use with HTTP or HTTPS. See HTTP_RegisterMethod.

Applications for new permanent method codes should be made to ROOL Allocations. The range 128-254 is reserved for private non-distributed modules. Method numbers 0 and 255 are reserved and must not be used.

Notes

The outgoing request may be augmented by the underlying protocol module. For example, AcornHTTP maintains a global cookie store and will transmit to the server any relevant cookies automatically, unless R0 bit 30 is set.

[RISC OS Open source page](#)

URL_ParseURL

SWI &83E07 · URL_Fetcher

This SWI is used to parse URLs into their constituent parts, enabling clients to extract the various fields from the URL in a reliable manner. The call is also capable of resolving a relative URL to produce a fully-qualified URL, and of reconstructing a full URL from a set of components.

Before calling

R0 Flags:

Bit 0 => · If set, R5 contains number of words in data block, else a default of 10 words is assumed.

Bit 1 => · If set, character codes 0 to 31 and 127 in the URL will be escaped (hex encoded, e.g. space becomes '%20') – only available in URL 0.42 or later. URL 0.38 through to 0.41 inclusive always escape these characters. Versions prior to 0.38 never do this.

Bits 31..2 · Reserved (0)

R1 Reason code:

0 => · Return component buffer requirements
 1 => · Return component data in specified buffers
 2 => · Construct full URL from component buffers
 3 => · 'Quick parse'

R2 Pointer to base URL

R3 Pointer to URL relative to base URL (or NULL if none)

R4 Pointer to data block of R5 words (unless R1=3, see below, or R0:0 is unset, in which case R4 points to a buffer of at least 10 words in length)

R5 If R0:0 set, size of R4 block in words

On return

R0 Flags:

Bits 31-0 Reserved (0)

All other registers preserved

SWI is not re-entrant

Interrupt status undefined

Data block at R4 is updated in line with entry reason code

In practice

Parse URLs to / from their constituent parts.

If R3 is non-NULL, it is assumed to point to a partial URL which needs to be resolved with respect to the base URL pointed to by R2. If R3 is NULL, then R2 is assumed to point to a full URL.

The data block referred to above is either a block of integers which will be updated to contain the size of the required buffer for each element, or a block containing pointers to buffers for the actual data.

All strings are zero-terminated and all lengths include space for the zero terminator.

The number of entries in the block is specified in R5 if R0:0 is set on entry. If R0:0 is clear, then the default value of 10 is assumed. The format of the data block is:

+0 – Usage: Fully canonicalised URL
 +4 – Usage: URL protocol (e.g. "http", "ftp") forced to lower-case
 +8 – Usage: Hostname (e.g. "www.acorn.com") forced to lower-case
 +12 – Usage: Port (e.g. "80")
 +16 – Usage: Username – used for FTP authentication and mailto
 +20 – Usage: Password – for FTP
 +24 – Usage: Account – for FTP
 +28 – Usage: Path (e.g. "pub/riscos/releases") [See note]
 +32 – Usage: Query – for HTTP, things after a query character
 +36 – Usage: Fragment – for HTTP, things after a hash character

It is anticipated that this SWI will be called twice: the first time to find the lengths of the buffers, and the second to retrieve a copy of the data into the buffers. The URLs pointed to by R2 and R3 (if used) need not be fully-qualified. e.g. R2 may point to "www.acorn.com/browser/". The fully canonicalised version of the URL at block+0 refers to a fully-qualified, canonicalised version of it, which in this example would be "http://www.acorn.com/browser/".

During canonicalisation, the port number will be elided if possible. See the discussion under SWI URL_ProtocolRegister for details of how URL discovers whether this is possible or not.

[Note] The path will not start with a '/' unless the URL being parsed explicitly specified one – this is in keeping with the URL specification, so for example, given the URL "http://www.acorn.com/browser/", then the path component is "browser/", and not "/browser/"; the slash between the hostname and path is a separator only, not a part of either component.

The entry reason codes are described below.

URL_ParseURL_ReturnLengths (R1 = 0)

Work out space required for URL components.

When R1 is 0 on entry to the SWI, the data block is treated as a block of unsigned 32-bit integers. The contents of the block are ignored on entry, but on exit are filled in with the lengths of the individual components of the URL. A value of zero is stored for a field which does not exist; non-zero values include space for a zero-byte terminator.

URL_ParseURL_ReturnData (R1 = 1)

Split a URL into its component parts.

When R1 is 1 on entry to the SWI, the data block is treated as a block of pointers to buffers to receive the components of the URL. Each of the pointers in the data block must be either zero, indicating that the caller is not interested in that field, or point to a buffer which is sufficiently long to receive the field. The client can ensure this by having previously used reason code 0 to determine the length required.

URL_ParseURL_ComposeFromComponents (R1 = 2)

Combine the components of a URL.

When R1 is 2 on entry to the SWI, the data block is treated as containing the broken down fields of a URL. Each of the pointers in the data block must be either zero or point to a buffer containing the null-terminated value of the component, with the exception of the full URL field, which is a pointer to a buffer to receive the fully canonicalised URL. This buffer is filled in on exit.

URL_ParseURL_QuickResolve (R1 = 3)

Quickly obtain a fully resolved URL.

When R1 is 3 on entry to the SWI, R4 points to a buffer for receiving the fully resolved URL. R5 is the length of the buffer. On exit, the buffer is filled in with the fully resolved URL obtained, and R5 is decreased by the length of the URL (including terminating zero byte). Hence R5 will be negative on exit if the buffer wasn't large enough. There is no fixed rule for calculating the minimum buffer length required for the answer. To guarantee that the buffer is large enough, it should be calculated as:

If R0:1 is set on entry, there is the potential for up to the entire URL to be hex encoded. In this case, you would need to multiply the above by three. URL 0.37 and earlier never hex encodes URLs. Note that URL 0.38, 0.39, 0.40 and 0.41 will always do this; the control through R0:1 was introduced in v0.42. Clients not knowing about this bit (therefore leaving R0:1 unset) will find that 0.42 or later do not automatically escape URLs, this being more sensible default behaviour on the whole.

Characters which are already hex encoded in URLs are left alone in all versions of the URL module.

Clients are strongly recommended to use this reason code if they wish to resolve a relative URL or canonicalise a URL and are only interested in the fully resolved and canonicalised form of the URL, since it is significantly faster than using reason code 0 and then reason code 1. To help reduce the chances of wildly over-allocating buffer space, setting of R0:1 is not recommended unless full hex escaping is definitely required.

[RISC OS Open source page](#)

URL_ProtocolDeregister

SWI &83E21 · URL_Fetcher

This call should be used by the protocol module to tell the URL module that it is no longer available. The URL module will raise the appropriate disconnect messages with its clients, and tell the protocol module the number of clients that were affected.

Before calling

R0 Flags: ·

Bits 0-31 · Reserved (0)

R1 SWI base

On return

R0 Flags: ·

Bits 0-31 · Reserved (0)

R1 Number of client sessions that were using this module

All other registers preserved

In practice

Deregister a protocol module from the URL module.

Interrupts

Interrupt state is undefined.

Re-entrancy

SWI is not re-entrant

In practice

Typically, it will be called during a protocol module's finalisation code. If the protocol module is deregistered successfully, then URL will issue a service call `Service_URLProtocolModule_ProtocolModule?` to inform any interested modules.

Related SWIs

- URL_ProtocolRegister

Related vectors

None

[RISC OS Open source page](#)

URL_ProtocolRegister

SWI &83E20 · URL_Fetcher

This call is used by a protocol fetcher module to register its SWI base and the type of URL that it accepts with the URL module. The SWIs that are accessible from this SWI base are defined in the Protocol module interface documentation. If the module cannot be registered (e.g. another module is already claiming that URL base), then an error will be returned. R3 is an integer version number and R4 is a pointer to a string containing more information which will be displayed by the *URLProtoShow command (or 0 if no descriptive text is provided).

Before calling

R0 Flags:

Bit 0 => · If set, R5 contains protocol flags word

Bit 1 => · If set, R6 contains the default port number

Bits 31..2 · Reserved (0)

R1 Protocol module's SWI base

R2 URL fetch scheme supported e.g. "http:" etc.

R3 Version number * 100 e.g. 116 => version 1.16

R4 Informational string. Up to 50 characters of descriptive text, e.g. "Acorn HTTP fetcher"

R5 Protocol flags word, if R0:0 set. See below

R6 Default port number, if R0:1 set. See below

On return

R0 Flags:

Bits 31..0 · Reserved (0)

All other registers preserved

SWI is not re-entrant.

Interrupt status is undefined

In practice

Register a protocol module with the URL module.

Typically, it will be called during a protocol module's initialisation code or on a callback set from the module's initialisation code. If the protocol module is registered successfully, then URL will issue a service call

Service_URLProtocolModule_ProtocolModule? to inform any interested modules.

If R0:0 is set, then R5 contains a protocol flags word. This is used to describe to URL how the resolver should treat URLs from this scheme. The current bits defined are:

Bit / Meaning when set

0 – Meaning when set: Path is not UNIX-like

1 – Meaning when set: No parsing should be performed on this scheme

2 – Meaning when set: Scheme allows "user@" to precede the hostname component

3 – Meaning when set: Hash (ASCII 35) allowed in hostname (e.g. for file: URLs)

4 – Meaning when set: No hostname component (e.g. mailto: URLs)

5 – Meaning when set: Remove leading "." components in pathname.

Note that the meanings of set bits are such that zero is a reasonable value to pass for unknown schemes. Note that if URL is requested to resolve URLs using schemes unknown to it, it will assume a protocol flags word value of zero. This may lead to inconsistent behaviour depending on whether the protocol module is loaded or not.

If R0:1 is set, then R6 contains the default port number for this scheme. This is used by the URL resolving code to determine if explicitly specified port numbers can be elided from the URL. For example, when constructing the canonicalised form of "http://www.acorn.com:80/", the port bit is dropped as it serves no useful purpose, leaving "http://www.acorn.com/".

The URL module is primed with knowledge of the following protocols:

It is not necessary for modules implementing those protocols to set either flag bit and hence there is no need for them to set R5 or R6.

[RISC OS Open source page](#)

URL_ReadData

SWI &83E03 · URL_Fetcher

This SWI is used to read the data pending from a request, find out how much data has been read on this call and how much more there is remaining to be read for the request. R2 is a pointer to a buffer on entry (and R3 is the size of the buffer), on exit the buffer contains the new data, R4 contains the amount of data written to the buffer and R5 contains the amount of data left to be read. If the amount of data left is unknown R5 will contain -1.

Before calling

R0 Flags:

Bits 31..0 Reserved (0)

R1 Session identifier

R2 Client buffer for received data

R3 Size of buffer pointed to by R2

On return

R0 Status word (see SWI URL_Status) with the addition of protocol-specific flags (see below)

R2 Preserved. Contents of buffer modified

R4 Number of bytes transferred to R2 buffer

R5 Number of bytes still to be read to complete object (if known) or -1 if unknown

All other registers preserved

SWI is not re-entrant

Interrupt status undefined

In practice

Read data pending from a request.

R0 always returns the protocol status code (see URL_Status) possibly with the addition of protocol-specific flags. For example, for HTTP and HTTPS requests, the underlying AcornHTTP protocol module will set bit 16 if there are cookies in the queue. These can be read using HTTP_EnumerateCookies.

In the event of all the data being read (R5 = 0 on exit), a call to URL_Stop is not required as this is performed automatically when URL_Deregister is called for the client session. Once all data has been read a call to URL_Status can return no meaningful information, simply indicating that the transfer has completed.

The data returned will take the form of a complete HTTP compatible response. Responses should use HTTP/1.0 if possible and avoid HTTP/1.1. For example, AcornHTTP will downgrade any higher version responses to HTTP/1.0, having taken care to remove any features applicable only to the higher version, such as chunked transfer encodings.

When this SWI is called, the URL module invokes the Protocol_ReadData SWI for the protocol module concerned with the request.

[RISC OS Open source page](#)

URL_Register

SWI &83E00 · URL_Fetcher

This SWI initialises a client session with the URL module and provides the client with a session identifier that can be used to monitor the status of the URL module within that client's context. The session identifier is unique for each client session that is registered with URL and is also used as an identifier in subsequent interactions with the URL module.

Before calling

R0 Flags:

Bits 31-0 reserved (0)

On return

R0 Reserved - currently 0

R1 Session identifier

All other registers preserved

In practice

Initialise a client session with the URL module.

Interrupts

Interrupt state is undefined.

Re-entrancy

SWI is not re-entrant.

In practice

Multiple registration by the same client application is permitted. This will provide the client with multiple identifiers to the URL module. Calling this SWI does not result in the calling of any protocol module SWIs.

The URL module imposes no limit on the number of concurrently registered sessions, other than having the required memory available in which to store details of the session.

[RISC OS Open source page](#)

URL_SetProxy

SWI &83E04 · URL_Fetcher

This call is used to set up a proxy server to use for a session with the URL module. If R1 is zero then the proxy is considered global and is used for all sessions. If R1 is a valid session identifier then the proxy server for that session only is set. R2 is a pointer to a string containing the base URL to pass the request on to when a proxy request is made. This is of the form "http://www-cache.demon.co.uk:8080/" (note the trailing '/'). A common error is to omit the port number. If the port number is not specified, then the default port number is used. See discussion under URL_ProtocolRegister regarding how the default port number is derived.

Before calling

R0 Flags:

Bits 0-31 reserved (0)

R1 Session identifier

R2 Address of buffer containing a URL base

R3 URL 'method' to proxy (address of URL fetch identifier to be proxied)

R4 0 => · Proxy request

1 => · Don't proxy request

· All other values reserved

On return

R0 Status word (see SWI URL_Status for details)

All other registers preserved

In practice

Set up a proxy server for a session with the URL module.

Interrupts

Interrupt state is undefined.

Re-entrancy

SWI is not re-entrant.

In practice

R3 is a pointer to a buffer containing the initial part of the URL to proxy - the URL scheme (e.g. "http:", "ftp:"). This system has the advantage that requests to certain hosts can be proxied and not others (e.g. by giving "http://www.acorn.co.uk/" as the scheme). However, if R4 is 1, this indicates that no matter how the proxy settings have been defined, requests to the base URL should not be proxied in this case (R3 is undefined). When a URL_GetURL request is received, the proxy settings are evaluated in the following order:

1 / Client no-proxy

2 – Client no-proxy: Client proxy

3 – Client no-proxy: Global no-proxy

4 – Client no-proxy: Global proxy

This is to ensure all client settings override global settings and thus remain safe for the given client - ie. a client which sets up a proxy server and then defaults all other URLs to no-proxy, can, no matter how the global settings are changed, be sure of where requests will end up. If R2=0 on entry, then all proxy settings for the specified session are cleared.

Calling this SWI does not result in any calls being made to protocol modules.

[RISC OS Open source page](#)

URL_Status

SWI &83E02 · URL_Fetcher

This SWI is used to monitor the transfer of data from a remote service. It is protocol independent - the exit status bits are common to all services. Clients must test this field bit-wise, since the value is cumulative.

Before calling

R0 Flags:

Bits 0-31 · Reserved (must be zero)

R1 Session identifier

On return

R0 Status word:

Bit 0 => · Connected to server

Bit 1 => · Sent request

Bit 2 => · Sent data

Bit 3 => · Initial response received

Bit 4 => · Transfer in progress

Bit 5 => · All data received

Bit 6 => · Transfer aborted

R1 Preserved

R2 Server response, as an "HTTP" response code (200, 401 etc.)

R3 Bytes read so far (total body data count)

R4 Total bytes to be transferred in whole transaction if known (approximate value only), or -1 if unknown

All other registers preserved

In practice

Obtain information on a session.

Interrupts

Interrupt state undefined.

Re-entrancy

SWI is not re-entrant.

In practice

Clients may not assume that the states returned in R0 will progress in any particular combination or order. However, the likely progression during a fetch for a resource being retrieved over a network (when the bits are combined into a single decimal value) is: 0,1,3,7,15,31 and then R0:5 set upon completion, and R0:6 set at any stage when an error has occurred.

Since each protocol module is returning its results according to the HTTP protocol, R2 can be treated as an HTTP response code whatever the URL being fetched. For example, the FileFetcher module will indicate file not found errors by setting the response code to 404 (HTTP's Not Found error code).

Note that in the case of, for example, an HTTP 400 (Forbidden) return, some explanatory data may be received, too. If the amount of data to be received is unknown, R4 will contain -1, however R3 will contain the number of bytes received so far. The R4 value should be treated as approximate, since the exact interpretation varies between protocols.

When this SWI is called, the URL module invokes the Protocol_Status SWI for the protocol module concerned with the request.

[RISC OS Open source page](#)

URL_Stop

SWI &83E05 · URL_Fetcher

This call aborts a current request if there is one associated with the session identifier. In the event of no request being associated with the identifier, an error is generated. The purpose of this SWI call is to provide the client with a way of enforcing the termination of a request. It is not called by the client just because all the data associated with the request has finished being transferred, although it may do that if it so chooses. The URL_Stop call will be made automatically by the URL module when the session is deregistered by the client using SWI URL_Deregister.

Before calling

R0 Flags:

Bits 31..0 · Reserved (0)

R1 Session identifier

On return

R0 Status word (see URL_Status for details)

All other registers preserved

SWI is not re-entrant

Interrupt status undefined

In practice

Abort a request placed with the URL module.

When this SWI is called, the URL module invokes the Protocol_Stop SWI for the protocol module concerned with the request.

[RISC OS Open source page](#)

USBDriver_DeregisterBus

SWI &54A41 · USBDriver

This SWI deregisters a host controller bus previously registered.

Before calling

R0 HC bus handle

On return

R0 Preserved

[RISC OS Open source page](#)

USBDriver_InsertTransfer

SWI &54A42 · USBDriver

This SWI is called by the host controller driver to notify USBDriver that a transfer has just started.

Before calling

R0 Pointer to usbd_xfer started

On return

R0 Preserved

In practice

When a transfer is requested by USBDriver via the transfer entry point for that bus, the host controller should perform any hardware specific steps to start the transfer then confirm success via this SWI.

Notes

The use of this intermediate SWI (as opposed to the HC simply starting the transfer) allows USBDriver to keep bus use statistics or for the HC to schedule the transfer in a queue for later.

[RISC OS Open source page](#)

USBDriver_RegisterBus

SWI &54A40 · USBDriver

This SWI registers a new host controller with USBDriver.

Before calling

R0 Pointer to USB SoftC Block block

R1 Version number the controller structure corresponds to

On return

R0 HC bus handle

R1 Preserved

Notes

If USBDriver knows that the API version in R1 incompatible with its current expectations, then the registration will be rejected.

The R1 parameter was introduced in USBDriver 0.49; for safety it's recommended that host controller modules check USBDriver_Version and refuse to register if USBDriver is too old to support API version checking.

[RISC OS Open source page](#)

USBDriver_ScheduleSoftInterrupt

SWI &54A44 · USBDriver

The host controller calls this SWI to inform the USBDriver that an interrupt occurred on the given bus.

Before calling

R0 Pointer to usbd_bus that interrupted

On return

R0 Preserved

Notes

The host controller's interrupt handler should determine the cause, quickly acknowledge the interrupt in the controller hardware, then call this SWI.

USBDriver will subsequently call the soft interrupt entry point for that bus to decode the interrupt cause completely, such as handling error conditions or determining which transaction completed. The use of this intermediate SWI (as opposed to the HC calling itself directly) allows USBDriver to keep bus use statistics and optionally defer work to another lower-than-interrupt priority if it is likely to take a long time or the overall system is busy.

[RISC OS Open source page](#)

USBDriver_TransferComplete

SWI &54A43 · USBDriver

This SWI is used by the host controller driver to notify USBDriver that the given transfer has completed.

Before calling

R0 Pointer to usbd_xfer that completed

On return

R0 Preserved

[RISC OS Open source page](#)

USBDriver_Version

SWI &54A45 · USBDriver

This SWI is intended to allow host controllers to check whether they are compatible with USBDriver prior to registering.

Before calling

--

On return

R0 Module version number * 100

Notes

This SWI was introduced in USBDriver 0.49.

[RISC OS Open source page](#)

VFPSupport_ActiveContext

SWI &58EC6 · VFPSupport

This call can be used to read the currently active context (or the context awaiting lazy activation).

Before calling

-

On return

R0 Currently active context

All other registers preserved

Notes

If no context is active, 0 will be returned.

[RISC OS Open source page](#)

VFPSupport_ChangeContext

SWI &58EC3 · VFPSupport

This call should be used to switch the active context. Specify 0 to switch to the null context (i.e. VFP/NEON hardware deactivated).

Before calling

R0 Context to activate

R1 Flags:

Bit 0: Lazy activation flag (0=activate now, 1=use lazy activation)

Bit 1: Application space changing (0=no change, 1=changing)

Other bits reserved, should be zero

On return

R0 Previously active context

All other registers preserved

Notes

If bit 0 of R0 is set, the context switch will occur in a lazy manner, i.e. the old context will only be saved and the new context will only be activated once the first VFP/NEON instruction is executed. Using lazy activation can provide improved performance in some situations, by avoiding redundant save/restore operations.

Bit 1 is used by the Wimp when a task switch occurs. Setting this bit is required to ensure that any context which is located in application space is saved correctly before the memory mapping changes and the new task is paged in.

VFPSupport_ExamineContext can be used to examine a context to determine whether it is active or not.

[RISC OS Open source page](#)

VFPSupport_CheckContext

SWI &58EC0 · VFPSupport

This call allows programs to check if it is possible to create a context with a given set of parameters, and how large the context save area will be (if created manually).

Before calling

R0 Flags:

Bit 0: User mode flag (0=user mode access not required, 1=user mode access required)

Bit 1: Application space flag (0=not in application space, 1=in application space)

Bits 30-31: Ignored

Other bits reserved, should be zero

R1 Number of doubleword registers required (1-32)

On return

R0 Required size of context save area

All other registers preserved

Notes

If it is not possible to create a context with the given features, an error will be returned.

See VFPSupport_CreateContext for more details of the flags in R0. Note that bits 30-31 of R0 are ignored, to allow the same set of flags to be passed to VFPSupport_CreateContext.

[RISC OS Open source page](#)

VFPSupport_CreateContext

SWI &58EC1 · VFPSupport

This call allows programs to create and optionally activate a new VFP context.

Before calling

R0 Flags:

Bit 0: User mode flag (0=user mode access not required, 1=user mode access required)

Bit 1: Application space flag (0=not in application space, 1=in application space)

Bit 30: Lazy activation flag (0=leave context inactive, 1=activate lazily. Supercedes bit 31)

Bit 31: Activation flag (0=leave context inactive, 1=activate now)

Other bits reserved, should be zero

R1 Number of doubleword registers required (1-32)

R2 Pointer to buffer to use for context, or 0

R3 FPSCR value to initialise the context with

On return

R0 Context pointer

R1 Previously active context (if R0 bit 30 or 31 was set), else preserved

All other registers preserved

In practice

Set R1 to the number of doubleword registers you require. When the context is activated registers D0 to Dn will be available to you; do not attempt to access registers outside your allocated range as your code will either abort or potentially corrupt the state of another context.

Notes

If it is not possible to create a context with the given features, an error will be returned.

The flag in bit 0 of R0 controls a security measure that is supported on some systems; if you only require use of VFP/NEON instructions from privileged modes then it is recommended that you do not set this bit. This will prevent user-mode code from being able to execute VFP/NEON instructions and corrupt your context while it is active.

The flag in bit 1 of R0 must be set if the context is to be stored in application space. If you are not providing your own memory buffer, this flag should be clear.

Bits 30 and 31 of R0 control whether the new context will be activated now or whether it will be left inactive. If bit 30 is set the context will be activated in a lazy manner, i.e. the old context will only be saved and the new context will only be activated once the first VFP/NEON instruction is executed. Using lazy activation can provide improved performance in some situations, by avoiding redundant save/restore operations. If bit 31 is set the context will be activated immediately.

R2 should either be a pointer to a word-aligned buffer of the size indicated by VFPSupport_CheckContext, or 0 to allow the VFPSupport module to manage the context memory itself. Note that if you are managing the context memory yourself, the following rules apply:

- If the context is located in application space, bit 1 of R0 must be set to ensure the context is saved correctly before your task is paged out. Even for applications designed to be single-tasking this flag should be set, as there is the possibility your code may be running in a taskwindow.
- The context save area can be moved, but only when the context is inactive. Use VFPSupport_ExamineContext to check whether the context is active prior to moving the buffer, or use VFPSupport_ChangeContext to forcibly deactivate it.
- You are not required to call VFPSupport_DestroyContext in order to destroy the context, but if you do not use VFPSupport_DestroyContext you must make sure the context is inactive before reusing the memory.

Also note that if R2 is supplied, some error checks are skipped; it is assumed that the values of R0 and R1 were previously OK'd by VFPSupport_CheckContext.

When the context is created, the FPSCR is initialised to the value given in R3, and FPSCR is cleared to zero. The actual floating point registers are undefined.

[RISC OS Open source page](#)

VFPSupport_DestroyContext

SWI &58EC2 · VFPSupport

This call allows programs to destroy a context previously created by VFPSupport_CreateContext.

Before calling

R0 Context to destroy

R1 Context to activate in its place

On return

R0 Context that's now active

All other registers preserved

Notes

If the context in R0 is currently active, then the context given in R1 will be activated in its place. Specify 0 to switch to the null context.

If the context in R0 is not currently active then R1 is ignored and the active context will not be modified.

[RISC OS Open source page](#)

VFPSupport_ElementaryFunctions

SWI &58ECA · VFPSupport

This call returns tables of VFP implementations of the elementary functions, jumping to the Nth table entry will run one of the following functions which are given their C standard function names for convenience here, but other languages may give them different names:

Before calling

R0 Flags (reserved for future use, set to 0)

On return

R0 Number of functions provided by this version of VFPSupport

R1 Pointer to branch table for single precision functions

R2 Pointer to branch table for double precision functions

All other registers preserved

0 · **sin(x)** – Equivalent FPA op: SIN · Equivalent VFP op: -

1 · **cos(x)** – Equivalent FPA op: COS · Equivalent VFP op: -

2 · **tan(x)** – Equivalent FPA op: TAN · Equivalent VFP op: -

3 · **asin(x)** – Equivalent FPA op: ASN · Equivalent VFP op: -

4 · **acos(x)** – Equivalent FPA op: ACS · Equivalent VFP op: -

5 · **atan(x)** – Equivalent FPA op: ATN · Equivalent VFP op: -

6 · **atan2(y,x)** – Equivalent FPA op: POL · Equivalent VFP op: -

7 · **log(x)** – Equivalent FPA op: LGN · Equivalent VFP op: -

8 · **log10(x)** – Equivalent FPA op: LOG · Equivalent VFP op: -

9 · **exp(x)** – Equivalent FPA op: EXP · Equivalent VFP op: -

10 · **pow(x,y)** – Equivalent FPA op: POW · Equivalent VFP op: -

11 · **fma(x,y,z)** – Equivalent FPA op: - · Equivalent VFP op: VFMA

Notes

When calling the above functions, the following rules apply:

- The functions will preserve 8 byte stack alignment
- D0-D7 (or S0-S15 for single precision), plus the usual APCS registers (a1-a4, ip, lr) may be corrupted during the call
- Flush to zero mode must be off

In the event of an exception during calculations, the corresponding FPSCR cumulative flags bits will be updated and optionally a floating point error thrown if enabled in the exception trap bits.

[RISC OS Open source page](#)

VFPSupport_ExamineContext

SWI &58EC4 · VFPSupport

This call is used to examine the state of a VFP context, for example to access the registers without activating the context itself.

Before calling

R0 Context to examine

R1 Flags:

Bit 0: Serialise context

Other bits reserved, should be zero

On return

R0 Flags:

Bit 0: User mode flag (0=user mode access not required, 1=user mode access required)

Bit 1: Application space flag (0=not in application space, 1=in application space)

Bit 29: Context is awaiting lazy activation (0=no, 1=yes)

Bit 30: Context status registers are active (0=no, 1=yes)

Bit 31: Memory allocation method (0=user allocated, 1=VFPSupport allocated)

R1 Number of doubleword registers (may be greater than number requested upon context creation)

R2 Register status bitmask

R3 Pointer to dump format descriptor block

R4 Context size

All other registers preserved

In practice

If the context being examined is currently active (or scheduled for lazy deactivation) then some of the registers in the save area may be out of date. To avoid this, set bit 1 of R0 to request that the context state is serialised.

Notes

On exit, R0 bits 0-1 correspond to the flags passed to VFPSupport_CreateContext, and R1 corresponds to the number of registers stored by the context (which may be larger than the requested number in some situations). The contents of R2 and bit 30 of R0 can be used to detect which registers have been saved to the context and which are currently active (and thus indicate that the saved value may be out of date).

R2 is a bitmask representing the status of each doubleword register. If bit 'n' is set then register Dn is active, otherwise it is saved.

R3 is a pointer to a dump format descriptor block, which describes the internal structure of the context area. By parsing the descriptor block you can manually read or modify the register state. The descriptor block is a sequence of words terminated by the value -1. For each word, the lower 16 bits indicate the field within the context, and the upper 16 bits give the offset (in bytes) of the field value within the area. The content of the descriptor block may change from system to system, or even depending on the current state of the context.

Defined field IDs are:

/ Description

0 – Description: Provides the offset to the FPSCR register

1 – Description: Provides the offset to the FPEXC register

2 – Description: Provides the offset to the FPINST register

3 – Description: Provides the offset to the FPINST2 register

4 – Description: Provides the offset to the FSTMX format word

5 – Description: Provides the offset to the start of the register dump

Items 0-4 are all one word in length. Item 5, the register dump, has its length defined by the number of doubleword registers in the context (R1). All items will be at a word aligned offset into the context.

If item 4 is present then it indicates that the register dump is in a VFP implementation-defined format. Otherwise the register dump is in standard format, i.e. a direct representation of the register contents, depending on how the context was being used. For VFP this will most likely be a series of single or double precision floating point values, whereas for NEON it might be integer, floating point, or any arbitrary data.

Note that a context which has FPEXC.EN (bit 30) clear will not have its data registers restored when it is next activated (only the special registers will be restored). VFPSupport creates contexts in this state in order to avoid pointlessly loading the registers with uninitialised data when the context is first activated. Therefore software which wishes to create an inactive context and then write to the data register dump will have to be careful to set FPEXC.EN to 1 otherwise the values will not be used when the context is activated.

[RISC OS Open source page](#)

VFPSupport_ExceptionDump

SWI &58EC9 · VFPSupport

This call allows for reading and writing of the VFP exception dump that VFPSupport maintains. VFPSupport will automatically fill the exception dump with a copy of the current VFP context whenever a trapped exception occurs and raises a RISC OS error. This call allows code to read that exception dump, or to create its own exception dump should it desire. The exception dump is a standard VFP context structure, and so can be examined using VFPSupport_ExamineContext.

Before calling

R0 Flags:

Bit 0: Get exception dump

Bit 1: Get exception context

Bit 2: Clear exception dump

Bit 3: Create exception dump

Other bits: Reserved, should be zero

R1 Context pointer (if bit 3 of flags set)

Otherwise ignored

On return

R0 Copy of exception dump (if bit 0 of flags set), otherwise preserved

R1 Exception context (if bit 1 of flags set), otherwise preserved

All other registers preserved

In practice

Behaviour of the call is controlled by the flags value in R0. Multiple flags can be specified at once. Their order of processing and their behaviour is described below:

- The create flag is handled first. If this flag is set, R1 is treated as a pointer to a VFP context to copy to the internal exception buffer. This will overwrite the current contents. If a null pointer is specified, it will result in the buffer being cleared.
- Next, the two get flags (bit 0 and bit 1) are checked. If bit 0 is set, R0 on exit will contain a copy of the exception context. This copy must be freed by the caller using OS_Module 7. The returned value will be zero if there is no exception context

available. An error will be returned if there is insufficient memory to copy the context.

- If bit 1 is set, the original address of the exception context will be returned – i.e. the value of R1 specified when the exception buffer was initialised. This value should only be used for identifying the context; no guarantee is made that the memory it points to is still valid. Zero will be returned if there is no exception context available.
- Finally, if bit 2 is set the exception buffer will be cleared.

All operations on the exception buffer occur in an atomic manner. For example, setting bits 0, 1 and 2 allow for the contents of the buffer to be read and then cleared. If a subsequent exception occurs halfway through the operation then it will not affect the returned results, but it will be correctly stored within the exception buffer. However, if lack of memory prevents the exception from being returned to the caller, then it will still be cleared from the internal buffer.

[RISC OS Open source page](#)

VFPSupport_FastAPI

SWI &58EC5 · VFPSupport

This call allows you to gain direct access to the most commonly used SWIs, to allow speed-critical systems to avoid the overhead of the kernel SWI dispatcher.

Before calling

-

On return

R0 Workspace pointer to pass in R12 to all routines

R1 CheckContext function pointer

R2 CreateContext function pointer

R3 DestroyContext function pointer

R4 ChangeContext function pointer

All other registers preserved

Notes

When calling the above functions, the following rules apply:

- The processor must be in a privileged mode
- R14_svc must be corruptible (i.e. if in an interrupt handler, you must have previously saved it)
- R12 must contain the given workspace pointer (R0)

Other than the above, the functions behave exactly like their SWI counterparts.

[RISC OS Open source page](#)

VFPSupport_Features

SWI &58EC8 · VFPSupport

This call can be used to read the various features about the system.

Before calling

R0 Reason code

On return

As defined by reason code in R0

In practice

Supported reason codes are:

/

0 – : Read system registers

1 – : Read trapped VFP exception capability

2 – : Read misc feature flags

3 – : Read extra system registers

[RISC OS Open source page](#)

VFPSupport_Features 0

SWI &58EC8 · VFPSupport

This call returns the values of the FPSID, MVFR0 and MVFR1 system registers. These registers describe which version of the VFP/NEON architecture is implemented and which optional features are supported.

Before calling

R0 0 (Reason code)

On return

R0 FPSID register value

R1 MVFR0 register value

R2 MVFR1 register value

All other registers preserved

In practice

VFPSupport_Features 3 can be used to read additional registers, e.g. MVFR2.

[RISC OS Open source page](#)

VFPSupport_Features 1

SWI &58EC8 · VFPSupport

This call returns information about which trapped VFP exceptions are supported; i.e. which exceptions will generate an appropriate RISC OS error if the exception occurs while the exception enable bit is set.

Before calling

R0 1 (Reason code)

On return

R0 Mask of valid exception enable bits

All other registers preserved

In practice

The value returned in R0 corresponds to the exception enable bits in the FPSCR. For example, if bit 8 is set, it indicates that invalid operation exceptions can be generated by setting bit 8 of the FPSCR (the IOE bit). If the value returned in R0 is zero then it indicates that no exceptions can be generated (e.g. the VFP implementation in Cortex-A8 CPUs).

If this call returns an error due to being unimplemented then it indicates an old version of VFPSupport is in use which does not have the VFP support code implemented (VFPSupport versions prior to 0.06). In this case enabling exceptions may result in undefined instruction aborts occurring for any exception-generating instruction. In addition, undefined instruction aborts may also occur if the hardware is used outside of RunFast mode.

[RISC OS Open source page](#)

VFPSupport_Features 2

SWI &58EC8 · VFPSupport

This call returns miscellaneous information about the VFP/NEON environment.

Before calling

R0 2 (Reason code)

On return

R0 Misc flags:

Bit 0: VFP short vectors are supported by hardware

Bit 1: VFP short vectors are supported by software

Other bits: Reserved (zero)

All other registers preserved

Short vector support

'Short vectors' refers to the use of VFP data processing instructions with non-zero values for LEN or STRIDE set in the FPSCR, as defined in the ARM ARM. It does not refer to NEON vectors. Usage of short vectors is deprecated by ARM, and most modern VFP implementations don't support them. On some machines VFPSupport is able to emulate support for them via software, but this is considerably slower than executing a sequence of VFP scalar instructions in a row, or using NEON instructions.

If bit 0 of R0 is set then it indicates that short vectors are supported by the hardware, and so short vector operations will execute quickly (unless support code is invoked for some other reason).

If bit 1 of R0 is set then it indicates that short vectors are being emulated via software, and so any such operations will execute slowly.

If neither bit is set then short vectors are not supported at all. If you attempt to use them then the exact behaviour will depend on the model of CPU in use – e.g. you may get an undefined instruction exception, or the instruction may be treated as a scalar op. Note that if there is no support for short vectors then the FPSCR may also have the LEN and STRIDE fields hard-wired to zero.

For versions of VFPSupport which don't implement this SWI (0.09 and below), use VFPSupport_Features 0 to read the MVFR0 register. If bits 24-27 of MVFR0 are zero then short vectors are not supported (in software or hardware). If the bits are non-zero then they are supported in hardware.

[RISC OS Open source page](#)

VFPSupport_Features 3

SWI &58EC8 · VFPsupport

This call returns the value of the MVFR2 system register, which was introduced in ARMv8. Together with the registers returned by VFPsupport_Features 0, these registers describe which version of the VFP/NEON architecture is implemented and which optional features are supported.

Before calling

R0 3 (Reason code)

On return

R0 MVFR2 register value

R1 Reserved (zero)

R2 Reserved (zero)

R3 Reserved (zero)

All other registers preserved

In practice

Registers R1-R3 are reserved for returning additional registers which are introduced in future CPU architecture versions.

Notes

This call was introduced in VFPsupport 0.13.

[RISC OS Open source page](#)

VFPSupport_Version

SWI &58EC7 · VFPsupport

This call can be used to read the version number of the module. For example for version 1.23 of the module, the value 123 will be returned.

Before calling

-

On return

R0 Module version number * 100

All other registers preserved

[RISC OS Open source page](#)

VideoOverlay_Create

SWI &59CC0 · VideoOverlay

Creates a new overlay associated with the indicated Wimp task. If the task exits, the overlay will be automatically destroyed. For the overlay to become visible, it must be attached to a window via VideoOverlay_SetWindow.

Before calling

R0 Pointer to Mode Selector Block describing the required properties of the overlay

R1 Aspect ratio to use when calculating returned min/max scale values

Top 16 bits = width

Bottom 16 bits = height

R2 Flags

Bit 0: Overlay must support scaling

Bits 1-31 reserved, should be zero

R3 Task handle

On return

R0 Overlay ID

R1 Flags:

Bits 0-7: Overlay type (0 = Z-Order, 1 = Basic, 2+ reserved)

Bits 8-31: Reserved

R2 Minimum width overlay can be scaled to

R3 Minimum height overlay can be scaled to

R4 Maximum width overlay can be scaled to

R5 Maximum height overlay can be scaled to

- All other registers preserved

Notes

See GraphicsV 20 for more details on overlay creation.

If a Mode change occurs, all Overlays will be destroyed by the GraphicsV driver. An application should watch for this occurring, reissue this call to recreate the overlay and call any additional VideoOverlay SWI's to transform/display the overlay if required.

Mode Selector Block

The Mode Selector Block must include some additional Mode Variable Numbers. Refer to Extended Framebuffer Format Specification for ModeFlags and NColour values:

Value / Name / Meaning

0 – Name: ModeFlags · Meaning: Data format and colour space information

3 – Name: NColour · Meaning: Maximum logical colour

13 – Name: MinScreenBanks · Meaning: Optional. If omitted, 1 overlay bank is assumed

[RISC OS Open source page](#)

VideoOverlay_Destroy

SWI &59CC1 · VideoOverlay

Destroys the given overlay.

Before calling

R0 Overlay ID

On return

- All registers preserved

[RISC OS Open source page](#)

VideoOverlay_DiscardBuffer

SWI &59CC5 · VideoOverlay

Discards a buffer, allowing the driver to potentially free the memory that was allocated to it. If necessary, the buffer will be unmapped first. If the overlay is currently displaying this buffer, then the overlay will be hidden.

Before calling

R0 Overlay ID

R1 Buffer number

On return

- All registers preserved

[RISC OS Open source page](#)

VideoOverlay_DisplayBuffer

SWI &59CC2 · VideoOverlay

Switches which buffer is being displayed by the overlay. A negative buffer number will hide the overlay. It is important to call this every time you have a new frame to display (even if the buffer number remains the same) - if software rendering is in use, then VideoOverlay will use this as an indicator that the overlay needs redrawing.

Before calling

R0 Overlay ID

R1 Buffer number

On return

- All registers preserved

[RISC OS Open source page](#)

VideoOverlay_MapBuffer

SWI &59CC3 · VideoOverlay

Maps an overlay bank/buffer into logical address space, potentially allocating the memory for it if the driver hasn't already done so.

Before calling

R0 Overlay ID

R1 Buffer number

On return

R0 Pointer to array giving buffer addresses:

Word 0: Plane 0 base logical address

Word 1: Plane 0 stride between image rows

Words 2+: Logical addresses and strides for any additional planes

- All other registers preserved

In practice

If this is a freshly-allocated buffer, the contents are undefined. Otherwise, the contents will be preserved.

Drivers may have limited logical address space available for mapping overlays, so in Wimp situations, it's recommended to unmap your overlay(s) before allowing the Wimp to swap out your task.

The array returned in R0 will remain valid until the bank is unmapped.

[RISC OS Open source page](#)

VideoOverlay_RedrawWindow

SWI &59CCA · VideoOverlay

The owner of the window which an overlay is attached to must call this SWI for every redraw rectangle associated with the window. Taking into account the current clipping rectangle set by `VideoOverlay_SetPosition`, VideoOverlay will fill the corresponding screen pixels with the correct colour in order to allow the alpha blending / TCK hardware to operate correctly. For overlays which are composited directly into the framebuffer (e.g. software fallback for Basic overlays), this will trigger the rendering to take place.

Before calling

R0 Overlay ID

R1 Pointer to block from `Wimp_RedrawWindow` / `Wimp_GetRectangle`

On return

- All registers preserved

In practice

Whenever necessary, VideoOverlay will automatically call `Wimp_ForceRedraw` in order to request that relevant areas of the window are redrawn.

[RISC OS Open source page](#)

VideoOverlay_SetPosition

SWI &59CC9 · VideoOverlay

Adjusts the position and clipping rectangle of the overlay within the window. These coordinates are specified in window coordinates. The X/Y offset indicates the position of the top-left corner of the overlay, from the user's perspective, prior to any clipping.

Before calling

R0 Overlay ID**R1** X offset**R2** Y offset**R3** Minimum X coordinate of clipping rectangle**R4** Minimum Y coordinate of clipping rectangle**R5** Maximum X coordinate of clipping rectangle**R6** Maximum Y coordinate of clipping rectangle

On return

- All registers preserved

[RISC OS Open source page](#)

VideoOverlay_SetScale

SWI &59CC7 · VideoOverlay

Adjusts the scale of the overlay.

Before calling

R0 Overlay ID**R1** Width**R2** Height**R3** Aspect ratio to use if the image size needs adjusting

Top 16 bits = width

Bottom 16 bits = height

On return

R1 Actual width**R2** Actual height

- All other registers preserved

In practice

Rounding errors or other hardware limits may prevent the chosen values from being used, in which case the driver will make the necessary adjustments (while maintaining the indicated aspect ratio) and return the actual values used.

[RISC OS Open source page](#)

VideoOverlay_SetWindow

SWI &59CC8 · VideoOverlay

Associates the overlay with a window. For the overlay to be displayed correctly, the owner of the window must call VideoOverlay_RedrawWindow for every redraw rectangle associated with that window. For Basic overlays, the owner must also be aware that any icons or other graphics which are rendered on top of the overlay will not be visible. Windows can appear on top of Basic overlays, but will cause VideoOverlay to fall back to software rendering for some or all of the overlay, adversely affecting performance.

Before calling

R0 Overlay ID

R1 Window handle

On return

- All registers preserved

In practice

Note that the window does not need to belong to the task that the overlay is associated with.

[RISC OS Open source page](#)

VideoOverlay_UnmapBuffer

SWI &59CC4 · VideoOverlay

Unmaps an overlay buffer from the logical address space.

Before calling

R0 Overlay ID

R1 Buffer number

On return

- All registers preserved

[RISC OS Open source page](#)

VideoOverlay_Vet

SWI &59CC6 · VideoOverlay

Allows the user to check whether an overlay with the specified properties can be created, and if so, what capabilities it will have.

Before calling

R0 Pointer to Mode Selector Block describing the required properties of the overlay

R1 Aspect ratio to use when calculating returned min/max scale values

Top 16 bits = width

Bottom 16 bits = height

R2 Flags

Bit 0: Overlay must support scaling

Bits 1-31 reserved, should be zero

R3 Task handle

On return

R1 Flags:

Bits 0-7: Overlay type (0 = Z-Order, 1 = Basic, 2+ reserved)

Bits 8-31: Reserved

R2 Minimum width overlay can be scaled to

R3 Minimum height overlay can be scaled to

R4 Maximum width overlay can be scaled to

R5 Maximum height overlay can be scaled to

- All other registers preserved

In practice

The ability for a driver to create an overlay may be influenced by the current screen mode, how much video memory is free, and the properties of any other overlays that may currently be in use. Therefore, the results of this call may be invalidated if any of those influencing factors change.

[RISC OS Open source page](#)

Wimp_AddMessages

SWI &400F6 · WindowManager

The purpose of this call is to update the list of message codes that an application can be informed of via `Wimp_Poll`.

Before calling

R0 Pointer to word array of message codes to add (terminated with zero word)

On return

--

Notes

Prior to RISC OS 4, using this call on an application that received all messages would limit the application to only the newly appended messages.

[RISC OS Open source page](#)

Wimp_AutoScroll

SWI &400FD · WindowManager

The purpose of this call is to control how the Wimp automatically scrolls an application's windows.

Before calling

R0 AutoScroll Entry Flags or 0 to cancel autoscrolling

R1 Pointer to AutoScroll Block

Ignored if R0 = 0 – disable AutoScroll

Used for entry if R0 bits 0 or 1 set and bit 7 clear – enable AutoScroll.

Filled at exit if R0 bit 7 is set – read AutoScroll status.

On return

R0 AutoScroll Status Flags

In practice

Typical use is to handle the window scrolling whilst dragging in a window either as part of the drag and drop protocol or when dragging one end of a selection.

In the latter case after a drag button Mouse_Click event you start a drag with Wimp_DragBox, call Wimp_AutoScroll R0=3 R1=AutoScrollBlock and when you get a User_Drag_Box event after the mouse button is released you call Wimp_AutoScroll R0=0 to switch off autoscrolling.

If bit 7 (Read Status) is set on entry other entry flags are ignored. If the Status Flags returned in R0 have bits 0 or 1 set (AutoScroll has been enabled) the AutoScrollBlock in R1 is filled with the values used when AutoScroll was enabled.

Notes

This call is application specific, and not system wide.

The window handle in the block passed in R1 when enabling AutoScroll must be owned by the calling task.

Only 1 window can use autoscroll at a time so when you are finished with it you should cancel it.

[RISC OS Open source page](#)

Wimp_BaseOfSprites

SWI &400EA · WindowManager

The purpose of this call is to find the memory addresses of the ROM and RAM parts of the Wimp's sprite pool. Note that the RMA area may move around (when sprites are added) and that Window Manager's with version greater than 3.5 use a dynamic area instead of the RMA.

Before calling

--

On return

R0 Base of ROM Sprite Area

R1 Base of RMA Sprite Area

In practice

You can use these addresses with `OS_SpriteOp`, but you should always use `Wimp_SpriteOp`.

The only real use for this SWI is for saving the ROM sprites to a file. An example code in BASIC is provided below.

```
SYS "Wimp_BaseOfSprites" TO ROMBase%,RMABase%
```

```
SYS "OS_SpriteOp",12,ROMBase%,"filename"
```

[RISC OS Open source page](#)

Wimp_BlockCopy

SWI &400EB · WindowManager

The purpose of this call is copy a rectangular block of a windows work area to another position. This is achieved by using the VDU block/copy primitive, before invalidating any areas which must be updated by the application itself.

Before calling

R0 Window Handle

R1 Source minimum x co-ordinate (inclusive)

R2 Source minimum y co-ordinate (inclusive)

R3 Source maximum x co-ordinate (exclusive)

R4 Source maximum y co-ordinate (exclusive)

R5 Destination minimum x co-ordinate

R6 Destination minimum y co-ordinate

On return

R0 - R6 Preserved

Notes

If the source block contains icons, then their visual representation will be moved, however, their bounding boxes will not. It will be up to the application to move the icons by deleting and re-creating them so that they are re-drawn correctly.

If this call is used in a very low colour depth mode and source data contains an ECF pattern, unless the destination is a multiple of the ECF size, then the newly copied area will be out of step with the existing pattern.

All co-ordinates are relative to the window's work area.

This call must not be made from inside a `Wimp_RedrawWindow` or `Wimp_UpdateWindow` loop.

[RISC OS Open source page](#)

Wimp_ClaimFreeMemory

SWI &400EE · WindowManager

The purpose of this call is to claim a specific amount of the Wimp's free memory.

Before calling

R0 0 to release

1 to claim

R1 Amount of memory required

On return

R0 Corrupt

R1 Amount of memory available

R2 Pointer to start address, or 0 if not enough

Notes

This call is deprecated.

This call is similar to `OS_ClaimScreenMemory`. It claims all the memory in the memory in the free pool. The memory must be accessed in SVC mode, and `Wimp_Poll` should not be called while memory is reserved.

[RISC OS Open source page](#)

Wimp_CloseDown

SWI &400DD · WindowManager

The purpose of this call is to inform the Wimp that a task is about to terminate.

Before calling

R0 Task handle returned by `Wimp_Initialise`

R1 "TASK" (&4B534154)

On return

R0 Corrupt

Notes

This must be called immediately before the task terminates by calling `OS_Exit`. If it is not, then the task has a last chance to trap its shutdown by means of the `Service_WimpCloseDown` (which will also be called if the task exits normally).

The task handle and magic word are only required if closing a task which is not presently paged in. e.g a module.

[RISC OS Open source page](#)

Wimp_CloseTemplate

SWI &400DA · WindowManager

The purpose of this call is to close the currently open template file, previously opened by `Wimp_OpenTemplate`.

Before calling

--

On return

R0 Corrupt

[RISC OS Open source page](#)

Wimp_CloseWindow

SWI &400C6 · WindowManager

The purpose of this call is to remove the window, from the list of active windows.

Before calling

R1 Pointer to window handle

R1 Pointer to window handle

R2 "TASK" (&4B534154)

R3 Window handle of parent window

On return

R0 Corrupt

R0 Corrupt

Notes

The Wimp will issue any necessary redraw messages for any currently displayed windows made visible.

There are two ways that this call can be used. The nested window method allows for windows that have a parent window to be closed. i.e. child windows.

[RISC OS Open source page](#)

Wimp_CommandWindow

SWI &400EF · WindowManager

The purpose of this call is open a text window in which VDU4-type output can be displayed, primarily for running older-style programs, that are text-based.

Before calling

R0 -1 · Command window is closed whether it was used or not

0 Same as -1, but if output, do a *Press SPACE or click mouse to continue

1 Command window is set to 'active'

>1 Pointer to title. Command window is set to 'pending', and next time VDU4 output is generated, the window will open

On return

R0 Corrupt

In practice

When the Wimp starts a task, it executes the following command.

SYS "Wimp_CommandWindow", "filename"

SYS "Wimp_CommandWindow", -1

[RISC OS Open source page](#)

Wimp_Createlcon

SWI &400C2 · WindowManager

This purpose of this call is to create an icon.

In practice

This SWI has two syntax forms: icon bar and window.

- The icon bar form creates and displays an icon on the icon bar. An icon can be positioned next to a specific icon, or displayed in order of priority.
- The window form creates an icon that is part of a window. It does not, however, automatically update the window.
- Wimp_Createlcon (icon bar syntax)
- Wimp_Createlcon (window syntax)

[RISC OS Open source page](#)

Wimp_Createlcon (icon bar syntax)

SWI &400C2 · WindowManager

This purpose of this call is to create and display an icon on the icon bar. An icon can be positioned next to a specific icon, or displayed in order of priority.

Before calling

R0 Icon Priority, or
Icon handle number
R1 Pointer to block
+0 Icon bar Position
+4 Icon Block

On return

R0 Icon handle

Icon bar Position

A list of all the different methods an icon can be positioned on the icon bar is in the Icon bar Position page.

The Wimp is able to group icons together on the icon bar if values -3 or -4 are used. Icons can also be positioned based in the order of their priority.

Icon Priority

This is used to specify the placement of an icon in relation to other icon priority values. Icons with higher values are placed towards the outer edge of the icon bar, which is determined by the scan direction.

For a list of common desktop module priority values, please see the Icon Priority page.

Icon Block

The icon block contains all the information required to create the icon itself. The block starts at R1+4 for this call. A full detailed description can be found in the Icon Block page.

[RISC OS Open source page](#)

Wimp_Createlcon (window syntax)

SWI &400C2 · WindowManager

The purpose of this call is to create an icon that is part of a window. To display the icon, the window must first be updated by using `Wimp_ForceRedraw`.

Before calling

R0 –

R1 Pointer to block

+0 Window handle

+4 Icon Block

On return

R0 Icon handle (unique within window)

Icon Block

The icon block contains all the information required to create the icon itself. The block starts at `R1+4` for this call. A full detailed description can be found in the Icon Block page.

[RISC OS Open source page](#)

Wimp_CreateMenu

SWI &400D4 · WindowManager

The purpose of this call is to create and display a menu structure. A menu window to open can be a traditional menu containing menu items or it can also be a window such as a dialog box.

Before calling

R1 Pointer to Menu Block or

Window handle, or

`-1` means close any active menu

R2 X co-ordinate of top-left corner of top level menu

R3 Y co-ordinate of top-left corner of top level menu

On return

R0 Corrupt

In practice

Having used this call, applications should return to its normal polling loop. Further sub-menus are opened via `Wimp_CreateSubMenu`. The Wimp maintains the menu tree until a mouse click is detected. If the click was within a menu, then a `Menu_Menu_Selection` event code is returned from `Wimp_Poll`, along with a list of selections. If the click was detected outside of a menu, then the Wimp closes all menus.

Notes

A menu is actually a window with its work area entirely covered by menu items. However, any window can be opened as a menu if `R1` contains the window handle.

The Wimp will automatically display a vertical scroll bar if a menu is taller than the current screen mode

When a menu is closed, the caret is automatically given back to wherever it was before the window was opened.

Pressing the Return key while the caret is inside a writable menu item is equivalent to a mouse click.

It is conventional that menu items clicked with the Adjust button should keep the menu structure open so further selections can be made. For this to happen, the Wimp temporarily marks the menu tree when a selection is made. To confirm which mouse button was clicked, `Wimp_GetPointerInfo` should then be called. If the adjust button was clicked, then `Wimp_CreateMenu` should be called before returning to `Wimp_Poll` so that the menu tree to be re-opened as before. The menu structure can also be modified (if required) before re-opening.

To aid with identifying which menu item was selected by the user, the `Wimp_DecodeMenu` can be used.

[RISC OS Open source page](#)

Wimp_CreateSubMenu

SWI &400E8 · WindowManager

The purpose of this call is to create a sub-menu.

Before calling

R1 Pointer to sub-menu block

R2 X co-ordinate of top left of sub-menu

R3 Y co-ordinate of top left of sub-menu

On return

R0 Corrupt

In practice

When a sub-menu is requested by the Wimp to be opened, it generates a `Message_MenuWarning` event code. Applications should respond by using this call to create the sub-menu.

The contents of R1 to R3 are obtained from event code above at offsets +20, +24 & +28

[RISC OS Open source page](#)

Wimp_CreateWindow

SWI &400C1 · WindowManager

The purpose of this call is to create a window and return the handle to the window so it can then be displayed.

Before calling

R1 Pointer to Window Block

On return

R0 Window Handle

In practice

Icons can also be created at the same time as the window itself.

[RISC OS Open source page](#)

Wimp_DecodeMenu

SWI &400D5 · WindowManager

This call is incredibly useful to decode menus by their contents rather than their item selection numbers. For example, a returned value of Edit.Paste would mean that the user has selected the 'Paste' item within the 'Edit' sub-menu.

Before calling

R1 Pointer to menu data structure

R2 Pointer to list of menu selections

R3 Pointer to buffer for answer

On return

R0 Corrupt

Notes

Buffer pointed to by R3 is updated to contain menu item texts, separated by a period sign .

The string is terminated with a CR control code.

[RISC OS Open source page](#)

Wimp_Deletelcon

SWI &400C4 · WindowManager

The purpose of this call is to delete an icons definition within a window. To update the window applications should then call Wimp_ForceRedraw.

Before calling

R1 +0 Window handle, or

-2 for icon bar

+4 Icon handle

On return

R1 Corrupt

In practice

Note: The icons memory will be reallocated if it is the last icon belonging to a window.

[RISC OS Open source page](#)

Wimp_DeleteWindow

SWI &400C3 · WindowManager

The purpose of this call is to close a window if it is open.

Before calling

R1 +0 Window handle

On return

R1 Corrupt

Notes

The memory used for the window is re-allocated upon closing of the window. Any indirect data is however, left alone.

[RISC OS Open source page](#)

Wimp_DragBox

SWI &400D0 · WindowManager

The purpose of this call is to initiate a drag operation.

Before calling

R1 Pointer to Drag Block, or

-1 to cancel drag

R2 "TASK" invokes use of R3

R3 Drag Box Flags

On return

R0 Corrupt

Notes

A drag can span several calls to Wimp_Poll, so applications must remember what is being dragged and why.

The termination of a non system drag is reported by the User_Drag_Box event code.

Aborted non system drags (for example when an error box is shown) are not reported.

[RISC OS Open source page](#)

Wimp_Extend

SWI &400FB · WindowManager

This call has multiple specific uses that are determined by the reason code passed via R0.

Before calling

R0 Reason Code

0-5 Internal use only

- 6 Stack Information (parent window)
- 7 Stack Information (top child window)
- 8 Stack Information (bottom child window)
- 9 Stack Information (sibling immediately below)
- 10 Stack Information (sibling immediately above)
- 11 Window Information
- 13 Sprite Suffix
- 14 Get icon validation string
- 15 Read slot size
- 257 Sprite Suffix (RISC OS Select compatible version)

Notes

Be careful when probing for supported features – unsupported reason codes will return with all registers preserved and no error.

[RISC OS Open source page](#)

Wimp_Extend (get validation string)

SWI &400FB · WindowManager

This call allows programs to easily read the validation string of arbitrary icons (e.g. icons owned by other tasks).

Before calling

- R0** 14 (reason code)
- R1** Window handle
- R2** Icon handle
- R3** Pointer to buffer for result (must not be located in application space)
- R4** Size of buffer, or
0 to read size required

On return

- R4** Reduced by validation string length

Notes

To determine the required buffer size, first call with **R4** = 0. If the call returns with **R4** = 0 then the icon has no validation string. Else, **-R4** provides the required buffer size to use when reading the string.

Care must be taken to ensure the buffer is not allocated from application space, as the Wimp will temporarily page out the current task when reading the string.

For convenience, the Wimp will null-terminate the string (the string pointed to by the icon block may be control-terminated)

This call was introduced by Wimp 5.64. On older Wimp versions, it's possible to use `Wimp_TransferBlock` to read the validation string. However using that SWI will be less efficient because there is no direct way to determine the string length.

[RISC OS Open source page](#)

Wimp_Extend (read slot size)

SWI &400FB · WindowManager

This call implements the same behaviour as `Wimp_SlotSize` when called with R0 and R1 set to -1. However, the return values are in units of pages instead of bytes. This allows accurate values to be returned on systems with over 2GB of RAM.

Before calling

R0 15 (reason code)

On return

R0 Size of current slot (pages)

R1 Size of next slot (pages)

R2 Size of free pool (pages)

Notes

The page size can be read via `OS_ReadMemMapInfo`.

This call was introduced in Wimp 5.67.

[RISC OS Open source page](#)

Wimp_Extend (sprite suffix)

SWI &400FB · WindowManager

This call with a reason code of 13, adds a sprite file suffix onto a pathname

Before calling

R0 13 (reason code)

R1 Pointer to file name

R2 Pointer to output buffer

R3 Size of buffer, or

0 to read size required

On return

R0 Corrupt

R3 Space left in buffer, or

Required size

Notes

It uses a multi-pass strategy so that it will fall back where a particular file is not available.

For more information on sprite files and use of sprite file suffixes, please read the page about system icons.

Due to a possibility that other versions of RISC OS use reason code 13 to provide unrelated functionality, it is advisable to use `Wimp_Extend (sprite suffix)` (RISC OS Select version) unless a check for a suitable OS version is made before the call.

[RISC OS Open source page](#)

Wimp_Extend (sprite suffix) (RISC OS Select)

SWI &400FB · WindowManager

This call with a reason code of 257, adds a sprite file suffix onto a pathname.

Before calling

R0 257 (reason code)

R1 Pointer to buffer containing file name

On return

R0 R1 on entry

R1 0

Notes

Icons are searched for using the same algorithm as Wimp_Extend (sprite suffix). If a suitable sprite file is found then the filename in the buffer will be updated, otherwise the buffer will be left untouched.

It is the caller's responsibility to make sure that the buffer is large enough to contain the updated filename - it's recommended that there's enough spare space for at least 8 extra characters to be inserted.

Compared to Wimp_Extend (sprite suffix), this call has the advantage that it is available in both the RISC OS Open and RISC OS Select versions of the Wimp, easing the task of writing software which is compatible with both OS versions. However unlike the RISC OS Open version of the call, no buffer size is specified, and so buffer overruns are possible.

Because Wimp_Extend does not return an error when an unknown reason code is given, you can check if this call is implemented on a given Wimp version by examining R1 on exit. If the call is implemented then R1 will have been set to zero.

[RISC OS Open source page](#)

Wimp_Extend (stack information)

SWI &400FB · WindowManager

This call with a reason code of 6 to 10, returns the handle of a window relative to the window specified in R1 (if provided)

Before calling

R0 Reason Code

6 Return handle of parent window

7 Return handle of top child window

8 Return handle of bottom child window

9 Return handle of sibling window immediately above

10 Return handle of sibling window immediately below

R1 Window handle, or -1 for top level

On return

R1 Next window handle, or -1 for none

Notes

A returned value of -1 means one of the following:

- 'no window'
- R1 was -1 on entry and R0 was not 7 or 8

Pane windows are not skipped when using this call.

[RISC OS Open source page](#)

Wimp_Extend (window information)

SWI &400FB · WindowManager

This call with a reason code of 11 is used to return specific information relating to the windows system icons. e.g. widths and heights.

Before calling

R0 11 (reason code)

R1 Pointer to 100 byte block

+0 Window handle, or

0 to return information based on a generic window with line borders enabled

On return

R0 Preserved

R1 Block preserved and updated

Notes

The 100 byte block points to the window handle on entry, but is updated on exit.

[RISC OS Open source page](#)

Wimp_ForceRedraw

SWI &400D1 · WindowManager

This SWI has two syntax forms: mark invalid area and extended.

In practice

- The mark invalid area form marks areas of windows as invalid so they can be redrawn later.
- The extended form allows an application to update the title bar text without updating the rest of the window.
- Wimp_ForceRedraw (mark invalid area syntax)
- Wimp_ForceRedraw (extended syntax)

[RISC OS Open source page](#)

Wimp_ForceRedraw (extended syntax)

SWI &400D1 · WindowManager

This syntax form allows an application to force a redraw of windows that belong to other applications.

Before calling

R0 Window handle

R1 "TASK" (&4B534154)

R2 3 to update title bar

On return

R0 Corrupt

In practice

It is also the preferred option to redraw a window's title bar, as the work area does not get redrawn.

[RISC OS Open source page](#)

Wimp_ForceRedraw (mark invalid area syntax)

SWI &400D1 · WindowManager

This syntax form marks an area of a screen as invalid, so it can be redrawn later.

Before calling

R0 Window handle, or

-1 for absolute screen co-ordinates

-2 for icon bar

R1 Minimum x of area

R2 Minimum y of area

R3 Maximum x of area

R4 Maximum y of area

On return

R0 Corrupt

Notes

Co-ordinates are relative to the window's work area, unless **R0** = -1 on entry, in which case absolute screen co-ordinates should be provided.

This call should not be used for animation, or other temporary changes. Under these circumstances `Wimp_UpdateWindow` should be used instead.

Under the nested wimp, windows belonging to other applications can be marked as invalid.

[RISC OS Open source page](#)

Wimp_GetCaretPosition

SWI &400D3 · WindowManager

The purpose of this call is to return information relating to the caret.

Before calling

R1 Pointer to block

On return

R0 Corrupt

block +0 Window handle, or -1 if none

+4 Icon handle, or -1 if none

+8 X offset

+12 Y offset

+16 Caret Flags, or -1 if not displayed

+20 Index into string, or -1 if writeable icon

Notes

The index value returned is a byte index, and not a character index. Therefore, there is only a distinction when the system alphabet is UTF8.

[RISC OS Open source page](#)

Wimp_GetIconState

SWI &400CE · WindowManager

The purpose of this call is to return the icon block information relating to a specific icon.

Before calling

R1 Pointer to block

+0 Window handle

+4 Icon handle

On return

R1 Pointer to block

+0 Window handle

+4 Icon handle

+8 Icon Block (32 bytes)

[RISC OS Open source page](#)

Wimp_GetMenuState

SWI &400F4 · WindowManager

The purpose of this call is to return the selected menu item and the sub-menus used. This is called the 'menu tree'.

Before calling

- R0** 0 => return the current menu tree state, or
1 => report tree leading up to, but not including, a window/icon pair
- R1** Pointer to buffer for result
- R2** Window handle of menu (if R0 = 1)
- R3** Icon handle of entry in menu (if R0 = 1)

On return

- R0** Corrupt
- R1** Returned values in buffer

Notes

The buffer is filled with a list of numbers in the same format as returned by Menu_Selection by Wimp_Poll.

The buffer is terminated with the value of -1.

The buffer will return null (-1) if:

- R0 = 1 and window or icon handle in R2/R3 is not in the tree, or
- R0 = 0 and the menu belongs to another application, or has been closed.

If the window is a dialog box, the list of sub-menus used will not include the dialog box.

[RISC OS Open source page](#)

Wimp_GetPointerInfo

SWI &400CF · WindowManager

The purpose of this call is to return the state of the mouse during the previous Wimp_Poll.

Before calling

- R1** Pointer to block

On return

- R0** Corrupt
- Block
- +0** Mouse x
- +4** Mouse y
- +8** Button State
- +12** Window handle (-1 => background, -2 => icon bar)
- +16** Icon Handle

Notes

As this call returns the pointer state during the previous Wimp_Poll, it is perfectly safe to use it after a long menu click delay to recreate the menu or as a means of dragging boxes.

[RISC OS Open source page](#)

Wimp_GetRectangle

SWI &400CA · WindowManager

The purpose of this call is to return the details of the next rectangle of the work area to be drawn.

Before calling

R1 Pointer to block to return result

On return

R0 0 if no more rectangles

R1 Block is updated with Rectangle Data

Notes

This call is used, repeatedly, immediately following a call of either Wimp_RedrawWindow or Wimp_UpdateWindow.

If this call follows an earlier Wimp_RedrawWindow, then the rectangles is also cleared to the background colour of the window. If, however, it follows a Wimp_UpdateWindow call, then the rectangle's contents are preserved.

[RISC OS Open source page](#)

Wimp_GetWindowInfo

SWI &400CC · WindowManager

The purpose of this call is to return the details of a windows state.

Before calling

R1 Pointer to block to return result

Block

+0 Window Handle

On return

R1 Block is updated

Block

+0 Window handle

+4 Window Block

Notes

This call also returns information of any icon created after the window itself was created.

If bit 0 of R1 is set, then the icons are not returned and a buffer size of 92 bytes is required for the Window Block up to and including the icon count.

[RISC OS Open source page](#)

Wimp_GetWindowOutline

SWI &400E0 · WindowManager

The purpose of this call is to return the co-ordinates of a rectangle that covers the window (including window furniture).

Before calling

R1 Pointer to block to return result

Block

+0 Window Handle

On return

R0 Corrupt

R1 Block is updated

Block

+0 Window handle

+4 Bounding box minimum x (inclusive)

+8 Bounding box minimum y (inclusive)

+12 Bounding box maximum x (exclusive)

+16 Bounding box maximum y (exclusive)

Notes

A window does not need to be open for this call to function, unless the Window Manger cannot handle nested windows.

[RISC OS Open source page](#)

Wimp_GetWindowState

SWI &400CB · WindowManager

This SWI has two syntax forms: original and extended.

In practice

- The original form returns information relating to a specific window.
- The extended form allows information relating to specific window and its parent window to be returned. This form requires a Window Manager with nested window capability.
- Wimp_GetWindowState (original syntax)
- Wimp_GetWindowState (extended syntax)

[RISC OS Open source page](#)

Wimp_GetWindowState (extended syntax)

SWI &400CB · WindowManager

The syntax of this call will return a windows visible state and its parent window handle and information relating to the child windows nested flags.

Before calling

R1 Pointer to block to return result

Block

+0 Window Handle

R2 "TASK" (&4B534154)

On return

R1 Block is updated

Block

+0 Window handle, or -2 for icon bar

+4 Visible area minimum x

+8 Visible area minimum y

+12 Visible area maximum x

+16 Visible area maximum y

+20 Scroll x offset

+24 Scroll y offset

+28 Window Locations

+32 Window Flags

R3 Parent window handle, or -1 if top level window

R4 Nested Window Flags

Notes

A Window Manager with nested window capability is required for this call to work.

[RISC OS Open source page](#)

Wimp_GetWindowState (original syntax)

SWI &400CB · WindowManager

The syntax of this call will return a windows visible state.

Before calling

R1 Pointer to block to return result

Block

+0 Window Handle

On return

R1 Block is updated

Block

+0 Window handle, or -2 for icon bar
+4 Visible area minimum x
+8 Visible area minimum y
+12 Visible area maximum x
+16 Visible area maximum y
+20 Scroll x offset
+24 Scroll y offset
+28 Window Locations
+32 Window Flags

Notes

"Visible area" gives the window in screen co-ordinates. "Scroll offset" gives the top-left hand corner of the window in user co-ordinates. It is only an offset when the top-left corner of the extent is (0,0) in user co-ordinates.

Another syntax form of this call can be used to cause information relating to a specific window and its parent window to be returned. This, however, requires a Window Manager with nested window capability.

[RISC OS Open source page](#)

Wimp_Initialise

SWI &400C0 · WindowManager

The purpose of this call is to register a task with the Wimp.

Before calling

R0 Last known Wimp version x 100
R1 "TASK" (&4B534154)
R2 Pointer to description string to display in Task Manager
R3 Pointer to list of acceptable User Messages

On return

R0 Current Wimp version x 100
R1 Task handle

Notes

This call should only be used once when starting up. And before a task closes, it should call Wimp_CloseDown.

The input R0 value should indicate the version of the Wimp that your application was developed for. The Wimp will use this information to adjust its behaviour, so that your program can still run as intended when running under a newer Wimp.

The interpretation of R3 depends on the input R0 value (and ultimately, on whether the current Wimp version is $\geq$ R0):

- If R0 is less than 300, or the Wimp version is less than 300, R3 is ignored, and all user messages will be delivered to the task
- If R0 is greater than or equal to 300, R3 is treated as a pointer to a zero-terminated list of user message codes. Only the user messages given in this list will be delivered to the task, resulting in a more responsive system when faced with e.g. applications which regularly send broadcast messages. Take care not to include Message_Quit in the list, as its value of zero will get mistaken for the list terminator. The Wimp will always deliver Message_Quit to tasks, regardless of any message filter which they have set up. Behaviour of empty lists and null pointers differs depending on Wimp version: If the Wimp version is before 310, an empty list is treated as "no messages are important" (other than Quit) If the Wimp version is 310 or later, an empty list is treated as "all messages are important". A null pointer is treated as "no messages are important" (other than Quit).

The task can add or delete messages that it can respond to by `Wimp_AddMessages` and `Wimp_RemoveMessages`.

[RISC OS Open source page](#)

Wimp_LoadTemplate

SWI &400DB · WindowManager

This SWI has two syntax forms: load and request size.

In practice

- The load form is used to load a window template into memory, ready for use.
- The request size form is used to return the memory size required to load a window template.
- `Wimp_LoadTemplate` (load syntax)
- `Wimp_LoadTemplate` (request size syntax)

[RISC OS Open source page](#)

Wimp_LoadTemplate (load syntax)

SWI &400DB · WindowManager

This purpose of this call is to load a window template into memory so it can then be used.

Before calling

- R1** Pointer to buffer for template
- R2** Pointer to workspace for indirected data
- R3** Pointer to end of indirect data workspace +1 byte
- R4** Pointer to 256 byte array for font reference, or -1 for no fonts
- R5** Pointer to (wildcard) name to match (must be 12 bytes word-aligned)
- R6** Position to search from, or 0 for the first call

On return

- R0** Corrupt
- R2** Pointer to remaining workspace
- R3** Reserved
- R4** Reserved
- R5** Template name returned in block
- R6** Position of next entry, or 0 if template not found

Notes

This call should only be used after the template file has been opened using `Wimp_OpenTemplate`. The specific template window can then be loaded using this call.

Indirect data is then copied by the Wimp to the workspace as referenced by R2. The font reference array is also updated if fonts are used by the template window.

It is possible to load a template window by searching using a wildcarded name. e.g. 'dialog*' by setting R6 to 0. As there may be more than one match, R6 upon exit will point to the next match, or 0 if it is last template found. As R5 is overwritten, the task must re-initialise the wildcard string every time it called.

When loading a template file, all references to any fonts are updated to current font handles. The Wimp calls `Font_FindFont` and replaces the original font numbers with the new font handle. It then increments the entry for that handle in the reference array. Please note, that this array should be initialized to zero before the first call to `Wimp_LoadTemplate`.

When a window is deleted, references to any fonts with handles in the range of 1 to 255 should be removed. This is achieved by calling `Font_LoseFont` the number of times given by that font's reference count in the array. An alternative method is to use a single reference array for all the windows, and only call `Font_LoseFont` the required number of times for each handle when the task terminates.

No errors are generated if a template could not be found. In this instance, `R6` is simply set to 0.

When loading templates, the call checks that the available indirect workspace is large enough. If it is not, an error will be returned.

If an error does occur when trying to load a template, then it is advisable to call `Wimp_CloseTemplate` and close the template file.

As pointers to memory blocks are used to return the template data, it is vital that enough memory is set aside. To calculate the size of buffer required to load a template file, the request size syntax form of this call should be used.

[RISC OS Open source page](#)

Wimp_LoadTemplate (request size syntax)

SWI &400DB · WindowManager

This syntax form of this call is to return the memory size required to load a window template.

Before calling

- R1** 0 to find the required buffer size
- R2** Pointer to workspace for indirected data
- R3** Pointer to end of workspace +1 byte
- R4** Pointer to 256 byte array for font reference, or -1 for no fonts
- R5** Pointer to (wildcard) name to match (must be 12 bytes word-aligned)
- R6** Position to search from, or 0 for the first call

On return

- R0** Corrupt
- R1** Size of window and icons
- R2** Size of indirect data

Notes

As pointers to memory blocks are used to return the template data, it is vital that enough memory is set aside. The space required to return the window definition in `R1` is 88 bytes, and a further 32 bytes for each icon. Further memory is required to contain any indirect data.

This is very useful to identify how much memory is required to load a template, before actually loading it using the load syntax form of this call.

No errors are generated if a template could not be found. In this instance, `R6` is simply set to 0.

If an error does occur when trying to load a template, then it is advisable to call `Wimp_CloseTemplate` and close the template file.

[RISC OS Open source page](#)

Wimp_OpenTemplate

SWI &400D9 · WindowManager

This purpose of this call is to open a template file ready for accessing.

Before calling

R1 Pathname of template file

On return

R0 Corrupt

[RISC OS Open source page](#)

Wimp_OpenWindow

SWI &400C5 · WindowManager

This SWI has two syntax forms: original and extended.

In practice

- The original form is used to open non-child windows that belong to the application.
- The extended form is used to open parent or child windows that may or may not belong to the application.
- Wimp_OpenWindow (original syntax)
- Wimp_OpenWindow (extended syntax)

[RISC OS Open source page](#)

Wimp_OpenWindow (extended syntax)

SWI &400C5 · WindowManager

The purpose of this call is to update the list of active windows. It is used when a window is to be opened for the first time, and when a window moves, scrolls or resizes.

Before calling

R1 Pointer to block, or

0 to flush all pending window opens to the screen

Block

+0 Window handle

+4 Visible area minimum x

+8 Visible area minimum y

+12 Visible area maximum x

+16 Visible area maximum y

+20 Scroll offset x

+24 Scroll offset y

+28 Window Locations

+32 New Window Flags if bit 0 of nesting flags is set

R2 "TASK" (&4B534154)

R3 Window handle, or

-1 for top level window

R4 Nested Window Flags

On return

R1 Pointer to block

Block

+0 Window handle

+4 Visible area minimum x

+8 Visible area minimum y

+12 Visible area maximum x

+16 Visible area maximum y

+20 Scroll offset x

+24 Scroll offset y

R2 Corrupt

Notes

The Wimp will issue any redraw messages to the application if necessary to ensure the window is updated.

This syntax form can open parent or child windows that may or may not belong to the application, however, this is only possible if Window Manager has nested window capability.

The original syntax form only opens parent windows that belong to the application.

[RISC OS Open source page](#)

Wimp_OpenWindow (original syntax)

SWI &400C5 · WindowManager

The purpose of this call is to update the list of active windows. It is used when a window is to be opened for the first time, and when a window moves, scrolls or resizes.

Before calling

R1 Pointer to block

Block

+0 Window handle

+4 Visible area minimum x

+8 Visible area minimum y

+12 Visible area maximum x

+16 Visible area maximum y

+20 Scroll offset x

+24 Scroll offset y

+28 Window Locations

On return

R1 Pointer to block

Block

- +0** Window handle
- +4** Visible area minimum x
- +8** Visible area minimum y
- +12** Visible area maximum x
- +16** Visible area maximum y
- +20** Scroll offset x
- +24** Scroll offset y

Notes

"Visible area" gives the window in screen co-ordinates. "Scroll offset" gives the top-left hand corner of the window in user co-ordinates. It is only an offset when the top-left corner of the extent is (0,0) in user co-ordinates.

The Wimp will issue any redraw messages to the application if necessary to ensure the window is updated.

This call is limited to opening parent windows that belong to the application. The extended syntax form can open parent or child windows that may or may not belong to the application.

[RISC OS Open source page](#)

Wimp_PlotIcon

SWI &400E2 · WindowManager

The purpose of this call is to display a 'virtual' icon.

Before calling

- R1** Icon Block
- R2** 0 for safety (see magic values for some OS versions)
- R4** X offset (if used outside of update/redraw loop)
- R5** Y offset (if used outside of update/redraw loop)

On return

- R0** Corrupt

Notes

The main use for this call is within a redraw or update loop, so that an application can display any virtual icons required.

The call can be used outside of a redraw or update loop, however, in this instance R4 and R5 must be used to specify screen co-ordinates where the icon will be displayed, rather than the co-ordinates passed within the Icon Block itself.

If used outside the redraw/update loop, the graphic area will be reset to the current screen mode. This is not good if outputting to a sprite.

[RISC OS Open source page](#)

Wimp_Poll

SWI &400C7 · WindowManager

The purpose of this call is poll the Wimp to check for any system wide events that the application may need to respond to. i.e. windows, menus, mouse, keyboard handling.

Before calling

R0 Poll Mask

R1 Pointer to 256 byte block

R3 Pointer to pollword in RMA (if R0 bit22 is set)

On return

R0 Reason Code

R1 Pointer to block of data

R2 Senders task handle

Notes

Every application must have poll loop whereby it calls Wimp_Poll.

The call returns specific event codes, and the application must then decide whether it wishes to respond to it.

If upon calling Wimp_Poll an escape condition is detected, the Wimp will cancel the condition and disable future escape conditions.

Any errors that occur inside Wimp_Poll are reported by the Wimp itself and not passed back to the applications. The only exception to this rule is when an error was caused due to bad parameter usage.

[RISC OS Open source page](#)

Wimp_PollIdle

SWI &400E1 · WindowManager

The purpose of this call is poll the Wimp to check for any system wide events that the application may need to respond to, but will return control to this task as soon as possible after the time stated in R2.

Before calling

R0 Poll Mask

R1 Pointer to 256 byte block

R2 Earliest monotonic time for return with null reason

R3 Pointer to pollword in RMA (if R0 bit22 is set)

On return

R0 Reason Code

R1 Pointer to block of data

R2 Senders task handle

Notes

This call effectively carries out the same function as Wimp_Poll, except that any Null reason codes that occur prior to the time given in R2 will not be delivered to the task. Delivery of other events is not affected.

This provides tasks which want to perform actions at scheduled times an efficient way to sleep until the next action is due to occur (at which point the Wimp will 'wake up' the task by delivering the null poll event). Note however that there is no guarantee that your task will regain control at precisely the time you specify. E.g. the system could be busy servicing another task, or a single-tasking program could have started.

If upon calling Wimp_PollIdle an escape condition is detected, the Wimp will cancel the condition and disable future escape conditions.

Any errors that occur inside Wimp_PollIdle are reported by the Wimp itself and not passed back to the applications. The only exception to this rule is when an error was caused due to bad parameter usage.

[RISC OS Open source page](#)

Wimp_ProcessKey

SWI &400DC · WindowManager

The purpose of this call is twofold:

Before calling

R0 Character Code

On return

R0 Corrupt

In practice

- Pass on un-used keypresses that the task has been notified of via the Wimp_Poll (Key_Pressed event code)
- Insert keypresses into the Wimp buffer

Notes

A task must pass on keypresses that it does not use, to ensure that other tasks get notified of them too.

Characters passed on take precedence over characters in the keyboard buffer when notifying tasks.

[RISC OS Open source page](#)

Wimp_ReadPalette

SWI &400E5 · WindowManager

The purpose of this call is read the palette and return the information in a designated block of memory.

Before calling

R1 Pointer to 20 word Palette Block

R2 "TRUE" (&45555254) to return true colour

On return

R0 Corrupt

Block Updated with palette definitions

Notes

The 20 word block of memory is updated to contain the palette information.

Applications should use this call to identify current Wimp palette settings.

If 'TRUE' is passed in R2, then true colour values are returned.

[RISC OS Open source page](#)

Wimp_ReadPixTrans

SWI &400ED · WindowManager

The purpose of this call is to inform the application how the Wimp would plot the icon independent of the current screen mode.

Before calling

R0 &0xx if sprite in system area

&1xx if sprite in user area and R2 = name

&2xx if sprite in user area and R2 = pointer

R1 Pointer to spritearea (0 for system, 1 for Wimp)

R2 Pointer to sprite name, or sprite itself

R6 Block to receive scale factors

R7 Block to receive colour translation table (2, 4 or 16 bytes)

On return

R0 Corrupt

Blocks at

R6 Contains Scale Factors

R7 Contains the Colour Translation Table

Notes

Sprites that were created with less than four bits per pixel, have their pixels mapped onto the Wimp's greyscale colours.

Applications wishing to plot a sprite using the best approximation to its actual colours should use ColourTrans.

[RISC OS Open source page](#)

Wimp_ReadSysInfo

SWI &400F2 · WindowManager

The purpose of this call is to retrieve information pertaining to the Wimp.

Before calling

R0 Wimp Items

On return

R0 Item value

R1 Item value 2 (if specified by Wimp Item in R0)

[RISC OS Open source page](#)

Wimp_RedrawWindow

SWI &400C8 · WindowManager

The purpose of this call is to redraw parts of a window that needs updating.

Before calling

R1 Pointer to block to containing window handle

On return

- Exits through Wimp_GetRectangle

Notes

This call is normally used after receiving a Redraw_Window_Request via Wimp_Poll.

This call must be the first Wimp operation after receiving a Redraw_Window_Request. This means you cannot, for example, create or delete other windows until after the redraw is complete.

Windows in need of updating are divided into discrete rectangular areas, and thus many rectangles may need redrawing. To ensure all necessary rectangle are updated, this call is used as part of a loop with Wimp_GetRectangle.

[RISC OS Open source page](#)

Wimp_RegisterFilter

SWI &400F5 · WindowManager

This call is used by the The Filter Manager, and should not be used unless a task wishes to replace the whole filter system.

Notes

Use The Filter Manager to register filters for specific tasks.

[RISC OS Open source page](#)

Wimp_RemoveMessages

SWI &400F7 · WindowManager

The purpose of this call is to remove the user_message codes that a task can receive from Wimp_Poll

Before calling

R0 Pointer to word array of Message Codes (terminated with zero word)

On return

- -

[RISC OS Open source page](#)

Wimp_ReportError

SWI &400DF · WindowManager

The purpose of this call is to display an error box to the user.

Before calling

R0 Pointer to standard SWI Error Block

R1 ReportError Flags

R2 Pointer to title text (if R1 bit 4 is set), or

Pointer to application name (if R1 bit 4 is not set), or

0 for basic error box title

R3 Sprite name (if bit 8 is set)

R4 Sprite Area (if bit 8 is set)

R5 Pointer to string containing list of user specific button names (comma delimited)

On return

R0 Corrupt

R1 0 = No selection

1 = OK

2 = Cancel

3 = Extra Button

4 = Extra Button

5 = Extra Button

Notes

The use of this call should be avoided where possible, due to the non-multitasking nature of these error boxes.

Application specific buttons can be displayed in an error box, by having a list of button names, separated by commas in R5.

The total size for all the user specific button names is 256 characters.

No more than 8 buttons can be displayed in an error box.

A user pressing Return will have the same effect as clicking on the default button.

Pressing Escape is the same as choosing the default cancel option on the error box.

The SWI error block must be zero terminated.

[RISC OS Open source page](#)

Wimp_ResizeIcon

SWI &400FC · WindowManager

The purpose of this call is to resize or move an icon already created within a window.

Before calling

R0 Window handle

R1 Icon handle

R2 Minimum X of icon bounding box

R3 Minimum Y of icon bounding box

R4 Maximum X of icon bounding box

R5 Maximum Y of icon bounding box

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

R5 Preserved

[RISC OS Open source page](#)

Wimp_SendMessage

SWI &400E7 · WindowManager

The purpose of this call is to send a message either to a particular task or to all tasks.

Before calling

R0 Reason Code

R1 Pointer to message block

R2 Destination task handle, or

Destination window handle, or

- 2 for creator of icon handle in R3, or

0 to broadcast to all (including self)

R3 Icon handle (if R2 = -2)

On return

R0 Corrupt

R2 Handle of destination task (if not broadcast)

Block Updated

Notes

All messages are passed around the Wimp using this call.

Tasks are notified of incoming messages via Wimp_Poll.

The type of message to be sent is dependant on the reason code value.

The message block in R1 should be in the format specified by the particular Reason Code being used in R0.

When replying to a message, R2 should be the task handle of the original message.

Reason code values of 1 to 12 can be used to inform another task about a window, mouse, icon etc...

Reason codes of 17 to 19 are user messages and are the most common type of message to send.

- User_Message (reason code 17) is used to inform a task of something.
- User_Message_Recorded (reason code 18) is used to inform a task of something but requires a message of acknowledgement in return. If no acknowledged message is received, the Wimp will send the message back to the originator (unless the message's type is not in the list of message types that the application accepts).
- User_Message_Acknowledge is reason code 19 is used to reply back to a task that sent a User_Message_Recorded message.

[RISC OS Open source page](#)

Wimp_SetCaretPosition

SWI &400D2 · WindowManager

The purpose of this call is to move the caret to a new position..

Before calling

R0 Window handle, or -1 to turn off

R1 Icon handle, or -1 for caret in window work area

R2 X offset

R3 Y offset

R4 Caret Flags, or -1 if none

R5 Index into string, or -1 if writeable icon

Notes

The index value is a byte index, and not a character index. Therefore, there is only a distinction when the system alphabet is UTF8.

Tasks are informed of a window gaining or losing input focus by Gain_Caret and Lose_Caret events via Wimp_Poll.

[RISC OS Open source page](#)

Wimp_SetColour

SWI &400E6 · WindowManager

The purpose of this call is to set the current graphic foreground or background colour and action to one of the Wimp's 16 standard colours.

Before calling

R0 Colour Value (1 byte see below)

Bits

0-3 Wimp Colour

4-6 GCOL action

7 0 for foreground

1 for background

On return

R0 Corrupt

Notes

When used in non-16 colour screen modes, the Wimp will translate the colours to the closest approximation. More details on this can be found in the Colour section.

[RISC OS Open source page](#)

Wimp_SetColourMapping

SWI &400F8 · WindowManager

The purpose of this call is to change the way the Wimp maps its own colours to physical colours.

Before calling

R0 Pointer to palette to be used for converting Wimp colours, or

-1 to use default Wimp palette

0 to use palette defined by Wimp_SetPalette

R2 Pointer to 1bpp mapping array (2 bytes)

R3 Pointer to 2bpp mapping array (4 bytes)

R4 Pointer to 4bpp mapping array (16 bytes)

R5 0

R6 0

R7 0

On return

--

Notes

The palette in R1 is 16 words long and in the BBGRRxx format.

[RISC OS Open source page](#)

Wimp_SetExtent

SWI &400D7 · WindowManager

The purpose of this call is to set the size of the work area dimensions of a specific window.

Before calling

R0 Window handle

R1 Pointer to block

Block

+0 New work area minimum x

+4 New work area minimum y

+8 New work area maximum x

+12 New work area maximum y

On return

R0 Corrupt

Notes

After setting the work area sizes, it is most likely that the Wimp will need to redraw the scroll bars to reflect the new total size of window.

The work area extent cannot be changed so that any part of the visible work area lies outside the extent. i.e. a windows current size cannot be changed, causing it to scroll.

[RISC OS Open source page](#)

Wimp_SetFontColours

SWI &400F3 · WindowManager

The purpose of this call is to set the colours used when displaying anti-aliased fonts.

Before calling

R1 Font background colour

R2 Font foreground colour

On return

R0 Corrupt

Notes

The font colours are not preserved across Wimp Polls.

[RISC OS Open source page](#)

Wimp_SetIconState

SWI &400CD · WindowManager

The purpose of this call is to set value of an icons Icon Flag by the setting of the individual bits.

Before calling

R1 Pointer to block

Block

+0 Window handle (-1 or -2 for icon bar)

+4 Icon handle

+8 EOR word

+12 Clear word

On return

R0 Corrupt

Notes

This call sets the icon flags 'state' at the bit level. It does this by the following method:

$\text{new state} = (\text{old state AND NOT 'clear word'}) \text{ EOR 'EOR word'}$

A combination of the EOR word and the clear word determines the value of the bits of the current icon flag. The table below shows how the bits are set when the EOR and clear words are used.

Clear / EOR / Meaning

0 – EOR: 0 · Meaning: Preserve the bit's status

0 – EOR: 1 · Meaning: Toggle the bit's state

1 – EOR: 0 · Meaning: Clear the bit

1 – EOR: 1 · Meaning: Set the bit

Any change to the icon flag will result in the icon being updated automatically on screen, however, if the justification of the icon is changed, and the icon has the input focus, then Wimp_SetCaretPosition should be called to make sure the caret

remains at the correct text position.

[RISC OS Open source page](#)

Wimp_SetMode

SWI &400E3 · WindowManager

The purpose of this call is to set the screen mode used by the Wimp.

Before calling

R0 Mode Specifier

On return

R0 Corrupt

Notes

The preferred method of changing the screen display is by specifying the selector block as it allows for many more types of display.

The palette and mouse pointer are reset upon use of this call.

Screen memory is reallocated, and the screen redrawn automatically.

Other languages documentation

Français

[RISC OS Open source page](#)

Wimp_SetPalette

SWI &400E4 · WindowManager

The purpose of this call is to edit the Wimp palette values.

Before calling

R1 Pointer to Palette Block (20 word size)

On return

R0 Corrupt

Notes

In 16 colour screen modes the video palette is also edited, and in other modes the Wimp's colour lookup table may change.

For the change to take effect, the screen must be redrawn.

It causes Service_WimpPalette to be issued.

[RISC OS Open source page](#)

Wimp_SetPointerShape

SWI &400D8 · WindowManager

The purpose of this call is to change the appearance of the mouse pointer and the hot point.

Before calling

R0 Shape number (0 to turn off)

R1 Pointer to shape data, or

-1 for no change

R2 Width in pixels (multiple of 4)

R3 Height in pixels

R4 Active point x offset (hot point)

R5 Active point y offset (hot point)

On return

R0 Corrupt

Notes

This call is now deprecated. To change the mouse pointer, Wimp_SpriteOp 36.

[RISC OS Open source page](#)

Wimp_SetWatchdogState

SWI &400FA · WindowManager

The purpose of this call is to implement a screenlocking mechanism, as found in screensavers and the alike.

Before calling

R0 0 to turn off, or

1 to turn on

R1 Password (must be same for off as for on)

On return

R0 Preserved

R1 Preserved

Notes

It is advisable to implement a password when turning on the watchdog. The password must then be supplied before the watchdog can be turned off. This will also ensure that other tasks cannot over-ride a screenlock.

[RISC OS Open source page](#)

Wimp_SlotSize

SWI &400EC · WindowManager

The purpose of this call is to read or set the amount of memory allocated to the current, or next application to be loaded.

Before calling

R0 New size of current slot, or

-1 to return size

R1 New size of next slot, or

-1 to return size

On return

R0 Size of current slot (bytes)

R1 Size of next slot (bytes)

R2 Size of free pool (bytes)

Notes

The use of 32bit integers means that the returned values may be incorrect on systems with over 2GB of RAM (especially the free pool size). To avoid this, consider using Wimp_Extend (read slot size), which will return the values in units of pages.

[RISC OS Open source page](#)

Wimp_SpriteOp

SWI &400E9 · WindowManager

The purpose of this call is to perform a sprite related operation within the Wimp.

Before calling

R0 Sprite Reason Code

R1 Unused

R2 Pointer to sprite name

... Specific to each reason code

On return

R1-R10 Results of OS_SpriteOp call

Notes

This call is actually mapped onto the more generic OS_SpriteOp call. However, &100 is automatically added to the value in R0, and R1 is set to the Wimp sprite area pointer.

Note that operations which attempt to modify sprites or the sprite area are generally not allowed and will produce an error. Consult the table at OS_SpriteOp for details on which operations are and aren't supported.

Additionally, using reason code 24 to read the address of a sprite is discouraged. The address may change across Wimp_Poll calls due to other sprites being merged into the area, and using the pointer to directly modify the sprite is forbidden as it may be held in ROM or a supposedly read-only filesystem such as ResourceFS. From a performance standpoint there is also little reason to use a direct pointer to a Wimp sprite, as the Wimp internally caches pointers to its sprites.

[RISC OS Open source page](#)

Wimp_StartTask

SWI &400DE · WindowManager

The purpose of this call is to create a new task and execute the given * command.

Before calling

R0 Pointer to *command string

On return

R0 Task handle of child, or
0 if it has exited

Notes

The SWI returns control when the new task exits, or calls Wimp_Poll.

This call can only be used from USR mode with an empty SVC stack, and only when it is the 'live' task. i.e gained control from Wimp_Initialise or Wimp_Poll.

It is not sufficient to drop from SVC mode into USR mode to call this SWI, and precludes starting tasks from within:

- Callbacks
- Interrupts
- Filters
- SWIs
- Vectors (events, etc...)
- Any other system call mechanism

[RISC OS Open source page](#)

Wimp_TextColour

SWI &400F0 · WindowManager

The purpose of this call is to specify the colour of VDU 4 text only.

Before calling

R0 0-3 · Wimp colour
7 0 for Foreground colour
1 for background colour

On return

R0 Corrupt

Notes

This call is only used with VDU 4 text.

To specify the text colour of anti-aliased fonts within the Wimp, please use Wimp_SetFontColours

[RISC OS Open source page](#)

Wimp_TextOp

SWI &400F9 · WindowManager

The purpose of this call is to ensure that the current anti-aliased desktop font is used correctly.

Before calling

R0 Bits 0 -7: Reason Code:

0 Set Text Colours

1 Calculate width of string

2 Plot text

3 Get split point

4 Truncate

Bits 8 - 31: Flags (reason code dependent):

On return

R0 Corrupt, or used to return values

Notes

The value of the reason code specifies the meaning of the call.

[RISC OS Open source page](#)

Wimp_TransferBlock

SWI &400F1 · WindowManager

The purpose of this call is to transfer a block of data from one task to another.

Before calling

R0 Handle of source task

R1 Pointer to source data

R2 Handle of destination task

R3 Pointer to destination buffer

R4 Length of buffer

On return

R0 Corrupt

Notes

This call is often used to transfer data to another task when a user has dragged data from one task to another.

The data transferred is byte-aligned.

[RISC OS Open source page](#)

Wimp_UpdateWindow

SWI &400C9 · WindowManager

The purpose of this call is to redraw the parts of a window that needs updating.

Before calling

R1 Pointer to block

Block

+0 Window handle

+4 Work area rectangle minimum x (inclusive)

+8 Work area rectangle minimum y (inclusive)

+12 Work area rectangle maximum x (exclusive)

+16 Work area rectangle maximum y (exclusive)

On return

R0 0 for no more to do, non-zero for update according to returned block

Notes

“Work area rectangle” means the area you wish to redraw given in user co-ordinates.

This call is similar to Wimp_GetRectangle, but with some important differences.

- Not all of the window has to be updated, only the specified rectangular area.
- Invalid rectangles are not cleared by the Wimp after redrawing using this call.
- Wimp_UpdateWindow can be called at anytime – not just in response to a Redraw_Window_Request.

This call exits via Wimp_GetRectangle, which specifies the co-ordinates of the rectangular parts needing updating (one at a time).

An example of how to use this call is shown in pseudo code form below:

```
SYS"Wimp_UpdateWindow",blk TO rectangles
WHILE rectangles
  update the contents of the returned rectangle
SYS"Wimp_GetRectangle",blk TO rectangles
ENDWHILE
```

[RISC OS Open source page](#)

Wimp_WhichIcon

SWI &400D6 · WindowManager

The purpose of this call is to search all icons within a specific window for an icon with an icon flag that matches the pattern specified in R3.

Before calling

R0 Window handle

R1 Block to contain icon handles

R2 Bit mask

R3 Bit settings to match

On return**R0** Corrupt**Block** Updated**Notes**

For each icon that matches the search criteria, its handle number is added to the block pointed to by R1.

The list of returned icon handles are terminated by a value of -1.

The mask value in R2 is used to specify which bits are used to compare against. An example of how this call works is shown below.

(icon flags AND R2) = (R3 AND R2)

[RISC OS Open source page](#)

Service calls

Notifications broadcast between modules. The return-register rules determine whether a handler claims a service or passes it on.

Service_ADFSPodule

Service &10800

This service is by the ADFS to enable other expansion cards to use their own hardware instead of those built into the machine. The expansion cards should claim the service call and update the passed registers to the values of its own hardware.

Before calling

R1 Reason Code 67584 (&10800)

R2 Address of current disc controller

R3 Address of IRQ status register for controller

R4 Mask for IRQ status register

When ANDed with the IRQ status register gives a non-zero value if IRQs are enabled

R5 Address of IRQ mask register for current disc controller

R6 Mask for IRQ mask register

When ORRed with the IRQ mask register, enables IRQ

On return

All registers preserved to pass on, else

R1 0 (claim service)

R2 Address of new disc controller

R3 Address of IRQ status register for new disc controller

R4 Mask for IRQ status register (AND)

R5 Address of IRQ mask register for controller

R6 Mask for IRQ mask register (OR)

[RISC OS Open source page](#)

Service_ADFSPoduleIDE

Service &10801

This service is by the ADFS to enable other IDE expansion cards to use their own hardware instead of those built into the machine. The expansion cards should claim the service call and update the passed registers to the values of its own hardware.

Before calling

R1 Reason Code 67585 (&10801)

R2 Address of current disc controller

R3 Address of IRQ status register for controller

R4 Mask for IRQ status register

When ANDed with the IRQ status register gives a non-zero value if IRQs are enabled

R5 Address of IRQ mask register for current disc controller

R6 Mask for IRQ mask register

When ORred with the IRQ mask register, enables IRQ

R7 Address of data read routine for current disc controller (0 for default)

R8 Address of write read routine for current disc controller (0 for default)

On return

All registers preserved to pass on, else

R1 0 (claim service)

R2 Address of new disc controller

R3 Address of IRQ status register for new disc controller

R4 Mask for IRQ status register (AND)

R5 Address of IRQ mask register for controller

R6 Mask for IRQ mask register (OR)

R7 Address of data read routine for current disc controller (0 for default)

R8 Address of write read routine for current disc controller (0 for default)

[RISC OS Open source page](#)

Service_ADFSPoduleIDEDying

Service &10802

This service is by an IDE expansion card to warn ADFS that the expansion card will be terminating and cease to be.

Before calling

R1 Reason Code 67586 (&10802)

On return

All registers preserved to pass on

[RISC OS Open source page](#)

Service_BufferStarting

Service &6F · Service_BuferStarting

This service is issued when the buffer manager starts.

Before calling

R1 111 (&6F)

On return

R1 Preserved

[RISC OS Open source page](#)

Service_CalibrationChanged

Service Call &5B · ColourTrans

This service is issued by ColourTrans when the ColourTrans_SetCalibration SWI is called.

Before calling

R1 91 (&5B)

On return

All registers preserved

Notes

It used whenever the screen has been calibrated.

This Service Call should never be claimed.

[RISC OS Open source page](#)

Service_ClaimFIQ

Service &0C

This service is issued by a module running as a foreground task (not IRQ), that wants to claim the FIQ hardware vector.

Before calling

R1 12 (&0C)

On return

All registers preserved to pass on, else

R1 0 (claim service)

Notes

Under RISC OS the FIQ vector can only be used by one claimant at a time. Once the vector has been claimed, the claimant is required to write their FIQ handler code directly to the processor vectors. 228 bytes of space are available to store the code, starting at (vector base + &1C). The vector base can be determined by examining bit 20 of R0 returned by OS_PlatformFeatures 0, or by OS_PlatformFeatures 32. Once the handler code is installed the driver can safely enable FIQs in the device and the system interrupt controller.

If you are the current owner of the FIQ vector and you receive this service call, you must release the vector and claim the service call. Because this call is made in the foreground, it is permissible to block for a short period of time until it is safe for you to stop using the vector.

[RISC OS Open source page](#)

Service_ClaimFIQinBackground

Service &47

This service is issued by a module running as a background task (i.e. IRQ) that wants to claim the FIQ hardware vector, or a foreground task that wants to poll availability of the vector.

Before calling

R1 71 (&47)

On return

R1 0 to claim service, else preserved to pass on

Notes

Unlike Service_ClaimFIQ, the current owner of the FIQ vector is not required to release the vector when he receives this call. Code wanting to claim the vector from the background must pay attention to whether the service call was claimed in order to determine whether the vector was released or not. If the service call was claimed ($R1=0$) then the FIQ vector was released and the caller is now the new owner. If the service call was not claimed ($R1 \neq 0$) then the current claimant is retaining ownership.

If you are the current owner of the FIQ vector and you receive this call, and you wish to relinquish the vector, then you should do so and signal this by claiming this service. Because this call may be made from the background, you should only claim the call if you can release the vector without blocking.

[RISC OS Open source page](#)

Service_CloseFile

Service Call &68 · FileSwitch

This service is called when the system requests that an object, and any children of that object be closed.

Before calling

R1 104 (&68)

R2 Pointer to canonical filename (null terminated)

R3 Number of files closed as a result of this service call

On return

R1 Preserved

R2 Preserved

R3 Number of files closed as a result of this service call

Notes

This Service Call should never claimed.

A module need not close the file, if this may potentially cause problems.

Before passing this service call on, you must increment R3 by the number of files that your code has closed.

[RISC OS Open source page](#)

Service_ColourPickerLoaded

Service &93

This service is for internal use only, and should not be used

Before calling

R1 147 (&93)

On return

--

[RISC OS Open source page](#)

Service_DCIDriverStatus

Service &9D

This service is issued by a driver when it starts or exits.

Before calling

R0 Driver Info Block

R1 157 (&9D)

R2 Status (0 if starting, 1 if terminating)

R3 DCI version supported

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

Notes

This service call must not be claimed.

When a protocol module receives this service call from a driver that is starting up (ie R2 = 0), it should add the driver to its list of device drivers. When a protocol module receives the call from a driver that is terminating (ie R2 = 1), it should scan its list of device drivers for a driver information block matching the one addressed by R0, and remove the corresponding driver from the list.

You may instead decide to keep a terminating driver on the list, but to mark it as inactive in case it later restarts. If you do this, you must not match driver information blocks by comparing their addresses, as when a driver restarts the block may well be in a different location. You instead have to match fields within the driver information block: the short name and unit number (at offsets 4 and 8 respectively) together uniquely identify a driver information block, and so a match of them is sufficient.

[RISC OS Open source page](#)

Service_DCIFrameTypeFree

Service &9E

This service is issued by the Internet module (or equivalent) when it relinquishes a frame type filter.

Before calling

R0 Driver Info Block

R1 158 (&9E)

R2 Frame type

R3 Address level

R4 Error level

On return

R0 -

R1 = 0 if the frame has been reclaimed by another module

R2 -

R3 -

[RISC OS Open source page](#)

Service_DCIProtocolStatus

Service &9F

This service is issued by a protocol module when it starts or exits. Clients of the protocol module (for example, Freeway is a client of Internet) may wish to track this in order to go dormant when the underlying protocol has gone, and re-register when it comes back.

Before calling

R0 Protocol module's private word pointer

R1 159 (&9F)

R2 Status (0 if starting, 1 if terminating)

R3 DCI version supported

R4 Pointer to protocol module's title string

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R3 Preserved

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_DesktopWelcome

Service Call &7C · WindowManager

This service call is broadcast prior to the desktop startup screen being displayed.

Before calling

R1 124 (&7C)

On return

R1 Value of 0 to claim and stop startup screen from appearing

Notes

The returning value in R1 should be claimed if a different startup screen is required, or to simply prevent the default from being displayed.

[RISC OS Open source page](#)

Service_DeviceDead

Service &79

This service is issued by DeviceFS when a specified device has been killed - usually because a second device of the same name has been registered.

Before calling

R0 0

R1 121 (&79)

R2 Device driver handle

R3 Pointer to device name, or 0 if whole device driver is being deregistered

On return

All registers must be preserved

Notes

If the entire driver is being deregistered, then this service is issued for each name in turn, and a final time with R3 = 0.

[RISC OS Open source page](#)

Service_DeviceFSCloseRequest

Service &81

This service is issued by DeviceFS when an open attempt on a device cannot be complete.

Before calling

R1 129 (&81)

R2 File handle

On return

Preserve all registers to pass on, else

R1 0 (claim, if you were able to close file in R2)

Notes

The reason for this service call being issued by DeviceFS are:

- That device already has its maximum number of streams open
- A non-full-duplex device has one or more streams open in the conflicting direction

Hopefully sufficient blocking streams can be closed, which will allow the open request to be fulfilled. If you opened the stream in R2, and it need no longer be kept open, then you should close it and claim the service. The kernel responds to this service to close the printer and serial devices if it has implicitly opened them.

[RISC OS Open source page](#)

Service_DeviceFSDying

Service &71

This service is issued by DeviceFS when it is about to be killed.

Before calling

R0 0

R1 113 (&71)

On return

R0 Preserved

R1 Preserved

Notes

All devices will have already been issued with Service_DeviceDead.

[RISC OS Open source page](#)

Service_DeviceFSStarting

Service &70

This service is issued by DeviceFS when it reinitialises.

Before calling

R1 112 (&70)

On return

R1 Preserved

Notes

Unlike Service_ResourceFSStarting, DeviceFS has the good sense to issue this service on a Callback, so its SWIs (e.g. DeviceFS_Register) can be called in the normal fashion.

[RISC OS Open source page](#)

Service_DiscDismounted

Service &7D

This service is issued when a disc is dismounted.

Before calling

R1 125 (&7D)

R2 Pointer to filing_system::disc

On return

R1 Preserved

R2 Preserved

Notes

This allows the Filer to close open directory displays. Disc name should be used in preference to disc number in the string in R2.

[RISC OS Open source page](#)

Service_DisplayChanged

Service &DE

This service call is issued by OS_ScreenMode 11 when the OS is switching to a different video driver, or when the kernel receives a Service_DisplayStatus 'configuration changing' service call for the current GraphicsV driver.

Before calling

R0 Reason

R1 222 (&DE)

R2 New display number

R3 Sub-reason:

0 = Mode will be changed

128 = Mode will not be changed

Reason Codes

/ Hex # / Action

0 – Hex #: &00 · Action: Display is in process of being changed

1 – Hex #: &01 · Action: Display change complete

128 – Hex #: &80 · Action: Display change about to begin

In practice

Under RISC OS 5 the service call is issued three times, for three different stages of the operation:

- The first call will be made with reason 128 (change about to begin). At this point in time the VDU state will not have changed in any way. Programs should cease any interrupt-driven VDU output or direct screen access, as the VDU state is about to become indeterminate.

- The second call will be made with reason 0 (change in progress), after the kernel has updated VDU variable 192 to contain the new driver number. However the rest of the VDU state will not have been updated yet. Software which deals with screen mode selection (e.g. the ScreenModes module) should recache its mode lists for the new driver.

- The third call will be made with reason 1 (change complete), once the kernel has fully switched over to the new driver and selected a new screen mode. At this point the VDU is stable again and it is safe for interrupt-driven VDU output to resume.

For all the stages, R2 contains the new display number (i.e. GraphicsV driver number).

If activation of the new driver fails for any reason, the kernel will attempt to restore the previous driver. This will result in a sequence of four service calls instead of the usual three; reasons 128 and 0 will be issued with R2 set to the new driver, followed by 0 and 1 for the original driver.

Notes

The value of R3 is a sub-reason code. Sub-reason code 0 is used from within OS_ScreenMode 11 to indicate that the operation is going to result in the screen mode being changed. Sub-reason code 128 is used from within Service_DisplayStatus to indicate that the screen mode isn't going to change. Other sub-reason codes may be defined in future.

Under RISC OS 5, the display number in R2 corresponds to the GraphicsV driver number.

Under RISC OS Select, the display number in R2 corresponds to the VideoV? driver number.

[RISC OS Open source page](#)

Service_DisplayFormatHelp

Service &6C

This service is issued by Help Format.

Before calling

R0 0

R1 108 (&6C)

On return

R0 Preserved to pass on

R1 Preserved to pass on

R0 Pointer to error block

R1 P0 (to claim)

Notes

For each of your formats you should display one line of the form. format - description e.g. L - 640K, 47 entry directories, old map, all ADFS

Only claim this service if there is an error while displaying the help text.

[RISC OS Open source page](#)

Service_DisplayStatus

Service &DF

This service call is issued by the kernel when video drivers register or deregister with the system, or by drivers when their configuration/capabilities change in a noteworthy way.

Before calling

R0 Reason

R1 223 (&DF)

R2 Display number

R3 Reason 1: Pointer to device descriptor

Other reason codes: Undefined

Reason Codes

/ Hex # / Action

0 – Hex #: &00 · Action: Device deregistered

1 – Hex #: &01 · Action: Device registered

128 – Hex #: &80 · Action: Device configuration changing

129 – Hex #: &81 · Action: Device configuration changed

Device registration and deregistration

Under RISC OS 5, the display number in R2 corresponds to the GraphicsV driver number. Reason 1 is issued when a driver calls OS_ScreenMode 65, just after the kernel marks the driver as being ready for use. However because GraphicsV does not use device descriptors, the descriptor pointed to by R3 will be empty (a single terminator word). Reason 0 is issued when a driver calls OS_ScreenMode 66, before the kernel marks the driver as having stopped.

Under RISC OS Select, the display number in R2 corresponds to the VideoV? driver number, and the service calls are issued at the appropriate times for the VideoV API.

Device configuration changes

Under RISC OS 5, drivers will issue reason codes 128 and 129 when their configuration changes in a noteworthy way, e.g. when the list of available pixel formats changes. Reason code 128 should be issued before the change takes place and reason code 129 should be issued after the change is complete. This allows the driver to warn software running in the background that the driver is entering an indeterminate state, and to notify it when the state change is complete.

The kernel will listen out for configuration change service calls which correspond to the current GraphicsV driver and transform them into the appropriate Service_DisplayChanged calls. Specifically, DisplayChanged reason 128 (PreChanging) will be issued when the DisplayStatus 128 is received, and DisplayChanged reasons 0 (Changing) and 1 (Changed) will be issued once the change is complete. This forwarding of service calls means that software which is only interested in the current GraphicsV driver only has one service call to listen out for (Service_DisplayChanged) instead of two.

[RISC OS Open source page](#)

Service_DrawObjectDeclareFonts

Service &45541 · DrawFile

This service is issued whenever SWI DrawFile_DeclareFonts encounters an object whose type it doesn't know.

Before calling

R0 Object type? (Draw File Format)

R1 &45541 (reason code)

R2 Pointer to the render state?

On return

R1 0 if recognised, otherwise Preserved

Notes

Extension modules should recognise the object type, declare the fonts in it and then claim the service. Errors should only be reported if there is no error presently known - the DrawFile module will continue after errors, but only report the first one.

[RISC OS Open source page](#)

Service_DrawObjectRender

Service &45540 · DrawFile

This service is issued whenever SWI DrawFile_Render encounters an object whose type it doesn't know.

Before calling

R0 Object type? (DrawFile format)

R1 &45540 (reason code)

R2 Pointer to the render state?

On return

R1 0 if recognised, otherwise Preserved

Notes

Extension modules should recognise the object type, process it and then claim the service. Errors should only be reported if there is no error presently known - the DrawFile module will continue after errors, but only report the first one.

[RISC OS Open source page](#)

Service_DynamicAreaCreate

Service &90

This service is issued to inform the TaskManager when a dynamic are has been created.

Before calling

R1 144 (&90)

R2 Area number of dynamic area just created

On return

R1 Preserved

R2 Preserved

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_DynamicAreaRemove

Service &91

This service is issued to inform the TaskManager when a dynamic are is about to be removed.

Before calling

R1 145 (&91)

R2 Area number of dynamic area about to be removed

On return

R1 Preserved

R2 Preserved

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_DynamicAreaRenum

Service &92

This service is issued to inform the TaskManager when a dynamic are is being renumbered.

Before calling

R1 146 (&92)

R2 Old Area number

R3 New Area number

On return

R1 Preserved

R2 Preserved

R3 Preserved

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_EconetDying

Service &56

This service is issued when Econet is about to quit.

Before calling

R1 86 (&56)

On return

R1 Preserved to pass on (do not claim)

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_EnumerateFormats

Service &6A

This service is issued to get information about available disc formats from softloaded modules.

Before calling

R1 106 (&6A)

R2 Pointer to linked list of format specifications

On return

R1 Preserved (do not claim)

R2 Pointer to extended linked list

Notes

It should be issued whenever this information is requested, in case new modules mean more formats are now available. The service must not be claimed – rather it should be passed on with an extended list.

The service issuer is responsible for freeing the list and all data from the RMA after use, or if an error occurs during generation.

[RISC OS Open source page](#)

Service_EnumerateFormats Block

Part 16 - Service Calls

The block should be built in this order:

Block

- +0 – Contents: Pointer to next block, or 0 to terminate list
- +4 – Contents: Pointer to RMA block with text for Format submenu
- +8 – Contents: Pointer to RMA block with help text for Format submenu
- +12 – Contents: SWI to call to get raw disc format info
- +16 – Contents: R3 parameter for this SWI
- +20 – Contents: SWI to call to lay down disc structure
- +24 – Contents: R0 parameter for this SWI
- +28 – Contents: Flags:
 - bit 0 set for native ADFS format
 - All other bits reserved, must be zero
- Allocate block in RMA
- Zero offsets 4 and 8 (pointers to text)
- Link into list
- Allocate block in RMA for text at offset 4
- Link this to the block
- Allocate block in RMA for text at offset 8
- Link this to the block
- Copy text into the two RMA blocks
- Fill in the rest of the block

This allows an error to be returned by claiming the service and returning the intact list.

[RISC OS Open source page](#)

Service_EnumerateNetworkDrivers

Service &9B

This service is issued by anyone wishing to get a list of current network drivers.

Before calling

R0 Current head of list, or 0 to start

R1 155 (&9B)

On return

R0 New head of list

Notes

When a protocol module receives this service call it should allocate a 2 word block from RMA using OS_Module 6 and form a chain by storing R0 into offset +0 and a pointer to its own Driver Info Block to +4, then passing back its new allocation in R0 for the next recipient in the list to add to.

It is the service call issuer's responsibility to free the chain of 2 word blocks using OS_Module 7 afterwards.

[RISC OS Open source page](#)

Service_EnumerateScreenModes

Service &8D

This service is issued to enumerate the available screen modes.

Before calling

R1 141 (&8D)

R2 Number of modes to skip

R3 Monitor type

R4 Memory bandwidth available (in bytes/sec)

R5 Total amount of video RAM in system (in bytes)

R6 Pointer to block to return data, or 0 to just count entries

R7 Size of block (in bytes) if R6<>0, or 0 if R6=0

On return

R0 Preserved ?

R1 0 to claim (further valid modes are available, but would not fit in the block), else preserved

R2 - (number of modes filled in)

R3 Preserved

R4 Preserved

R5 Preserved

R6 Pointer to byte after last one filled in, or preserved if 0 on entry

R7 Amount of unused space in block, or -(amount of space need in block) if counting

Notes

Modules return information on all modes they provide that work on the specified monitor type and which require no more than the specified memory bandwidth and video memory.

OS_ScreenMode 2 provides a front-end for applications; you should use it rather than issuing this service call yourself.

By setting R6 and R7 to zero, clients can find the amount of space required to hold all the returned modes; they can then issue the call again to actually read the information. Alternatively, clients can use a fixed size buffer, and repeatedly issue the call until it is no longer claimed. When using this method, R2 on entry - the number of modes to skip this iteration - should be set to: (previous R2 on entry) - (R2 on exit).

[RISC OS Open source page](#)

Service_EnumerateScreenModes Block

Part 16 - Service Calls

For format 0, the rest of the block has the following format:

Block

Offset / Contents

+0 - Contents: Size of entry in bytes (multiple of 4)

+4 – Contents: Flags:

Bit 0: Must be 1

Bits 1-7: Block format

Bits 8-31: Additional format-specific flags

Offset / Contents

+8 – Contents: X resolution (in pixels)

+12 – Contents: Y resolution (in pixels)

+16 – Contents: Log2BPP value

+20 – Contents: Frame rate (in Hz, to nearest integer)

+24 – Contents: Mode name, null terminated, and then padded with nulls until it is word aligned. (For unnamed modes this will simply be a single word whose value is 0).

For format 1, the rest of the block has the following format:

Offset / Contents

+8 – Contents: X resolution (in pixels)

+12 – Contents: Y resolution (in pixels)

+16 – Contents: NColour value

+20 – Contents: ModeFlags value

+24 – Contents: Log2BPP value

+28 – Contents: Frame rate (in Hz, to nearest integer)

+32 – Contents: Mode name, null terminated, and then padded with nulls until it is word aligned. (For unnamed modes this will simply be a single word whose value is 0).

Block formats 2 and above are currently undefined.

Currently, no formats make use of the extra flags in bits 8-31 of the word at +4.

Notes

Block format 1, introduced with RISC OS 5.21, is an extended version of format 0 to allow the full range of pixel formats supported by the OS to be expressed. For backwards-compatibility, code which generates mode provider blocks should generate a format 0 block wherever possible. See Valid Mode Variable Combinations for a list of all pixel formats understood by various OS versions, and which formats can be expressed as a simple Log2BPP value (i.e. a format 0 block).

Software which processes mode provider blocks must be careful to check the low byte of the word at +4 and skip blocks with unknown or invalid formats.

[RISC OS Open source page](#)

Service_Error

Service &06

This service is issued after an error, but before the error handler is called.

Before calling

R0 Pointer to error block

R1 6 (&06)

On return

R0 Preserved

R1 Preserved

Notes

This service all should not be claimed.

[RISC OS Open source page](#)

Service_FilerDying

Service &4F

This service is issued when the Filer task is closed down. Filer module tasks should close down in sympathy when they receive this service (see Service_StartedFiler).

Before calling

R1 79 (&4F)

On return

R1 Module's taskhandle variable set to zero

[RISC OS Open source page](#)

Service_FilterManagerDying

Service Call &88 · FilterManager

This service is issued when The Filter Manager is in the process of quitting, to inform filters that they have been deregistered.

Before calling

R1 136 (&88)

On return

All registers preserved

Notes

This service call must not be claimed.

The timing of the service call means that it is not possible to call FilterManager SWIs from within the service call.

[RISC OS Open source page](#)

Service_FilterManagerInstalled

Service Call &87 · FilterManager

This service is issued when The Filter Manager starts to indicate that filters can be (re)registered with it.

Before calling

R0 Filter manager version x 100

R1 135 (&87)

On return

All registers preserved

Notes

This service call must not be claimed.

The service call is issued from a callback, so it is safe to use FilterManager SWIs from within the service call.

[RISC OS Open source page](#)

Service_FindNetworkDriver

Service &84

This service is issued when ...

Before calling

R1 132 (&84)

On return

R1 Preserved ?

Notes

This service is part of the AUN Driver Control Interface.

More information is required to complete this service call.

[RISC OS Open source page](#)

Service_FindProtocols

Service &41580

This service call is used in your terminal emulator to find all available compatible protocol modules.

Before calling

R1 Reason Code 267648 (&41580)

R2 Lowest TIP version supported × 100

R3 Last TIP version known × 100

R4 Emulator flags

On return

R1 0 to claim, else preserved to pass on

R2 Pointer to protocol name string (null terminated)

R3 Base SWI number of protocol module

R4 Pointer to protocol information block

R5 Protocol flags

Emulator Flags

0 – Value: 0 · Meaning: Emulator does not support file transfer calls

1 · Emulator supports file transfer calls

1 – 2 – Value: 00 · Meaning: Direction of link not important

01 · One-way link required – protocol to emulator

- 10 · One-way link required - emulator to protocol
- 11 · Two-way link required
- 3 – Value: 0 · Meaning: Bits 1 – 2 are minimum requirement
- 1 · Bits 1 – 2 are exact requirement
- 4 – 31 – Value: 0 · Meaning: All other bits are reserved and must be zero

Protocol Flags

Bits / Value / Meaning

- 0 – Value: 0 · Meaning: Can open new link
- 1 · Cannot open new link, or not useful
- 1 – Value: 0 · Meaning: Protocol does not support file transfer SWIs
- 1 · Protocol supports file transfer SWIs
- 2 – Value: 0 · Meaning: Protocol does not support Protocol_DirOp?
- 1 · Protocol supports Protocol_DirOp?
- 3 – 31 – Value: 0 · Meaning: All other bits are reserved and must be zero

Note: If the protocol is mainly for file transfer (such as Ftp) and the terminal emulator doesn't support file transfer calls (bit 0 of R3 was clear on entry) the protocol module should set bit 0 to show it's 'not useful'.

Notes

To use this SWI, you should follow these steps:

- Repeatedly issue this service call until it is not claimed - without polling the Wimp in the meantime
- Issue Service_FindProtocolsEnd

The protocol module checks to see if:

- It uses a version of the Acorn TIP in the range supported by the terminal emulator
- It supports links in the direction required by the terminal emulator

If one of the above isn't true, the protocol module must not claim the call - that is, it must return with registers preserved.

If both the above are true it must claim the call - that is, it must return with the values shown above in the section entitled On exit. It must then set an internal flag so it doesn't claim this call again until it receives a Service_FindProtocolsEnd.

[RISC OS Open source page](#)

Service_FindProtocolsEnd

Service &41581

This service call is used in your terminal emulator to indicate the end of your search for available protocols.

Before calling

R1 Reason Code 267649 (&41581)

On return

R1 0 to claim, else preserved to pass on

Notes

Protocol modules must change their internal flag so they respond again to Service_FindProtocols calls - from whatever terminal emulator the calls originate. They must not claim this call.

[RISC OS Open source page](#)

Service_FontsChanged

Service Call &6E · FontManager

This service is issued by the Font Manager to notify any module-based applications that they need to update their list of available fonts.

Before calling

R1 110 (&6E)

On return

All registers preserved

Notes

To retrieve the list of available fonts, an application should call Font_ListFonts.

[RISC OS Open source page](#)

Service_FreewayStarting

Service &95 · Freeway

The purpose of this service is to notify that Freeway is starting up.

Before calling

R1 &95 (reason code)

R2 Flags (all reserved, SBZ)

On return

R1 Preserved (must not be claimed)

[RISC OS Open source page](#)

Service_FreewayTerminating

Service &96 · Freeway

The purpose of this service is to notify that Freeway is terminating.

Before calling

R1 &96 (reason code)

R2 Flags (all reserved, SBZ)

On return

R1 Preserved (must not be claimed)

[RISC OS Open source page](#)

Service_FSRedeclare

Service Call &40 · FileSwitch

This service is called when the FileSwitch module has been reinitialised.

Before calling

R1 64 (&40)

On return

R1 Preserved to pass on (do not claim)

Notes

This Service Call should never claimed.

If you are in a filing system, you should make yourself known to FileSwitch by calling OS_FSControl 12.

[RISC OS Open source page](#)

Service_Hardware

Service Call &D9 · Hardware Abstraction Layer

Register interface

Before calling

R0 Bits 0-7 Reason code

Bits 8-31 flags (dependent on reason code)

R1 Service_Hardware (&D9)

- Other registers dependent on reason code

On return

- Depends on reason code

Reason Codes

/ Hex / Meaning

0 – Hex: &00 · Meaning: Device added

1 – Hex: &01 · Meaning: Device being removed

[RISC OS Open source page](#)

Service_Hardware 0

Service Call &D9 · Hardware Abstraction Layer

This service call is issued by the OS after a new device has been added to the system.

Before calling

R0 Bits 0-7 Reason code (0)

Bits 8-31 flags (reserved, will be 0)

R1 Service_Hardware (&D9)

R2 Device

On return

- All registers must be preserved

[RISC OS Open source page](#)

Service_Hardware 1

Service Call &D9 · Hardware Abstraction Layer

This service call is issued by the OS before a device is removed from the system. Claiming the service call by setting R1 to 0 will block the removal of the device.

Before calling

R0 Bits 0-7 Reason code (1)

Bits 8-31 flags (reserved, will be 0)

R1 Service_Hardware (&D9)

R2 Device

On return

R1 0 to object to device being removed

R0 error block

Other registers must be preserved

Else all registers must be preserved

[RISC OS Open source page](#)

Service_Help

Service &09

This service is issued when a *Help command is issued.

Before calling

R0 Pointer to command line

R1 9 (&09)

On return

All registers preserved to pass on, else

R1 0 (claim service)

Notes

This service call should not be claimed unless you wish to augment or replace the *Help command.

[RISC OS Open source page](#)

Service_IdentifyDisc

Service &69

This service is issued to identify a disc format. FileCore claims it for formats it recognises.

Before calling

R1 105 (&69)

R2 Pointer to buffer for format name

R3 Buffer length

R5 Pointer to disc record

R6 Sector cache handle

R7 Pointer to FileCore instance private word

On return

R1 Preserved

R2 Preserved

R3 Preserved

R5 Preserved

R6 New sector cache handle

R8 Preserved

R1 0 to claim

R2 Filetype number for given disc

R3 Preserved ?

R5 Pointer to modified disc record

R6 New sector cache handle

R8 Preserved

Notes

- (Unless any of the 4 values are zero) check the disc record's sector size, sectors per track, density and lowest sector ID number. Pass on the service if the sector scheme is unrecognised
- Update the values in the disc record for disc size, sequence sizes, double step and heads.
- Read the sector contents and check them - if it does not recognise these then pass on the service with R6 updated
- Update the values in the disc record for disc cycle id and disc name
- Fill the buffer in R2 with text suitable for the current format menu entry
- Claim the service call

If any of sector size, sectors per track, density and lowest sector ID number are 0 then these were not supplied by FileCore_MiscOp 0 - this should only happen for hard discs.

The heads field of the disc record should be updated in line with the sequence sides bit - if this bit becomes cleared add one to the heads, if this bit becomes set subtract one (but do not reduce heads below 0).

The sector contents should be read with FileCore_DiscOp 9 with R1 = flags 01×0 binary (1 second timeout, ignore escape, no alternative defect list) pointer to the alternative disc record supplied R2 = the disc address given by the root directory address on the disc record supplied

[RISC OS Open source page](#)

Service_IdentifyFormat

Service &6B

This service is issued to identify how to layout a disc format.

Before calling

R0 Pointer to format name (see Format Specifier)

R1 107 (&6B)

On return

R0 Preserved

R1 Preserved

R0 Preserved

R1 0 to claim

R2 SWI to call to get raw disc format info

R3 R3 parameter for this SWI

R4 SWI to call to lay down disc structure

R5 R0 parameter for this SWI

Notes

If DOSFS recognised the format name (which is different to FileCore names) then R2 would be DOSFS_DiscFormat and R4 DOSFS_LayoutStructure.

[RISC OS Open source page](#)

Service_IIC

Service &81100 · IIC

This service provides a variety of lookup operations relating to the IIC bus.

Before calling

R1 &81100

R2 Reason code

Reason Codes

/ Hex # / Action

0 – Hex #: &00 · Action: Lookup description

Notes

The reason code passed by R2 on entry determines the operation of the service call.

[RISC OS Open source page](#)

Service_IIC 0

Service &81100 · IIC

This service converts an IIC bus and address to a string name. It is offered to driver modules to provide a definitive name for IIC devices they recognise, otherwise a default string will be used.

Before calling

R1 &81100

R2 0

R3 Bits 0-7: Address of device, bit 0 clear

Bits 24-31: IIC bus number

On return

R1 0 to claim service, else preserved

R2 Pointer to zero terminated string

R3 Preserved

[RISC OS Open source page](#)

Service_International

Service &43 · Part 16 – Service Calls

This service provides a variety of conversion and information functions.

Before calling

R1 67 (&43)

R2 Reason code

Reason Codes

0 – Hex #: &00 · Action: Convert country name to number

1 – Hex #: &01 · Action: Convert alphabet name to number

2 – Hex #: &02 · Action: Convert country number to name

3 – Hex #: &03 · Action: Convert alphabet number to name

4 – Hex #: &04 · Action: Convert country number to alphabet number

5 – Hex #: &05 · Action: Define range of characters

6 – Hex #: &06 · Action: Info – New keyboard selected

7 – Hex #: &07 · Action: Define character from UCS

8 – Hex #: &08 · Action: Return UCS conversion table

9 – Hex #: &09 · Action: Convert country number to ISO3166-1 two letter abbreviation

Notes

The reason code passed by R2 on entry determines the operation of the service call.

[RISC OS Open source page](#)

Service_International 0

Service &43 · Part 16 – Service Calls

This service converts a country name to a number.

Before calling

R1 67 (&43)

R2 0

R3 Pointer to country name (null terminated)

On return

R1 0 to claim service, else preserved

R4 Country number, or preserved if unrecognised

Notes

If a module provides additional countries, it should check this service – if the name matches (case insensitive, allow abbreviations with ".") the one provided by the module, it should claim this service.

[RISC OS Open source page](#)

Service_International 1

Service &43 · Part 16 – Service Calls

This service converts an alphabet name to a number.

Before calling

R1 67 (&43)

R2 1

R3 Pointer to alphabet name (null terminated)

On return

R1 0 to claim service, else preserved

R4 Country number, or preserved if unrecognised

Notes

If a module provides additional alphabets it should check this service – if the name matches (case insensitive, allow abbreviations with ".") one provided by the module it should claim this service.

[RISC OS Open source page](#)

Service_International 2

Service &43 · Part 16 – Service Calls

This service converts a country number to an unterminated name.

Before calling

R1 67 (&43)

R2 2

R3 Country number

R4 Pointer to buffer

R5 Length of buffer

On return

R1 0 to claim service, else preserved

R2 Preserved

R3 Preserved

R4 Preserved

R5 Number of characters put into buffer, or preserved if not recognised

Notes

If a module provides additional countries it should check this service – if the number matches one provided by the module it should claim this service.

[RISC OS Open source page](#)

Service_International 3

Service &43 · Part 16 – Service Calls

This service converts an alphabet number to an unterminated name.

Before calling

R1 67 (&43)

R2 3

R3 Alphabet number

R4 Pointer to buffer

R5 Length of buffer

On return

R1 0 to claim service, else preserved

R2 Preserved

R3 Preserved

R4 Preserved

R5 Number of characters put into buffer, or preserved if not recognised

Notes

If a module provides additional alphabets it should check this service – if the number matches one provided by the module it should claim this service.

[RISC OS Open source page](#)

Service_International 332

Service &43 · Part 16 – Service Calls

Return character code mask for Alphabet.

In practice

Entry

R1 / &43 (International)

R2 – &43 (International): 332 (ChrCodeMask)

R3 – &43 (International): Alphabet number

R4 – &43 (International): &FF (important)

Exit

R2 / Updated with flags:

- Updated with flags:: b0-1: multicode type
- Updated with flags:: 0=none (eg Latin1)
- Updated with flags:: 1=codepoints (eg UTF-8)
- Updated with flags:: 2=sequences (glyph composition)
- Updated with flags:: 3=reserved

R4 – Updated with flags:: Bitmask for character codes

Notes

This is the API that documents the significant bits of R0 in OS_WriteC and WrchV. Other bits MUST be ignored.

R4 is defined to be contiguous, but is not necessarily whole bytes – e.g. Unicode is &1FFFFFF (21 bit).

[RISC OS Open source page](#)

Service_International 4

Service &43 · Part 16 – Service Calls

This service converts a country number to an alphabet number.

Before calling

R1 67 (&43)

R2 4

R3 Country number

On return

R1 0 to claim service, else preserved

R2 Preserved

R3 Preserved

R4 Preserved

R5 Alphabet number, or preserved if not recognised

Notes

If a module provides additional countries it should check this service – if the number matches one provided by the module it should claim this service.

[RISC OS Open source page](#)

Service_International 5

Service &43 · Part 16 – Service Calls

This service defines a range of characters for a given alphabet number.

Before calling

R1 67 (&43)

R2 5

R3 Alphabet number

R4 ASCII of first character

R5 ASCII of last character

On return

R1 0 to claim service, else preserved

R2 Preserved

R3 Preserved

R4 Preserved

R5 Preserved

Notes

If a module provides additional countries it should check this service – if the number matches one provided by the module it should claim this service, and call VDU 23 to define any characters in the range (R4-R5) inclusive defined in this alphabet.

[RISC OS Open source page](#)

Service_International 6

Service &43 · Part 16 – Service Calls

This service is generated when the keyboard selection is changed.

Before calling

R1 67 (&43)

R2 6

R3 New keyboard number

R4 Alphabet number

On return

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_International 7

Service &43 · Part 16 – Service Calls

This service defines a character from UCS.

Before calling

R1 67 (&43)

R2 7

R3 Character in system font to redefine

R4 UCS character (&00000000 – &7FFFFFFF)

On return

R1 0 to claim service, else preserved

R2 Preserved

R3 Preserved

R4 Preserved

Notes

Any module providing UCS character definitions should check to see if it can provide an 8×8 definition for the specified UCS character. If it can, it should claim the service call and define the character using VDU 23, otherwise all registers should be preserved.

This call is intended as an interim measure until the kernel supports large character sets. The International module provides several hundred character definitions – this call provides access to them.

[RISC OS Open source page](#)

Service_International 8

Service &43 · Part 16 – Service Calls

This service returns an UCS conversion table for a given alphabet number.

Before calling

R1 67 (&43)

R2 8

R3 Alphabet number

On return

R1 0 to claim service, else preserved

R2 Preserved

R3 Preserved

R4 Pointer to table if recognised, else preserved

Notes

Any module providing additional alphabets should check to see if it can provide a conversion table for the given alphabet number. If it can, it should claim the service and return a pointer to the table in R4, otherwise return with all register preserved.

The table has 256 32-bit entries, one for each character, giving the equivalent UCS code. If a character is not defined in the alphabet, its entry should contain &FFFFFFFF (not a valid UCS character).

Characters 0-31 and 127 are control codes in RISC OS – their entries in the tables of all alphabets should be 0-31 and 127 to guarantee a sensible translation to UTF8.

It is nonsensical to issue this service call for alphabet UTF8 (111). No module should claim such a call.

[RISC OS Open source page](#)

Service_International 9

Service &43 · Part 16 – Service Calls

This service converts a country number to the unterminated 2 letter abbreviation specified in ISO3166-1. Conventionally, this is the country's top level domain name.

Before calling

R1 67 (&43)

R2 9

R3 Country number

R4 Pointer to buffer

R5 Length of buffer

On return

R1 0 to claim service, else preserved

R2 Preserved

R3 Preserved

R4 Preserved

R5 Number of characters put into buffer, or preserved if not recognised

Notes

The UK uses 'uk' in preference to its allocated 'gb' abbreviation.

If a module provides additional countries it should check this service – if the number matches one provided by the module it should claim this service.

[RISC OS Open source page](#)

Service_InternetStatus

Service &B0

This service is issued by a protocol module such as Net or Internet.

Before calling

R0 Subreason code

R1 176 (&B0) (reason code)

On return

R0 Preserved

R1 Preserved

In practice

All other registers dependant upon the subreason code in R0

Subreason Codes

- 0 – Event: Interface address has been changed
- 1 – Event: Receives a network map from a gateway station
- 2 – Event: Interface up/down
- 3 – Event: Dynamic boot start (internal use only)
- 4 – Event: Dynamic boot reply (internal use only)
- 5 – Event: Dynamic boot offer (internal use only)
- 6 – Event: Dynamic boot request (internal use only)
- 7 – Event: Dynamic boot inform (internal use only)
- 8 – Event: Duplicate IP address seen
- 9 – Event: Media status change

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_InvalidateCache

Service Call &82 · ColourTrans

This service is issued by ColourTrans when the cache is flushed by ColourTrans.

Before calling

R1 130 (&82)

On return

All registers preserved

Notes

This Service Call should never be claimed.

[RISC OS Open source page](#)

Service_KeyHandler

Service &44

This service is issued after OS_InstallKeyHandler is called, or after a reset, to inform keyboard handlers what sort of keyboard is present.

Before calling

R1 68 (&44)

R2 Keyboard ID

On return

R1 Preserved

R2 Preserved

Notes

This service must not be claimed

[RISC OS Open source page](#)

Service_LookupFileType

Service &42

This service is issued when a filetype lookup in the File\$Type_XXX variable fails.

Before calling

R1 66 (&42)

R2 Filetype

On return

R1 0 to claim service (filetype known), else preserved

R2 First four characters, if known

R3 Last four characters, if known

Notes

This service call has limited use, as System Variables are more useful.

[RISC OS Open source page](#)

Service_Memory

Service Call &11 · WindowManager

This call is used when the contents-addressable memory in the memory controller is about to be re-mapped, which alters the memory map of the system. This call should be claimed if re-mapping should not take place.

Before calling

R0 Required amount of memory to change application space (signed value)

R1 17 (&11)

R2 Current Active Object pointer (CAO)

On return

R1 Value of 0 prevents re-mapping taking place

In practice

A module will be initially provided the current slot size for its application workspace (beginning at &8000). However, modules do not generally need this area, as they use RMA memory instead. Therefore, when an application calls Wimp_Initialise, the Wimp inspects the CAO, and if the value is within application workspace the Wimp does nothing. If, however, the CAO is outwith application workspace, then the Wimp will reduce the current slot size to zero automatically.

Some modules, e.g. BASIC, do require application workspace, and therefore the Wimp uses this call just before returning the application to its free pool.

Notes

RISC OS 5.24 & 5.26 contain a bug which will cause the sign of R0 to be lost when OS_ChangeDynamicArea is used to resize application space while the Wimp is inactive; application space shrinks will be reported as if they were a grow operation.

[RISC OS Open source page](#)

Service_MemoryMoved

Service &4E

This service is issued when OS_ChangeDynamicArea has just finished, and is used by the Wimp to tidy up.

Before calling

R1 78 (&4E)

On return

R1 Preserved to pass on (do not claim)

Notes

This service must not be claimed

[RISC OS Open source page](#)

Service_MessageFileClosed

Service &5E

This service is issued by MessageTrans to warn that all files have been closed.

Before calling

R0 0 (not sure if actually required)

R1 94 (&5E)

On return

R0 Preserved to pass on (do not claim)

R1 Preserved to pass on (do not claim)

Notes

This service call must not be claimed.

It is only needed by applications which use direct pointers into message files – they must re-open message files with MessageTrans_OpenFile and recache pointers.

If applications are using direct pointers (e.g. indirected icons) with MessageTrans_MakeMenus then they will need to close menus and re-make them if this service is issued.

Use Wimp_GetMenuState to check whether your own application's menu is open – close it if it is. h4. See also

- MessageTrans
- Wimp_GetMenuState
- MessageTrans_MakeMenus
- MessageTrans_OpenFile
- Service Calls

[RISC OS Open source page](#)

Service_MIDI

Service &58

This service provides a variety of MIDI related notifications.

Before calling

R0 Reason code

R1 88 (&58)

Reason Codes

/ Hex # / Action

0 – Hex #: &00 · Action: Initialise

1 – Hex #: &01 · Action: Remove

Notes

The reason code passed by R0 on entry determines the operation of the service call.

[RISC OS Open source page](#)

Service_MIDI 0

Service &58 · Part 16 – Service Calls

This service is issued when MIDI module is about to be initialised.

Before calling

R0 0

R1 88 (&58)

On return

R0 Preserved

R1 Preserved

Notes

This is service call is for advanced use only, where the actual address of the MIDI module or its workspace is being implicitly or explicitly used.

Service_MIDI 1 is used to inform that a MIDI module is about to be removed.

[RISC OS Open source page](#)

Service_MIDI 1

Service &58 · Part 16 – Service Calls

This service is issued when MIDI module is about to be removed.

Before calling

R0 1

R1 88 (&58)

On return

R0 Preserved

R1 Preserved

Notes

This is service call is for advanced use only, where the actual address of the MIDI module or its workspace is being implicitly or explicitly used.

Service_MIDI 0 is used to inform that a MIDI module is about to be initialised.

[RISC OS Open source page](#)

Service_ModeChange

Service &46

This service is issued when a mode change has taken place.

Before calling

R1 70 (&46)

R2 Mode Specifier

R3 Monitor type

On return

R1 Preserved to pass on (do not claim)

R2 Preserved

R3 Preserved

Notes

This service must not be claimed

[RISC OS Open source page](#)

Service_ModeChanging

Service &89

This service is issued when a mode change about to take place (or after Service_PreReset).

Before calling

R1 137 (&89)

R2 Mode number

R3 Monitor type

On return

R1 Preserved to pass on (do not claim)

R2 Preserved (do not claim)

R3 Preserved (do not claim)

Notes

This service must not be claimed.

It is intended to be used by modules that claim PaletteV for particular combinations of mode and monitor.

[RISC OS Open source page](#)

Service_ModeExtension

Service &50

This service is used by the kernel to look up information about a mode which the kernel does not recognise. Most significantly, it is the mechanism by which the ScreenModes module communicates mode timings to the kernel when an MDF is in use.

Before calling

R1 80 (&50)

R2 Mode Specifier (number or selector, NOT a Sprite Mode Word)

R3 Monitor type, or -1 for 'unspecified'

R4 Bandwidth limit (bytes/sec) 1

R5 Memory limit (bytes) 1

On return

R1-R4 Preserved to pass on, else:

R1 0 to claim service

R2 Preserved

R3 Pointer to VIDC List

R4 Pointer to Mode Workspace List (if a mode number was provided in R2 on entry)

Else zero (if a Mode Selector Block was provided on entry)

Notes

For numbered modes, the contents of the mode workspace list must be identical no matter the monitor type selected. Only the contents of the VIDC list may differ by monitor type.

Under RISC OS 5, implementers of Service_ModeExtension handlers are expected to ensure that any VIDC list returned has been validated against the current GraphicsV driver.

For monitor types 0-6 and 8, the kernel has several built in tables of mode timings and workspace parameters that describe the built in numbered modes. Prior to RISC OS 5.28, these tables were effectively private to the kernel, limiting the ability for external software to generate its own VIDC lists by calling Service_ModeExtension. In RISC OS 5.28 and later this has been rectified, by adding a Service_ModeExtension handler to the kernel which is able to query those tables and return a VIDC list which is valid for the current GraphicsV driver. The kernel's Service_ModeExtension handler supports mode lookup by either mode number or (sufficiently accurate) Mode Selector Block. The addition of the Service_ModeExtension handler to the kernel also allowed for removal of some special-case logic that existed for handling those numbered modes during mode vetting (OS_CheckModeValid) and mode setting (e.g. OS_ScreenMode). Now, those code paths are fully reliant on Service_ModeExtension for generating VIDC lists.

Mode workspace lists and VIDC lists returned by this call may be in a temporary buffer and at risk of being overwritten by the next call to this service call - when calling Service_ModeExtension, care should be taken to copy them to your own buffer ASAP.

1 The R4 and R5 parameters were added to the service call specification in RISC OS 3.5. They allow ScreenModes to filter the MDF entries by the VRAM size and memory bandwidth limit of the machine.

ScreenModes 0.33 and above (RISC OS 5.20+) ignores R4, and instead relies on GraphicsV_VetMode to check for any relevant bandwidth limits. This is because on modern systems, the video driver is in a much better position to determine any bandwidth limitations than the kernel.

[RISC OS Open source page](#)

Service_ModeFileChanged

Service &94 · ScreenModes

This service is issued by ScreenModes when a new Monitor Definition File has been loaded, or when some other event has caused the list of available screen modes to change (e.g. when Service_DisplayChanged is issued in response to a configuration change in a video driver).

Before calling

R1 148 (&94)

On return

All registers preserved

Notes

This is a broadcast message that should not be claimed.

[RISC OS Open source page](#)

Service_ModeTranslation

Service &51

This service is issued by OS_CheckModeValid or a mode change if the requested mode isn't supported by the current monitor type / video hardware.

Before calling

R1 81 (&51)

R2 Mode number, shadow bit clear

R3 Monitor type

On return

R1-R3 Preserved to pass on, else:

R1 0 to claim service

R2 Substitute mode (mode number or mode Mode Selector Block)

R3 Preserved

In practice

Implementors of Service_ModeTranslation should check to see if they recognise the input mode and monitor type, and then check to see if they know of any suitable substitute mode (e.g. with similar resolution and colour depth) which is supported by the hardware. If such a mode is found, they can claim the call and return the new mode in R2.

Notes

This call is only issued for mode numbers, not Mode Selector Blocks.

For monitor types 0-6 and 8, the kernel has built in logic for handling translation of the built in numbered modes. Prior to RISC OS 5.28, this logic was directly integrated into the VDU driver, limiting the ability for external software to utilise or override the logic (e.g. Service_ModeTranslation would not be called for mode+monitor type combinations which were capable of being handled by the kernel's logic). In RISC OS 5.28, the logic was moved to a Service_ModeTranslation handler, placing the built in modes on an even footing with extension modes and other monitor types.

[RISC OS Open source page](#)

Service_ModulePostFinal

Service &DB

This service call is made by the kernel after a module has been finalised, including during module reinitialisation. At the time the call is made the module will not be linked into the module chain, so its SWIs etc are not available.

Before calling

R0 Module address

R1 219 (&DB)

R2 Pointer to module title

R3 0 for base instantiation, else R3 = pointer to postfix string

R4 BCD version number (derived from module's help string)

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

In practice

This service call is intended to inform clients of a module that any registrations they made with it are no longer active, and any associated handles are stale. Along with Service_ModulePostInit, this avoids the need for each module to have its own dedicated initialisation/finalisation service calls.

For modules which need to give their clients more warning (i.e. warning them before finalisation instead of after), it is also acceptable for modules to make this service call themselves, during their finalisation sequence. However there is currently no way of determining whether the call came from the kernel or from the module – programmers must consult the documentation of each module to determine whether the service call is issued manually or not.

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_ModulePostInit

Service &DA

This service call is made after a module has been (re)initialised and after it has been linked into the module chain.

Before calling

R0 Module address

R1 218 (&DA)

R2 Pointer to module title

R3 0 for base instantiation, else R3 = pointer to postfix string

R4 BCD version number (derived from module's help string)

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

Notes

This service call must not be claimed.

This means that any SWIs etc provided by the module are available (in contrast, during any service calls issued by the module's own initialisation code, the module is not yet linked into the chain).

This service call is intended to allow clients of a module to (re)register themselves with it.

[RISC OS Open source page](#)

Service_ModulePreInit

Service &B9

This service called is raised after the module has been loaded, prior to the module initialisation entry being called.

Before calling

R0 Module address

R1 185 (&B9)

R2 Module length + 4

On return

R0 Preserved

R1 0 if claimed, preserved otherwise

R2 Preserved

Notes

This service call is intended to allow run-time patching of modules. After patching, OS_SynchroniseCodeAreas will be called.

[RISC OS Open source page](#)

Service_MonitorLeadTranslation

Service &76

This service is issued by OS_ReadSysInfo 1 to determine the monitor's lead type.

Before calling

R1 118 (&76)

R2 Monitor lead ID

On return

R1-R5 Preserved to pass on, else

R1 0 (to claim service)

R2 Preserved

R3 Default screen mode for this monitor type

R4 Monitor type

R5 Sync type:

0 => separate syncs

1 => composite sync

Notes

This service is issued by OS_ReadSysInfo 1 if the configured value for any of Mode, Monitor Type or Sync is "auto", to determine these parameters from the monitor lead.

Older machines (e.g. A400 series vintage) cannot read monitor lead IDs.

[RISC OS Open source page](#)

Service_MouseTrap

Service Call &52 · WindowManager

This service is issued to inform applications that it has detected mouse movement.

Before calling

R0 Mouse x co-ordinate

R1 82 (&52)

R2 Button state (from OS_Mouse)

R3 Time of mouse event (from OS_ReadMonotonicTime)

R4 Mouse y co-ordinate

On return

- All registers preserved

Notes

This is often used by applications that wish to record mouse movements and the state of the mouse buttons so they can be played back. This can be very useful to produce applications that simulate human interaction. e.g. Demonstration software.

Applications that record the mouse positions and states, should also monitor INKEY values as well. Both positive and negative.

[RISC OS Open source page](#)

Service_NetFS

Service &55

This service by NetFS to inform NetFiler that things may have changed.

Before calling

R1 85 (&55)

On return

R1 Preserved to pass on (do not claim)

Notes

This service call must not be claimed.

As an example, this service call may be issued when any of the following have occurred:

- *Ligon
- *Bye
- *Sdisc
- *Mount

[RISC OS Open source page](#)

Service_NetFSDying

Service &5F

This service is issued when NetFS is about to quit.

Before calling

R1 95 (&5F)

On return

R1 Preserved

[RISC OS Open source page](#)

Service_NetworkDriverStatus

Service &8B

This service is issued when ...

Before calling

R1 139 (&8B)

On return

R1 Preserved ?

Notes

This service is part of the AUN Driver Control Interface.

More information is required to complete this service call.

[RISC OS Open source page](#)

Service_NewApplication

Service &2A

This service is issued when a new application is about to be started.

Before calling

R1 42 (&2A)

On return

R1 0 to claim service (prevents application starting), else preserved

Notes

Usually this occurs after a *Go, or *RMRun or *Run operation. If you provide another means of starting an application that does not include one of these, you should issue this Service and act accordingly.

[RISC OS Open source page](#)

Service_PagesSafe

Service &8F

This service is issued to inform recipients that the pages specified in R3 have been replaced by the pages specified in R4.

Before calling

R1 143 (&8F)

R2 Number of entries in page block array

R3 Pointer to 32 bit page block array before move

R4 Pointer to 32 bit page block array after move

On return

All registers preserved

Notes

This service call must not be claimed.

The logical addresses in both page blocks will match, and the page number and physical address fields give the pages which have moved.

Because of the page block format used, this call will only report changes to pages which have 32 bit addresses. On versions of RISC OS which support larger addresses, the OS will issue Service_PagesSafe64 after this call, describing all the pages

which have moved.

[RISC OS Open source page](#)

Service_PagesSafe64

Service &E4

This service is issued to inform recipients that the pages specified in R3 have been replaced by the pages specified in R4.

Before calling

R1 228 (&E4)

R2 Number of entries in page block array

R3 Pointer to 64 bit page block array before move

R4 Pointer to 64 bit page block array after move

On return

All registers preserved

In practice

This service is an extended form of Service_PagesSafe which uses 64 bit addresses. See the Service_PagesUnsafe documentation for full details of the behaviour of this group of service calls.

Notes

This service call must not be claimed.

The logical addresses in both page blocks will match, and the page number and physical address fields give the pages which have moved.

[RISC OS Open source page](#)

Service_PagesUnsafe

Service &8E

This service is issued to inform recipients that the pages specified are about to be swapped with different pages, e.g. as a result of a Dynamic Area PreGrow handler requesting pages which are currently in use by another area. DMA operations involving these pages should be suspended until Service_PagesSafe is received.

Before calling

R1 142 (&8E)

R2 32 bit page block array describing the affected pages, with all attributes filled in

R3 Number of entries in page block array

On return

All registers preserved

In practice

For example:

- Physical page 1 is currently mapped to logical address &8000
- A Dynamic Area PreGrow handler requests that physical page 1 be added to its DA
- The kernel detects that page 1 is already in use, so it takes a page from the free pool (e.g. page 200), and copies the contents of page 1 into it.
- The address translation for the pages is updated so that page 200 is now located at &8000. Page 1 gets added to the dynamic area.

By copying the contents of page 1 into page 200 and placing it at &8000, the kernel ensures that software which accesses memory by logical address (i.e. most software) will be completely unaffected by the fact that page 1 is now owned by the dynamic area.

However hardware devices and hardware drivers typically interact with memory using physical addresses. E.g. if a file is being loaded into memory at &8000, the device driver would initiate a DMA transfer to physical page 1. Under normal circumstances this would result in the requested data appearing at &8000 once the DMA transfer completes. But if page 1 gets reassigned during the transfer, and the device driver isn't notified of this, the data will appear in the DA's memory instead of appearing at &8000. This is why `Service_PagesUnsafe` exists - to provide DMA drivers the opportunity to pause their operation until the page has been swapped by the CPU (and then resume the operation using the new mapping, provided by `Service_PagesSafe`).

Note that the page copy & remap operation is performed with IRQs disabled and with the FIQ vector temporarily claimed by the kernel. This ensures that any interrupt handlers which are using the pages (by logical address) do not need to be aware of the copy operation.

Notes

This service call must not be claimed.

Because of the page block format used, this call will only report changes to pages which have 32 bit addresses. On versions of RISC OS which support larger addresses, the OS will issue `Service_PagesUnsafe64` before this call, describing all the pages. Specifically, service calls will be issued in the following sequence:

- `Service_PagesUnsafe64`, describing all pages
- `Service_PagesUnsafe`, describing only the pages which have 32 bit addresses (and which be given replacements with 32 bit physical addresses)
- `Service_PagesSafe`, describing only the pages which have 32 bit addresses (and which have been given replacements with 32 bit physical addresses)
- `Service_PagesSafe64`, describing all pages

Because the 64 bit versions always surround calls of the 32 bit versions, any software which sees a 64 bit call can ignore the following 32 bit calls.

[RISC OS Open source page](#)

Service_PagesUnsafe64

Service &E3

This service is an extended form of `Service_PagesUnsafe` which uses 64 bit addresses. See the `Service_PagesUnsafe` documentation for full details of the behaviour of this group of service calls.

Before calling

R1 227 (&E3)

R2 64 bit page block array describing the affected pages, with all attributes filled in

R3 Number of entries in page block array

On return

All registers preserved

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_PCI

Service &C3 · PCI

This service provides a variety of lookup operations relating to the PCI Manager.

Before calling

R1 195 (&C3)

R2 Reason code

Reason Codes

/ Hex # / Action

0 – Hex #: &00 · Action: Lookup description

1 – Hex #: &01 · Action: Lookup vendor description

Notes

The reason code passed by R2 on entry determines the operation of the service call.

[RISC OS Open source page](#)

Service_PCI 0

Service &C3 · PCI

This service converts a PCI function handle to a string name.

Before calling

R1 195 (&C3)

R2 0

R3 Function handle

On return

R1 0 to claim service, else preserved

R2 Pointer to zero terminated string

R3 Preserved

Notes

Service_PCI 1 is used to convert a PCI vendor ID to a string name.

[RISC OS Open source page](#)

Service_PCI 1

Service &C3 · PCI

This service converts a PCI vendor ID number to a string name.

Before calling

R0 Vendor ID

R1 195 (&C3)

R2 1

On return

R0 Preserved

R1 0 to claim service, else preserved

R2 Pointer to zero terminated string

Notes

Service_PCI 0 is used to convert a PCI function handle to a string name.

[RISC OS Open source page](#)

Service_PDriverChanged

Service &7F

This service is issued when PDriver changes the currently selected printer.

Before calling

R1 127 (&7F)

R2 Printer driver number

On return

R0 Preserved

R1 Preserved

Notes

The !Printers application will issue Message_SetPrinter? when the printer is changed.

[RISC OS Open source page](#)

Service_PDriverGetMessages

Service &78

This service is issued when a PDriver module is about to open
Resources:\$.Resources.PDrivers.Messages.

Before calling

R1 120 (&78)

On return

R1-R3 Preserved to pass on, else

R1 0 (claimed)

R3 Pointer to block (20 bytes in size)

Notes

If another PDriver module has already opened this file, it should claim the service and return the block pointed to by R3.

If a PDriver module issues this service and it goes unclaimed, it should allocate 20 bytes with OS_Module 6, pass the block to MessageTrans_OpenFile, and set the last word (usage count) to 1.

If the service is claimed, then the PDriver module should increase the usage count in the block by 1.

Either way, when the PDriver module is finished with the file, it should decrease the usage count in the block by 1. If this reduces the count to 0, it should call MessageTrans_CloseFile and then OS_Module 7 to free the block.

[RISC OS Open source page](#)

Service_PDriverGetMessages Block

Part 16 – Service Calls

Block

Size / Contents

16 – Contents: MessageTrans block, from MessageTrans_OpenFile

4 – Contents: Usage count

[RISC OS Open source page](#)

Service_PDriverStarting

Service &65

This service is issued when the PDriver module starts.

Before calling

R1 101 (&65)

On return

R1 Preserved

Notes

Printer drivers should call PDriver_DeclareDriver? to declare themselves.

[RISC OS Open source page](#)

Service_PDumperDying

Service &67

This service is issued when the PDriverDP module is about to quit.

Before calling

R1 103 (&67)

On return

R1 Preserved

[RISC OS Open source page](#)

Service_PDumperStarting

Service &66

This service is issued when the PDriverDP module starts.

Before calling

R1 102 (&66)

On return

R1 Preserved

Notes

Printer drivers should call PDriver_MiscOpForDriver? to declare themselves.

[RISC OS Open source page](#)

Service_Portable

Service &8A · Portable

This service call is issued by the Portable module before and after performing certain activities.

Before calling

R1 138 (&8A)

R2 Reason code

- Other registers dependent on call

On return

- Registers dependent on call

[RISC OS Open source page](#)

Service_Portable 0

Portable

This service is issued by the Portable module before powering down hardware.

Before calling

R1 &8A (Service call number)

R2 0 (reason code)

R3 Bitmask of which ports are being powered down

On return

R3 Bitmask – set bits mean no objection to power down

Notes

If this results in $R3 = 0$, the service should be claimed to prevent it being needlessly passed round remaining modules.

See also Service_Portable 1, which is issued when powering up hardware.

[RISC OS Open source page](#)

Service_Portable 1

Portable

This service is issued by the Portable module before powering up hardware.

Before calling

R1 &8A (Service call number)

R2 1 (reason code)

R3 Bitmask of which ports are being powered up

On return

R3 Bitmask – set bits mean no objection to power up

Notes

If this results in $R3 = 0$, the service should be claimed to prevent it being needlessly passed round remaining modules.

See also Service_Portable 0, which is issued when powering down hardware.

[RISC OS Open source page](#)

Service_Portable 3

Portable

This service is issued by the Portable module when the system is being placed in a state of suspension via SWI Portable_Stop.

Before calling

R1 &8A (Service call number)

R2 3 (reason code)

On return

- All registers preserved

Notes

At the time the service call is made, all hardware will still be powered on.

[RISC OS Open source page](#)

Service_Portable 4

Portable

This service is issued by the Portable module when the system has returned from a state of suspension (after being suspended by SWI Portable_Stop.)

Before calling

R1 &8A (Service call number)

R2 4 (reason code)

R3 Freeze failure reason code:

0 = Freeze was successful

1 = Failed to power down hard discs

2 = Failed to stop sound system

3 = Failed to blank screen

4 = Portable_Control call failed

5 = Portable_Control failed to power down something

6 = Battery manager busy

Other values: Reserved (interpret as freeze failure)

On return

- All registers preserved

Notes

At the time the service call is made, all hardware will have been returned to the power state it was in prior to suspension.

[RISC OS Open source page](#)

Service_Portable 5

Portable

This service is issued by the Portable module in response to SWI Portable_Stop, in order to check whether any software objects to the system being suspended.

Before calling

R1 &8A (Service call number)

R2 5 (reason code)

On return

R1 0 (service call claimed) to prevent system suspend

&8A (preserved) to allow system suspend

Notes

If any software objects to the system being suspended it should claim the service call by setting R1 to 0.

[RISC OS Open source page](#)

Service_Portable Ports

Portable

Ports

Bits / Hardware

0 – Hardware: Econet

3 – Hardware: Serial oscillator and buffer

14 – Hardware: Floppy Disc Controller oscillator

[RISC OS Open source page](#)

Service_Portable Reason Codes

Portable

Reason Codes

/ Type

0 – Type: PowerDown

1 – Type: PowerUp

2 – Type: TidyUp?

3 – Type: Freeze

4 – Type: Unfreeze

5 – Type: FreezeRequest

[RISC OS Open source page](#)

Service_PostInit

Service &73

This service is issued after all the ROM modules have been initialised.

Before calling

R1 115 (&73)

On return

R1 Preserved

Notes

This service must not be claimed.

It is unlikely that you will be able to use this service unless you are part of the OS or live on a Podule.

[RISC OS Open source page](#)

Service_PostReset

Service Call &27 · WindowManager

This service is issued at the end of a machine reset.

Before calling

R1 39 (&27)

On return

R1 Preserved. Do not claim

Notes

This service must not be claimed.

As MessageTrans does not close message files at the end of a machine reset, applications that wish to close their message files should call MessageTrans_CloseFile at this point.

[RISC OS Open source page](#)

Service_PreModeChange

Service &4D

This service allows a module to object to a mode change, or substitute a different mode.

Before calling

R1 77 (&4D)

R2 Mode Specifier

On return

R1 Preserved to pass on

R2 Preserved to pass on

R1 0 (claim service)

R2 0

R0 Pointer to error block

R1 0 (claim service)

R1 Preserved to pass on

R2 New Mode Specifier

Notes

Service_ModeTranslation offers a better method to support new monitor types.

[RISC OS Open source page](#)

Service_PreReset

Service &45

This service is issued when Break is released, just before a software generated reset occurs. This allows expansion card software to reset devices, as there is no hardware reset signal on the expansion card bus.

Before calling

R1 69 (&45)

On return

R1 Preserved to pass on (do not claim)

Notes

This service must not be claimed

[RISC OS Open source page](#)

Service_Print

Service &41

This service is for internal use only.

Before calling

R1 65 (&41)

On return

--

Notes

This Service call must not be claimed.

[RISC OS Open source page](#)

Service_ProtocolDying

Service &83

This service is issued when Econet protocol is dying.

Before calling

R1 131 (&83)

On return

R1 Preserved ?

Notes

This service is part of the AUN Driver Control Interface.

[RISC OS Open source page](#)

Service_ProtocolNameToNumber

Service &41582

This service call is used in your terminal emulator to request the conversion of a protocol name to a base SWI number.

Before calling

R1 Reason Code 267650 (&41582)

R2 Pointer to protocol name (null terminated)

On return

R1 0 to claim, else preserved to pass on

R2 Base SWI number for protocol

Notes

If a protocol module recognises the protocol name it must claim the call and return the base SWI number of the protocol. Otherwise it must pass the call on.

[RISC OS Open source page](#)

Service_ReAllocatePorts

Service &48

This service is issued when Econet restarts.

Before calling

R1 72 (&48)

On return

R1 0 to claim service, else Preserved to pass on

Notes

This service must not be claimed.

[RISC OS Open source page](#)

Service_ReleaseFIQ

Service &0B

This service must be issued by a module when it voluntarily releases the FIQ hardware vector.

Before calling

R1 11 (&0B)

On return

All registers preserved to pass on, else

R1 0 (claim service)

Notes

A module can claim this vector if it wants to take over ownership of the FIQ vector. If the call is left unclaimed then the kernel will take ownership, installing a default handler which disables and ignores any received FIQ.

This call should only be issued when a module voluntarily releases control of the FIQ vector. It should not be issued in response to Service_ClaimFIQ or Service_ClaimFIQinBackground.

[RISC OS Open source page](#)

Service_ResourceFSDying

Service &5A

This service is issued by ResourceFS just before it removes itself as a filing system.

Before calling

R1 90 (&5A)

On return

R1 Preserved to pass on (do not claim)

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_ResourceFSStarted

Service &59

This service is issued whenever files are added or removed from ResourceFS.

Before calling

R1 89 (&59)

On return

R1 Preserved to pass on (do not claim)

Notes

This service call must not be claimed.

This service call allows programs using files in ResourceFS to check if things have changed.

[RISC OS Open source page](#)

Service_ResourceFSStarting

Service &60

This service is issued by ResourceFS when it reinitialises

Before calling

R1 96 (&60)

R2 Address to call

R3 Workspace for ResourceFS

On return

R1-R3 Preserved to pass on (do not claim)

Notes

This service call must not be claimed.

As it is not linked into the module chain at this point, its SWIs cannot be called, so programs should call the code addressed by R2 directly:

STMFD sp!, {r0, lr} ; stack actual return address

ADR r0, files ; pointer to resource file block

MOV lr, pc ; emulate BL

MOV pc, r2 ; call ResourceFS code

LDMFD sp!, {r0, pc} ; and return

Unlike ResourceFS_RegisterFiles, the routine addressed by R2 does not issue Service_ResourceFSStarted for each block registered. Instead it issues one Service_ResourceFSStarted after all modules have received this service call.

[RISC OS Open source page](#)

Service_RTCSynchronised

Service &DD

This service is issued when the softcopy of the RTC has been re-initialised from the hardware RTC, e.g. via OS_ResyncTime.

Before calling

R1 221 (&DD)

On return

R1 Preserved

Notes

This service call is for informational purposes only and must not be claimed.

It indicates that the UTC time that is maintained by the kernel has been updated. The TerritoryManager will also report an updated local time, however note that TerritoryManager will only update the DST state when it receives the service call, so attempting to read the local time from within this service call may return incorrect values. Service_TimeZoneChanged will be issued by the territory manager if and when the timezone information has been corrected.

This call is generated by RISC OS 5.22 and later, and RISC OS Select.

[RISC OS Open source page](#)

Service_ScreenBlanked

Service &7A

This service is issued just after the monitor screen is blanked.

Before calling

R1 122 (&7A)

On return

R1 Preserved

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_ScreenRestored

Service &7B

This service is issued just after the monitor screen is restored.

Before calling

R0 0 for normal restoration

R1 123 (&7B)

Flags passed in R4 to ScreenBlanker_Control 2 if screen was flashed

On return

R0 Preserved

R1 Preserved

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_SCSIAttached

Service &20102

Issued when a new device or bus is registered with SCSIIdriver. A hardware driver may issue this itself to indicate that a device has been attached to an ID already registered with SCSIIdriver.

Before calling

R0 Device ID

R1 Reason Code 131330 (&20102)

R2 Device ID mask

On return

All registers preserved

In practice

The mask indicates whether a bus, a device or a unit (potentially multiple) has been attached; someone wanting to rescan the bus should scan all devices with IDs such that AND R2 = R0.

Example:

/

Bus 0, device 1 attached: – : R0 = &01 (0:1.0), R2 = &1F (3:7.0)

Bus 3, device 4, unit 5 attached: – : R0 = &BC (3:4.5), R2 = &FF (3:7.7)

Bus 2 attached: – : R0 = &10 (2:0.0), R2 = &18 (2:0.0)

[RISC OS Open source page](#)

Service_SCSIDetached

Service &20103

Issued when a new device or bus is deregistered with SCSIdriver. A hardware driver may issue this itself to indicate that a device has been detached without deregistering from SCSIdriver.

Before calling

R0 Device ID

R1 Reason Code 131331 (&20103)

R2 Device ID mask

On return

All registers preserved

In practice

The mask indicates whether a bus, a device or a unit (potentially multiple) has been detached.

Example:

/

Bus 0, device 1 detached: – : R0 = &01 (0:1.0), R2 = &1F (3:7.0)

Bus 3, device 4, unit 5 detached: – : R0 = &BC (3:4.5), R2 = &FF (3:7.7)

Bus 2 detached: – : R0 = &10 (2:0.0), R2 = &18 (2:0.0)

[RISC OS Open source page](#)

Service_SCSIDying

Service &20101

Issued before SCSIdriver dies. The SCSI system is still alive at this point, but a hardware driver may have shut down if it received this service call before you.

Before calling

R1 Reason Code 131329 (&20101)

On return

All registers preserved

[RISC OS Open source page](#)

Service_SCSIStarting

Service &20100

Issued as SCSIdriver initialises. The SWIs will not yet be available at this point - use a callback to call them if you need to.

Before calling

R1 Reason Code 131328 (&20100)

On return

All registers preserved

[RISC OS Open source page](#)

Service_SDIODescribeFIC

Service &81046 · SDIODriver

This service is used to to acquire a textual description of an SDIO device.

Before calling

R0 Function Interface Code

R1 &81046 (Reason Code)

On return

R1 0 to claim

R2 Pointer to zero-terminated string

Notes

This service call is used, for example, by *SDIODevices.

[RISC OS Open source page](#)

Service_SDIOSlotAcquired

Service &81044 · SDIODriver

This service is used to notify the OS that a slot has been acquired.

Before calling

R1 &81044 (Reason Code)

R2 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

On return

R1 Preserved (do not claim)

Notes

This service call is issued just after exclusive access to an SDIO slot has been acquired using SWI SDIO_Control.

[RISC OS Open source page](#)

Service_SDIOSlotAttached

Service &81040 · SDIODriver

This service is used to notify the OS that there is a slot attached to the machine.

Before calling

R1 &81040 (Reason Code)

R2 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

Bit 16 set => slot contains only non-removable IO device(s)

Bit 17 set => slot is removable but has non-functional card detect line

Bit 18 set => slot contains non-removable memory devices(s)

On return

R1 Preserved (do not claim)

[RISC OS Open source page](#)

Service_SDIOSlotDetached

Service &81041 · SDIODriver

This service is used to notify the OS that a slot has been detached from the machine.

Before calling

R1 &81041 (Reason Code)

R2 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

Bit 16 set => slot contains only non-removable IO device(s)

Bit 17 set => slot is removable but has non-functional card detect line

Bit 18 set => slot contains non-removable memory devices(s)

On return

R1 Preserved (do not claim)

[RISC OS Open source page](#)

Service_SDIOSlotReleased

Service &81045 · SDIODriver

This service is used to notify the OS that a slot has been released.

Before calling

R1 &81045 (Reason Code)

R2 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

On return

R1 Preserved (do not claim)

Notes

This service call is issued just before exclusive access to an SDIO slot is released. It is particularly useful in reducing the overhead of frequently issued SD commands, because it enables software to determine whether anyone else might have changed the card state (deselected the card, set a different block size and so on). In the common case where this doesn't happen, it means a number of SD commands can be omitted, allowing increased throughput.

To guard against reentrancy, the procedure that should be employed is thus:

- Allocate two flags, flag A to say that nobody else claimed the slot since you last did, and flag B to say that you have currently claimed the slot
- Call SDIO_Control to claim the slot
- SDIO_Control returns (if this fails then stop here)
- Set B
- Issue SD commands - fewer will be needed if A is set
- Set A
- Call SDIO_Control to release the slot
- Service_SDIOSlotReleased handler is entered with B set on entry; clear B
- At this point, interrupts may happen and further instances of Service_SDIOSlotReleased may occur with B now clear on entry; clear A if so
- SDIO_Control (from step 7) returns
- More instances of Service_SDIOSlotReleased may still occur with B clear on entry; clear A if so

This service call may have an additional benefit to enable a background process to be woken up if it previously found that the slot was not available. However you should note that the slot cannot be acquired from within the service call handler. Instead, you should schedule the background process to execute at a later time, for example using OS_CallAfter, OS_AddCallBack or the RTSupport module.

[RISC OS Open source page](#)

Service_SDIOUnitAttached

Service &81042 · SDIODriver

This service is used to notify the OS that there is a unit attached to the machine.

Before calling

R1 &81042 (Reason Code)

R2 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

Bits 16 - 31 => relative card address (RCA)

R3 Bits 0 - 7 = unit number

Bits 8-15 = function interface code (FIC) for IO, or 0 for memory (1 = UART, 2 = Bluetooth Type-A etc.)

Bit 16 set => device is write-protected

Bit 17 set => MMC (rather than SD/SDIO) device

Bits 18-31 reserved, 0

R4 Low word of card capacity (for memory cards)

R5 High word of card capacity (for memory cards)

On return

R1 0 to claim, else preserved

[RISC OS Open source page](#)

Service_SDIOUnitDetached

Service &81043 · SDIODriver

This service is used to notify the OS that a unit detached from the machine.

Before calling

R1 &81043 (Reason Code)

R2 Bits 0 - 7 = slot number

Bits 8 - 15 = bus number

Bits 16 - 31 => relative card address (RCA)

R3 Bits 0 - 7 = unit number

Bits 8-15 = function interface code (FIC) for IO, or 0 for memory (1 = UART, 2 = Bluetooth Type-A etc.)

Bit 16 set => device is write-protected

Bit 17 set => MMC (rather than SD/SDIO) device

Bits 18-31 reserved, 0

R4 Low word of card capacity (for memory cards)

R5 High word of card capacity (for memory cards)

On return

R1 0 to claim, else preserved

[RISC OS Open source page](#)

Service_ShutDown

Service Call &7E · WindowManager

This service call is broadcast by the Task manager when it is asked to perform a shutdown.

Before calling

R1 126 (&7E)

On return

R1 0 to claim and stop shutdown

Notes

To prevent a shutdown, it should be claimed. e.g. RamFS uses this to warn the user that there are unsaved files in the RAM disc.

[RISC OS Open source page](#)

Service_ShutdownComplete

Service Call &80 · WindowManager

This service call is broadcast at the end of a shutdown, when the shutdown message is displayed on screen.

Before calling

R1 128 (&80)

On return

- This service call should not be claimed

[RISC OS Open source page](#)

Service_Sound

Service &54

This service is issued by the sound modules when the sound configuration changes or when sound subsystems start and stop.

Before calling

R0 Reason code

R1 84 (&54)

On return

R0 Preserved

R1 Preserved

[RISC OS Open source page](#)

Service_Sound 10

Part 16 - Service Calls

This service call is made by SoundDMA when sound output is stopping, i.e. when sound is turned off by a call to SWI Sound_Enable. It will be made after the last buffer fill request has occurred. It is provided as a hint for any software that buffers audio in realtime, e.g. video playback software that needs to keep audio in sync with video, internet radio services, etc.

Before calling

R0 10 (reason code)

R1 &54 (Service call number)

On return

- All registers preserved

Notes

This call may be called with interrupts disabled; software should be careful not to re-enable interrupts or perform slow operations which will hinder the interrupt latency of the system.

Use SWI Sound_ReadSysInfo 1 to determine whether the version of SoundDMA present on the system generates this service call.

This service call should not be claimed.

[RISC OS Open source page](#)

Service_Sound 11

Part 16 - Service Calls

This service call is made by SoundChannels when the automatic tuning feature transitions from on to off, or vice-versa. By making this state change easy to detect, it allows other systems to use it as a trigger to enable or disable their own backwards-compatible modes. For example, SoundScheduler will disable tempo compensation when automatic tuning is disabled.

Before calling

R0 11 (reason code)

R1 &54 (Service call number)

R2 Automatic tuning state (1=off, 2=on)

On return

- All registers preserved

Notes

This service call should not be claimed.

[RISC OS Open source page](#)

Service_Sound 8

Part 16 – Service Calls

This service call is made by SoundDMA when the sample rate or buffer size of the sound system is changed.

Before calling

R0 8 (reason code)

R1 &54 (Service call number)

R2 New sample rate (1/1024Hz)

R3 New buffer size (samples per buffer)

On return

– All registers preserved

Notes

Typically this call will be made with interrupts disabled, to ensure all software is notified of the parameter change before the next buffer fill (using the new parameters) occurs. If the call is made with interrupts disabled, software should be careful not to re-enable interrupts or perform slow operations which will hinder the interrupt latency of the system.

Use SWI Sound_ReadSysInfo 1 to determine whether the version of SoundDMA present on the system generates this service call.

This service call should not be claimed.

[RISC OS Open source page](#)

Service_Sound 9

Part 16 – Service Calls

This service call is made by SoundDMA when sound output is starting, i.e. when sound is turned on by a call to SWI Sound_Enable. It will be made before the first buffer fill request occurs. It is provided as a hint for any software that buffers audio in realtime, e.g. video playback software that needs to keep audio in sync with video, internet radio services, etc.

Before calling

R0 9 (reason code)

R1 &54 (Service call number)

On return

– All registers preserved

Notes

This call may be called with interrupts disabled; software should be careful not to re-enable interrupts or perform slow operations which will hinder the interrupt latency of the system.

Use SWI Sound_ReadSysInfo 1 to determine whether the version of SoundDMA present on the system generates this service call.

This service call should not be claimed.

[RISC OS Open source page](#)

Service_Sound Reason Codes

Part 16 – Service Calls

Reason Codes

- 0 – Type: DMA handler starting
- 1 – Type: DMA handler dying
- 2 – Type: Channel handler starting
- 3 – Type: Channel handler dying
- 4 – Type: Scheduler starting
- 5 – Type: Scheduler dying
- 6 – Type: SharedSound starting
- 7 – Type: SharedSound dying
- 8 – Type: Sound config changing
- 9 – Type: Sound enabled
- 10 – Type: Sound disabled
- 11 – Type: Automatic tuning enabled/disabled

[RISC OS Open source page](#)

Service_StartedFiler

Service &4C

This service is issued by the Filer after resident Filer module tasks have started, to allow tasks that have failed to initialise to reset.

Before calling

R1 76 (&4C)

On return

R1 Preserved (do not claim)

Notes

For more information, see Service_StartFiler.

[RISC OS Open source page](#)

Service_StartedWimp

Service Call &4A · WindowManager

This service call is broadcast by the Desktop after resident module tasks have started, to allow tasks that have failed to initialise to reset.

Before calling

R1 74 (&4A)

On return

- This service call should not be claimed

In practice

See Service_StartWimp for details of the procedure.

[RISC OS Open source page](#)

Service_StartFiler

Service &4B

This service is issued by the Filer to start up resident filer module tasks by passing the returned command to Wimp_StartTask.

Before calling

R0 Task handle of Filer

R1 75 (&4B)

On return

R0 Pointer to command start module, else preserved to pass on

R1 0 to claim service, else preserved to pass on

Notes

Module tasks which wish to start should claim this service, passing their start command in R0. The Filer will re-issue this service until no more modules claim it.

To prevent infinite loops if an error occurs in starting a task, modules should:

- Set their taskhandle to 0 on; Initialise (see Module Initialisation Code?) Service_PostReset Service_StartedFiler
- Only claim this service if their taskhandle is zero (i.e. they are not already running, and not attempting to re-start)
- Set their taskhandle to a flag value (-1) when they claim this service
- Do either; Set their taskhandle based on the return result of Wimp_Initialise if their command is called, or set their taskhandle back to 0 on Service_StartedFiler (failed to initialise)
- Set their taskhandle back to 0 when they quit using Wimp_CloseDown

Filer tasks should also check for Service_FilerDying, and close down if they receive it.

[RISC OS Open source page](#)

Service_StartUpFS

Service Call &12 · FileSwitch

This service call is called when the system wants to start up a filing system.

Before calling

R1 18 (&12)

R2 Filing system number

On return

R1 Preserved (never claim)

R2 Preserved

Notes

This Service Call should never claimed.

This is the old way to start up a filing system, the preferred method is to use OS_FSControl 14.

[RISC OS Open source page](#)

Service_StartWimp

Service Call &49 · WindowManager

This service is broadcast by the Wimp to start up resident module tasks by passing the returned *command to Wimp_StartTask.

Before calling

R1 73 (&49)

On return

R0 Pointer to *command to start module

R1 0 to claim call

Notes

Module tasks which wish to start should claim this service, passing their start *command in R0. The Wimp will re-broadcast this service until no more modules claim it.

Filer module tasks should wait for Service_StartFiler instead.

To prevent infinite loops if an error occurs in starting a task, modules should:

- set their taskhandle to 0 when initialising, Service_PostReset, or Service_StartedWimp
- only claim this service if their taskhandle is zero. i.e. they are not already running, and not attempting to restart
- set their taskhandle to a flag value (-1) when they claim this service
- set their taskhandle based on the return result of Wimp_Initialise if their command is called, or set their taskhandle back to 0 on Service_StartedWimp
- set their taskhandle back to 0 when they quit using Wimp_CloseDown

[RISC OS Open source page](#)

Service_SwitchingOutputToSprite

Service &72

This service is issued immediately after output has been switched to, or from a sprite, or mask by OS_SpriteOp 60 or OS_SpriteOp 61.

Before calling

R1 114 (&72)

R2 R0 value as passed to OS_SpriteOp

R3 R1 value as passed to OS_SpriteOp

R4 R2 value as passed to OS_SpriteOp

R5 R3 value as passed to OS_SpriteOp

On return

R1 Preserved

R2 Preserved

R3 Preserved

R4 Preserved

R5 Preserved

[RISC OS Open source page](#)

Service_SyntaxError

Service &8C

This service is issued just before a syntax error is returned from a module *Command.

Before calling

R1 140 (&8C)

R2 Pointer to entry in command table?

R3 Pointer to base of module

R4 Pointer to command string in module

On return

R0 Pointer to error block, else preserved to pass on

R1 0 to claim, else preserved

R2 Preserved

R3 Preserved

R4 Preserved

Notes

This service is issued before a syntax error is returned from a *Command resident in a module, which allows a module to localise messages for the current territory.

This might be the way in which RISC OS 3.60 translates the MessageTrans tokens to form a valid message.

[RISC OS Open source page](#)

Service_TaskManagerAcknowledgements

Service &42680

This service allows modules to provide licence information, copyright or other messages. It is used by the TaskManager in RO4/6 to compile the text displayed when the Information button is pressed in the OS version window.

Before calling

R1 &42680

R2 0 to inform of a change

otherwise ptr to Code to call with text information:

Entry

R0 = Priority (usually &1000)

R1 = Module title,0

R2 = Licence text or other message,0

Exit

all preserved

On return

R1 Preserve (if claimed, nothing will be displayed)

R2 Preserve

In practice

Also known as Service_GetCredits.

Note that this API does not define a Workspace pointer for the Code, so the caller will probably require a veneer, sadly.

The text provided to the Code will not necessarily persist afterwards, so must be copied by the Code.

As the caller may cache the results, modules that respond to this service must issue it with R2=0 on initialisation/finalisation to tell interested parties to discard their cached text.

The priority is used to sort the messages before display, low to high. Messages of the same priority are sorted in reverse chronological order. Example priorities:

0 / OS modules

&1000 – OS modules: SpriteExtend

&2000 – OS modules: AppleTalk

&5000 – OS modules: CDFSSoftATAPI

1<<20 – OS modules: nemo's ModCredits module

(The ModCredits module adds all the Module copyright strings to the collected information)

[RISC OS Open source page](#)

Service_TerritoryManagerLoaded

Service &64

This service is issued when the territory manager starts.

Before calling

R1 100 (&64)

On return

R1 Preserved

Notes

This service call is issued to indicate that territory modules should register themselves with Territory_Register.

[RISC OS Open source page](#)

Service_TerritoryStarted

Service &75

This service is issued when the machine's territory is changed.

Before calling

R1 117 (&75)

On return

R1 Preserved

Notes

This service must not be claimed.

ROM modules will use this service to re-open their messages files, where as RAM modules can ignore this service.

[RISC OS Open source page](#)

Service_TimeZoneChanged

Service &81080

This service is sent when the current timezone is changed, by any means. This could be due to the clock being synchronised, a change in territory, timezone, or DST settings, or automatic daylight savings changeover.

Before calling

R1 &81080

R2 b0 = set if DST is active

b1 = set if DST is managed automatically

R3 new offset in signed centiseconds

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_UKByte

Service &07

This service is issued when an unknown OS_Byte code is given.

Before calling

R0 Pointer to error block

R1 7 (&07)

R2 OS_Byte number

R3 R1 passed to OS_Byte command

R4 R2 passed to OS_Byte command

On return

All registers preserved to pass on, else

R1 0 (claim service)

R3 R1 to return to caller

R4 R2 to return to caller

Notes

ByteV vector should be used in preference to trapping this service.

[RISC OS Open source page](#)

Service_UKCommand

Service &04

This service is issued when a command is unknown.

Before calling

R0 Pointer to command

R1 4 (&04)

On return

R0 Preserved to pass on, else

0 if command successful, else pointer to error

R1 Preserved to pass on, else

0 (to claim service)

Notes

This service is issued when a command is unknown, before the filing system is called to attempt to 'run' the command (See OS_FSControl 4).

NetFS uses this service to implement file server commands. Use the command table? in a Relocatable Module if possible to implement new *Commands.

[RISC OS Open source page](#)

Service_UKCompression

Service &B7

This service is used to provide patches for RISC OS 3.70 where required. It is called when an Absolute file (filetype is &FF8) is run.

Before calling

R0 0 = First pass - unsqueeze

1 = Second pass - patch

All others reserved

R1 183 (&B7)

R2 Load address

R3 Size

R4 Execute address

R5 Filename (not canonicalised)

On return

R0 Preserved

R1 0 if claimed (all changes have been performed), else preserved

R2 Preserved

R3 New size

R4 New execute address

R5 Preserved

Notes

Only if the file does not contain an AIF header, will it be called with R0 with a value of 0. After patching OS_SynchroniseCodeAreas will be called.

[RISC OS Open source page](#)

Service_UKConfig

Service &28

This service is issued when a *Configure command is issued with an unknown parameter.

Before calling

R0 Pointer to command tail, or 0 for syntax line

R1 40 (&28)

On return

R0 Error code

R1 0 to claim service (option recognised), else preserved

Notes

You should use the module command table? to trap specific configure options in preference to this service.

[RISC OS Open source page](#)

Service_UKStatus

Service &29

This service is issued when a *Status command is issued with an unknown parameter.

Before calling

R0 Pointer to command tail, or 0 for syntax line

R1 41 (&29)

On return

R0 Preserved

R1 0 to claim service (option recognised), else preserved

Notes

You should use the module command table? to trap specific status options in preference to this service.

[RISC OS Open source page](#)

Service_UKWord

Service &08

This service is issued when an unknown OS_Word code is given.

Before calling

R1 8 (&08)

R2 OS_Word number

R3 Pointer to OS_Word data block

On return

All registers preserved to pass on, else

R1 0 (claim service)

Notes

WordV vector should be used in preference to trapping this service.

[RISC OS Open source page](#)

Service_URLModule

AcornSSL

This service call is issued by the AcornSSL module to communicate important events.

Before calling

R0 0 (Service_URLModule_SSL_Started)

R1 &83E02 (Service_URLModule_SSL)

R2 version number * 100

R0 1 (Service_URLModule_SSL_Dying)

R1 &83E02 (Service_URLModule_SSL)

R2 version number * 100

On return

All registers must be preserved. Do not claim this service call.

All registers must be preserved. Do not claim this service call.

[RISC OS Open source page](#)

Service_URLProtocolModule

URL_Fetcher

This service communicates vital information between the URL_Fetcher and related protocol modules.

Before calling

R0 Reason code - reason for the service call (see below)

R1 &83E00 (Service_URLProtocolModule)

All other registers are reason code dependent

On return

All registers must be preserved, unless claiming the service call. In all the currently defined cases, the service call must not be claimed. Protocol modules must ignore reason codes that they do not understand.

(Service &83E00)

This service call is issued by the URL protocol module to communicate important events to the protocol modules.

Reason Codes

/ Hex # / Action

0 – Hex #: &00 · Action: URL module has initialised

0 – Hex #: &01 · Action: URL module is dying

All other reason codes are reserved and must not be used.

[RISC OS Open source page](#)

Service_URLProtocolModule 0

URL_Fetcher

Upon receiving this service call, protocol modules should re-register with the new URL module by issuing SWI URL_ProtocolRegister as usual. It must assume that any previous registration is no longer valid.

Before calling

R0 0 · URL module has initialised

R1 &83E00 · Service_URLProtocolModule

R2 version · Version number of URL module * 100

On return

All registers must be preserved. The service call must not be claimed. Protocol modules must ignore reason codes that they do not understand.

In practice

This service call must not be claimed.

[RISC OS Open source page](#)

Service_URLProtocolModule 1

URL_Fetcher

Upon receiving this service call, protocol modules should note that the URL module has gone away and not attempt to talk to it any more until a future Service_URLProtocolModule/URLModuleStarted service call arrives.

Before calling

R0 1 · URL module is dying

R1 &83E00 · Service_URLProtocolModule

R2 version · Version number of URL module * 100

On return

All registers must be preserved. The service call must not be claimed. Protocol modules must ignore reason codes that they do not understand.

In practice

This service call must not be claimed.

[RISC OS Open source page](#)

Service_USB

Service &D2

This service provides a variety of USB related notifications.

Before calling

R0 Reason code

R1 210 (&D2)

Reason Codes

/ Hex # / Action

0 – Hex #: &00 · Action: Attach

1 – Hex #: &01 · Action: Connected

2 – Hex #: &02 · Action: Reserved

3 – Hex #: &03 · Action: Starting

4 – Hex #: &04 · Action: Dying

Notes

The reason code passed by R0 on entry determines the operation of the service call.

[RISC OS Open source page](#)

Service_USB 0

Service &D2 · Part 16 – Service Calls

This service is issued when a new USB device has been attached to the USB system for any drivers which may be interested in it.

Before calling

R0 0

R1 210 (&D2)

R2 Pointer to USB Service Call block

On return

R0 Preserved

R1 Preserved

R2 Preserved

Notes

This service call must not be claimed.

[RISC OS Open source page](#)

Service_USB 1

Service &D2 · Part 16 – Service Calls

This service is issued with OS_ServiceCall by any application or module which wishes to get information on all of the currently connected devices without needing to keep track of Attach (see Service_USB 0) and DeviceDead (See Service_DeviceDead).

Before calling

R0 1

R1 210 (&D2)

R2 0

On return

R0 Preserved

R1 Preserved

R2 Pointer to linked list of USB Service Call blocks

Notes

It is the responsibility of the issuer of the call to free the memory used by the returned descriptors in the RMA.

Each item in the linked list is created using OS_Module 6 and should be freed with OS_Module 7.

Do not claim this call, other USB drivers in the module chain may wish to add further blocks to the list.

[RISC OS Open source page](#)

Service_USB 3

Service &D2 · Part 16 – Service Calls

This service is issued notify the (re)starting of the USB system and for any clients of USBDriver to (re)register themselves.

Before calling

R0 3

R1 210 (&D2)

On return

R0 Preserved

R1 Preserved

Notes

You must not claim this service call.

[RISC OS Open source page](#)

Service_USB 4

Service &D2 · Part 16 – Service Calls

This service is issued to inform that the USB system is shutting down.

Before calling

R0 4

R1 210 (&D2)

On return

R0 Preserved

R1 Preserved

Notes

You must not claim this service call.

[RISC OS Open source page](#)

Service_ValidateAddress

Service &6D

This service is issued when OS_ValidateAddress encounters an unrecognised area.

Before calling

R1 109 (&6D)

R2 R0 as passed to OS_ValidateAddress

R3 R1 as passed to OS_ValidateAddress

On return

R1 0 to indicate area is valid, else &6D to pass on

R2 Preserved ?

R3 Preserved ?

Notes

It is not called for areas which should be mapped.

[RISC OS Open source page](#)

Service_WimpCloseDown

Service Call &53 · WindowManager

This is used to notify when the Window Manager is about to close a task.

Before calling

R0 0 if Wimp_CloseDown

>0 if starting new task

R1 83 (&53)

R2 Taskhandle

On return

R0 Preserved, or set to Error Block to stop start-up

Notes

This service call is broadcast when a task exits using Wimp_CloseDown, or when Wimp_Initialise is called from within a task that is already initialised.

If a task exits without calling Wimp_CloseDown or if an error occurs and the task does not trap it, then the Wimp will call this service so that it still has a chance to shut any jobs it may have active.

If the 'starting task' form of this service is called then the usual response is to return an error in R0 similar to 'Wimp is currently active'.

[RISC OS Open source page](#)

Service_WimpPalette

Service Call &5D · WindowManager

This service call is broadcast by the Window Manager when Wimp_SetPalette is called to set the Wimp's palette.

Before calling

R1 93 (&5D)

On return

- All registers preserved. Do not claim

Notes

It can be used to tell when the palette has been changed.

[RISC OS Open source page](#)

Service_WimpRegisterFilters

Service Call &86 · WindowManager

This service call allows the Filter Manager to install filters with the Window Manager.

Before calling

R1 134 (&86)

On return

--

Notes

It is used when the Wimp resets the Filter table back to its default state.

Filters should be registered with Wimp_RegisterFilter.

More information on filters is available in the FilterManager section.

[RISC OS Open source page](#)

Service_WimpReportError

Service Call &57 · WindowManager

This service call is used to temporarily suspend the trapping of VDU output so that an error window can be displayed. However, if R0 = 1, then it will re-activate the trapping of VDU output as before.

Before calling

R0 0 = Error window opening

1 = Error window closing

R1 87 (&57)

On return

- All registers preserved

[RISC OS Open source page](#)

Service_WimpSaveDesktop

Service Call &5C · WindowManager

The task manager broadcasts this call before Message_SaveDesktop. It is used to allow modules to save state to a desktop boot file.

Before calling

R0 Flag word as per Message_SaveDesktop

R1 Reason Code &5C

R2 File handle of file to be written to

On return

R0 Pointer to Error Block, or preserved if not claimed

R1 0 to claim, or preserved

- All other registers preserved

In practice

When a module receives this service code, it should write out any * Commands to the specified file handle.

[RISC OS Open source page](#)

Service_WimpSpritesMoved

Service Call &85 · WindowManager

This service call is broadcast if the Wimp sprite pools have to move.

Before calling

R1 133 (&85)

R2 Pointer to ROM area

R3 Pointer to RAM area

On return

- All registers preserved. Do not claim

Notes

The service call must not be claimed.

[RISC OS Open source page](#)

*Commands

Command-line and desktop commands, with syntax, parameters, and relevant notes.

*Access

The purpose of this *Command is to change the attributes of all objects matching the wildcarded specification.

Syntax

*Access object_spec [attributes]

Parameters

object_spec / Valid (wildcarded) pathname specifying file or directory

attributes – Valid (wildcarded) pathname specifying file or directory: Following attributes are allowed:

L – Valid (wildcarded) pathname specifying file or directory: Lock object against deletion by any user

W – Valid (wildcarded) pathname specifying file or directory: Write permission for you

R – Valid (wildcarded) pathname specifying file or directory: Read permission for you

/ – Valid (wildcarded) pathname specifying file or directory: Separator between your permissions and the public's

W – Valid (wildcarded) pathname specifying file or directory: Write permission for the public (on NetFS)

R – Valid (wildcarded) pathname specifying file or directory: Read permission for the public (on NetFS)

Notes

These attributes control whether you can run, read from, write to and delete a file.

[RISC OS Open source page](#)

*AddApp

BootCommands

This *Command stores objects whose names match the argument pattern into Resources:\$.Apps.

Syntax

*AddApp [directory.]pattern

Parameters

directory / Valid pathname specifying a directory

pattern – Valid pathname specifying a directory: Wildcarded pattern to match

Notes

'Resources:\$.Apps' is the location that stores entries for the Apps icon on the icon bar.

[RISC OS Open source page](#)

*AddFS

This *Command adds the name of a remote file server's disc to the list of those already known by NetFS.

Syntax

```
*AddFS file_server_number [disc_number [:]disc_name]
```

Parameters

file_server_number / File server number to add to list

disc_number – File server number to add to list: Disc number of the disc to add

disc_name – File server number to add to list: Disc name to add

Notes

If only the file server is specified, then all its discs will be removed from the list.

For the file_server_number, both a net number and a station number must be supplied.

[RISC OS Open source page](#)

*AddMap

This *Command adds an entry to the Netl address map from the IPv4 address space to Econet address space.

Syntax

```
*AddMap <IP_address> <net>
```

Parameters

IP_address / Fully qualified IPv4 address

net – Fully qualified IPv4 address: Econet net number

Notes

The map is held in Choices:Internet.AUNMap and can be added to so that an AUN mapping is present after reset by editing it.

[RISC OS Open source page](#)

*AddTinyDir

Pinboard

This *Command adds the icon of a file, application or directory to the icon bar.

Syntax

```
*AddTinyDir [object]
```

Parameters

object · Valid pathname specifying a file, application or directory

Notes

If no pathname is given, and there is no TinyDir icon already present on the icon bar, a blank directory icon is added to the icon bar.

[RISC OS Open source page](#)

*AddToRMA

BootCommands

This *Command attempts to increase the size of the RMA by the given amount.

Syntax

*AddToRMA amount[K|M]

Parameters

amount · Amount of free memory to add (in bytes, kilobytes, or megabytes)

Notes

The memory will be taken from the free pool.

[RISC OS Open source page](#)

*ADFS

This *Command selects the ADFS (Advanced Disc Filing System) as the current filing system.

Syntax

*ADFS

Parameters

None.

[RISC OS Open source page](#)

*Alias

This *Command sets an alias for a command.

Syntax

*Alias alias command

Parameters

alias / Alias variable

command – Alias variable: Specify a command

Notes

None.

[RISC OS Open source page](#)

*Alphabet

This *Command selects an alphabet, according to a country name or alphabet name.

Syntax

*Alphabet [country_name_] [alphabet_name]

Parameters

country_name · Valid country name. A list of countries can be obtained with *Countries

alphabet_name · Valid alphabet name. A list of alphabets can be obtained with *Alphabets. A list is available [here](#)

Notes

With no parameter, this command displays the currently selected alphabet.

[RISC OS Open source page](#)

*Alphabets

This *Command lists all the alphabets currently supported.

Syntax

*Alphabets

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*Append

This *Command opens an existing file for adding more data to the end of it.

Syntax

*Append filename

Parameters

filename · Valid pathname specifying an existing file

Notes

Each line of input is transformed using OS_GSTrans before it is appended to the file.

Pressing Escape finishes the input.

[RISC OS Open source page](#)

*AppPath

BootCommands

This *Command appends a path element to a path variable, removing any earlier occurrences of the path if present.

Syntax

*AppPath variable element

Parameters

variable / A system variable to append to

element – A system variable to append to: The path to append

[RISC OS Open source page](#)

*AppSize

BootCommands

This *Command moves memory into or out of the RMA.

Syntax

*AppSize size[K|M]

Parameters

size[K|M] · Desired number of (kilo|mega)bytes of memory to leave in application space

Notes

Prior to RISC OS 3.5, this command was used to move free memory out of the RMA and into application space.

On RISC OS 3.5 and above the addition of the kernel-managed free pool dynamic area means that adjusting the size of the RMA will typically result in memory being added to or removed from the free pool, rather than application space. Therefore although this command can be used to shrink the RMA to its minimum size (by specifying a suitably large size parameter), it cannot be used to control the size of application space.

To solve this problem, a new command, *AppSlot was introduced.

[RISC OS Open source page](#)

*AppSlot

BootCommands

This *Command attempts to resize application space to the indicated size.

Syntax

*AppSlot size[K|M]

Parameters

size[K|M] · Desired number of (kilo|mega)bytes of memory to leave in application space

Notes

On OS versions older than RISC OS 3.5, this command behaves in exactly the same way as *AppSize, by adjusting the size of the RMA.

On RISC OS 3.5 and above this command will take memory from (or add memory to) the free pool.

This command should be used in preference to *AppSize.

[RISC OS Open source page](#)

*ARP

This *Command displays and can modify the Internet-to-Physical-address translation tables located in the Internet module which are used by the address resolution protocol, ARP.

Syntax

*ARP <host_> *ARP -a *ARP -d <host_> *ARP -s <host_> <_phys_addr> [temp] [pub] *ARP -f <filename>

Parameters

host · Internet host specified either by name (which must be present in the host name data base InetDBase\$Path hosts) or by address (using the standard Internet dot notation)

-a · Display all the ARP entries currently in its table

-d · Delete an entry for host

-s · Create an ARP entry for the host called host with the physical address phys_addr. The entry will be permanent unless the word temp is given in the command. The entry will be 'published' if the word pub is given. This system will then act as an ARP or Reverse ARP server, responding to requests for host's physical address even though the host address is not its own

phys_addr · Physical address of host given in Ethernet format (ie six hexadecimal bytes separated by colons)

-f · Cause the file filename to be read and multiple entries to be set in the ARP tables

filename · Valid pathname specifying a file containing multiple entries to be set in the ARP table

Notes

Without flags, the command displays the current ARP entry for the host.

If you don't know the physical address of an interface, you can use the *InetInfo command to find it.

[RISC OS Open source page](#)

*Audio

This *Command can be used to turn the sound system on or off, or to select the default audio controller device.

Syntax

*Audio on|off|deviceid

Parameters

on or off / Turn sound system on or off

deviceid – Turn sound system on or off: Select default controller device by specifying ID string

Notes

When the sound system is switched off all Channel Handlers and Scheduler activity is stopped. Software interrupts are still allowed, even though no sound will be produced.

[RISC OS Open source page](#)

*Back

This *Command swaps the current and previously selected directories in the current filing system.

Syntax

*Back

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*BackDrop

Pinboard

This *Command displays a sprite on the desktop background.

Syntax

*BackDrop [-centre|-scale|-tile|-remove] [filename]

Parameters

-centre / Align sprite to centre of desktop

-scale – Align sprite to centre of desktop: Scale sprite to cover the complete desktop background

-tile – Align sprite to centre of desktop: Tile sprite across desktop

-remove – Align sprite to centre of desktop: Remove the current background

filename – Align sprite to centre of desktop: Valid pathname, specifying a sprite file

Notes

If no filename is specified, the current backdrop's placing is altered.

If the sprite file contains multiple sprites, then the first sprite will be used.

[RISC OS Open source page](#)

*Backup

This *Command copies the used part of one floppy disc to another.

Syntax

*Backup source_drive dest_drive [Q]

Parameters

source_drive / Number of source drive (0 – 3)

dest_drive – Number of source drive (0 – 3): Number of destination drive (0 – 3)

Q – Number of source drive (0 – 3): Speed up copy process

Notes

The Q parameter speeds up the copy process by using the application work area as a buffer (if required), so fewer disc accesses are required.

Free space is not copied.

This command is only applicable to floppy disc drives.

[RISC OS Open source page](#)

*BASIC

BASIC

This *Command starts the ARM BBC BASIC V interpreter.

Syntax

*BASIC -help *BASIC [-chain] filename *BASIC -quit filename *BASIC -load filename *BASIC @start,end *BASIC -chain @start,end *BASIC -load @start,end *BASIC -quit @start,end

Parameters

-help / Return information on the BASIC module, including options

-chain – Return information on the BASIC module, including options: Load and execute program and then return to immediate mode when programs stops

-quit – Return information on the BASIC module, including options: Load and execute program and then exit BASIC interpreter

-load – Return information on the BASIC module, including options: Load filename only

@start,end – Return information on the BASIC module, including options: Load program that is already in memory at start address. The end address is the byte following the end of the program

Notes

If no parameters are specified, the interpreter is simply loaded awaiting for input by the user.

@start,end addresses must be in a fixed format of @xxxxxxx,xxxxxxx with no spaces and be in uppercase.

[RISC OS Open source page](#)

*BASIC64

BASIC

This *Command starts the ARM BBC BASIC VI interpreter.

Syntax

*BASIC64 -help *BASIC64 [-chain] filename *BASIC64 -quit filename *BASIC64 -load filename *BASIC64 @start,end

*BASIC64 -chain @start,end *BASIC64 -quit @start,end

Parameters

-help / Return information on the BASIC module, including options

-chain – Return information on the BASIC module, including options: Load and execute program and then return to immediate mode when programs stops

-quit – Return information on the BASIC module, including options: Load and execute program and then exit BASIC interpreter

-load – Return information on the BASIC module, including options: Load filename only

@start,end – Return information on the BASIC module, including options: Load program that is already in memory at start address. The end address is the byte following the end of the program

Notes

BASIC will select the most appropriate version of BASIC VI for the machine, if a Vector Floating Point unit is available BASICVFP will be used in preference to the FPA variant.

If no parameters are specified, the interpreter is simply loaded awaiting for input by the user.

@start,end addresses must be in a fixed format of @xxxxxxx,xxxxxxx with no spaces and be in uppercase.

[RISC OS Open source page](#)

*BASICFPA

BASIC

This *Command starts the ARM BBC BASIC VI interpreter.

Syntax

*BASICFPA -help *BASICFPA [-chain] filename *BASICFPA -quit filename *BASICFPA -load filename *BASICFPA @start,end

*BASICFPA -chain @start,end *BASICFPA -quit @start,end

Parameters

-help / Return information on the BASIC module, including options

-chain – Return information on the BASIC module, including options: Load and execute program and then return to immediate mode when programs stops

-quit – Return information on the BASIC module, including options: Load and execute program and then exit BASIC interpreter

-load – Return information on the BASIC module, including options: Load filename only

@start,end – Return information on the BASIC module, including options: Load program that is already in memory at start address. The end address is the byte following the end of the program

Notes

Specifically the FPA version of BASIC VI will be started, which may be not be the most appropriate version of BASIC VI for the machine, use *BASIC64 in preference.

[RISC OS Open source page](#)

*BASICVFP

BASIC

This *Command starts the ARM BBC BASIC VI interpreter.

Syntax

*BASICVFP -help *BASICVFP [-chain] filename *BASICVFP -quit filename *BASICVFP -load filename *BASICVFP @start,end
*BASICVFP -chain @start,end *BASICVFP -quit @start,end

Parameters

-help / Return information on the BASIC module, including options

-chain – Return information on the BASIC module, including options: Load and execute program and then return to immediate mode when programs stops

-quit – Return information on the BASIC module, including options: Load and execute program and then exit BASIC interpreter

-load – Return information on the BASIC module, including options: Load filename only

@start,end – Return information on the BASIC module, including options: Load program that is already in memory at start address. The end address is the byte following the end of the program

Notes

Specifically the VFP version of BASIC VI will be started, which may be not be the most appropriate version of BASIC VI for the machine, use *BASIC64 in preference.

[RISC OS Open source page](#)

*BlankTime

This *Command sets the time in seconds before the screen blanks.

Syntax

*BlankTime [W|O] [time]

Parameters

W / End screen blanking if writing to screen

O – End screen blanking if writing to screen: Do not end screen blanking if writing to screen

time – End screen blanking if writing to screen: Duration of inactivity before screen blanks (seconds)

Notes

If, during the specified BlankTime there is no activity (i.e. no keyboard or mouse input is received, and – with the W option – there is no writing to the screen) then the screen then blanks. If no option is specified, then O is assumed.

The blank time is only retained until the next reset.

[RISC OS Open source page](#)

*BreakClr

Debugger

This *Command removes the breakpoint at the specified address or register value.

Syntax

*BreakClr [addr_]_reg]

Parameters

addr / Address of breakpoint to clear (hexadecimal address)

reg – Address of breakpoint to clear (hexadecimal address): Register containing address of breakpoint to clear

Notes

If no parameters are given the command removes all breakpoints.

[RISC OS Open source page](#)

*BreakList

Debugger

This *Command lists all currently set breakpoints.

Syntax

*BreakList

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*BreakSet

Debugger

This *Command sets a breakpoint at the address specified by literal number or register value.

Syntax

*BreakSet addr_]_reg

Parameters

addr / Address of breakpoint to set (hexadecimal address)

reg – Address of breakpoint to set (hexadecimal address): Register containing address of breakpoint to set

Notes

Without parameter the command removes all breakpoints.

[RISC OS Open source page](#)

*Build

This *Command opens a new file and directs subsequent data into it.

Syntax

*Build filename

Parameters

filename · Valid pathname specifying a file

Notes

Each line of input is transformed using OS_GSTrans before it is appended to the file.

Pressing Escape finishes the input.

If the file already exists, it is over-written with the subsequent input data.

Lines are terminated using ASCII code 13 (&0D).

[RISC OS Open source page](#)

*Bye

This *Command ends a filing session by closing all files and unsetting all directories and Libraries.

Syntax

*Bye [[:]file_server] (NetFS syntax)

Parameters

file_server · File server name

Notes

Other actions are filing system dependent, and where relevant may include dismounting discs, parking the heads of hard discs, and logging off the current fileserver.

[RISC OS Open source page](#)

*Cache

This *Command turns the cache on or off.

Syntax

*Cache [on|off]

Parameters

on or off · Turn cache on or off

Notes

With no parameter, this command gives the cache's current state.

[RISC OS Open source page](#)

*Canonical

BootCommands

This *Command reads the given system variable, canonicalises the contents, then writes that back into the system variable.

Syntax

*Canonical variable

Parameters

variable · System variable whose contents are to be canonicalised

[RISC OS Open source page](#)

*Cat

This *Command lists all the objects in the directory given by the argument.

Syntax

*Cat directory

Parameters

directory · Valid pathname specifying a directory

Notes

If no directory is specified, then the current directory is listed.

This command can be abbreviated to `'*.'`, provided the *Set the system variable Alias\$. to a different value from its default.

[RISC OS Open source page](#)

*CDDevices

This *Command displays information about any CD devices that are connected.

Syntax

*CDDevices

Parameters

None.

Notes

The information returned includes drive, product name, capacity, firmware revision, and their SCSI ID (displayed as device, LUN, and card; or as zeroes for non-SCSI devices).

[RISC OS Open source page](#)

*CDFS

This *Command selects the CD-ROM Filing System as the current filing system.

Syntax

*CDFS

Parameters

none.

Notes

This command is only available if you have a CD-ROM drive fitted.

[RISC OS Open source page](#)

*CDir

This *Command creates a directory with the specified pathname.

Syntax

*CDir directory [size_in_entries]

Parameters

directory / Valid pathname specifying a directory

size_in_entries – Valid pathname specifying a directory: Number of entries the directory should hold before it needs to be expanded.

Notes

On NetFS, you can also give the size of the directory using the size_in_entries parameter

[RISC OS Open source page](#)

*CDSpeed

This *Command sets or returns the speed of the CD drive, or of the current drive if none is specified.

Syntax

*CDSpeed [drive [speed]]

Parameters

drive / Valid CD drive number

speed – Valid CD drive number: New read speed for that drive

1 = standard speed

2 = double speed

255 = maximum speed

[RISC OS Open source page](#)

*ChangeDynamicArea

This *Command changes the size of the font cache, system sprite area and/or RAM disc.

Syntax

*ChangeDynamicArea [-FontSize n[K]] [-SpriteSize n[K]] [-RamFSSize n[K]]

Parameters

n · Size of area (a suffix of K states kilobytes, will assume bytes in the absence of any suffix)

Notes

The main use of this command is within desktop boot files to set up memory allocations.

An error is generated if memory cannot be allocated.

[RISC OS Open source page](#)

*ChannelVoice

This *Command assigns a voice to a channel.

Syntax

*ChannelVoice channel voice_number[_voice_name]

Parameters

channel / Channel number (1 – 8)

voice_number – Channel number (1 – 8): Voice number (1 – 16; or 0 to mute)

voice_name – Channel number (1 – 8): Voice name as returned by *Voices

Notes

By default, only the first of the eight voices will be available. To make others available, use the BASIC VOICES n command. Do not confuse the BASIC VOICE keyword with the *Voices *Command.

[RISC OS Open source page](#)

*CheckMap

This *Command checks that the map of an E- or F-format disc has the correct checksums and is consistent with the directory tree.

Syntax

*CheckMap [disc_spec]

Parameters

disc_spec · Name or number of disc drive

Notes

If only one copy of the map is good, this command lets you rewrite the bad one with the information in the good one.

If bad map information cannot be cured, you should backup all of your files and then reformat your hard disc. Contact your supplier for more information.

[RISC OS Open source page](#)

*Close

This *Command closes all open files on the current filing system.

Syntax

*Close

Parameters

None.

Notes

It is possible to close all open files on a specific filing system by preceding the command with the filing system name. e.g. *adfs: Close.

This command must not be used within the desktop environment, as it may close files required by other applications.

[RISC OS Open source page](#)

*ColourTrans

This *Command is for internal use only. Do not use it.

Syntax

*ColourTrans...

Parameters

None.

Notes

This command is for internal use only. Do not use it.

[RISC OS Open source page](#)

*ColourTransLoadings

ColourTrans

The purpose of this *Command is to set the red, green and blue weightings used when trying to match colours.

Syntax

*ColourTransLoadings redweight greenweight blueweight

Parameters

redweight / Amount of weight to apply to red colour

greenweight – Amount of weight to apply to red colour: Amount of weight to apply to green colour

blueweight – Amount of weight to apply to red colour: Amount of weight to apply to blue colour

Notes

This command is for internal use and should not be used.

[RISC OS Open source page](#)

*ColourTransMap

ColourTrans

The purpose of this *Command is to set up the calibration table from its parameters. The number of parameters passed for each component must have been already specified using *ColourTransMapSize command.

Syntax

*ColourTransMap RRGGBBDD RRGGBBDD RRGGBBDD etc...

Parameters

RRGGBBDD · Number to be placed in calibration table (8 digit hexadecimal number)

Notes

This command is for internal use and should not be used.

[RISC OS Open source page](#)

*ColourTransMapSize

ColourTrans

The purpose of this *Command is to configure the parameters that will be passed via the next *ColourTransMap command.

Syntax

*ColourTransMapSize n1 n2 n3

Parameters

n1 · Number of parameters for component 1 to be passed in *ColourTransmap

n2 · Number of parameters for component 2 to be passed in *ColourTransmap

n3 · Number of parameters for component 3 to be passed in *ColourTransmap

Notes

This command is for internal use and should not be used.

[RISC OS Open source page](#)

*Compact

This *Command collects together free space on a disc by moving files.

Syntax

*Compact [disc_spec]

Parameters

disc_spec · Name or number of disc drive

Notes

If no parameter is passed, then the compaction operation is performed on the current disc.

This command works on either floppy or hard discs.

[RISC OS Open source page](#)

*Configure

This *Command configures a particular setting. There are many different settings, each with different parameters. The full list is available below.

Available Configure *Commands

- *Configure ADFSBuffers
- *Configure ADFSDirCache
- *Configure AutoDST
- *Configure Baud

- *Configure Boot
- *Configure BootNet
- *Configure BST
- *Configure Cache
- *Configure Caps
- *Configure CDROMBuffers
- *Configure CDROMDrives
- *Configure Country
- *Configure Data
- *Configure Delay
- *Configure Dir
- *Configure Drive
- *Configure DST
- *Configure DumpFormat
- *Configure FileSystem
- *Configure Floppies
- *Configure FontMax
- *Configure FontMax1
- *Configure FontMax2
- *Configure FontMax3
- *Configure FontMax4
- *Configure FontMax5
- *Configure FontSize
- *Configure FreewayAutoAddress
- *Configure FS
- *Configure GMT
- *Configure HardDiscs
- *Configure IDEDiscs
- *Configure Ignore
- *Configure Language
- *Configure Lib
- *Configure LMNameServer
- *Configure LMTransport
- *Configure Loud
- *Configure Mode
- *Configure MonitorType
- *Configure MouseStep
- *Configure MouseType
- *Configure NoBoot
- *Configure NoCaps
- *Configure NoDir
- *Configure NoDST

- *Configure NoNum
- *Configure NoScroll
- *Configure Num
- *Configure NVidia
- *Configure Print
- *Configure PrinterBufferSize
- *Configure PS
- *Configure Quiet
- *Configure RamFSSize
- *Configure Repeat
- *Configure RMASize
- *Configure ScreenSize
- *Configure Scroll
- *Configure SCSIFSDrive
- *Configure ShareBoot
- *Configure ShCaps
- *Configure SoundDefault
- *Configure SoundSystem
- *Configure SpriteSize
- *Configure Step
- *Configure Sync
- *Configure SystemSize
- *Configure Territory
- *Configure TimeZone
- *Configure Truncate
- *Configure TV
- *Configure WimpAutoFrontIconBar?
- *Configure WimpAutoFrontDelay?
- *Configure WimpAutoMenuDelay
- *Configure WimpAutoScrollDelay?
- *Configure WimpButtonType?
- *Configure WimpClickSubmenu?
- *Configure WimpDoubleClickDelay
- *Configure WimpDoubleClickMove
- *Configure WimpDragDelay
- *Configure WimpDragMove
- *Configure WimpFlags
- *Configure WimpFont
- *Configure WimpIconBarSpeed?
- *Configure WimpIconBarAcceleration?
- *Configure WimpIconBarLayout?
- *Configure WimpIconiseButton?

- *Configure WimpMenuDragDelay
- *Configure WimpMode
- *Configure WimpSpritePrecedence?

[RISC OS Open source page](#)

*Configure ADFSBuffers

This *Command sets the configured number of 1Kb file buffers reserved for ADFS to speed up operations on open files.

Syntax

*Configure ADFSBuffers n

Parameters

n · Number of buffers

Notes

A value of 1 sets a default value appropriate to the computer's RAM size; a value of 0 disables fast buffering on open files.

[RISC OS Open source page](#)

*Configure ADFSDirCache

This *Command sets the configured amount of memory reserved for the directory cache.

Syntax

*Configure ADFSDirCache size[K]

Parameters

size · Number of kilobytes reserved

Notes

A value of 0 sets a default value appropriate to the computer's RAM size.

[RISC OS Open source page](#)

*Configure AutoDST

This *Command sets the configured value for daylight saving time to be automatically adjusted by the computer.

Syntax

*Configure AutoDST

Parameters

None.

Notes

The territory contains a table of rules to define when the change to and from daylight saving time occurs. To manually apply a change, use *Configure DST or *Configure NoDST instead.

[RISC OS Open source page](#)

*Configure Baud

This *Command sets the configured baud rate for the serial port.

Syntax

*Configure Baud n

Parameters

n · Baud rate number from this list

Notes

The change takes effect following the next reset.

[RISC OS Open source page](#)

*Configure Boot

This *Command sets the configured boot action so that a power-on, reset or Ctrl-Break runs a boot file.

Syntax

*Configure Boot

Parameters

None.

Notes

If the shift key is pressed when a machine is powered on, or reset, then no boot will take place.

The boot process looks for the &.!Boot file, and if found runs it, as set by the *Opt 4 command.

A boot process can be disabled at any time by issuing *FX 255 command.

NoBoot is the default setting.

Any change takes effect on the next power-on or hard reset.

[RISC OS Open source page](#)

*Configure BootNet

This *Command sets the configured state for whether or not the AUN software is to be loaded.

Syntax

*Configure BootNet On|Off

Parameters

On / AUN will be used

Off – AUN will be used: AUN is disabled

Notes

The default value is off.

[RISC OS Open source page](#)

*Configure BST

This *Command sets the configured timezone to UTC+00:00 with daylight saving set to ON. It is only available when the UK territory is active, other territories will accept their local name for summer time instead.

Syntax

*Configure BST

Parameters

None.

Notes

Use *Configure DST to change the daylight saving flag without affecting the timezone.

[RISC OS Open source page](#)

*Configure Cache

This *Command sets the configured state of the cache on or off.

Syntax

*Configure Cache On|Off

Parameters

On / The caches are enabled by default

Off – The caches are enabled by default: The caches are disabled

Notes

None.

[RISC OS Open source page](#)

*Configure Caps

This *Command sets the configured value for Caps Lock to 'ON'.

Syntax

*Configure Caps

Parameters

None.

Notes

The default value is NoCaps, which can be set by *Configure NoCaps.

[RISC OS Open source page](#)

*Configure CDROMBuffers

This *Command sets the configured amount of memory reserved for CD-ROM buffering, in kilobytes.

Syntax

*Configure CDROMBuffers size[K]

Parameters

size · Size of memory to reserve in kilobytes; Must be one of the following values:

0, 8, 16, 32, 64, 128, 256 or 512

Notes

If a value other than one of those above is given, then the greatest permitted value less than the given value will be used.

[RISC OS Open source page](#)

*Configure CDROMDrives

This *Command sets the configured number of CD-ROM drives recognised at power on.

Syntax

*Configure CDROMDrives n

Parameters

n · Number of drives, in the range of 0 – 28

[RISC OS Open source page](#)

*Configure Country

This *Command sets the configured alphabet and keyboard layout.

Syntax

*Configure Country country_name

Parameters

country_name · Valid country name. A list of countries can be obtained with *Countries

Notes

None.

[RISC OS Open source page](#)

*Configure Data

This *Command sets the configured data format for the serial port.

Syntax

*Configure Data n

Parameters

n · Data word format (0 – 7)

0 · 7 – Parity: Even · Stop bits: 2

1 · 7 – Parity: Odd · Stop bits: 2

2 · 7 – Parity: Even · Stop bits: 1

3 · 7 – Parity: Odd · Stop bits: 1

4 · 8 – Parity: None · Stop bits: 2

5 · 8 – Parity: None · Stop bits: 1

6 · 8 – Parity: Even · Stop bits: 1

7 · 8 – Parity: Odd · Stop bits: 1

Notes

The default value is 4.

The change takes effect following the next reset.

[RISC OS Open source page](#)

*Configure Delay

This *Command sets the configured delay in centiseconds before keys start to auto-repeat.

Syntax

*Configure Delay n

Parameters

n · Delay in centiseconds

Notes

The default value is 32.

A value of zero disables auto-repeat.

[RISC OS Open source page](#)

*Configure Dir

This *Command sets the configured disc mounting so that discs are mounted at power on. NoDir is the default setting (as set by *Configure NoDir).

Syntax

*Configure Dir

Parameters

None.

Notes

This configuration setting has been ineffectual since RISC OS 3.00, when FileSwitch manages disc mounting.

[RISC OS Open source page](#)

*Configure Drive

This *Command sets the configured ADFS drive number that is selected at power on.

Syntax

*Configure Drive n

Parameters

n · Drive number (0 – 7)

Notes

Floppy drives can be drive numbers 0 to 3, and hard discs are numbered 4 to 7.

[RISC OS Open source page](#)

*Configure DST

This *Command sets the configured value for daylight saving time to ON.

Syntax

*Configure DST

Parameters

None.

Notes

The time zone is set when you configure your computer's territory, not by this command.

[RISC OS Open source page](#)

*Configure DumpFormat

This *Command sets the configured format used by the *Dump, *List and *Type commands.

Syntax

*Configure DumpFormat n

Parameters

n · Number from 0 – 15. Bottom two bits define how control character codes are displayed:

0 = GStFormat (eg |A for ASCII 1)

1 = Full stop '.'

2 = <d>, where d is a decimal number

3 = <&h>, where h is a hexadecimal number

Notes

If bit 2 is set, characters that have their top bit set are treated as printable, otherwise they treated as control characters.

If bit 3 is set, characters which have their top bit set are ANDed with &7F before processing so the top bit is no longer set.

The default value is 4.

The change takes effect immediately.

[RISC OS Open source page](#)

*Configure FileSystem

This *Command sets the configured filing system to be used at power on or hard reset.

Syntax

*Configure FileSystem fs_name|_fs_number

Parameters

fs_name / Filing system Name (e.g. ADFS, SCSI, SDFS, Net, RAM)

fs_number – Filing system Name (e.g. ADFS, SCSI, SDFS, Net, RAM): Filing System number

Notes

To specify a Filing System by name, FileSwitch must already have that name registered. This is because FileSwitch converts the name to a number for use.

If the configured Filing System is invalid, then an error will be returned on every subsequent command that tries to access the currently selected filing system, until a valid Filing System is selected.

[RISC OS Open source page](#)

*Configure Floppies

This *Command sets the configured number of floppy drives recognised at power on.

Syntax

*Configure Floppies n

Parameters

n · Number of floppy drives, in the range of 0 – 4

[RISC OS Open source page](#)

*Configure FontMax

This *Command sets the configured maximum size of the font cache.

Syntax

*Configure FontMax m K|n

Parameters

m K / Amount of memory to reserve (kilobytes)

n – Amount of memory to reserve (kilobytes): Number of 4k chunks to reserve

Notes

While *Configure FontSize is used to set the amount of memory reserved for the font cache, this command is used to specify extra memory that the Font Manager may need.

[RISC OS Open source page](#)

*Configure FontMax1

This *Command sets the configured maximum height at which to scale from a bitmap font rather than from an outline font.

Syntax

*Configure FontMax1 max_pointsize

Parameters

max_pointsize · Maximum height (pointsize) of font to scale from bitmap to outline font

Notes

This setting is only valid with 4bpp screen modes.

Before this setting is used, the following checks are made;

- Look for f9999x9999 file of the correct size, then;
- Look for x90y45 font of the correct size;
- Check values in FontMax2 and FontMax3, then;
- Check values in FontMax4 and FontMax5, then;
- If the above checks fail to produce an output will it consider the value of FontMax1.

Pixel height = height in points x pixels per inch / 72

The pixel height varies depending on the different resolution of screen mode.

[RISC OS Open source page](#)

*Configure FontMax2

This *Command sets the configured maximum height at which to scale from outlines to anti-aliased bitmaps, rather than to 1 bit per pixel bitmaps.

Syntax

*Configure FontMax2 max_height

Parameters

max_height · Maximum height (points) of font to scale from outlines to anti-aliased bitmaps

Notes

Before this setting is used, the following checks are made;

- Look for f9999x9999 file of the correct size, then;
- Look for x90y45 font of the correct size;
- If the above checks fail to produce an output will it consider the value of FontMax2.

Pixel height = height in points x pixels per inch / 72

The pixel height varies depending on the different resolution of screen mode.

[RISC OS Open source page](#)

*Configure FontMax3

This *Command sets the configured maximum height at which to retain bitmaps in the cache, rather than the outlines from which they were converted.

Syntax

*Configure FontMax3 max_height

Parameters

max_height · Maximum height (points) of fonts which retain bitmaps in the cache

Notes

Unlike the other FontMax__n__ values, FontMax3 values affects fonts in both 4bpp and 1bpp screen modes.

Use in 4 bpp Screen modes

Before this setting is used, the following checks are made;

- Look for f9999x9999 file of the correct size, then;
- Look for x90y45 font of the correct size;
- If the above checks fail to produce an output will it consider the value of FontMax3.
- If font pixel height is less or equal to heights specified in FontMax2 and FontMax3, then check values in FontMax4 and FontMax5, and finally FontMax1. Any resultant bitmap will be kept in the cache.
- If font pixel height is greater than height specified in FontMax3 then do not cache bitmap. Instead cache the outline.

Use in 1bpp Screen modes

Before this setting is used, the following checks are made;

- Look for b9999x9999 file of the correct size, then;
- If the above checks fails, it looks for the Outline file, and scaling it to give 1bpp bitmap.
- If font pixel height less or equal to heights specified in FontMax3 then any resultant bitmap will be kept in the cache.
- If font pixel height is greater than height specified in FontMax3 then do not cache the resultant bitmap. Instead cache the outline.

$\text{Pixel height} = \text{height in points} \times \text{pixels per inch} / 72$

The pixel height varies depending on the different resolution of screen mode.

[RISC OS Open source page](#)

*Configure FontMax4

This *Command sets the configured maximum width at which to use horizontal subpixel anti-aliasing.

Syntax

*Configure FontMax4 max_width

Parameters

max_width · Maximum width of font to use horizontal subpixel anti-aliasing (in Points)

Notes

This setting is only valid with 4bpp screen modes.

Before this setting is used, the following checks are made;

- Look for f9999x9999 file of the correct size, then;
- Look for x90y45 font of the correct size;
- If the above checks fail to produce an output will it consider the values of FontMax2 and FontMax3, then FontMax4 and FontMax5. If the font pixel width is less than or equal to FontMax 4, then it will look for the Outline file, and construct 4 anti-aliased bitmaps for each character. If the font pixel height is less than or equal to the height in FontMax5, then both horizontal and vertical subpixel anti-aliasing occurs. This equates to 16 bitmaps to be constructed per character.

Pixel width = width in points x pixels per inch / 72

The pixel height varies depending on the different resolution of screen mode.

[RISC OS Open source page](#)

*Configure FontMax5

This *Command sets the configured maximum height at which to use vertical subpixel anti-aliasing.

Syntax

*Configure FontMax5 max_height

Parameters

max_height · Maximum height of font to use vertical subpixel anti-aliasing (in Points)

Notes

This setting is only valid with 4bpp screen modes.

Before this setting is used, the following checks are made;

- Look for f9999x9999 file of the correct size, then;
- Look for x90y45 font of the correct size;
- If the above checks fail to produce an output will it consider the values of FontMax2 and FontMax3, then FontMax4 and FontMax5. If the font pixel width is less than or equal to FontMax 4, then it will look for the Outline file, and construct 4 anti-aliased bitmaps for each character. If the font pixel height is less than or equal to the height in FontMax5, then both horizontal and vertical subpixel anti-aliasing occurs. This equates to 16 bitmaps to be constructed per character.

Pixel height = height in points x pixels per inch / 72

The pixel height varies depending on the different resolution of screen mode.

[RISC OS Open source page](#)

*Configure FontSize

This *Command sets the configured amount of memory reserved for the font cache.

Syntax

*Configure FontSize size K

Parameters

size K · Memory to reserve for font cache (Kilobytes)

Notes

This memory is claimed when the Font Manager is first initialised. If enough memory could not be reserved, then Font Manager is started with what is available.

The amount of memory reserved by this call is the minimum amount of memory that will be used for the font cache.

[RISC OS Open source page](#)

*Configure FreewayAutoAddress

This *Command determines whether, after scanning the available network interfaces that could be used, Freeway should automatically pick an IP address if all no suitable interface was found to be configured by other means.

Syntax

*Configure FreewayAutoAddress On|Off

Parameters

On / Freeway will automatically choose an address

Off – Freeway will automatically choose an address: Freeway will not automatically choose an address

Notes

Typically the boot sequence assigns IP addresses to interfaces, but in a discless machine it would be necessary to automatically assign one in order to be able to mount a remote share from which to boot (so the setting would be 'On' in this scenario).

The address will be chosen from the 10.x.x.x range reserved for private networks, with the lower 3 octets calculated from a hash of the interface name and its MAC address.

[RISC OS Open source page](#)

*Configure FS

This *Command sets the configured default file server for NetFS.

Syntax

*Configure FS file_server

Parameters

file_server · Name or number of file server

Notes

The default value is 0.254

[RISC OS Open source page](#)

*Configure GMT

This *Command sets the configured timezone to UTC+00:00 with daylight saving set to OFF. It is only available when the UK territory is active, other territories will accept their local name for standard time instead.

Syntax

*Configure GMT

Parameters

None.

Notes

Use *Configure NoDST to change the daylight saving flag without affecting the timezone.

[RISC OS Open source page](#)

*Configure HardDiscs

This *Command sets the configured number of ST506 hard disc drives recognised at power on.

Syntax

*Configure HardDiscs n

Parameters

n · Number of drives, in the range of 0 - 2

Notes

This command is only used to set the number of old-style ST506 hard disc drives, the type used in the 1980s in early model RISC OS computers. This configuration option is not available in versions of ADFS with ST506 support omitted.

[RISC OS Open source page](#)

*Configure IDEDiscs

This *Command sets the configured number of IDE hard disc drives recognised at power on.

Syntax

*Configure IDEDiscs n

Parameters

n · Number of drives, in the range of 0 - 2

Notes

This command is used to set the number of parallel ATA (PATA) IDE hard disc drives. ADFS 3.34 and later will autodetect such drives, and therefore this configuration option is no longer available in those versions.

[RISC OS Open source page](#)

*Configure Ignore

This *Command sets the configured printer ignore character.

Syntax

*Configure Ignore [ASCII_code]

Parameters

ASCII_code · ASCII code (0 – 255)

Notes

The default value is 10 (ASCII linefeed). A value of 0, sets the ignore character to ASCII code 0.

Any change made, will only take effect on the next hard reset.

[RISC OS Open source page](#)

*Configure Language

This *Command sets the configured language used at power-on.

Syntax

*Configure Language module_number

Parameters

module_number · Number of module to be run at power-on

Notes

The unusual word 'Language' for this command is for backwards compatibility with earlier versions of the Operating Systems.

The default value is the desktop.

The module_number is specified by the number returned by *Modules command. Numbers may differ depending on versions of the Operating System, and whether modules have been added or removed.

If a machine is configured to go to the desktop, and to run a file using *Exec, it will not enter the desktop. A machine must be configured as one or the other, but not both. This is because the desktop module checks on entry if a file is being run using *Exec, and if so, exits.

[RISC OS Open source page](#)

*Configure Lib

This *Command sets the configured library selected by NetFS after logon.

Syntax

*Configure Lib [0|1]

Parameters

0 or 1

Notes

A value of 0 specifies the default library, while a value of 1 specifies \$.ArthurLib.

[RISC OS Open source page](#)

*Configure LMNameServer

This *Command sets the master browser address which LanManFS will query to get a list of servers on the local network.

Syntax

*Configure LMNameServer <IP_address>

Parameters

IP_address · Fully qualified IPv4 address

Notes

For SMB networks this address will be contacted to retrieve a list of servers known on the domain which is held in the FS setting. If the name server setting is unset then LanManFS will listen for a browser election to try to determine the master browser automatically – this may not work if the network administrator has prevented broadcast messages on the network.

[RISC OS Open source page](#)

*Configure LMTransport

This *Command sets the transport protocol which LanManFS uses.

Syntax

*Configure LMTransport NetBEUI|IP

Parameters

NetBEUI / NetBIOS Extended User Interface

IP – NetBIOS Extended User Interface: Internet Protocol

Notes

While NetBEUI has slightly lower packet overheads, it is not routable between networks, so has fallen from favour – it is no longer widely supported by the majority of SMB servers. The Internet Protocol implements NetBIOS over TCP/IP, and is the default selected after a CMOS reset since RISC OS 5.20.

[RISC OS Open source page](#)

*Configure Loud

This *Command sets the configured volume for the beep to the loudest volume setting.

Syntax

*Configure Loud

Parameters

None.

Notes

The change takes effect on the next reset.

[RISC OS Open source page](#)

*Configure Mode

This *Command sets the configured screen mode.

Syntax

*Configure Mode screen_mode|auto

Parameters

screen_mode / Display mode to use on device power-up

auto – Display mode to use on device power-up: Automatic setting of appropriate screen mode

Notes

This command is identical to the *Configure WimpMode command.

[RISC OS Open source page](#)

*Configure MonitorType

This *Command sets the configured monitor type.

Syntax

*Configure MonitorType n|Auto|EDID

Parameters

n / Monitor type number

Auto – Monitor type number: Automatic monitor type selection via the monitor ID pins

EDID – Monitor type number: Automatic configuration from the monitor's EDID

Notes

The configured monitor type can also be set to one of the numeric types by holding down the corresponding key on the numeric keypad while the computer is switched on.

On RISC OS 3.5 and above, the configured monitor type is typically only relevant up until the point where the boot sequence loads a MDF (or EDID file) via *LoadModeFile, at which point the ScreenModes module will take care of managing the list of modes that are made available to the system.

Setting the configured monitor type to 7 (MDF) is not recommended as the kernel's built in mode list does not contain any entries for that monitor type.

The "Auto" setting is only fully supported on Archimedes-era machines. RiscPC / IOMD machines do not implement the full monitor detection circuitry, and the introduction of Monitor Definition Files largely rendered the use of the numeric monitor types obsolete. Additionally, monitors which support EDID may confuse the detection system, and RISC OS 5.20 dropped support for the detection completely (any "Auto" monitor will be treated as type 3, VGA)

EDID support was introduced in RISC OS 5.23. Note that IOMD / VIDC hardware is unable to read EDID; setting the monitor type to EDID on such a machine is unlikely to be very useful.

[RISC OS Open source page](#)

*Configure MouseStep

This *Command sets the configured value of the pointer-speed multiplication factor that determines how fast the pointer moves when the mouse is moved.

Syntax

*Configure MouseStep n

Parameters

n · Speed (1 - 127)

Notes

Although values up to 127 are accepted, anything above 6 is impractical.

OS_Word 21,2 can also be used to set the mouse step in real-time.

[RISC OS Open source page](#)

*Configure MouseType

This *Command sets the configured pointer device to be used.

Syntax

*Configure MouseType device_type

Parameters

device_type · Type of device to use; Must be one of the following values provided here

[RISC OS Open source page](#)

*Configure NoBoot

This *Command sets the configured boot action so that a Shift-power-on, shift-reset or Shift-Break runs a boot file.

Syntax

*Configure NoBoot

Parameters

None.

Notes

If the shift key is pressed when a machine is powered on, or reset, then a boot will take place.

The boot process looks for the &.!Boot file, and runs it if it finds it, as set by the *Opt 4 command.

A boot process can be disabled at any time by issuing *FX 255 command.

NoBoot is the default setting.

Any change takes effect on the next power-on or hard reset.

[RISC OS Open source page](#)

*Configure NoCaps

This *Command sets the configured value for Caps Lock to 'OFF'.

Syntax

*Configure NoCaps

Parameters

None.

Notes

The default value is NoCaps. To change the configured value use *Configure Caps.

[RISC OS Open source page](#)

*Configure NoDir

This *Command sets the configured disc mounting status so that discs are not mounted at power on. This is the default setting.

Syntax

*Configure NoDir

Parameters

None.

Notes

This configuration setting has been ineffectual since RISC OS 3.00, when FileSwitch manages disc mounting.

[RISC OS Open source page](#)

*Configure NoDST

This *Command sets the configured value for daylight saving time to OFF.

Syntax

*Configure NoDST

Parameters

None.

Notes

The time zone is set when the computer's territory is configured, not by this command.

[RISC OS Open source page](#)

*Configure NoNum

This *Command sets the configured value for Numeric Lock to 'OFF'.

Syntax

*Configure NoNum

Parameters

None.

Notes

The default value is to have Numeric Lock 'ON'.

[RISC OS Open source page](#)

*Configure NoScroll

This *Command sets the configured scrolling so the screen does not scroll upwards at the end of a line.

Syntax

*Configure NoScroll

Parameters

None.

Notes

The default value is Scroll.

For more information on NoScroll, please see VDU 23,16.

[RISC OS Open source page](#)

*Configure Num

This *Command sets the configured value for Numeric Lock to 'ON'.

Syntax

*Configure Num

Parameters

None.

Notes

The default value is to have Numeric Lock 'ON'.

[RISC OS Open source page](#)

*Configure NVidia

This *Command allows for various options relating to red/blue swapping of the display to be specified.

Syntax

*Configure NVidia [-device <pcidevno>] -auto | -manual [-swap8bpp] [-swap16bpp] [-swap32bpp] [-alt16bpp]

Parameters

-device <pcidevno> / PCI device number to configure (as per *PCIDevices)

If omitted, configuration settings will be applied to all cards recognised by the driver

-auto – PCI device number to configure (as per *PCIDevices): Enable automatic configuration (default)

-manual – PCI device number to configure (as per *PCIDevices): Enable manual configuration. The following options become available:

-swap8bpp – PCI device number to configure (as per *PCIDevices): Red-blue swap 8bpp modes

-swap16bpp – PCI device number to configure (as per *PCIDevices): Red-blue swap 16bpp modes

-swap32bpp – PCI device number to configure (as per *PCIDevices): Red-blue swap 32bpp modes

-alt16bpp – PCI device number to configure (as per *PCIDevices): Alternate red-blue swapping of mouse and gamma table in 16bpp modes

Notes

Some options only affect the configuration of the hardware, while other options will affect the configuration of the OS (e.g. whether the "LTRGB" screen modes introduced in RISC OS 5.21 are used). Additionally, many machines use graphics cards which have had the red and blue lines swapped over on the VGA connector. This means that if your display is incorrect in some colour depths, some experimentation may be required in order to find the settings that work best for your machine and the software you use.

[RISC OS Open source page](#)

*Configure Print

The purpose of this *Command is to set the configured default destination for printed output.

Syntax

*Configure Print n

Parameters

n / Printer destination (0 – 7): Consists of:

- 0 – Printer destination (0 – 7): Consists of:: Null (no output)
- 1 – Printer destination (0 – 7): Consists of:: Parallel port
- 2 – Printer destination (0 – 7): Consists of:: Serial port
- 3 – Printer destination (0 – 7): Consists of:: Path in system variable PrinterType\$3 (reserved for user)
- 4 – Printer destination (0 – 7): Consists of:: Network printer (handled through Netprint)
- 5 – 7 – Printer destination (0 – 7): Consists of:: Path in system variables PrinterType\$5, 6 or 7

Notes

For any value of n_, the path held in the variable PrinterType\$_n is used.

The change takes effect on the next hard reset.

[RISC OS Open source page](#)

*Configure PrinterBufferSize

The purpose of this *Command is to set the configured amount of memory reserved for printer buffering.

Syntax

*Configure PrinterBufferSize m K|n

Parameters

m K / Kilobytes of memory to reserve

n – Kilobytes of memory to reserve: Number of pages of memory to reserve (0 – 127)

Notes

This command should not be used.

[RISC OS Open source page](#)

*Configure PS

The purpose of this *Command is to set the configured default network printer server.

Syntax

*Configure PS printer_server

Parameters

printer_server · Name or station number of printer server

Notes

It is not necessary to be logged in to use a printer server.

[RISC OS Open source page](#)

*Configure Quiet

The purpose of this *Command is to set the configured volume for the beep to the half the maximum volume.

Syntax

*Configure Quiet

Parameters

None.

Notes

The change takes effect on the next reset.

[RISC OS Open source page](#)

*Configure RamFSSize

The purpose of this *Command is to set the configured amount of memory reserved for the RAM Filing System to use.

Syntax

*Configure RamFSSize m K|n

Parameters

m K / Number of kilobytes to reserved

n – Number of kilobytes to reserved: Number of pages of memory reserved (0 – 127)

Notes

Does not take effect until the next hard reset.

The default value is 0, which disables the RAM filing system.

[RISC OS Open source page](#)

*Configure Repeat

The purpose of this *Command is to set the configured interval in centiseconds between the generation of auto-repeat keys.

Syntax

*Configure Repeat n

Parameters

n · Interval in centiseconds

Notes

The default value is 8.

To completely disable auto-repeat, set the delay to zero; *Configure Delay 0 will do this.

[RISC OS Open source page](#)

*Configure RMASize

The purpose of this *Command is to set the configured extra area of memory reserved for relocatable modules.

Syntax

*Configure RMASize m K|n

Parameters

m K / Kilobytes of memory to reserve

n – Kilobytes of memory to reserve: Number of pages of memory to reserve (0 – 127)

Notes

If the parameter passed is 0, no extra memory is reserved for relocatable modules.

[RISC OS Open source page](#)

*Configure ScreenSize

The purpose of this *Command is to set the configured extra area of memory reserved for screen size.

Syntax

*Configure ScreenSize m K|n

Parameters

m K / Kilobytes of memory to reserve

n – Kilobytes of memory to reserve: Number of pages of memory to reserve (0 – 127)

Notes

If no parameter is passed, no extra memory is reserved for screen size.

[RISC OS Open source page](#)

*Configure Scroll

The purpose of this *Command is to set the configured scrolling so the screen scrolls upwards at the end of a line.

Syntax

*Configure Scroll

Parameters

None.

Notes

The default value is Scroll. It can be changed by *Configure NoScroll.

For more information on Scroll, please see VDU 23,16.

[RISC OS Open source page](#)

*Configure SCSI Drive

If the startup filesystem is configured as 'SCSI' then the drive number should be specified.

Syntax

*Configure SCSI Drive n

Parameters

n · Drive number (0 - 7)

Notes

USB memory sticks can be drive numbers 0 to 3, and USB hard discs are numbered 4 to 7.

[RISC OS Open source page](#)

*Configure ShareBoot

This *Command sets the remote disc name which ShareFS will contact in order to boot from if it's the configured filing system.

Syntax

*Configure ShareBoot discname

Parameters

discname · Name of a remote shared disc on the network

Notes

Should the remote discname be unavailable on boot ShareFS will wait a while for it to become available. Once mounted, ShareFS runs Share::discname.\$!ShareBoot on the remote disc.

[RISC OS Open source page](#)

*Configure ShCaps

This *Command sets the configured value for Caps Lock to 'ON', while Shift key generates lower case letters.

Syntax

*Configure ShCaps

Parameters

None.

Notes

This command does two things;

- Configure Caps Lock to 'ON'
- Generate lower case letters when the shift key is held down.

If the configure value is 'Caps' (set by *Configure Caps), then the shift key does not generate lower case letters.

[RISC OS Open source page](#)

*Configure SoundDefault

This *Command sets the configured speaker setting, volume and voice.

Syntax

*Configure SoundDefault speaker volume voice_number

Parameters

speaker / 0 to disable internal speaker

1 to enable internal speaker

volume – 0 to disable internal speaker: 0 – 7 (quiet to loud)

voice_number – 0 to disable internal speaker: Voice number (1 – 16)

Notes

None.

[RISC OS Open source page](#)

*Configure SoundSystem

This *Command sets the configured default sound system.

Syntax

*Configure SoundSystem 8bit | 16bit [oversampled] | n

Parameters

8bit / Standard 8-bit sound system

16bit – Standard 8-bit sound system: Standard 16-bit sound system

oversampled – Standard 8-bit sound system: Perform sample interpolation

n – Standard 8-bit sound system: CMOS RAM settings (0 - 7)

Notes

None.

[RISC OS Open source page](#)

*Configure SpriteSize

This *Command sets the configured size of the extra area of memory reserved for the system sprite size.

Syntax

*Configure SpriteSize m K|n

Parameters

m K / Kilobytes of memory to reserve

n – Kilobytes of memory to reserve: Number of pages of memory to reserve (0 - 127)

Notes

If no parameter is passed, no extra memory is reserved for the system sprite size.

Changes will not take effect until the next reset.

[RISC OS Open source page](#)

*Configure Step

This *Command sets the configured step rate of one or all floppy disc drives.

Syntax

*Configure Step n [drive]

Parameters

n / Step time (in milliseconds)

drive – Step time (in milliseconds): Drive number (0 - 3)

Notes

The default n value is 3 milliseconds. If no drive parameter is specified, then the step rate is set for all floppy disc drives.

[RISC OS Open source page](#)

*Configure Sync

This *Command sets the configured type of synchronisation for vertical sync output.

Syntax

*Configure Sync 0|1|Auto

Parameters

0 / Vertical sync

1 – Vertical sync: Composite sync

Auto – Vertical sync: Automatic detection

Notes

None.

[RISC OS Open source page](#)

*Configure SystemSize

This *Command sets the configured size of the extra area of memory reserved for the system heap.

Syntax

*Configure SystemSize m K|n

Parameters

m K / Kilobytes of memory to reserve

n – Kilobytes of memory to reserve: Number of pages of memory to reserve (0 – 63)

Notes

If the parameter passed is 0, no extra memory is reserved for the system heap.

[RISC OS Open source page](#)

*Configure Territory

This *Command sets the configured default territory for the machine.

Syntax

*Configure Territory territory

Parameters

territory · Valid territory name. A list is available [here](#) or by issuing a *Territories command

Notes

Use this command with caution; if you set a territory that is unavailable your computer will not start, and so you will have to reset your CMOS RAM.

[RISC OS Open source page](#)

*Configure TimeZone

This *Command sets the configured local time offset from UTC.

Syntax

*Configure TimeZone [+|-] hours [:minutes]

Parameters

hours / UTC offset in hours

minutes – UTC offset in hours: UTC offset in minutes

Notes

The time offset must be in the range -13:45 to +13:45, and must be an exact multiple of 15 minutes.

[RISC OS Open source page](#)

*Configure Truncate

This *Command sets the configured value for whether or not filenames are truncated when too long for a filing system to handle.

Syntax

*Configure Truncate On|Off

Parameters

On / Long filenames truncated

Off – Long filenames truncated: Long filenames not truncated

Notes

None.

[RISC OS Open source page](#)

*Configure TV

This *Command sets the configured vertical screen alignment and screen interlace.

Syntax

*Configure TV [vert_align [[,],interlace]]

Parameters

vert_align / Adjust vertical screen alignment. Possible values of:

0 - 3: lines up by this amount

255 - 252: lines down 1 to 4 lines (respectively)

interlace – Adjust vertical screen alignment. Possible values of: Switch screen interlace setting. Possible values of:

0 = Interlace on

1 = Interlace off

[RISC OS Open source page](#)

*Configure WimpAutoMenuDelay

This *Command sets the configured time the pointer must rest over a menu item before its submenu (if any) is automatically opened.

Syntax

*Configure WimpAutoMenuDelay delay

Parameters

delay · Time in 1/10 th of a second units

Notes

This will only be of use, if the automatic opening of submenu's is enabled using *Configure WimpFlags.

[RISC OS Open source page](#)

*Configure WimpDoubleClickDelay

This *Command sets the configured time after a single click during which a double-click is accepted.

Syntax

*Configure WimpDoubleClickDelay delay

Parameters

delay · Time in 1/10 th of a second units

Notes

A pending double-click will be immediately cancelled if any of the following events occur:

- Wimp_DragBox is called

- Pointer moves by more than a specified number of OS units
- Mouse is not clicked during the allocated time, as configure by this command

[RISC OS Open source page](#)

*Configure WimpDoubleClickMove

This *Command sets the configured distance from the position of a single click within which a double click is accepted.

Syntax

*Configure WimpDoubleClickMove distance

Parameters

distance · Distance in OS units

Notes

A pending double-click will be immediately cancelled if the pointer moves by more than a specified number of OS units

[RISC OS Open source page](#)

*Configure WimpDragDelay

*Configure WimpDragDelay delay

This *Command sets the configured time before an automatically opened submenu is closed.

Syntax

*Configure WimpDragDelay delay

Parameters

delay · Time in 1/10 th of a second units

Notes

During this time the pointer can be moved over other menu entries without closing the submenu. This makes it easy to reach the submenu.

[RISC OS Open source page](#)

*Configure WimpDragMove

This *Command sets the configured distance from the position of a single click that the pointer has to move for a drag to be started.

Syntax

*Configure WimpDragMove distance

Parameters

distance · Distance in OS units

Notes

None.

[RISC OS Open source page](#)

*Configure WimpFlags

This *Command sets the configured default actions when dragging windows.

Syntax

*Configure WimpFlags n

Parameters

Bit – A value between 0 and 255. Bits as;: Meaning when set

0 – A value between 0 and 255. Bits as;: Window position drags are continuously redrawn

1 – A value between 0 and 255. Bits as;: Window resizing drags are continuously redrawn

2 – A value between 0 and 255. Bits as;: Horizontal scroll drags are continuously redrawn

3 – A value between 0 and 255. Bits as;: Reserved

4 – A value between 0 and 255. Bits as;: No beep is generated when error box appears

5 – A value between 0 and 255. Bits as;: Windows can be dragged partly off screen to right and bottom

6 – A value between 0 and 255. Bits as;: Windows can be dragged partly off screen in all directions

7 – A value between 0 and 255. Bits as;: Open submenus automatically

Notes

None.

[RISC OS Open source page](#)

*Configure WimpFont

This *Command sets the configured value for the font to use on the desktop.

Syntax

*Configure WimpFont n

Parameters

n · Number specifying the font to use (0 – 15)

Notes

A parameter of 0 sets the font from Wimp\$Font system variables; a parameter of 1 sets the System font; parameters 2 – 15 set a ROM font.

[RISC OS Open source page](#)

*Configure WimpMenuDragDelay

*Configure WimpAutoMenuDelay

This *Command sets the configured time in 1/10 second units during which menu activity is disabled after a menu has been automatically opened.

Syntax

*Configure WimpMenuMenuDelay delay

Parameters

delay · Time in 1/10 th of a second units

Notes

During this time the pointer can be moved over other menu entries without causing the submenu to be closed.

[RISC OS Open source page](#)

*Configure WimpMode

This *Command sets the configured screen mode used by the wimp.

Syntax

*Configure WimpMode screen_mode|auto

Parameters

screen_mode / Display mode to use on device power-up

auto – Display mode to use on device power-up: Automatic setting of appropriate screen mode

Notes

This command is identical to the *Configure Mode command.

[RISC OS Open source page](#)

*Continue

Debugger

This *Command resumes execution after a breakpoint, using the saved state.

Syntax

*Continue

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*Copy

This *Command copies between directories the object(s) that match the given wildcard specification.

Syntax

*Copy source_spec destination_spec [[~]options]

Parameters

dest_spec – Valid (wildcarded) pathname specifying a file or directory: Valid (wildcarded) pathname specifying a file or directory

options · Valid options (Upper/lowercase letters, separated by spaces) · Default ?

A(ccess) · Force destination access to same as source · ON

C(onfirm) · Prompt for confirmation of each copy · ON

D(elete) · Delete the source object after copy · OFF

F(orce) · Force overwriting of existing object · OFF

L(ook) · Look at destination before loading source file · OFF

N(ewer) · Copy only if source more recent than destination · OFF

P(rompt) · Prompt for disc to be changed as needed in copy · OFF

Q(uick) · Use application workspace as a buffer · OFF

R(ecurse) · Copy subdirectories and contents · OFF

S(tamp) · Restamp date-stamped files after copying · OFF

(s)T(ructure) · Copy only the directory structure · OFF

V(erbose) · Print information on each object copied · ON

Notes

Options are taken from the system variable Copy\$Options, and those given to the command.

To turn on a specific option, it must be included in the option list. To ensure a specific option is turned off, then it must be preceded with the tilde symbol (~).

The leafname of the destination must specify a filename, or the wildcard symbol '*', which in the latter case will use the same pathname as the source file.

It is important to note, that an infinite loop can easily be caused by copying a directory into one of its sub-directories.

[RISC OS Open source page](#)

*CopyBoot

This *Command copies the boot block from one MS-DOS floppy disc over the boot block of another.

Syntax

*CopyBoot source_drive dest_drive

Parameters

source_drive / Number of the source floppy drive (0 - 3)

dest_drive – Number of the source floppy drive (0 - 3): Number of the destination floppy drive (0 - 3)

Notes

It does not place the system files on a disc, it simply copies the boot block from one disc to another. The system files still need to be copied from an appropriate MS DOS disc.

[RISC OS Open source page](#)

*Count

This *Command adds up the sizes of data held in one or more objects that match the given wildcard specification.

Syntax

*Count object_spec [[~]options]

Parameters

object_spec / Valid (wildcarded) pathname specifying a file or directory

options · Valid options (Upper/lowercase letters, separated by spaces). · Default

C(onfirm) · Prompt for confirmation of each copy · OFF

R(ecurse) · Count subdirectories and contents · ON

V(erbose) · Print information on each object copied · OFF

Notes

Options are taken from the system variable Copy\$Options, and those given to the command.

The size of data returned consists of the actual amount of data each object is comprised of, rather than the amount of disc space it occupies.

To turn on a specific option, it must be included in the option list. To ensure a specific option is turned off, then it must be preceded with the tilde symbol (~).

[RISC OS Open source page](#)

*Countries

This *Command lists all the countries currently supported.

Syntax

*Countries

Parameters

None.

Notes

With no parameter, this command displays the currently selected alphabet.

[RISC OS Open source page](#)

*Country

This *Command selects the appropriate alphabet and keyboard layout for a given country.

Syntax

*Country [country_name]

Parameters

country_name · Valid country name. A list of countries can be obtained with *Countries

Notes

None.

[RISC OS Open source page](#)

*Create

This *Command creates an empty file.

Syntax

*Create filename [length [exec_addr [load_addr]]]

Parameters

filename / Valid pathname specifying a file

length – Valid pathname specifying a file: Number of bytes to reserve

exec_addr – Valid pathname specifying a file: Address to jump to after loading (applicable to programs only)

load_addr – Valid pathname specifying a file: Address at which file is loaded into memory (when *Loaded)

Notes

No data is transferred to the file.

The optional load and execution addresses and length are in hexadecimal.

If the load address and execution address parameters are omitted, then a file with a filetype of &FFD (Data) and a timestamp will be created.

The default value for the length parameter and load_address parameter is both zero.

[RISC OS Open source page](#)

*Debug

Debugger

This *Command causes entry to the debugger.

In practice

It can be used from BASIC to invoke a command line shell.

Syntax

*Debug

Parameters

None.

Notes

Press Escape to exit.

[RISC OS Open source page](#)

*Defect

This *Command reports what object contains a defect, or, if none, maps the defect out.

Syntax

*Defect disc_spec disc_addr

Parameters

disc_spec / Name of disc or number of disc drive

disc_addr – Name of disc or number of disc drive: Address where defect exists

Notes

The optional load and execution addresses and length are in hexadecimal.

The disc_addr parameter must be a multiple of 256. i.e. must end in '00'

[RISC OS Open source page](#)

*Delete

This *Command erases a single named file or empty directory.

Syntax

*Delete object_spec

Parameters

object_spec · Valid (wildcarded) pathname specifying a file or empty directory

Notes

Wildcards must not be used in the last component of the pathname. i.e the filename

[RISC OS Open source page](#)

*DeskFS

DeskFS

This *Command selects the Desktop Filing System as the current filing system.

Syntax

*DeskFS

Notes

This command is only applicable for RISC OS 2. This command is not available for subsequent versions of RISC OS and should not be used.

[RISC OS Open source page](#)

*Desktop

WindowManager

This *Command initialises all desktop facilities, then starts the Desktop.

Syntax

*Desktop [command]-File filename]

Parameters

command / *command to be passed to Wimp_StartTask after desktop starts

filename – *command to be passed to Wimp_StartTask after desktop starts: Valid pathname specifying a file

Notes

It also runs an optional *Command or file of *Commands. This can be used to load several applications.

[RISC OS Open source page](#)

*Desktop_

WindowManager

The purpose of this *Command is to start ROM-resident desktop utilities.

Syntax

*Desktop_...

Parameters

None.

Notes

This command is reserved for internal use, and must not be used. The only exception is *Desktop_SetPalette?

[RISC OS Open source page](#)

*DeviceInfo

This *Command displays detailed information about driver-module activity.

Syntax

*DeviceInfo Device

Parameters

Device · Device driver name to use

Notes

Note that this command is supplied by the driver that comes with a network interface, rather than by the Operating System. Each of the standard Acorn drivers provides such a command. We expect third party drivers to provide a corresponding command; you should see the documentation supplied for the command name.

It is presented mainly as an aid to trouble-shooting, should you require it.

[RISC OS Open source page](#)

*DHCPExecute

DHCP

This *Command instructs the DHCP module to control the configuration of the specified interface.

Syntax

*DHCPExecute [-e] [-b] [-w] [-p] <interface>

Parameters

-e / Places any error message into Inet\$Error variable

-b – Places any error message into Inet\$Error variable: Blocks until the interface is bound

-w – Places any error message into Inet\$Error variable: Waits for the interface to appear

-p – Places any error message into Inet\$Error variable: Assigns a private IP address if the DHCP request times out

interface – Places any error message into Inet\$Error variable: Name of the interface to configure

Notes

The interface is usually formed from the driver abbreviation and unit number, for example ey1 is unit 1 for EtherY?. DHCP can be asked to wait, triggering callbacks, until the named interface appears otherwise it is an error if the interface is not present.

[RISC OS Open source page](#)

*DHCPInfo

DHCP

This *Command displays the internal state of the DHCP module.

Syntax

*DHCPInfo [interface]

Parameters

interface · Name of the interface to display state for, or all interfaces if omitted

[RISC OS Open source page](#)

*DimTime

This *Command sets the time in seconds before the screen dims.

Syntax

*DimTime [time]

Parameters

time · Duration of inactivity before screen dims (seconds)

Notes

If, during the specified DimTime there is no activity (i.e. no keyboard or mouse input is received) then the screen then dims. The dim time is only retained until the next reset.

[RISC OS Open source page](#)

*Dir

This *Command sets a directory as the currently selected directory.

Syntax

*Dir [directory]

Parameters

directory · Valid pathname of a directory

Notes

It selects a directory (by default the user root directory) as the current directory on a filing system.

[RISC OS Open source page](#)

*Dismount

This *Command ensures that it is safe to finish using a disc.

Syntax

*Dismount [disc_spec]

Parameters

disc_spec · Name or number of the disc drive

Notes

Buffers are flushed, files are closed, local caches discarded, directories and libraries unset, disc names annulled and, if appropriate, disc-reading heads are parked.

[RISC OS Open source page](#)

*DMAChannels

DMAManager

This *Command lists information about currently registered DMA channels.

Syntax

*DMAChannels

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*Do

BootCommands

This *Command passes its argument to OS_GSTrans and executes the result.

Syntax

*Do command

Parameters

command · Command to pass to OS_GSTrans, before execution

[RISC OS Open source page](#)

*DOSMap

This *Command specifies a mapping between an MS-DOS extension and a RiscOS file-type.

Syntax

*DOSMap [MS-DOS_extension [filetype]]

Parameters

MS-DOS_extension / MS-DOS file extension (up to 3 characters in length)

filetype – MS-DOS file extension (up to 3 characters in length): 3 digit filetype number, or filetype name

Notes

The filetype number should be preceded with an ampersand symbol, otherwise it is interpreted as a filetype name.

If the only parameter given is an MSDOS extension, then the mapping (if any) for that extension is cancelled. If no parameter is given, then all current mappings are listed.

The mappings are only retained until the next reset.

[RISC OS Open source page](#)

*Drive

This *Command sets the current drive.

Syntax

*Drive drive

Parameters

drive · Name or number of the disc drive, from 0 to 7

Notes

This sets the current drive if *NoDir is set.

[RISC OS Open source page](#)

*Dump

This *Command displays the contents of a file, in hexadecimal and ASCII codes.

Syntax

*Dump filename [file_offset [start_addr]]

Parameters

filename / Valid pathname specifying a file

file_offset – Valid pathname specifying a file: Offset from beginning of file

start_addr – Valid pathname specifying a file: Display as if the file were in memory starting at this address

Notes

The optional file offset and start address are in hexadecimal.

The start_addr parameter defaults to the file's load address. If a file is timestamped, then the load address is treated as if it is zero.

The format of data to display can be set using *Configure DumpFormat.

[RISC OS Open source page](#)

*Echo

This *Command displays a string on the screen (after translating it using OS_GSTrans)

Syntax

*Echo string

Parameters

string · String to display

Notes

This command is very useful in command scripts, to check the progress of a script.

[RISC OS Open source page](#)

*Eject

This *Command ejects the disc from the given CD/DVD drive.

Syntax

*Eject [drive]

Parameters

drive · Valid CD/DVD drive number

Notes

If no drive parameter is passed, then the current drive is used.

[RISC OS Open source page](#)

*EnumDir

This *Command creates a file of leafnames, that match the given wildcarded pattern, from the given directory.

Syntax

*EnumDir directory output_file [pattern]

Parameters

directory / Valid pathname specifying a directory

output_file – Valid pathname specifying a directory: Valid pathname specifying a file

pattern – Valid pathname specifying a directory: Wildcarded match specification

Notes

The default pattern value is `'*'`.

The current directory can be specified by using the `@` symbol.

[RISC OS Open source page](#)

*Error

This *Command generates an error with the given error number and explanatory text.

Syntax

*Error [error_number] text

Parameters

error_number / Error number

text – Error number: String of error message (printable characters only)

Notes

If no error number is passed, then a default value of 0 will be used.

[RISC OS Open source page](#)

*Eval

This *Command evaluates an integer or string expression.

Syntax

*Eval expression

Parameters

expression · Valid Command Line expression

Notes

For a comprehensive list of expressions that can be used with this command please see the CLI Expressions? page.

Floating point numbers cannot be used with this command.

[RISC OS Open source page](#)

*Ex

This *Command lists all the objects in a directory together with their corresponding file information.

Syntax

*Ex [directory]

Parameters

directory · Valid pathname specifying a directory

Notes

The default directory is the current directory.

[RISC OS Open source page](#)

*Exec

This *Command instructs the Operating System to take its input from the specified file.

Syntax

*Exec [filename]

Parameters

filename · Valid pathname specifying a file

Notes

If no parameter is given, the current exec file is closed.

[RISC OS Open source page](#)

*FileInfo

This *Command returns full file information for the specified object(s).

Syntax

*FileInfo object_spec

Parameters

object_spec · Valid (wildcarded) pathname specifying one or more file or directories

Notes

None.

[RISC OS Open source page](#)

*Filer_Action

This *Command starts a Filer_Action task running.

Syntax

*Filer_Action

Parameters

None.

Notes

This command can only be used within the desktop environment. An error will be generated if this command is issued outside the desktop environment.

The task sets its own slot size automatically. If the task does not receive a 'start operation' message before the next null event, then it will quit.

To issue this command, you must call Wimp_StartTask with R0 pointing to the string 'Filer_Action'.

[RISC OS Open source page](#)

*Filer_Boot

This *Command boots the specified desktop application by running its !Boot file.

Syntax

*Filer_Boot application

Parameters

application · Valid pathname specifying an application

Notes

This command can only be used within the desktop environment, or from within a Desktop boot file.

[RISC OS Open source page](#)

*Filer_CloseDir

This *Command closes a directory display on the Desktop, and any displayed subdirectories.

Syntax

*Filer_CloseDir directory

Parameters

directory · Valid pathname specifying a directory window to be closed

Notes

This command can only be used within the desktop environment, or from within a Desktop boot file.

[RISC OS Open source page](#)

*Filer_DClickHold

This *Command sets the time in centiseconds that the second mouseclick must be held down to simulate the action of Shift-doubleclick. It can also be used to disable the behaviour, by specifying a time of 0.

Syntax

*Filer_DClickHold centiseconds

Parameters

centiseconds · Time in centiseconds, or 0 to disable doubleclick-and-hold behaviour.

Notes

This command was first introduced in RISC OS 5.16.

[RISC OS Open source page](#)

*Filer_Layout

This *Command sets the default layout for newly opened directory displays.

Syntax

*Filer_Layout [-Largelcons|-SmallIcons|-FullInfo] [-SortByName|-SortByType|-SortBySize|-SortByDate] [-ReverseSort]
[-NumericalSort]

Parameters

- SmallIcons – Alternative: -si · Meaning: Display small icons
- Largelcons – Alternative: -li · Meaning: Display large icons
- FullInfo – Alternative: -fi · Meaning: Display small icons
- SortByName – Alternative: -sn · Meaning: Display sorted by name
- SortByType – Alternative: -st · Meaning: Display sorted by type
- SortByDate – Alternative: -sd · Meaning: Display sorted by date
- SortBySize – Alternative: -ss · Meaning: Display sorted by size
- ReverseSort – Alternative: -rs · Meaning: Sort in reverse order
- NumericalSort – Alternative: -ns · Meaning: Sort digits in names as numbers

[RISC OS Open source page](#)

*Filer_OpenDir

This *Command opens a directory display on the Desktop.

Syntax

*Filer_OpenDir directory [x y [width height]] [switches]

Parameters

- SmallIcons – Alternative: -si · Meaning: Display small icons
- Largelcons – Alternative: -li · Meaning: Display large icons
- FullInfo – Alternative: -fi · Meaning: Display small icons
- SortByName – Alternative: -sn · Meaning: Display sorted by name
- SortByType – Alternative: -st · Meaning: Display sorted by type
- SortByDate – Alternative: -sd · Meaning: Display sorted by date
- SortBySize – Alternative: -ss · Meaning: Display sorted by size
- ReverseSort – Alternative: -rs · Meaning: Sort in reverse order¹
- NumericalSort – Alternative: -ns · Meaning: Sort digits in names as numbers²

¹ Available from Filer 2.22 and above

² Available from Filer 2.34 and above

Each parameter, except for switches, can be preceded by a keyword (shown below)

Keyword / Alternative / Precedes Parameter

- directory – Alternative: -dir · Precedes Parameter: directory
- topleftx – Alternative: -x0 · Precedes Parameter: x
- toplefty – Alternative: -y0 · Precedes Parameter: y
- width – Alternative: -w · Precedes Parameter: width
- height – Alternative: -h · Precedes Parameter: height

Notes

This command can only be used within the desktop environment, or from within a Desktop boot file.

Pathnames should always include the filing system to avoid any ambiguities.

[RISC OS Open source page](#)

*Filer_Options

This *Command sets the default options for Filer operations. The options can also be set from the Filer setup plug-in of Configure.

Syntax

*Filer_Options [-ConfirmAll] [-ConfirmDeletes] [-Verbose] [-Force] [-Newer] [-Faster]

Parameters

Switch / Meaning

- ConfirmAll – Meaning: Prompt for confirmation of all operations
- ConfirmDeletes – Meaning: Prompt for confirmation of deletes only
- Verbose – Meaning: Provide an information window during operations
- Force – Meaning: Force overwrites of existing objects

-**Newer** – Meaning: Copy only if the source is more recent than the destination

-**Faster** – Meaning: Perform operation faster

[RISC OS Open source page](#)

*Filer_Run

This *Command performs the equivalent of double-clicking on an object in a directory display.

Syntax

*Filer_Run [-Shift|-NoShift] object

Parameters

object / Valid pathname specifying an object

-**Shift** – Valid pathname specifying an object: Observe the state of the SHIFT key on the keyboard²

-**NoShift** – Valid pathname specifying an object: Ignore state of the SHIFT key on the keyboard¹

¹ Available from Filer 2.19 and above

² Available from Filer 2.40 and above

Notes

This command can only be used within the desktop environment, or from within a Desktop boot file.

Pathnames should always include the filing system to avoid any ambiguities.

The SHIFT key will be tested by the Filer. If pressed and the Shift switch given the Filer will instead open the containing directory, the default if neither option is specified is NoShift.

[RISC OS Open source page](#)

*Filer_Truncation

This *Command sets the width that long filenames are truncated to for newly opened directory displays.

Syntax

*Filer_Layout [-LargeIconDisplay] [-SmallIconDisplay] [-FullInfoDisplay]

Parameters

Switch / Alternative / Meaning

-**LargeIconDisplay** – Alternative: -LID · Meaning: Truncate width when showing large icons

-**SmallIconDisplay** – Alternative: -SID · Meaning: Truncate width when showing small icons

-**FullInfoDisplay** – Alternative: -FID · Meaning: Truncate width when showing full info

[RISC OS Open source page](#)

*Filer_Visibility

This *Command sets the visibility of special directories.

Syntax

*Filer_Visibility [-IncludeOS | -ExcludeOS]

Parameters

Switch / Meaning

-**IncludeOS** – Meaning: Show special directories such as !Boot (this is the default)

-**ExcludeOS** – Meaning: Hide special directories such as !Boot

[RISC OS Open source page](#)

*Filters

FilterManager

This *Command lists all currently active pre- and post- Wimp_Poll filters.

Syntax

*Filters

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*FontCat

FontManager

This *Command lists the fonts available in Font\$Path or in the given directory.

Syntax

*FontCat [directory]

Parameters

directory · Valid pathname specifying a directory to search

Notes

If no directory is specified, then the directory as specified by Font\$Path variable is used.

[RISC OS Open source page](#)

*FontInstall

FontManager

This *Command adds a directory to the list of those scanned for fonts.

Syntax

*FontInstall [directory]

Parameters

directory · Valid pathname specifying a directory to add to Font\$Path

Notes

It also rescans the directory, even if it was already known to the Font Manager.

If no pathname is specified, all directories in Font\$Path are rescanned.

[RISC OS Open source page](#)

*FontLibrary

FontManager

This *Command sets a directory as the font library, replacing the previous library in the list of those scanned for fonts.

Syntax

*FontLibrary directory

Parameters

directory · Valid pathname specifying a directory

Notes

The font library is set by altering the Font\$Prefix system variable to the specified pathname.

[RISC OS Open source page](#)

*FontList

FontManager

The purpose of this *Command is to display details about the font cache.

Syntax

*FontList

Parameters

None.

Notes

The size of the cache, its free space and the fonts stored in the cache are returned.

[RISC OS Open source page](#)

*FontMerge

FontManager

This *Command merges new fonts into an existing !Fonts directory.

Syntax

*FontMerge source [destination]

Parameters

source / Directory of fonts to merge

destination – Directory of fonts to merge: Directory of destination !Fonts directory

[RISC OS Open source page](#)

*FontRemove

FontManager

This *Command removes a directory from the list of those scanned for fonts.

Syntax

*FontRemove [directory]

Parameters

directory · Valid pathname specifying a directory to remove from Font\$Path

Notes

The directory is removed from the list by removing the pathname from the system variable Font\$Path.

[RISC OS Open source page](#)

*Format

This *Command repairs a new floppy disc for use, or erases a used disc for re-use.

Syntax

*Format drive [format [disc_name]] [Y]

Parameters

drive / Number of the disc drive (0 - 3)

format – Number of the disc drive (0 - 3): Type of disc format required (available types here)

disc_name – Number of the disc drive (0 - 3): Name of new disc

Y – Number of the disc drive (0 - 3): No prompt for confirmation required

Notes

The default disc format to use is 'F'. This format offers improved file management and reduced file fragmentation.

[RISC OS Open source page](#)

*Free

This *Command displays the total free space remaining on a disc.

Syntax

*Free [disc_spec_] *Free [:file_server_] [_username] (NetFS syntax)

Parameters

disc_spec / Name of disc or number of the disc drive

file_server – Name of disc or number of the disc drive: Name of File Server

user_name – Name of disc or number of the disc drive: Name of user for network

Notes

For NetFS, Free displays a user's total free space, as well as the total free space for the disc.

[RISC OS Open source page](#)

*FreePool

BootCommands

This *Command will resize application space to match the current Wimp next slot size. Any memory which is moved out of application space will be moved into the free pool.

Syntax

*FreePool

Notes

This command requires RISC OS 3.5, as it operates by resizing the free pool dynamic area.

[RISC OS Open source page](#)

*FreezeTime

Portable

This *Command allows the user to read or set the delay until the machine will automatically enter freeze mode (i.e. suspend-to-RAM).

Syntax

*FreezeTime delay

Parameters

delay · Delay in seconds

Notes

If no delay is specified, the current delay will be reported.

Specifying a delay of 0 will disable the automatic freeze functionality.

The machine will only automatically enter suspend mode if the screen saver is active; therefore if the suspend time specified here is lower than the screen saver delay time, the suspend will be postponed until the screensaver starts.

See also SWI Portable_SleepTime which allows the delay to be set in units of centiseconds.

[RISC OS Open source page](#)

*FS

This *Command selects the current file server, restoring that file server's context (for example, its current directory).

Syntax

*FS [[:]file_server]

Parameters

file_server · Name or number of file server

Notes

If no argument is supplied, information is given about your current file server, followed by any non-current servers.

[RISC OS Open source page](#)

*FSLock_ChangePassword

This *Command changes the locked filing system and password.

Syntax

*FSLock_ChangePassword FSname [New_Pswd [New_Pswd [Old_Pswd]]]

Parameters

FSname / Filing System name

New_Pswd – Filing System name: New file locking password

Old_Pswd – Filing System name: Current file locking password

Notes

The new password must be entered twice to ensure the value is correct.

[RISC OS Open source page](#)

*FSLock_Lock

This *Command locks the computer from the partially unlocked state.

Syntax

*FSLock_Lock

Parameters

None.

Notes

If the machine is fully locked, an error message will be returned.

[RISC OS Open source page](#)

*FSLock_Status

This *Command displays the device's current lock state.

Syntax

*FSLock_Status

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*FSLock_Unlock

This *Command unlocks the device.

Syntax

*FSLock_Unlock [-full_] [_password]

Parameters

-full / Fully unlock the device

password – Fully unlock the device: Current file locking password

Notes

If the -full switch is given, then the machine will be fully unlocked; otherwise a partial unlock will be done, lasting only until the next reset.

[RISC OS Open source page](#)

*FwAddNet

Freeway

This *Command extends Freeway's network by adding the given net (which may be beyond the subnet to which you are connected).

Syntax

*FwAddNet <CIDR address>

Parameters

CIDR address · Qualified IPv4 address to add

Notes

The CIDR address encodes both the net that is being added and the net mask needed to broadcast on that net.

[RISC OS Open source page](#)

*FwShow

Freeway

This *Command displays all currently known Freeway objects.

Syntax

*FwShow

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*FX

This *Command calls OS_Byte to alter status variables, and to perform other closely-related actions.

Syntax

*FX reason_code [[,] r1 [[,] r2]]

Parameters

reason_code / OS_Byte reason code (0 to 255)

r1 – OS_Byte reason code (0 to 255): Register value of R1

r2 – OS_Byte reason code (0 to 255): Register value of R2

Notes

The reason code determines which action to perform. See OS_Byte for the complete list.

[RISC OS Open source page](#)

*Go

This *Command calls machine code at the given address (default &8000), passing an optional environment string.

Syntax

*Go [hex_addr] [; command tail]

Parameters

hex_addr / Address of machine code to call

command tail – Address of machine code to call: Command tail to pass to machine code

Notes

If no parameter is passed, the default address of &8000 is used.

The “;” to set the command tail is optional and only required if no address is specified. The command name used is “GO”, so the environment becomes:

GO <command tail>

Code called this way will be entered in USER with the stack and R14 undefined, code is expected to return via OS_Exit and set up its own stack and environment handlers.

[RISC OS Open source page](#)

*GOS

This *Command enters command line mode, and hence allows you to type * Commands.

Syntax

*GOS

Parameters

None.

Notes

You must be careful calling this command from applications that do not ‘shell’ new applications. e.g. calling this command when writing a BASIC program will result in the loss of unsaved work.

To end a session, type *Quit.

[RISC OS Open source page](#)

*GPIODevices

GPIO

This *Command lists all of the known GPIO port devices in the computer, along with their current pin states.

In practice

For each device, the first row shows whether a pin is configured as an input or and output. The second row shows whether a pin is currently high or low.

Pins are shown from highest numbered to lowest.

Syntax

*GPIODevices

Parameters

None.

[RISC OS Open source page](#)

*Help

This *Command returns brief information about each command.

Syntax

*Help [keyword]

Parameters

keyword / Name of command to get help on

Special keywords

Commands – Name of command to get help on: Lists all the available utility commands

FileCommands – Name of command to get help on: Lists all the commands relating to filing systems

Modules – Name of command to get help on: Lists the names of all currently loaded modules, with their version numbers and creation dates

Station – Name of command to get help on: Displays the current network and station numbers of your machine

Syntax – Name of command to get help on: Explains the format used for syntax messages

Notes

When this command is issued at the normal Command Line prompt, a screen of information will be displayed at a time, and will wait until the shift key is pressed before displaying another screen of information (if any).

Abbreviations are allowed, so by typing *Help Con., will return information on both *Configure and *Continue.

[RISC OS Open source page](#)

*HOff

Hourglass

This *Command turns the hourglass off.

Syntax

*HOff

Notes

This command was added to the Hourglass module in version 2.15. Prior to that, it had been implemented as a utility located in the library folder of the universal boot structure.

Note that unlike SWI Hourglass_Off, this command is guaranteed to disable the hourglass, as it internally calls Hourglass_Smash.

[RISC OS Open source page](#)

*HOn

Hourglass

This *Command turns the hourglass on.

Syntax

*HOn

Notes

This command was added to the Hourglass module in version 2.15. Prior to that, it had been implemented as a utility located in the library folder of the universal boot structure.

[RISC OS Open source page](#)

*I am

This *Command selects NetFS and logs you on to a file server.

Syntax

*I am [[:]file_server_number [:file_server_name] user_name [[:Return]password]

Parameters

file_server_number / File server number to log on to

file_server_name – File server number to log on to: File server name to log on to

user_name – File server number to log on to: User name to log on to desired network

password – File server number to log on to: Password as used to log on to network

Notes

Your user name and password are checked by the file server against the password file, before allowing you access.

[RISC OS Open source page](#)

*IconSprites

WindowManager

This *Command merges the sprites in a file with those in the Wimp sprite area.

Syntax

*IconSprites filename

Parameters

filename · Valid name of sprite file

Notes

None.

[RISC OS Open source page](#)

*IF

This *Command executes a *Command conditionally, depending on the value of an expression.

Syntax

*IF expression Then command [Else command]

Parameters

expression / Integer expression

command – Integer expression: Valid *Command

Notes

For a comprehensive list of expressions, please see the CLI Expressions? page. This command only allows for integer expressions, including variable names (enclosed in angled brackets).

If a variable is to be compared to a string, both must be enclosed in double quotes.

[RISC OS Open source page](#)

*IfConfig

This *Command configures network interface parameters.

Syntax

*IfConfig [-e] interface [inet <__host__> [<__parameters__>]]

Parameters

-e · Place any fatal error report string in the system variable Inet\$error instead of writing it to the standard output

interface · Two character name (as defined by the manufacturer) and unit number (starting from 0 for the first of that type fitted) of an interface – e.g. en0 (Acorn Ethernet 2 unit 0), ea1 (Acorn Ethernet 3 unit 1), ec0 (Econet)

host · Internet host specified either by name (which must be present in the host name data base InetDBase\$path hosts) or by address (using the standard Internet dot notation)

parameters · Following parameters are available:

up · Mark an interface 'up'. This may be used to enable an interface after going IfConfig - down. It happens automatically when setting the first address on an interface. If the interface was reset when previously marked down, the hardware will be re-initialised

down · Mark an interface 'down'. When an interface is marked 'down', the system will not attempt to transmit messages through that interface. If possible, the interface will be reset to disable reception as well. This action does not automatically disable routes using the interface

arp · Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default), for mapping between Internet addresses and Ethernet addresses

-arp · Disable the use of the Address Resolution Protocol

metric <n> · Set the routing metric of the interface to n (default 0). The routing metric is used by the routing protocol. Higher metrics have the effect of making a route less favourable; metrics are counted as additional hops to the destination network or host

Netmask <mask> · Specify how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, as an Internet address using the standard dot notation, or as a pseudo-network name listed in the network table Inet\$Dbase networks. The mask contains 1's for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion

broadcast · Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 1's

Notes

The IfConfig program is used to assign an Internet address to a network interface supported by the Internet module, and/or to configure network interface parameters. It must be used at Internet module startup time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address or other operating parameters. If you don't use any of the optional parameters, the IfConfig program displays the current configuration for a network interface.

[RISC OS Open source page](#)

*IfThere

BootCommands

This *Command checks for the presence of a given object, and perform one of two possible commands depending on the result of the check.

Syntax

```
*IfThere object_spec Then true_cmd [Else false_cmd]
```

Parameters

object_spec / Valid (wildcarded) pathname to a file or directory

true_cmd – Valid (wildcarded) pathname to a file or directory: Command to perform if check is true

false_cmd – Valid (wildcarded) pathname to a file or directory: Command to perform if check is false

Notes

This command uses OS_File 17 to perform the check.

[RISC OS Open source page](#)

*Ignore

This *Command sets the printer ignore character.

Syntax

*Ignore [ASCII_code]

Parameters

ASCII_code · ASCII code (0 - 255)

Notes

The default value is 10 (ASCII linefeed). A value of 0, sets the ignore character to ASCII code 0. No parameter passed, the NoIgnore state is set, so that all characters are printed.

The character set by this command is filtered out when using OS_Byte 5. OS_Byte 6 performs the exact same function as this command, while OS_Byte 246 reads and writes the printer ignore character. OS_Byte 182 controls the NoIgnore state.

[RISC OS Open source page](#)

*IICDevices

IIC

This *Command scans the indicated IIC bus and lists the devices that are detected.

Syntax

*IICDevices [bus]

Parameters

bus · Bus number (zero-based)

Notes

If no bus number is provided, all buses will be scanned.

[RISC OS Open source page](#)

*InetGateway

This *Command enables or disables IP layer packet forwarding (i.e. gateway operation) if multiple network interfaces are present.

Syntax

*InetGateway [On|Off]

Parameters

On / Enable IP layer packet forwarding

Off – Enable IP layer packet forwarding: Disable IP layer packet forwarding

Notes

The default state is off.

[RISC OS Open source page](#)

*InetInfo

This *Command displays information and statistics about the current state of the Internet module, which forms a part of the AUN software.

Syntax

*InetInfo [r] [i] [p]

Parameters

r / Return information of internal resource usage

i – Return information of internal resource usage: Return information of interface installed

p – Return information of internal resource usage: Return information of protocols

Notes

Most of the information displayed is runic in nature. It is presented mainly as an aid to trouble-shooting, should you require it.

[RISC OS Open source page](#)

*Info

This *Command returns file information for the specified object(s).

Syntax

*Info object_spec

Parameters

object_spec · Valid (wildcarded) pathname specifying one or more file or directories

Notes

None.

[RISC OS Open source page](#)

*InitStore

Debugger

This *Command fills user memory with the specified value.

Syntax

*InitStore [value | reg]

Parameters

value / Word to fill in user memory

reg – Word to fill in user memory: Register value to fill in user memory

Notes

If no parameter is specified then the value &E6000010 (which is an illegal instruction) is stored.

[RISC OS Open source page](#)

*Install_CheckSystem

Installer

This *Command ensures that versions of modules present in OS-version subdirectories of <System\$Dir> are not older than those in previous-version subdirectories, or the "root" Modules subdirectory.

Syntax

*Install_CheckSystem

Parameters

None.

Notes

Any that are removed in a manner consistent with *Install_Remove.

[RISC OS Open source page](#)

*Install_DeMerge

Installer

This *Command removes any entries from base_file that match the section, company and application of an entry in changes_file, unless the version number of the entry in base_file is newer than the one in changes_file.

Syntax

*Install_DeMerge base_file changes_file dest_file

Parameters

base_file / the "base" source file

changes_file – the "base" source file: the "changes" source file

dest_file – the "base" source file: the destination file

Notes

The result of this command is placed in dest_file.

If dest_file is to be overwritten or deleted, it is first copied into the backup directory. Details of the operation are appended to the log file.

[RISC OS Open source page](#)

*Install_Log

Installer

This *Command appends a free form line of text to the log file whose location was set with *Install_LogDir previously.

Syntax

*Install_Log [string]

Parameters

string · line of text to append

Notes

The text is processed similarly to *Echo in that characters are first passed through GStrans, though in order to ensure the log file remains purely a text log any unprintable control characters are converted to a period.

[RISC OS Open source page](#)

*Install_LogDir

Installer

This *Command defines the directory into which Installer will write the log file Log and build the backup tree.

Syntax

*Install_LogDir [directory]

Parameters

directory · the new log directory

Notes

Backed-up files will have the same path relative to directory.Backup as they used to have relative to directory. This means that directory must be higher up the hierarchy than any file to be installed or deinstalled, otherwise it cannot be backed up.

The default is for the log directory to be undefined, so this command will typically be used at the beginning of an installation script.

If no argument is given, the current log directory is printed.

[RISC OS Open source page](#)

*Install_Merge

Installer

This *Command combines the entries of two obeyfiles (`base_file` and `changes_file`), placing the result in another file.

Syntax

*Install_Merge `base_file` `changes_file` `dest_file`

Parameters

`base_file` / the "base" source file

changes_file – the "base" source file: the "changes" source file

dest_file – the "base" source file: the destination file

Notes

The `dest_file` may be the same as one of the source files.

The `changes_file` may optionally contain a set of rules for the positioning of new entry sections, companys or applications. These consist of any number of special one-line entries, starting |Section, |Company or |App, as appropriate. Each of these lines has the following format:

```
[[Section|Company|App] rule [rule [rule ...]]]
```

rule is either:

- specifier>specifier meaning create after in file
- specifier<specifier meaning create before in file

The lack of spaces is significant, and must be adhered.

specifier is either:

- case-insensitive single-word section, company or application
- * to refer to: any specifier (when used on the left) all specifiers (when used on the right)

Rules are used from the first appropriate entry in the file to the last, and left-to-right within each line.

Entries from `changes_file` will be merged one by one into `base_file`. This will be achieved thus:

- Scan `base_file` for the first entry with a section matching that of the new entry. If there are none, scan the command line for the first applicable rule. If no rule applies, *>* defaults
- Scan all consecutive entries in `base_file` with the same section for the first one with a company matching that of the new entry. If there are none, scan the command line for the first applicable rule. If no rule applies, *>* defaults
- Scan all consecutive entries in `base_file` with the same section and company for the one that also matches the application in the new entry. If there is none, scan the command line for the first applicable rule. If no rule applies, *>* defaults. If there already was such an entry, compare the version number; if the new entry is a matching or earlier version, use the entry from `base_file`, otherwise use the entry from `changes_file`.

If `dest_file` is to be overwritten, it is first copied into the backup directory. Details of the operation are appended to the log file; each successful entry update is listed separately (along with new version number and any applicable old version number).

Examples

Example 1:

The "Add to Apps" Boot window will use this command: `*Install_Merge <Choices$Write>.Boot.PreDesktop tempfile <Choices$Write>.Boot.PreDesktop`

where tempfile contains the line: `|App Configure>*`

The section ResApps and company Acorn can be assumed to already exist (they are part of the default PreDesktop file). However, all other Acorn ResApps entries should come first.

Example 2:

The "Run" Boot window will use this command: `*Install_Merge <Choices$Write>.Boot.Desktop tempfile <Choices$Write>.Boot.Desktop`

where tempfile contains the line: `|Section Run>Boot Run>Auto`

The new section Run should be created after Boot, or after Auto ("Auto tasks") if Boot doesn't yet exist. Positioning of the Acorn Configure entry within the Run section is not expected to be critical.

Example 3:

The nested Wimp installer, were it to be re-written to use this Installer module, would use: `*Install_Merge <Choices$Write>.Boot.PreDesktop changesfile <Choices$Write>.Boot.PreDesktop`

where changesfile contains the lines:

```
|Section Installation>Comments |Company Acorn<*
```

Installation may be a new section on some machines. Acorn should be the first company within the installation section. Since this is, at the time of writing, the first entry in Acorn Installation, we'll leave ordering of applications to later installers.

In the event that another entry is written at a later date, which needs to be added to PreDesktop after the Wimp installation entry, a clever approach needs to be taken (because there's a chance the Wimp will be installed at a later time, and because of the default rule `>`, it would be added afterwards in PreDesktop).

The new installer should first attempt to install an empty entry called Acorn Wimp 0.00 Installation, then install its own entry after the Wimp one. If a genuine Wimp entry already exists, the 0.00 entry is simply ignored. If the Wimp installer were only to be run later, the dummy entry's position would override the (default) `*>*` application rule in the above example.

Because of this, non-empty entries must, in every case, always have version numbers of at least 0.01.

[RISC OS Open source page](#)

*Install_NoLogDir

Installer

This *Command undefines the log directory, and would typically be used at the end of an installation script.

Syntax

```
*Install_NoLogDir
```

Parameters

None.

Notes

When there is no defined log directory, the log file data is written to the screen instead.

[RISC OS Open source page](#)

*Install_Remove

Installer

This *Command moves the specified file to its backup directory, and details of the operation are appended to the log file.

Syntax

*Install_Remove source_file

Parameters

source_file · the file to be moved to the backup location

Notes

Any file of the same name already in the backup directory is first deleted.

In the case of modules, the version number multiplied by 100 is appended to the name of the file in the backup directory, and the version number is mentioned in the log file.

In the case of fonts, the Messages files in the first fonts directory above source_file are updated to reflect the removal of the font.

[RISC OS Open source page](#)

*Install_Update

Installer

This *Command copies a file if source_file is newer than dest_file.

Syntax

*Install_Remove source_file dest_file

Parameters

source_file / the file to be copied

dest_file – the file to be copied: the file to be replaced

Notes

Any overwritten file is first copied into the backup directory, and details of the operation are appended to the log file.

In the case of modules, the version number multiplied by 100 is appended to the name of the file in the backup directory, and the old and new version numbers are mentioned in the log file.

In the case of fonts, the Messages files in the first application directory above dest_file are updated to include any new font.

The comparative newness of files is determined by the following steps:

- if both files are modules, the version number of the modules (or if they match, the timestamps of the files);
- otherwise, a simple timestamp comparison is used; in certain special cases, the timestamp is determined by a special method: untyped files are assigned a timestamp of the beginning of time; fonts are assigned the most recent timestamp of the files in the font directory (non-recursively); directories are assigned the most recent timestamp of the files in the directory (recursively); voids (where there is currently nothing) are checked to see if they are located below a file, rather than a directory; if the former, then the file's timestamp is used.

[RISC OS Open source page](#)

*Key

This *Command assigns a string to a Function Key.

Syntax

*Key keynumber [string]

Parameters

keynumber / Number (0 – 15)

string – Number (0 – 15): String that is GStrans compatible

Notes

Before a value is assigned, it is first transformed using OS_GStrans.

[RISC OS Open source page](#)

*Keyboard

This *Command selects the appropriate keyboard layout for a given country.

Syntax

*Keyboard [country_name]

Parameters

country_name · Valid country name. A list of countries can be obtained with *Countries

Notes

With no parameter, this command displays the currently selected keyboard layout.

[RISC OS Open source page](#)

*LanMan

*LanManFS

This *Command selects LanManFS (Local area network Manager Filing System) as the current filing system.

Syntax

*LanManFS

Parameters

None.

[RISC OS Open source page](#)

*LCat

This *Command lists all the objects in the named library subdirectory.

Syntax

*LCat [directory]

Parameters

directory · Valid pathname specifying a subdirectory of the current library

Notes

If no subdirectory is given, then the current library is used.

[RISC OS Open source page](#)

*LEx

This *Command lists all the objects in the named library subdirectory, including their file information.

Syntax

*LEx [directory]

Parameters

directory · Valid pathname specifying a subdirectory of the current library

Notes

If no subdirectory is given, then the current library is used.

[RISC OS Open source page](#)

*Lib

This *Command selects a directory (the default is filing system dependent) as the current library on the filing system.

Syntax

*Lib [directory]

Parameters

directory · Valid pathname specifying a directory

Notes

If no directory is given, then the action taken will depend on the Filing System is currently open.

[RISC OS Open source page](#)

*List

This *Command displays the contents of the named files (as per *Configure DumpFormat).

Syntax

*List [-File] filename [-TabExpand]

Parameters

-File / Optional parameter to precede filename (has no effect)

filename – Optional parameter to precede filename (has no effect): Valid pathname specifying a file

-TabExpand – Optional parameter to precede filename (has no effect): Tab characters to be expanded to 8 spaces

Notes

Each line is numbered.

[RISC OS Open source page](#)

*ListFS

This *Command displays a list of the file servers which NetFS is able to recognise.

Syntax

*ListFS [-Force]

Parameters

-Force · Force list to update before it is displayed

Notes

The optional argument forces the list to be updated before it is displayed.

[RISC OS Open source page](#)

*ListPS

This *Command displays a list all the currently available printer servers, optionally showing their status as well.

Syntax

*ListPS [-Full]

Parameters

-Full · Show status of each printer server

Notes

The list of available printer servers returned is based on the order in which they reply to this command.

[RISC OS Open source page](#)

*Load

This *Command loads the named file.

Syntax

*Load filename [load_addr]

Parameters

filename / Valid pathname specifying a file

load_addr – Valid pathname specifying a file: Specifies a file's load address

Notes

The optional load address is in hexadecimal.

If no load address is specified, then the filetype is checked:

- If there is no filetype, it is loaded at its own load address
- For any specified filetype, the corresponding Alias\$@LoadType_nnn variable is looked up to determine how the file should be loaded. If there is no corresponding Alias\$@LoadType_nnn variable, then an error is generated.

[RISC OS Open source page](#)

*LoadCMOS

BootCommands

This *Command loads a file into the computer's CMOS RAM, preserving only the network station number, the current year and the DST flag.

Syntax

*LoadCMOS filename

Parameters

filename · Valid pathname to a file

Notes

All other configured values are replaced by those stored in the file.

[RISC OS Open source page](#)

*LoadFontCache

FontManager

This *Command loads a file that was previously saved using *SaveFontCache back into the font cache.

Syntax

*LoadFontCache filename

Parameters

filename · Valid pathname specifying a file

Notes

This command is for internal use only, and must not be used.

[RISC OS Open source page](#)

*LoadModeFile

ScreenModes

This *Command loads a Monitor Definition File or EDID file into memory.

Syntax

*LoadModeFile filename

Parameters

filename · Valid pathname to a file

Notes

This call makes all the screen modes defined in the file available for use, while removing all modes defined in any previously loaded file.

Support for reading EDID files was introduced by ScreenModes 0.39.

Currently EDID files are detected by looking for a valid EDID block 0 header at the start of the file, rather than making the decision based on the file's filetype.

[RISC OS Open source page](#)

*Lock

This *Command locks the disc in the given CD-ROM drive, disabling the Eject button.

Syntax

*Lock [drive]

Parameters

drive · Valid CD/DVD drive number

Notes

If no drive parameter is passed, then the current drive is locked.

[RISC OS Open source page](#)

*Logon

This *Command logs on to a given file server, if username and password are both valid.

Syntax

*Logon [[:]file_server_number [:file_server_name] user_name [[:Return]password]

Parameters

file_server_number / File server number to log on to

file_server_name – File server number to log on to: File server name to log on to

user_name – File server number to log on to: User name to log on to desired network

password – File server number to log on to: Password as used to log on to network

[RISC OS Open source page](#)

*Map

This *Command displays a disc's free space map.

Syntax

*Map [disc_spec]

Parameters

disc_spec · Name of disc or number of disc drive

Notes

If no disc is specified, the operation is performed on the current disc.

[RISC OS Open source page](#)

*Memory

Debugger

This *Command displays the values in memory, grouped into 4-byte words unless specified otherwise.

Syntax

*Memory [P] [B|H|D] addr1_reg1 *Memory [P] [B|H|D] addr1_reg1 [+|-]addr2_reg2 *Memory [P] [B|H|D] addr1_reg1
+|-addr2_reg2 +addr3_reg3

Parameters

P / Specifies a physical rather than logical address (optional)

B|H|D – Specifies a physical rather than logical address (optional): Display bytes, halfwords or double words (optional)

addr1_reg1 – Specifies a physical rather than logical address (optional): Address or register containing start of display (hexadecimal)

addr2_reg2 – Specifies a physical rather than logical address (optional): Address or register containing offset (hexadecimal)

addr3_reg3 – Specifies a physical rather than logical address (optional): Address or register containing offset (hexadecimal)

Notes

If only the start address is specified, then 256 bytes will be returned starting from this address.

If two addresses are specified, then they specify the start and end addresses (unless specifying an offset by use of + or - symbols).

If three addresses are specified, then the second address is an offset from the start address, and the third address specifies the end address.

By default, the output is grouped into 4-byte words. This grouping can be changed by using the B, H or D flags (see above). Currently, halfword and doubleword access is only possible on CPUs that support the required instructions (LDRH, LDRD).

On kernels that support OS_Memory 14 and OS_Memory 15, the P flag causes physical memory to be mapped in temporarily (first flushing the cache to avoid inaccurate data being read). Otherwise, the system is assumed to be IOMD-based, where the 512MB of physical address space is mapped in at &80000000.

[RISC OS Open source page](#)

*MemoryA

Debugger

This *Command displays and alters the values in memory, grouped into 4-byte words unless specified otherwise.

Syntax

*MemoryA [P] [B|H|D] addr_1_reg1 [value_1_reg2]

Parameters

P / Specifies a physical rather than logical address (optional)

B|H|D – Specifies a physical rather than logical address (optional): Display bytes, halfwords or double words (optional)

addr_1_reg1 – Specifies a physical rather than logical address (optional): Address or register containing start of display (hexadecimal)

value – Specifies a physical rather than logical address (optional): Value to write into specified memory location (hexadecimal)

reg2 – Specifies a physical rather than logical address (optional): Register containing value to write into specified memory location (hexadecimal)

Notes

None.

[RISC OS Open source page](#)

*MemoryI

Debugger

This *Command disassembles memory into ARM or Thumb instructions.

Syntax

*MemoryI [P] [T] addr_1_reg1 *MemoryI [P] [T] addr_1_reg1 [+|-]addr2_reg2 *MemoryI [P] [T] addr_1_reg1 [+|-]addr2_reg2 +addr3_reg3

Parameters

P / Specifies a physical rather than logical address (optional)

T – Specifies a physical rather than logical address (optional): Indicates that disassembly should be produced for the Thumb instruction set, not ARM

addr1_ | reg1 – Specifies a physical rather than logical address (optional): Address or register containing start of display (hexadecimal)

addr2_ | reg2 – Specifies a physical rather than logical address (optional): Address or register containing offset (hexadecimal)

addr3_ | reg3 – Specifies a physical rather than logical address (optional): Address or register containing offset (hexadecimal)

Notes

If only the start address is specified, then 24 instructions will be disassembled starting from this address.

If two addresses are specified, then it specifies the start and end address (unless specifying an offset by use of + or - symbols).

If three addresses are specified, then the second address is an offset from the start address, and the third address specifies the end address.

On kernels that support OS_Memory 14 and OS_Memory 15, the P flag causes physical memory to be mapped in temporarily (first flushing the cache to avoid inaccurate data being read). Otherwise, the system is assumed to be IOMD-based, where the 512MB of physical address space is mapped in at &80000000.

[RISC OS Open source page](#)

*MimeMap

MimeMap

This *Command displays the line from the MimeMap for the corresponding MIME type.

Syntax

*MimeMap [&xxx | .ext | mime/type | Filetype]

Parameters

&xxx / RISC OS 12 bit file type

Filetype – RISC OS 12 bit file type: RISC OS file type name

.ext – RISC OS 12 bit file type: DOS extension

mime/type – RISC OS 12 bit file type: IANA MIME type

Notes

There may be more than one DOS extension that maps to a given RISC OS file type.

[RISC OS Open source page](#)

*MixVolume

SoundControl

This *Command is used to alter the volume settings of a specific mixer channel.

Syntax

*MixVolume system category index mute gain

Parameters

system / Mixer system/device

category – Mixer system/device: Category number

index – Mixer system/device: Index within category

mute – Mixer system/device: 1 = mute, 0 = unmute

gain – Mixer system/device: Gain in units of 1/16dB

Notes

The system parameter can either be the ID string of the audio controller device that the mixer is associated with (see `Sound_EnumerateControllers`), or the number 0 to specify the mixer that's associated with the current default audio controller.

Gain is specified in units of 1/16dB, so for example -16 corresponds to -1dB.

[RISC OS Open source page](#)

*ModelList

This *Command lists all the loaded colour models.

Syntax

*ModelList

Parameters

None.

[RISC OS Open source page](#)

*Modules

This *Command displays information about all installed relocatable modules.

Syntax

*Modules

Parameters

None.

Notes

The information returned includes the number allocated to each module, its position in memory, the address of its workspace, and its name.

[RISC OS Open source page](#)

*Mount

This *Command prepares a disc for general use.

Syntax

*Mount [disc_spec] *Mount [:]disc_spec (netFS syntax)

Parameters

disc_spec · Name of disc or number of disc drive

Notes

A disc is prepared for general use by the current directory to its root directory, setting the library directory (if it is currently unset) to \$.Library, and unsetting the User Root Directory.

For NetFS, Mount selects a disc from the file server by setting the current directory, the library directory and the User Root Directory.

This command performs exactly the same function as *SDisc.

[RISC OS Open source page](#)

*NameDisc

This *Command changes a disc's name.

Syntax

*NameDisc disc_spec new_name

Parameters

disc_spec / Name of disc or number of disc drive

new_name – Name of disc or number of disc drive: Name of new disc (up to 10 characters in length)

Notes

*NameDisk command provides exactly the same functionality.

[RISC OS Open source page](#)

*NameDisk

This *Command changes a disc's name.

Syntax

*NameDisk disc_spec new_name

Parameters

disc_spec / Name of disc or number of disc drive

new_name – Name of disc or number of disc drive: Name of new disc (up to 10 characters in length)

Notes

*NameDisc command provides exactly the same functionality.

[RISC OS Open source page](#)

*Net

This *Command selects the Network Filing System as the current filing system.

Syntax

*Net

Parameters

None.

[RISC OS Open source page](#)

*NetMap

This *Command displays the AUN map table for a specific network.

Syntax

*NetMap [net_number]

Parameters

net_number · Number of network

Notes

If no network number is supplied, then information relating to all network will be returned.

The map table shows the net number of each net, its name, and its Internet address.

[RISC OS Open source page](#)

*NetProbe

This *Command reports if a remote station is accessible and active, and hence can be reached from the local station and network.

Syntax

*NetProbe net_number_.station_number

Parameters

net_number / Remote station 'net' number

station_number – Remote station 'net' number: Remote station 'station' number

Notes

None.

[RISC OS Open source page](#)

*NetStat

This *Command displays the current status of any network interface(s) configured for AUN.

Syntax

*NetStat [-a]

Parameters

-a · Return all information, rather than limited amount

Notes

The optional parameter -a gives extra information, including traffic counters and full IP addresses.

[RISC OS Open source page](#)

*Networks

This *Command displays the current AUN routing table.

Syntax

*Networks

Parameters

None.

Notes

This shows the names of any local networks; and also the names of those remote networks that the station knows how to reach, and the gateway that it will use to do so.

[RISC OS Open source page](#)

*NoDir

This *Command unsets the current directory.

Syntax

*NoDir

Parameters

none.

Notes

It selects a directory (by default the user root directory) as the current directory on a filing system.

[RISC OS Open source page](#)

*NoLib

This *Command unsets the library directory.

Syntax

*NoLib

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*NoURD

This *Command unsets the User Root Directory (URD).

Syntax

*NoURD

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*Obey

Obey

This *Command executes a file of *commands.

Syntax

*Obey [[-v][-c][-m] [filename [parameters]]]

Parameters

-v / Echo each line before it is executed

-c – Echo each line before it is executed: Cache filename, and then execute it

-m – Echo each line before it is executed: lines to be read from specified memory address
filename – Echo each line before it is executed: Valid pathname specifying a file
parameters – Echo each line before it is executed: String to be passed (separated by spaces)

Notes

*Obey with no parameters terminates the current and any parent obey files. The same holds for *Error.

Arguments substitution is performed on each line, using parameters passed in the command.

If using the memory option, the file in memory must be zero-terminated.

In certain circumstances the command will not return, for example if the Obeyfile starts a new application, or *Quit is used.

Note that a *Quit line will not terminate the file.

[RISC OS Open source page](#)

*Opt 1

This *Command sets the filing system message level (for operations involving loading, saving or creating a file) for the current filing system.

Syntax

*Opt 1 [[,]n]

Parameters

n / Message level (0 - 3)

0 = Filename printed

1 = Filename, hexadecimal addresses and length printed

2 =Filename, and either date stamp and length, or hexadecimal

3 = Load and exec addresses printed

Notes

This command is Filing System specific

[RISC OS Open source page](#)

*Opt 4

This *Command sets the boot action for the current filing system.

Syntax

*Opt 4 [[,]n]

Parameters

n / Boot option (0 - 3)

0 = No boot action

1 = *Load boot file

2 = *Run boot file

3 = *Exec boot file

Notes

The *Exec setting will override the configured language setting. If you wish to enter the Desktop environment after executing a file, then the file should end with the *Desktop command.

[RISC OS Open source page](#)

*Pass

This *Command changes the password on the current fileserver.

Syntax

*Pass [old_password [new_password]]

Parameters

old_password / Existing password (if any)

new_password – Existing password (if any): New password (if any)

Notes

The maximum password length is dependent on the file server.

[RISC OS Open source page](#)

*Path

This *Command provides a convenient way to set a path variable, so that reference to a full pathname can be made using the shorthand 'path:'.

Syntax

*Path path full_path

Parameters

path / New path variable

full_path – New path variable: Full pathname of file, application or directory

Notes

None.

[RISC OS Open source page](#)

*PathMacro

This *Command provides a convenient way to set a path variable as a macro, so that a full pathname can be referenced using the shorthand 'path:'.

Syntax

*PathMacro path full_path

Parameters

path / New path macro name

full_path – New path macro name: Full pathname of file, application or directory

Notes

None.

[RISC OS Open source page](#)

*PCIDevices

PCI

This *Command lists all of the known PCI devices in the computer.

Syntax

*PCIDevices

Parameters

None.

[RISC OS Open source page](#)

*PCIInfo

PCI

This *Command gives detailed information about a given PCI function.

Syntax

*PCIInfo <function>

Parameters

function · PCI function number determined from *PCIDevices

Notes

There can be several functions per PCI peripheral, so the total number of PCI functions on which information can be obtained can exceed the number of PCI cards fitted.

[RISC OS Open source page](#)

*Pin

Pinboard

This *Command adds a file, application or directory to the desktop pinboard, positioning its icon at the given coordinates (in OS units).

Syntax

*Pin object x y

Parameters

object / Valid pathname specifying a file, application or directory

x – Valid pathname specifying a file, application or directory: X co-ordinate (in OS units)

y – Valid pathname specifying a file, application or directory: Y co-ordinate (in OS units)

Notes

None.

[RISC OS Open source page](#)

*Pinboard

Pinboard

This *Command initialises the pinboard, removing any existing pinned icons and backdrop.

Syntax

*Pinboard [-Grid]

Parameters

-Grid · Turn on grid locking

Notes

The grid locking mechanism is turned-off by default.

[RISC OS Open source page](#)

*PinboardOptions

Pinboard

This *Command allows the user to set the options used by the Pinboard.

Syntax

*PinboardOptions [switches]

Switches

Switch / Meaning if switch provided / Meaning if switch not provided

-Grid – Meaning if switch provided: Turn on grid lock · Meaning if switch not provided: Turn off grid lock

-IconiseTo<Location> – Meaning if switch provided: Windows are iconised and tidied to: IconBar, TopLeft, BottomLeft, TopRight, BottomRight · Meaning if switch not provided: Iconise to location of Pointer

-IconiseStackVertical – Meaning if switch provided: Vertical stacking of iconised window icons · Meaning if switch not provided: Icons are stacked horizontally

-TidyTo<Location> – Meaning if switch provided: File icons are tidied to: TopLeft, BottomLeft, TopRight, BottomRight · Meaning if switch not provided: File icons are tidied to TopLeft location

-TidyStackVertical – Meaning if switch provided: Vertical stacking of file icons · Meaning if switch not provided: Horizontal stacking of file icons

Notes

None.

[RISC OS Open source page](#)

*Ping

This *Command sends ICMP ECHO_REQUEST packets to network hosts.

Syntax

*Ping [-r] [-v]<_host_> [<_net_>] [<_count_>]

Parameters

-r · Bypass the normal routing tables and send directly to a host on an attached network. (If the host is not on a directly attached network an error is returned.) You can use this option to 'ping' a local host through an interface that has no route through it

-v · Display verbose output: any ICMP packets other than ECHO RESPONSE that are received are listed

host · Internet host specified either by name (which must be present in the host name data base <InetDBase\$Path>hosts) or by address (using the standard Internet dot notation)

packetsize · Size of packet to send (default is 64 bytes)

count · Number of packets to send (default is 1)

Notes

A network, particularly a Wide Area Network, can be a large and complex aggregation of network hardware connected together by gateways. Tracking a single-point hardware or software failure can be difficult. The Ping program utilises the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams (or pings) have an IP and ICMP header, and then an arbitrary number of padding bytes used to fill out the packet. When using Ping for fault isolation you should first run it on your local host, to verify that your local network interface is up and running. Then you should 'ping' hosts and gateways that are further and further away. If you don't give a count the Ping program sends just one datagram and expects a single ECHO_RESPONSE to be returned. If you give a count Ping sends one datagram per second up to that number, and prints one line of output for every ECHO_RESPONSE returned. No output is produced if there is no response. Packet loss statistics are computed and a brief summary is displayed when all responses have been received, or when the program times out, or when you terminate the program. RISC OS Ping messages are not time-stamped.

[RISC OS Open source page](#)

*PipeCopy

This *Command copies a file one byte at a time to one or two output files.

Syntax

*PipeCopy source_file destination_file1 [destination_file2]

Parameters

source_file / Valid pathname specifying a file

destination_file1 – Valid pathname specifying a file: Valid pathname specifying destination file

destination_file2 – Valid pathname specifying a file: Valid pathname specifying optional second destination file

Notes

None.

[RISC OS Open source page](#)

*Play

This *Command plays from the specified audio track to the end of the disc in the given CD_ROM drive.

Syntax

*Play track [drive]

Parameters

track / Track from which to start playing (0- 99)

drive – Track from which to start playing (0- 99): Valid CD/DVD drive number

Notes

If no drive parameter is passed, then the current drive is used.

[RISC OS Open source page](#)

*PlayList

This *Command lists the tracks whether audio or data on the disc in the given CD-ROM drive, together with their start time and the total CD time.

Syntax

*PlayList [drive]

Parameters

drive · Valid CD/DVD drive number

Notes

If no drive parameter is passed, then the current drive is used.

[RISC OS Open source page](#)

*PlayMSF

This *Command plays a piece of audio from the disc in the given CD-ROM drive.

Syntax

*PlayMSF mins_:secs_:frames mins_:secs_:frames [drive]

Parameters

mins / Number of minutes from the start

secs – Number of minutes from the start: Number of seconds from the start

frames – Number of minutes from the start: Number of frames from the start

drive – Number of minutes from the start: Valid CD/DVD drive number

Notes

If no drive parameter is passed, then the current drive is used.

The start and stop times are specified in minutes, seconds and frames from the start of the disc.

[RISC OS Open source page](#)

*PoduleLoad

This *Command copies a file into an expansion card's RAM.

Syntax

*PoduleLoad expansion_card_number filename [offset]

Parameters

expansion_card_number / Card number as returned by *Podules

filename – Card number as returned by *Podules: Valid pathname specifying a file

offset – Card number as returned by *Podules: Offset into code space (hexadecimal by default)

Notes

None.

[RISC OS Open source page](#)

*Podules

This *Command displays a list of the installed expansion cards and extension ROMs.

Syntax

*Podules

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*PoduleSave

This *Command copies an expansion card's ROM into a file.

Syntax

*PoduleSave expansion_card_number filename [offset]

Parameters

expansion_card_number / Card number as returned by *Podules

filename – Card number as returned by *Podules: Valid pathname specifying a file

offset – Card number as returned by *Podules: Offset into code space (hexadecimal by default)

Notes

None.

[RISC OS Open source page](#)

*Pointer

This *Command turns the mouse pointer on or off.

Syntax

*Pointer [0|1]

Parameters

0 or 1 · Turn mouse pointer off or on

Notes

None.

[RISC OS Open source page](#)

*PowerOff

This *Command turns off the computer without further prompting.

Syntax

*PowerOff

Parameters

None.

Notes

If the computer doesn't support software control of the power supply, this command does nothing and will return without raising an error.

[RISC OS Open source page](#)

*PrepPath

BootCommands

This *Command prepends a path element to a path variable, removing any earlier occurrences of the path if present.

Syntax

*PrepPath variable element

Parameters

variable / A system variable to prepend to

element – A system variable to prepend to: The path to prepend

[RISC OS Open source page](#)

*Print

This *Command displays the contents of the named file by sending each byte to the VDU.

Syntax

*Print filename

Parameters

filename · Valid pathname specifying a file

Notes

All characters codes are sent to the VDU, and hence it is possible for some strange visual effects to occur. No characters codes are filtered out.

[RISC OS Open source page](#)

*PS

This *Command changes the default printer server, checking that the new one exists.

Syntax

*PS printer_server

Parameters

printer_server · Name or number of printer server

Notes

If no argument is supplied, then the current printer server and its status is returned.

[RISC OS Open source page](#)

*QSound

This *Command generates a sound after a given delay.

Syntax

*QSound channel amplitude pitch duration beats

Parameters

channel / Channel number (1 – 8)

amplitude – Channel number (1 – 8): Linear scale: 0 silent – &FFF1 loud, or;

Logarithmic scale: &100 silent – &17F loud (a change of 16 will halve/double the amplitude)

pitch – Channel number (1 – 8): Pitch: 0 – 255. Each unit represents a quarter of a semitone. Value of 53 is middle C, or 256 to 32767 (&100 to &7FF) where top 3 bits specify octave, and bottom 12 bits specify fraction of octave. Value of 16384 (&4000) is middle C

duration – Channel number (1 – 8): Duration: 0 to 32767 (&0 – &7FFF) in twentieths of a second. Value of 255 gives a note of infinite duration

beats – Channel number (1 – 8): Beats delay before sound is generated. Based on the tempo set by *Tempo

Notes

To generate a sound after without delay use *Sound.

[RISC OS Open source page](#)

*Quit

This *Command causes exit from the current application.

Syntax

*Quit

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*RAM

This *Command selects the RAM Filing System as the current filing system.

Syntax

*RAM

Parameters

None.

[RISC OS Open source page](#)

*ReadMimeMap

MimeMap

This *Command (re)reads the MimeMap file held in the InetDBase which is used to convert to and from RISC OS filetypes.

Syntax

*ReadMimeMap

Parameters

None.

[RISC OS Open source page](#)

*Remove

This *Command erases a single named file or empty directory.

Syntax

*Remove object_spec

Parameters

object_spec · Valid (wildcarded) pathname specifying a file or empty directory

Notes

No error message is given if the object does not exist.

Wildcards must not be used in the last component of the pathname. i.e. the name of the file or directory.

[RISC OS Open source page](#)

*RemoveTinyDir

Pinboard

This *Command removes the icon of a file, application or directory that was previously placed on the icon bar by the *AddTinyDir command.

Syntax

*RemoveTinyDir [object]

Parameters

object - Valid pathname specifying a file, application or directory

Notes

If no pathname is given, all such icons are removed from the icon bar.

[RISC OS Open source page](#)

*RemPath

BootCommands

This *Command removes all occurrences of a path element from a path variable.

Syntax

*RemPath variable element

Parameters

variable / A system variable to search

element – A system variable to search: The path to find and remove

[RISC OS Open source page](#)

*Rename

This *Command changes the name of an object, within the same storage unit.

Syntax

*Rename object new_name

Parameters

object / Valid pathname specifying a file or directory

new_name – Valid pathname specifying a file or directory: Valid pathname specifying a file or directory

Notes

To move objects between discs or filing systems, use the *Copy command with the D(elete) option set.

Locked objects cannot be renamed. They must first be renamed by using *Access command.

[RISC OS Open source page](#)

*Render

The his *Command displays the contents of a Draw file

Syntax

*Render [-file] filename [m00 m01 m10 m11 m20 m21] [-bbox] [-suppress] [-flatness flatness]

Parameters

-file filename / Valid pathname specifying a Draw file to be rendered

m00,m01,m10,m11 – Valid pathname specifying a Draw file to be rendered: Four decimal numbers specifying the first four elements of the transformation matrix

m20,m21 – Valid pathname specifying a Draw file to be rendered: Two decimal numbers giving the other two elements of the transformation matrix

-bbox – Valid pathname specifying a Draw file to be rendered: Render bounding boxes around objects as dotted lines

-suppress – Valid pathname specifying a Draw file to be rendered: Suppress rendering of objects

-flatness flatness – Valid pathname specifying a Draw file to be rendered: Decimal number specifying flatness used to render curved paths (OS units)

Notes

You can optionally transform the output with a transformation matrix, render the bounding boxes around objects as dotted red rectangles, suppress the rendering of the objects themselves, and change the flatness used to render curved paths.

[RISC OS Open source page](#)

*Repeat

BootCommands

This *Command scans a given directory, applying a command to everything it finds, within the limits of the other parameters.

Syntax

*Repeat command directory [-Directories | -File | -Applications | -Type type] [-CommandTail cmdtail] [-Tasks]

Parameters

directory – Command to apply to objects in given directory: Valid pathname for a directory

-Directories – Command to apply to objects in given directory: Apply the command only to directories

-Applications – Command to apply to objects in given directory: Apply the command only to applications

-Files – Command to apply to objects in given directory: Apply the command only to files

-Type – Command to apply to objects in given directory: Apply the command onto to files with a specific filetype

type – Command to apply to objects in given directory: A filetype number or text string

-CommandTail – Command to apply to objects in given directory: Postfix the found object with cmdtail

cmdtail – Command to apply to objects in given directory: Command tail to apply to objects

-Tasks – Command to apply to objects in given directory: Apply the command as a Wimp task

verbose – Command to apply to objects in given directory: Give an indication of progress

[RISC OS Open source page](#)

*ResolverConfig

This *Command (re)reads the system variables which the Resolver uses to determine which servers to use for internet name resolution.

Syntax

*ResolverConfig

Parameters

None.

[RISC OS Open source page](#)

*ResourceFS

This *Command selects the Resource Filing System as the current filing system.

Syntax

*ResourceFS

Parameters

None.

[RISC OS Open source page](#)

*RMClear

This *Command deletes all relocatable modules from the module area.

Syntax

*RMClear

Parameters

None.

Notes

This command deactivates all relocatable modules in the module area, deletes them, and frees up their workspace. Use this command only with extreme caution, as it is so drastic in its effects.

ROMModules are not affected by this command. These modules can be disabled by either *RMKill or *Unplug.

[RISC OS Open source page](#)

*RMEnsure

This *Command checks that a module is present and is the given version (or a more recent one).

Syntax

*RMEnsure module_title version_number [command]

Parameters

module_title / Title of installed module

version_number – Title of installed module: Version number to check against

command – Title of installed module: Command Line command

Notes

A command, optionally given as a third parameter, is executed if the module is not present or the version number of the loaded module is lower than the number supplied as the second parameter to *RMEnsure. If no third parameter is supplied, an error will instead be generated if the module is not present or does not satisfy the version number check.

[RISC OS Open source page](#)

*RMFaster

This *Command makes a ROM module faster by copying it from ROM to RAM.

Syntax

*RMFaster module_title

Parameters

module_title · Title of installed ROM module

Notes

Use this command only with extreme caution, as some modules may not work correctly if moved.

[RISC OS Open source page](#)

*RMInsert

This *Command reverses the action of a previous *Unplug command, but without reinitializing any modules.

Syntax

*RMInsert module_title [ROM_section]

Parameters

module_title / Title of any ROM module

ROM_section – Title of any ROM module: ROM section to restrict command to

Notes

If a ROM section number is specified, then the unplug bit or all versions of the specified module present in the given section.

If no ROM section number is specified, then it clears the unplug bit for all versions of the module in the machine.

[RISC OS Open source page](#)

*RMKill

This *Command deactivates and deletes a relocatable module.

Syntax

*RMInsert module_title[_%instantiation]

Parameters

module_title / Title of any currently installed module

instantiation – Title of any currently installed module: Instantiation of module in question

Notes

Use this command only with extreme caution, as it may be drastic in its effects.

[RISC OS Open source page](#)

*RMLoad

This *Command loads and initializes a relocatable module.

Syntax

*RMLoad filename [module_init_string]

Parameters

filename / Valid pathname specifying a module file

module_init_string – Valid pathname specifying a module file: Used to pass parameters to modules

Notes

The file must have a valid module filetype (&FFA), else it will not be loaded.

The optional initialisation string can be used to pass parameters to modules so they initialise in a particular way (if required). For example, how much memory it should claim etc.

[RISC OS Open source page](#)

*RMReInit

This *Command reinitializes a relocatable module, reversing the action of any previous *RMKill or *Unplug command.

Syntax

*RMReInit module_title [module_init_string]

Parameters

module_title / Title of any currently installed module

module_init_string – Title of any currently installed module: Used to pass parameters to modules

Notes

Use this command with extreme caution, as it may be drastic and unexpected effects.

[RISC OS Open source page](#)

*RMRun

This *Command runs a relocatable module, first loading and initializing it if necessary.

Syntax

*RMRun _ filename_ [module_init_string]

Parameters

filename / Valid pathname specifying a module file

module_init_string – Valid pathname specifying a module file: Used to pass parameters to modules

Notes

The file must have a valid module filetype (&FFA), else it will not be loaded.

The optional initialisation string can be used to pass parameters to modules so they initialise in a particular way (if required). For example, how much memory it should claim etc.

[RISC OS Open source page](#)

*RMTidy

This *Command collects together free space in the module area by moving and reinitializing all the modules it contains.

Syntax

*RMTidy

Parameters

None.

Notes

This command is deprecated, and has no effect on RISC OS 3.5 and later.

[RISC OS Open source page](#)

*ROMModules

This *Command displays information about all installed relocatable modules in ROM.

Syntax

*ROMModules

Parameters

None.

Notes

The information returned includes the number allocated to each module, whether it is part of system or in expansion cards or in an extension ROM, its status and its name.

[RISC OS Open source page](#)

*Route

This *Command directly manipulates the Internet module's network routing tables.

Syntax

*Route [-e] [-f] add [net_]host destination gateway metric_ *Route [-e] [-f] delete [net_]host destination gateway *Route -a

Parameters

-e · Place any fatal error report string in the system variable Inet\$Error instead of writing it to the standard output

-f · Delete all gateway entries in the Internet module's routing tables. If this is used in conjunction with an add command, the tables are flushed first

[net_]host · Specify net or host

destination · Internet host or network specified either by name (which must be present in the respective host or network name data base InetDBase\$Path) or by address (using the standard Internet dot notation)

gateway · Next-hop gateway to which packets should be addressed

metric · Count giving the number of hops to the destination

-a · Display the routing tables

Notes

The metric must be zero if the destination is on a directly-attached network, and non-zero if the route utilises one or more gateways. If you're adding a route with metric 0, the gateway given is the address of this host on the common network, indicating the interface to be used for transmission.

Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with destination. The optional keywords net and host force the destination to be interpreted as a network or a host, respectively. Otherwise, if the destination has a local address part of 0 or if it's the symbolic name of a network, then the route's presumed to be to a network; else the route's presumed to be to a host. All symbolic names specified for a destination or gateway are looked up first as a host name; if this fails, the name is then looked up as a network name.

[RISC OS Open source page](#)

*RouteDTraceOff

RouteD

This *Command stops the debug trace from RouteD.

Syntax

*RouteDTraceOff

Parameters

None.

[RISC OS Open source page](#)

*RouteDTraceOn

RouteD

This *Command starts the debug trace from RouteD.

Syntax

*RouteDTraceOn <level> [filename]

Parameters

level / Verbosity of debug to emit

filename – Verbosity of debug to emit: File into which to record the debug

Notes

If the filename is omitted then debug will instead be printed out direct to the screen.

[RISC OS Open source page](#)

*Run

This *Command loads and executes a file, optionally passing a list of parameters to it.

Syntax

*Run filename [parameters]

Parameters

filename / Valid pathname specifying a file

parameters – Valid pathname specifying a file: Command Line tail

Notes

The given pathname is searched for in the directories listed in the System Variable Run\$Path. If a matching object is a directory then it is ignore, unless it contains a !Run file. The valid first match is used.

- If the file has no filetype, then the file's load and execution addresses are used.
- If the filetype is &FF8 (Absolute) it is loaded at address &8000.
- For any specified filetype, the corresponding Alias\$@RunType_nnn variable is looked up to determine how the file should be loaded. If there is no corresponding Alias\$@RunType_nnn variable, then an error is generated.

[RISC OS Open source page](#)

*SafeLogon

BootCommands

This *Command checks whether the user is already logged on to the given file server, and if not, uses *Logon to log on.

Syntax

*SafeLogon [[:]file_server_number |:]file_server_name] user_name [[:Return]password]

Parameters

file_server_number / File server number to log on to

file_server_name – File server number to log on to: File server name to log on to

user_name – File server number to log on to: User name to log on to desired network

password – File server number to log on to: Password as used to log on to network

[RISC OS Open source page](#)

*SATADevices

SATADriver

This *Command lists all of the SATA devices attached to the computer.

Syntax

*SATADevices

Parameters

None.

[RISC OS Open source page](#)

*Save

This *Command copies the given area of memory to the named file.

Syntax

*Save filename start_addr end_addr [exec_addr [load_addr]]

or

*Save filename start_addr + length [exec_addr [load_addr]]

Parameters

filename / Valid pathname specifying a file

start_addr – Valid pathname specifying a file: Address to start saving

end_addr – Valid pathname specifying a file: Address of last byte saved +1

length – Valid pathname specifying a file: Number of bytes to save

exec_addr – Valid pathname specifying a file: Address to jump to after loading (applicable to programs only; default is start_addr)

load_addr – Valid pathname specifying a file: Address at which file is loaded into memory (when *Loaded; default is start_addr)

Notes

The length and addresses and length are in hexadecimal.

[RISC OS Open source page](#)

*SaveCMOS

BootCommands

This *Command saves the computer's CMOS RAM settings to a file, to allow the settings to be restored by a future call to *LoadCMOS

Syntax

*SaveCMOS filename

Parameters

filename · Name of file to save the settings to

[RISC OS Open source page](#)

*SaveFontCache

FontManager

This *Command saves the current contents of the font cache to a file.

Syntax

*SaveFontCache filename

Parameters

filename · Valid pathname specifying a file

Notes

This command is for internal use only. Do not use it.

[RISC OS Open source page](#)

*SaveModeFile

ScreenModes

This *Command saves the currently active list of mode definitions to disc, creating a Monitor Definition File. It is perhaps most useful when the machine has been configured to use EDID, as it will allow you to see the full list of modes that were extracted from the EDID (compared to, e.g. the display manager, which will only show the modes which the GPU is capable of displaying).

Syntax

*SaveModeFile filename

Parameters

filename · Name of file to create

Notes

Specifying vdu: as the filename will cause the output to be printed directly to the screen.

Other languages documentation

Français

[RISC OS Open source page](#)

*SChoose

SpriteUtils

This *Command selects a sprite from the system sprite area for use in subsequent sprite plotting operation.

Syntax

*SChoose sprite_name

Parameters

sprite_name · Valid name of sprite in system sprite area

Notes

None.

[RISC OS Open source page](#)

*SCopy

SpriteUtils

This *Command makes a copy of the source sprite within the system sprite area, and renames it as the destination sprite.

Syntax

*SChoose source_sprite dest_sprite

Parameters

source_sprite / Valid name of sprite in system sprite area

dest_sprite – Valid name of sprite in system sprite area: Name of destination sprite (to be saved into system sprite area)

Notes

An error is generated if a sprite already exists with the same name as the destination, in the system sprite area.

[RISC OS Open source page](#)

*ScreenLoad

SpriteUtils

This *Command loads the contents of a sprite file into the graphics window.

Syntax

*ScreenLoad filename

Parameters

filename · Valid pathname, specifying a sprite file

Notes

Depending on the information stored in the sprite file, the screen mode and palette may be changed to accommodate the sprite.

The first sprite in the file is plotted at the bottom left corner of the graphics window. Generally this will be the bottom left corner of the screen, as any mode change will have reset the graphics window such that it covers the entire screen.

[RISC OS Open source page](#)

*ScreenSave

SpriteUtils

This *Command saves the contents of the graphics window and the current palette to a sprite file.

Syntax

*ScreenSave filename

Parameters

filename · Valid pathname, specifying a sprite file

Notes

The sprite file created as a result of this command will contain one sprite named 'screendump'.

[RISC OS Open source page](#)

*SCSI

This *Command selects the SCSI filing system as the current filing system.

Syntax

*SCSI

Parameters

This command is only available if you have a SCSI expansion card fitted.

[RISC OS Open source page](#)

*SCSIDevices

SCSIDriver?

This *Command lists all of the known SCSI devices in the computer.

Syntax

*SCSIDevices

Parameters

None.

[RISC OS Open source page](#)

*SCSIUSB_PopUpDelay

This *Command sets the delay between plugging in a USB mass storage peripheral and the drive appearing on the icon bar.

Syntax

*SCSIUSB_PopUpDelay [delay]

Parameters

delay · Time in centiseconds

Notes

If the delay is omitted, the currently set delay is shown.

[RISC OS Open source page](#)

*SDelete

SpriteUtils

This *Command deletes one or more sprites from the system sprite area.

Syntax

*SDelete sprite_name1 [sprite_name2...]

Parameters

sprite_name1 / Valid name of sprite in system sprite area

sprite_name2 – Valid name of sprite in system sprite area: Optional sprite names to be deleted

Notes

Any error that occurs will stop this operation immediately, and no further sprites will be deleted.

[RISC OS Open source page](#)

*SDFS

This *Command selects the SDIO filing system as the current filing system.

Syntax

*SDFS

Parameters

This command is only available if you have a SDFS interface or socket.

[RISC OS Open source page](#)

*SDIODevices

SDIODriver

This *Command prints out a table of information about the currently known SDIO devices.

Syntax

*SDIODevices

Parameters

None.

Notes

Typical output might look like the following:

```
Bus Slot RCA Fun Description Capacity Vendor Product Rev Date
0 0 0002 0 MMC card 32 Mbytes Unknown MMC 0.0 1997-12
```

```

0 0 0003 0 MMC card 1 Gbytes Unknown 001-GB 1.5 2005-06
1 0 0001 0 SDXC card 128 Gbytes SanDisk SD128 2.1 2006-04
2 0 1234 0 SDHC card 4 Gbytes Panasonic SD04G 1.0 2010-08
2 0 1234 1 SDIO camera
3 0 5678 1 SDIO UART
3 1 9ABC 1 SDIO Bluetooth Type-A
3 2 DEF0 1 SDIO unknown function interface code (66)
3 2 DEF0 2 SDIO unknown function interface code (255)

```

[RISC OS Open source page](#)

*SDIOSlots

SDIODriver

This *Command prints out a table of information about the current configuration of each SDIO slot.

Syntax

*SDIODevices

Parameters

None.

Notes

Typical output might look like the following:

```

Bus Slot Voltage Width Frequency
0 0 1.8 V 8-bit 52 MHz DDR
1 0 1.8 V 4-bit 208 MHz SDR
2 0 3.0 V 4-bit 50 MHz SDR
3 0 1.8 V 1-bit 25 MHz SDR
3 1 1.8 V 1-bit 25 MHz SDR
3 2 3.3 V 1-bit 25 MHz SDR

```

[RISC OS Open source page](#)

*SDisc

This *Command prepares a disc for general use.

Syntax

*SDisc [disc_spec] *SDisc [:]disc_spec (netFS syntax)

Parameters

disc_spec · Name of disc or number of disc drive

Notes

A disc is prepared for general use by the current directory to its root directory, setting the library directory (if it is currently unset) to \$.Library, and unsetting the User Root Directory.

For NetFS, Mount selects a disc from the file server by setting the current directory, the library directory and the User Root Directory.

This command performs exactly the same function as *Mount.

[RISC OS Open source page](#)

*Set

This *Command assigns a string value to a system variable.

Syntax

*Set var_name value

Parameters

var_name · Variable name, or a wildcarded specification for a single variable name

value · String value to GSTrans, and then assign to specified system variable

Notes

Before a value is assigned as a string to a system variable, it is first converted using OS_GSTrans.

[RISC OS Open source page](#)

*SetEval

This *Command evaluates an expression and assigns its value to a system variable.

Syntax

*SetEval var_name expression

Parameters

var_name / Valid variable name

expression – Valid variable name: Valid Command Line expression

Notes

For a comprehensive list of expressions that can be used with this command see the Expressions page.

[RISC OS Open source page](#)

*SetMacro

This *Command assigns an expression to a system variable. The expression is evaluated each time the variable is used.

Syntax

*SetMacro var_name macro

Parameters

var_name · Variable name, or a wildcarded specification for a single variable name

macro · String value to assign to specified system variable

Notes

Unlike the *Set command, the macro value is not converted using OS_GSTrans before it is assigned. Instead, it is GSTrans'd each time the variable is read.

For a comprehensive list of expressions that can be used with this command see the Expression page.

[RISC OS Open source page](#)

*SetPS

This *Command changes the default printer server, without checking that the new one exists.

Syntax

*SetPS printer_server

Parameters

printer_server · Name or number of printer server

Notes

This command does not check if the specified printer server name or number is valid, so an error for an invalid printer server will not be generated until trying to print something.

If no argument is supplied, it sets the current printer server to the default printer server (as set by *Configure PS).

[RISC OS Open source page](#)

*SetStation

This *Command sets a station-number.

Syntax

*SetStation [station_number]

Parameters

station_number · Station number (2 – 254)

Notes

The station number is stored in the allocated CMOS RAM, so when the device is switched off the information is retained.

If no value is supplied, a value will be requested. An invalid number will result in the current version being preserved.

[RISC OS Open source page](#)

*SetType

This *Command sets the file type of the named file to the given textual file type or hexadecimal number.

Syntax

*SetType filename filetype

Parameters

filename / Valid pathname specifying a file

filetype – Valid pathname specifying a file: Filetype text description or 3 digit hexadecimal filetype number

Notes

If the file does not have a date stamp, then it is stamped with the current time and date.

Although the command allows for the three digit hexadecimal code to be used without an Ampersand sign, it is highly recommended that one always be used. This is to stop any ambiguities with three character filetype names.

[RISC OS Open source page](#)

*SFlipX

SpriteUtils

This *Command reflects a sprite in the system sprite area about its X axis, i.e. it performs a vertical flip.

Syntax

*SFlipX sprite_name

Parameters

sprite_name · Valid name of sprite in system sprite area

[RISC OS Open source page](#)

*SFlipY

SpriteUtils

This *Command reflects a sprite in the system sprite area about its Y axis, i.e. it performs a horizontal flip.

Syntax

*SFlipY sprite_name

Parameters

sprite_name · Valid name of sprite in system sprite area

Notes

This command is equivalent to turning a sprite to face the opposite horizontal direction.

[RISC OS Open source page](#)

*SGet

SpriteUtils

This *Command gets a sprite from a rectangular area of the screen, defined by the most recent graphics positions (inclusive), and then saves it in the system sprite area.

Syntax

*SGet sprite_name

Parameters

sprite_name · Valid name of new sprite in system sprite area

Notes

None.

[RISC OS Open source page](#)

*Shadow

This *Command sets which bank of screen memory is used on subsequent mode changes.

Syntax

*Shadow [0|1]

Parameters

0 / Use shadow bank

1 – Use shadow bank: Use non-shadow bank

Notes

if no parameter is passed, then the shadow bank is used.

For shadow banks to be utilised, there must be at least double amount of memory available in the screen area memory, for the required screen mode.

This command is provided only for backwards compatibility.

[RISC OS Open source page](#)

*Share

This *Command makes a local directory available as a shared disc.

Syntax

*Share directory [disc_name] [-protected] [-cdrom] [-noicon] [-readonly]

Parameters

directory / Valid pathname of a directory

disc_name – Valid pathname of a directory: Name to use for the shared disc

-protected – Valid pathname of a directory: Make directory protected (default is unprotected)

-cdrom – Valid pathname of a directory: Indicates that shared directory is on a CD-ROM

-noicon – Valid pathname of a directory: Prevent an icon appearing for the shared disc

-readonly – Valid pathname of a directory: Specify shared disc as read only

Notes

If no disc_name is provided, the name of the directory is used, or in the case of root directory, the disc name itself.

[RISC OS Open source page](#)

*ShareFS

This *Command selects the Shared Filing System as the current filing system.

Syntax

*ShareFS

Parameters

None.

[RISC OS Open source page](#)

*ShareFSIcon

This *Command adds an icon to the icon bar for a remote shared disc.

Syntax

*ShareFSIcon disc_name

Parameters

disc_name · Name of the remote shared disc

[RISC OS Open source page](#)

*ShareFSLogoff

This *Command disconnects from an Access+ remote shared object.

Syntax

*ShareFSLogoff disc_name

Parameters

disc_name · Name of the remote shared object

[RISC OS Open source page](#)

*ShareFSLogon

This *Command connects to an Access+ remote shared object.

Syntax

*ShareFSLogon disc_name [password]

Parameters

disc_name / Name of the remote shared object

password – Name of the remote shared object: Optional password for protected objects

[RISC OS Open source page](#)

*ShareFSWindow

This *Command changes the size of the ShareFS transmission window, or with no parameter, report its current size.

Syntax

*ShareFSWindow size

Parameters

size · Size of the ShareFS transmission window

Notes

You should not change the size, unless you know what you are doing.

[RISC OS Open source page](#)

*Shares

This *Command reports the local directory currently available as a shared disc.

Syntax

*Shares

Parameters

None.

Notes

Reports the local directory that was set using the *Share command.

[RISC OS Open source page](#)

*ShellCLI

ShellCLI

This *Command invokes a command shell from a Wimp application, starting it as a *WimpTask.

Syntax

*ShellCLI

Parameters

None.

Notes

You can only issue this command from within an active task.

This command uses Shell_Create and Shell_Destroy SWI calls.

[RISC OS Open source page](#)

*ShellCLI_Task

TaskWindow

This *Command runs an application in a window.

Syntax

*ShellCLI_Task xxxxxxxx zzzzzzzz

Parameters

xxxxxxx / Task handle of parent task (8 digit hexadecimal number)

zzzzzzzz – Task handle of parent task (8 digit hexadecimal number): Handle number used by parent to identify the task (8 digit hexadecimal)

Notes

This command is now deprecated, and must not be used. The *TaskWindow command provides the same functionality.

[RISC OS Open source page](#)

*ShellCLI_TaskQuit

TaskWindow

This *Command quits the current task window.

Syntax

*ShellCLI_TaskQuit

Parameters

None.

Notes

This command is intended for use only within desktop applications.

[RISC OS Open source page](#)

*Show

This *Command displays the name, type and current value of the system variables matching the given pattern.

Syntax

*Show [var_spec]

Parameters

var_spec · Variable name, or a wildcarded specification for a set of variable names

Notes

If no variable name is given, all system variables are displayed.

[RISC OS Open source page](#)

*ShowFPRegs

Debugger

This *Command displays the state of the FPA floating point coprocessor at the time of the last floating point exception. This includes the value of the data registers, the control register, and the hardware type.

Syntax

*ShowFPRegs

Parameters

None.

Notes

Unlike *ShowRegs, this exception dump is only updated when a floating point exception occurs.

The memory address where the exception dump is stored is also displayed.

[RISC OS Open source page](#)

*ShowFree

This *Command displays the total free space remaining on a device within a window on the desktop.

Syntax

*ShowFree -FS fs_name device

Parameters

-FS / Prefix to specify filing system name

fs_name – Prefix to specify filing system name: Name of filing system

device – Prefix to specify filing system name: Name of device to return free space

Notes

This command will only work with registered filing system using Free_Register.

[RISC OS Open source page](#)

*ShowRegs

Debugger

This *Command displays the ARM register contents for the saved state.

Syntax

*ShowRegs

Parameters

None.

Notes

A saved state can be caught on the following traps:

- Address exception
- Break point
- Data abort
- Prefetch abort
- Undefined instruction
- Floating point and vector floating point exceptions

The memory address where the registers are stored is also displayed.

[RISC OS Open source page](#)

*ShowVFPRegs

Debugger

This *Command allows the contents of a VFP/NEON coprocessor context to be displayed. It has three modes of operation, described below. For each mode, the address of the context, the context flags, and the contents of the data and special registers will be displayed.

Syntax

*ShowVFPRegs [A <addr>|C|E]

Parameters

A <addr> · Show the contents of the context at the specified hexadecimal address (optional)

C · Show the contents of the currently active context (optional)

E · Show the contents of the context that last caused an exception (optional)

Notes

If no parameters are specified, the command acts as if '*ShowVFPRegs E' was issued, i.e. the contents of the exception context will be displayed.

Note that the exception context displayed by '*ShowVFPRegs E' is often only updated when a VFP exception occurs. Other types of exception are unlikely to update it.

[RISC OS Open source page](#)

*ShrinkRMA

BootCommands

This *Command will try to shrink the relocatable module area to its minimum size.

Syntax

*ShrinkRMA

Notes

This call only removes the free pages from the end of the RMA. Free space due to fragmentation within the RMA will not be affected.

[RISC OS Open source page](#)

*Shut

This *Command closes all open files on all filing systems.

Syntax

*Shut

Parameters

None.

Notes

Because this command closes all open files, this command must not be used within a multi-tasking environment. This is because it may close files belonging to other applications, resulting in errors and loss of work.

[RISC OS Open source page](#)

*ShutDown

This *Command prepares all filing systems for a system shutdown.

Syntax

*ShutDown

Parameters

None.

Notes

ShutDown closes all open files on all filing systems, logs off all NetFS file servers and parks hard disc heads.

Because this command closes all open files, this command must not be used within a multi-tasking environment. This is because it may close files belonging to other applications, resulting in errors and loss of work.

[RISC OS Open source page](#)

*SInfo

SpriteUtils

This *Command displays information on the system sprite workspace.

Syntax

*SInfo

Parameters

The returned information consists of the amount of system sprite workspace currently reserved, and the amount of free space, and the number of sprites defined.

Notes

None.

[RISC OS Open source page](#)

*SList

SpriteUtils

This *Command lists the names of all the sprites in the system sprite area.

Syntax

*SList

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*SLoad

SpriteUtils

This *Command loads a sprite file into the system sprite area.

Syntax

*SLoad filename

Parameters

filename · Valid pathname, specifying a sprite file

Notes

If insufficient memory is available, then an error will be returned, and nothing will be loaded. Furthermore, sprites already in memory will be lost.

[RISC OS Open source page](#)

*SMerge

SpriteUtils

This *Command merges the sprites in a file with those in the system sprite area.

Syntax

*SMerge filename

Parameters

filename · Valid pathname, specifying a sprite file

Notes

If insufficient memory is available, then an error will be returned, and nothing will be loaded.

And sprites in memory with the same name as one being merged will be overwritten.

Although the action is to merge, there must be sufficient memory to hold both the new file and original sprites. This is because the sprite file is first loaded into memory, and then the original sprites are replaced (if any have the same name).

[RISC OS Open source page](#)

*SNew

SpriteUtils

This *Command deletes all the sprites in the system sprite area.

Syntax

*SNew

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*Sound

This *Command generates a sound with no delay.

Syntax

*sSound channel amplitude pitch duration

Parameters

channel / Channel number (1 – 8)

amplitude – Channel number (1 – 8): Linear scale: 0 silent – &FFF1 loud, or;

Logarithmic scale: &100 silent – &17F loud (a change of 16 will halve/double the amplitude)

pitch – Channel number (1 – 8): Pitch: 0 – 255. Each unit represents a quarter of a semitone. Value of 53 is middle C, or 256 to 32767 (&100 to &7FF) where top 3 bits specify octave, and bottom 12 bits specify fraction of octave. Value of 16384 (&4000) is middle C

duration – Channel number (1 – 8): Duration: 0 to 32767 (&0 – &7FFF) in twentieths of a second. Value of 255 gives a note of infinite duration

Notes

To generate a sound after a specified amount of time use *QSound.

[RISC OS Open source page](#)

*SoundGain

This *Command controls the gain of the 8-bit sound system, relative to the 16-bit sound system.

Syntax

* SoundGain gain

Parameters

gain · Gain setting, in the range 0 to 7

Notes

The gain is specified in units of 3dB, therefore 0 corresponds to +0dB and 7 corresponds to +21dB.

Specifying values above zero may result in clipping.

[RISC OS Open source page](#)

*Speaker

This *Command turns the internal speakers on or off.

Syntax

*Speaker on|off

Parameters

on or off · Turn internal speakers on or off

Notes

It may still be possible to hear a low level of sound emanating from the speakers. This is because the command mutes the speakers, rather than disabling them.

[RISC OS Open source page](#)

*Spool

This *Command is to sends the VDU commands appearing on the screen to the specified file.

Syntax

*Spool [filename]

Parameters

filename · Valid pathname specifying a file

Notes

The spooling of data to a file is managed internally by OS_BPut. If OS_BPut generates an error, the spool file will be closed, and the error is then returned from OS_WriteC.

If no filename is given, the current spool file is closed.

The outputting of data continues until either a *Spool or *SpoolOn command (with or without a filename) is issued.

It is possible to temporarily disable the spool file without closing it by issuing OS_Byte 3.

[RISC OS Open source page](#)

*SpoolOn

This *Command appends the VDU commands appearing on the screen to the specified file.

Syntax

*SpoolOn [filename]

Parameters

filename · Valid pathname specifying a file

Notes

The spooling of data to a file is managed internally by OS_BPut. If OS_BPut generates an error, the spool file will be closed, and the error is then returned from OS_WriteC.

If no filename is given, the current spool file is closed.

The outputting of data continues until either a *Spool or *SpoolOn command (with or without a filename) is issued.

It is possible to temporarily disable the spool file without closing it by issuing OS_Byte 3.

[RISC OS Open source page](#)

*SRename

SpriteUtils

This *Command renames a sprite within the system sprite area.

Syntax

*SRename old_sprite new_sprite

Parameters

old_sprite / Name of sprite to be renamed

new_sprite – Name of sprite to be renamed: New name to be given to sprite

Notes

An error will be generated if a sprite with the new_sprite name already exists.

Sprite names can contain any sequence of printable characters, but cannot include a space.

There is no differentiation between uppercase and lowercase.

[RISC OS Open source page](#)

*SSave

SpriteUtils

This *Command saves the system sprite area as a sprite file.

Syntax

*SSave filename

Parameters

filename · Valid pathname, specifying a sprite file

Notes

None.

[RISC OS Open source page](#)

*Stamp

This *Command sets the date stamp on a file to the current time and date.

Syntax

*Stamp filename

Parameters

filename · Valid pathname specifying a file

Notes

If the file has not previously been date stamped, it is also given file type Data (&FFD).

[RISC OS Open source page](#)

*StartDesktopTask

TaskManager

This *Command starts a new task next time the desktop environment is entered. Tasks run using this command from outside the desktop will be queued until that time.

Syntax

*StartDesktopTask [command]

Parameters

command · *Command to issue

[RISC OS Open source page](#)

*Status

This *Command displays the value of a configuration option in the CMOS RAM.

Syntax

*Status [option]

Parameters

option · Name of configured option

Notes

If no option is specified, the values of all configuration options are shown.

[RISC OS Open source page](#)

*Stereo

This *Command sets the position in the stereo image of a sound channel.

Syntax

*Stereo channel position

Parameters

channel / Channel number (1 – 8)

position – Channel number (1 – 8): Left (-127) to Right (+127)

Notes

None.

[RISC OS Open source page](#)

*Stop

This *Command stops the playing of the disc in the given CD-ROM drive.

Syntax

*Stop [drive]

Parameters

drive · Valid CD/DVD drive number

Notes

If no drive parameter is passed, then the current drive is used.

[RISC OS Open source page](#)

*Supported

This *Command lists the drive types recognised by CDFS.

Syntax

*Supported

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*TaskWindow

TaskWindow

This *Command starts a background task, which will open a task window if it needs to get input, or to output a character to the screen.

Syntax

*TaskWindow [command] [[-wimplot] m K] [[-name] taskname] [-ctrl] [-display] [-quit] [-nice n] [-task &xxxxxxx] [-txt &xxxxxxx]

Parameters

m – *Command to issue (background task): Memory to allocate to task

taskname – *Command to issue (background task): Name of task

-ctrl – *Command to issue (background task): Allow control characters through

-display – *Command to issue (background task): Open task window immediately (else wait until a character is to be displayed

-quit – *Command to issue (background task): Specify that task must quit after the command

n – *Command to issue (background task): Set the number of time slices to give the task

-task &xxxxxxx – *Command to issue (background task): Task handle of calling task (8 digit hexadecimal number)

-txt &xxxxxxx – *Command to issue (background task): Handle number used by parent to identify the task (8 digit hexadecimal)

Notes

Fields passed to this command that consist of more than one word must be enclosed in double quotes.

You can only issue this command from within an active task.

[RISC OS Open source page](#)

*Tempo

This *Command sets the *Sound and *QSound system tempo.

Syntax

*Tempo tempo

Parameters

tempo · Tempo: 0 to &FFFF (default is &1000)

Notes

The default value of &1000 is equivalent to one beat per centisecond. Doubling or halving the tempo is equal to doubling of halving the tempo. e.g. A tempo &800 is equivalent to one beat every two seconds.

[RISC OS Open source page](#)

*Territories

* Territories

This *Command sets list the currently loaded territory modules.

Syntax

*Territories

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*Territory

This *Command sets the current territory, or with no parameter displays the currently selected territory.

Syntax

*Territory [territory_name]

Parameters

territory_name · Valid territory name. A list of territories can be obtained with *Territories

Notes

None.

[RISC OS Open source page](#)

*Time

This *Command displays the day, date and time.

Syntax

*Time

Parameters

None.

Notes

The day, date and time is returned in the same format as per OS_Word 14,0

[RISC OS Open source page](#)

*ToolSprites

WindowManager

This *Command merges the sprites in a file with those in the Wimp's pool of border sprites, which are used to redraw window borders.

Syntax

*ToolSprites filename

Parameters

filename · Valid name of sprite file

Notes

A redraw of the screen (such as a mode change) is advisable if any of the border sprites have been changed.

[RISC OS Open source page](#)

*Tuning

This *Command sets the overall tuning for the *Sound (and *QSound) system.

Syntax

*Tuning relative_change *Tuning auto <on|off>

Parameters

relative_change · -16383 to +16383 (0 to reset to default)

Notes

This command has two forms.

The first form controls the manual tuning. When setting manual tuning, a value of 0 can be specified to reset to the default value.

The second form of the command controls automatic tuning. If automatic tuning is on, the tuning value will be automatically adjusted to match changes in the sound system sample rate. If automatic tuning is enabled then any attempt to set the manual tuning value will be silently ignored.

Automatic tuning first became available in RISC OS 5.19 (SoundChannels 1.37), and is enabled by default.

[RISC OS Open source page](#)

*TV

This *Command sets the vertical screen alignment and screen interlace.

Syntax

*TV [vert_align [[,]interlace]]

Parameters

vert_align / Adjust vertical screen alignment. Possible values of:

0 - 3: lines up by this amount

255 - 252: lines down 1 to 4 lines (respectively)

interlace – Adjust vertical screen alignment. Possible values of: Switch screen interlace setting. Possible values of:

0 = Interlace on

1 = Interlace off

[RISC OS Open source page](#)

*Type

This *Command displays the contents of the named file using the configured *Dump format.

Syntax

*Type [-File] filename [-TabExpand]

Parameters

-File / Optional prefix to filename, but has no effect

filename – Optional prefix to filename, but has no effect: Valid pathname specifying a file

-TabExpand – Optional prefix to filename, but has no effect: Expand tab characters (ASCII 9) to 8 spaces

Notes

To display the contents of a file but with line numbers, use *List instead.

The format of data to display can be set using *Configure DumpFormat.

[RISC OS Open source page](#)

*Unalias

This *Command unsets an alias for a command.

Syntax

*Alias alias

Parameters

alias · Alias variable

Notes

None.

[RISC OS Open source page](#)

*Unlock

This *Command unlocks the disc in the given CD-ROM drive, enabling the Eject button.

Syntax

*Unlock [drive]

Parameters

drive · Valid CD/DVD drive number

Notes

If no drive parameter is passed, then the current drive is unlocked.

[RISC OS Open source page](#)

*Unplug

This *Command kills and disables all copies of a ROM resident module.

Syntax

*Unplug [module_title [ROM_section]]

Parameters

module_title / Title of any ROM module

ROM_section – Title of any ROM module: ROM section to restrict command to

Notes

If no parameters are given, the unplugged ROM modules are listed.

[RISC OS Open source page](#)

*Unset

This *Command deletes a system variable.

Syntax

*Unset var_spec

Parameters

var_spec · Variable name, or a wildcarded specification for a single variable name

Notes

If no variable name is given, all system variables are displayed.

[RISC OS Open source page](#)

*UnShare

This *Command removes, or unsets a directory currently available as a shared disc.

Syntax

*UnShare disc_name

Parameters

disc_name · Name to use for the shared disc

Notes

This command makes a local directory no longer available as shared disc.

[RISC OS Open source page](#)

*Up

This *Command moves the current directory up the directory structure by the specified number of levels.

Syntax

*Up [levels]

Parameters

levels · Number of directory levels to move up (0 - 128)

Notes

If no number is passed, the directory is moved up one level.

*Up is equivalent to *Dir^.

[RISC OS Open source page](#)

*URD

This *Command sets the User Root Directory (URD).

Syntax

*URD [directory]

Parameters

directory · Valid pathname of a directory

Notes

If no directory is specified, the URD is set to the root directory.

[RISC OS Open source page](#)

*URLProtoShow

URL_Fetcher

This *Command lists all the protocol modules currently registered with the URL_Fetcher module, and lists their SWI bases.

Syntax

*URLProtoShow

[RISC OS Open source page](#)

*USB Buses

*USB Buses shows a list of the USB buses attached to the computer. Syntax: *USB Buses

[RISC OS Open source page](#)

*USBConfInfo

*USBConfInfo shows configuration information for a USB device as listed by USBDevices. Syntax:

*USBConfInfo

[RISC OS Open source page](#)

*USBDevices

*USBDevices shows a list of the USB devices attached to the computer. Syntax: *USBDevices

[RISC OS Open source page](#)

*USBDevInfo

*USBDevInfo shows information on a USB device as listed by USBDevices. Syntax: *USBDevInfo

[RISC OS Open source page](#)

*USBQuirk

*USBQuirk adds, removes, or lists the quirk flags for the specified vendor/product/device. Syntax:

*USBQuirk <A | R | L>

In practice

This is a method to override misbehaviour in USB devices.

[RISC OS Open source page](#)

*USBReset

*USBReset forces the resetting of device by disconnecting it in software, and making the hub driver start the port again. Syntax: *USBReset

In practice

Note: Root hubs cannot be reset.

[RISC OS Open source page](#)

*USBSetConfig

*USBSetConfig sets the configuration of a USB device as listed by USBDevices. Syntax: *USBSetConfig

[RISC OS Open source page](#)

*USBSetInterface

*USBSetInterface sets the alternate setting for an interface on a USB device. Syntax: *USBSetInterface

[RISC OS Open source page](#)

*VCGenCmd

This *command requires the BCMVideo module, which is part of the Raspberry Pi ROM.

In practice

*VCGenCmd sends a text-based command to the VideoCore OS

It reads system configuration and other parameters.

See <https://www.raspberrypi.org/documentation/configuration/config-txt/README.md> for details.

Syntax

*VCGenCmd command

Parameters

get_config int / integer settings

get_config str – integer settings: string settings

get_config specific setting – integer settings: get value of the specific setting

See https://elinux.org/RPI_vcgencmd_usage for further information

Examples

*VCGenCmd get_config int *VCGenCmd get_config str *VCGenCmd get_config arm_freq *VCGenCmd measure_temp

[RISC OS Open source page](#)

*Verify

This *Command checks that the whole disc is readable.

Syntax

*Verify [disc_spec]

Parameters

disc_spec · Name of disc or number of disc drive

Notes

If no disc is specified, then the operation is performed on the current disc.

This command can be safely used on both hard and floppy discs.

[RISC OS Open source page](#)

*Voices

This *Command displays a list of the installed voices by name and number, and shows which voice is assigned to each of the eight channels.

Syntax

*Voices

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*Volume

This *Command sets the maximum overall volume of the 8 bit sound system.

Syntax

*Volume volume

Parameters

volume · 1 (quiet) to 127 (loud)

Notes

None.

[RISC OS Open source page](#)

*Where

Debugger

This *Command describes the region in memory of an address or register.

Syntax

*Where [<addr|reg>]

Parameters

None.

Notes

None.

[RISC OS Open source page](#)

*WhichDisc

This *Command returns the unique ID number for the disc in the given CD-ROM drive, or of the current drive if none is specified.

Syntax

*WhichDisc [drive]

Parameters

drive · Valid CD drive number

Notes

None.

[RISC OS Open source page](#)

*WimpKillSprite

WindowManager

This *Command removes a sprite from the Wimp's sprite area.

Syntax

*WimpKillSprite spritename

Parameters

spritename · Name of sprite in the Wimp sprite pool

Notes

An error is returned if the sprite cannot be found in the Wimp's RAM sprite pool.

[RISC OS Open source page](#)

*WimpMode

WindowManager

This *Command changes the current screen mode used by the Desktop.

Syntax

*WimpMode screen_mode

Parameters

screen_mode · Mode number of screen, or Mode String

Notes

This command is not supported when issued from within a task window.

[RISC OS Open source page](#)

*WimpPalette

WindowManager

This *Command uses a palette file to set the Wimp's colour palette.

Syntax

*WimpPalette filename

Parameters

filename · Valid name of palette file (filetype &FED)

Notes

None.

[RISC OS Open source page](#)

*WimpScroll

WindowManager

This *Command configures the mouse scroll wheel behaviour in the desktop.

Syntax

*WimpScroll [-Type] [-LineScroll] [-Speed]

Parameters

-Type / The style of scrolling (see notes).

-**LineScroll** – The style of scrolling (see notes).: Makes scrolling in text editors move the window only by whole multiples of a line of text.

-**Speed** – The style of scrolling (see notes).: Sets how far a window is scrolled for each wheel nudge, in multiples of 4 OS units.

Notes

The scrolling styles available are:

- Pointer – The window over which the pointer is currently
- Focus – The window with the input focus
- FocusOrPointer – The window with input focus, else under the pointer
- FavourHigher – As FocusOrPointer, preferring the one highest up the window stack in the event of a tie

With no arguments, the current settings are reported.

[RISC OS Open source page](#)

*WimpSlot

WindowManager

This *Command changes the memory allocation for the current and (optionally) the next Wimp task.

Syntax

*WimpSlot [-Min] minsize[K|M|G] [-Max maxsize[K|M|G]] [-Next nextsize[K|M|G]]

Parameters

minsize / Minimum of application space required

maxsize – Minimum of application space required: Maximum amount of application space required

nextsize – Minimum of application space required: Amount of memory that will be allocated to the next application

Notes

If the current WimpSlot is below min or above max, it will be adjusted up to min or down to max. If WimpSlot is currently within the min/max range it will remain unchanged.

If not enough memory is available an error will be generated.

[RISC OS Open source page](#)

*WimpTask

WindowManager

This *Command starts up a new task from within another task.

Syntax

*WimpTask command

Parameters

command · *Command which is to be passed

Notes

The passed *Command is issued via Wimp_StartTask.

This command will exit via OS_Exit if issued from outside a Wimp task.

Note that this command is not safe to use from BASIC. Use *StartDesktopTask, or Wimp_Initialise then Wimp_StartTask instead.

[RISC OS Open source page](#)

*WimpVisualFlags

WindowManager

The purpose of this *Command is to modify settings used to change the appearance within the Desktop.

Syntax

*WimpVisualFlags options

Parameters

- All – Abbreviation: -A · Description: Turn all flags on
- FontBlending – Abbreviation: -FB · Description: Use font blending in icons
- Fully3DIconBar – Abbreviation: -F3DIB · Description: Make the iconbar have a full 3D border
- 3DWindowBorders – Abbreviation: -3DWB · Description: Give all menus and dialogue boxes a 3D border
- TexturedMenus – Abbreviation: -TM · Description: Give all menus a textured background
- UseAlternateMenuBg – Abbreviation: -UAMB · Description: Use a different background tile for menus
- RemoveIconBoxes – Abbreviation: -RIB · Description: Remove the filled box from behind the text in text+sprite icons
- NoIconBoxesInTransWindows – Abbreviation: -NIBITW · Description: Remove the filled box from icons on windows similar to the pinboard
- NoIconBarBorder – Abbreviation: -NIBB · Description: Remove the outline from the icon bar
- WindowOutlineOver – Abbreviation: -WOO · Description: Plot the window outline over the tool icons
- WindowOutlineColour <&RRGGBB> – Abbreviation: -WOC · Description: Set the colour of the window outline
- WindowBorderFaceColour <&RRGGBB> – Abbreviation: -WBFC · Description: Set the colour of the top left portion of the window border
- WindowBorderOppColour <&RRGGBB> – Abbreviation: -WBOC · Description: Set the colour of the bottom right portion of the window border
- MenuBorderFaceColour <&RRGGBB> – Abbreviation: -MBFC · Description: Set the colour of the top left portion of the menu border
- MenuBorderOppColour <&RRGGBB> – Abbreviation: -MBOC · Description: Set the colour of the bottom right portion of the menu border
- ButtonBorderFaceColour <&RRGGBB> – Abbreviation: -BBFC · Description: Set the colour of the top left portion of button icons
- ButtonBorderShallowColour <&RRGGBB> – Abbreviation: -BBSC · Description: Set the colour of the shallow ridge and channel icons
- ButtonBorderOppColour <&RRGGBB> – Abbreviation: -BBOC · Description: Set the colour of the bottom right portion of button icons
- ButtonColour <&RRGGBB> – Abbreviation: -BC · Description: Set the colour of the top of button icons
- ButtonWellColour <&RRGGBB> – Abbreviation: -BWC · Description: Set the colour of the well around default action buttons
- ButtonHighlightColour <&RRGGBB> – Abbreviation: -BHC · Description: Set the colour of action buttons when selected

Notes

Options that are not specified are turned off.

[RISC OS Open source page](#)

*WimpWriteDir

WindowManager

This *Command sets the direction of the text entry for writeable icons to either the default for the current territory, or the reverse.

Syntax

*WimpWriteDir 0|1

Parameters

0 or 1 · 0 for default for current territory, 1 for reverse of default

Notes

The setting of this command also affects the direction in which text inside icons is printed.

[RISC OS Open source page](#)

*Wipe

This *Command deletes one or more objects that match the given wildcard specification.

Syntax

*Wipe object_spec [[~]options]

Parameters

object_spec / Valid (wildcarded) pathname specifying a file or directory

options · Valid options (Upper/lowercase letters, separated by spaces). · Default

C(onfirm) · Prompt for confirmation of each copy · ON

F(orce) · Force deletion of locked objects · OFF

R(ecurse) · Delete subdirectories and contents · OFF

V(erbose) · Print information on each object deleted · ON

Notes

Options are taken from the system variable Wipe\$Options, and those given to the command.

To turn on a specific option, it must be included in the option list. To ensure a specific option is turned off, then it must be preceded with the tilde symbol (~).

[RISC OS Open source page](#)

*X

BootCommands

This *Command executes the given command, trapping any error and storing its message text in the X\$Error system variable.

Syntax

*X command

Parameters

command · Command to execute

Notes

If X\$Error is already set, its value will not be modified.

[RISC OS Open source page](#)

Messages

Defined message and protocol references from the linked documentation.

Message_AlarmGoneOff

Code &501 · WindowManager

This message is broadcast by !Alarm when an application goes off.

Message Block

+20 · Name of application (null terminated)

+n · Application's identifier (null terminated)

Notes

If the message is not acknowledged, then !Alarm prompts the user to start the application (with a non-multitasking error box dialog box), and defers the alarm for 1 minute, before restarting this message broadcast procedure.

[RISC OS Open source page](#)

Message_AlarmSet

Code &500 · WindowManager

This message can be sent used to set/cancel alarms.

Message Block

+20 · 0 · Set alarm (local time)

1 · Set alarm (UTC)

2 · Cancel alarm

+24 · Depends on reason code at +20

Notes

If the message is recorded, then the application !Alarm acknowledges with 0 if successful or a null terminated error string. (Presumably this is in the error block at +20, but unsure. If it is, then the message block format is inconsistent, depending on whether it is to or from the application).

When setting an alarm using local time, it is possible to add out-of-range alarms to the database, which !Alarm will fault on redraw (but not fault on save).

When the task goes off !Alarm broadcasts Message_AlarmGoneOff, which your application should acknowledge.

[RISC OS Open source page](#)

Message_AppControl

Code &15 · WindowManager

This message is used to control applications in a generic manner.

Message Block

+20 · Protocol version number (1)

+24 · AppControl Flags

+28 · AppControl Operations

[RISC OS Open source page](#)

Message_ClaimEntity

Code &F · WindowManager

This message should be broadcast to all tasks an application claims the caret/selection or clipboard.

Message Block

+20 · Flags

Notes

None.

[RISC OS Open source page](#)

Message_ClaimEntity Flags

WindowManager

All other bits are reserved and must be set to zero.

Bit / Meaning when set

0 and 1 – Meaning when set: Caret or selection being claimed

2 – Meaning when set: Clipboard being claimed

[RISC OS Open source page](#)

Message_ColourPickerCloseDialogueRequest

WindowManager

This message is sent by ColourPicker to inform you that the picker wishes to close, usually because the 'OK' or 'Cancel' buttons have been pressed.

Message_ColourPickerCloseDialogueRequest

Message Block

+20 · ColourPicker handle

Notes

The task should respond by calling ColourPicker_CloseDialogue

[RISC OS Open source page](#)

Message_ColourPickerColourChanged

WindowManager

This message is sent by ColourPicker to inform you that the colour has changed.

Message Block

+20 · ColourPicker handle

+24 · ColourPicker Changed Flags

+28 · ColourPicker Colour Descriptor

Notes

None.

[RISC OS Open source page](#)

Message_ColourPickerColourChoice

Code &47700 · WindowManager

This message is sent by ColourPicker to inform the task that the 'OK' button has been clicked.

Message Block

+20 · ColourPicker handle

+24 · ColourPicker Choice Flags

+28 · ColourPicker Colour Descriptor

Notes

This message may or may not be followed by a Message_ColourPickerCloseDialogRequest.

The ColourPicker handle is returned when creating the ColourPicker window calling ColourPicker_OpenDialogue.

[RISC OS Open source page](#)

Message_ColourPickerOpenParentRequest

WindowManager

This message is sent by ColourPicker to ask you to open the parent of this window.

Message_ColourPickerOpenParentRequest

Message Block

+20 · ColourPicker handle

Notes

ColourPicker normally issues this message when the Adjust mouse button is pressed over the close icon.

[RISC OS Open source page](#)

Message_DataLoad

Code &3 · WindowManager

This message is sent by the Filer to ask an application to load a file, or by a task in response to a Message_DataSaveAck message.

+20 · Destination window handle
+24 · Destination icon handle
+28 · Destination screen x
+32 · Destination screen y
+36 · Estimated size, or -1 if unsafe
+40 · Filetype of data
+44 · Full pathname
+n · Column of file in current selection
+n+4 · Row of file in current selection

Notes

When replying to a Message_DataSaveAck message, the data-saving task should write its data to the temporary file specified, and reply by sending this message.

The reply to this message should be sent as a User_Message_Recorded so that if it returns unanswered, the sender can delete the temporary file and report an error.

Possible responses from data-loading task:

- Task sends Message_DataLoadAck message (file successfully loaded)
- Task does not reply (error loading data / not interested in filetype)

[RISC OS Open source page](#)

Message_DataLoadAck

Code &4 · WindowManager

This message is sent by a task that is loading data, in response to the Message_DataLoad message.

Message Block

+20 · Destination window handle
+24 · Destination icon handle
+28 · Destination screen x
+32 · Destination screen y
+36 · Estimated size
+40 · Filetype of data
+44 · Full pathname

Notes

A task that is capable of loading the particular type of file (as specified in the Message_DataLoad message) should try and load the file. If successful, then the task should reply using this message. To reply, all that is necessary is to change the message type to 4, and fill in the 'your_ref' field.

In addition, if the file was a temporary file, the task loading has the responsibility to delete it. The best way to determine this seems to be that messages from the Filer (file loading rather than data transfer) will have their 'your_ref' field set to 0, whereas Message_DataLoad messages which followed on from a Message_DataSaveAck will not.

[RISC OS Open source page](#)

Message_DataOpen

Code &5 · WindowManager

This message is broadcast when the user double clicks on a file.

Message Block

+20 · Window handle of directory viewer

+24 · unused

+28 · X offset of icon clicked

+32 · Y offset of icon clicked

+36 · 0

+40 · Filetypes (&1000 for directories, &2000 for applications, &3000 for untyped files)

+44 · Full pathname

Notes

If the receiver wants to load the file, it should acknowledge this message by returning a Message_DataLoadAck message.

If no task acknowledges the message, the filer will try to run it.

[RISC OS Open source page](#)

Message_DataRequest

Code &10 · WindowManager

This message is broadcast by a task who wants to paste the contents of the clipboard into its own document

Message Block

+20 · Window handle

+24 · Internal handle

+28 · X

+32 · Y

+36 · 4 (send clipboard)

+40 · List of filetypes in order of preference (terminated by -1)

Notes

When the owner of the clipboard receives this message, it should choose the earliest filetype in the list that it can provide (or use native format if none) and reply with a Message_DataSave message.

When the recipient receives the clipboard data, it should check the filetype and just ignore the message if it cannot handle it.

The recipient should ignore this message if bit 2 isn't set.

[RISC OS Open source page](#)

Message_DataSave

Code &1 · WindowManager

This message is used to indicate that a task wants to save data to a receiver.

Message Block

- +20 · Destination window handle
- +24 · Destination icon handle
- +28 · Destination screen x
- +32 · Destination screen y
- +36 · Estimated size
- +40 · Filetype of data (&1000 for directory, &2000 for application, -1 for untyped)
- +44 · Proposed leafname (zero terminated)

Notes

If this message is ignored by the receiver, then the task originator should do nothing.

An application that receives this message should store the 'my_ref' of each DataSave message, to check against any future Message_DataSaved message you might receive.

Normal use:

- User has terminated a drag
- Get the first four words with Wimp_GetPointerInfo
- Send the message to mentioned window

Possible use:

- Task sends Message_DataSaveAck (wishes to use file transfer)
- Task sends Message_RAMFetch (wishes to use memory transfer)
- Task does not reply (not interested in files of a specific filetype)

[RISC OS Open source page](#)

Message_DataSaveAck

Code &2 · WindowManager

This message is used to reply to a Message_DataSave message to indicate that the receiver should save their data to the specified file.

Message Block

- +20 · Destination window handle
- +24 · Destination icon handle
- +28 · Destination screen x
- +32 · Destination screen y
- +36 · Estimated size, or -1 if unsafe
- +40 · Filetype of data
- +44 · Full leafname, or "<Wimp\$Scrap>"

Notes

Words at 20 to 32 are preserved from the original message.

When the filer replies using this message, it replaces the leafname from the DataSave message with the full pathname of the file, and leaves the word at 36 as the estimated file size.

Normal action on receiving a DataSave message:

- Alter the pathname to "<Wimp\$Scrap>"
- Set the word at 36 to -1, to indicate that the file will be temporary
- Send this message in reply

Possible responses from other task:

- Task sends Message_DataLoad message (data written to a specified file)
- Task does not reply (error saving data / not listening)

[RISC OS Open source page](#)

Message_DataSaved

Code &D · WindowManager

This message is sent back to the originator of a saved file if at some later time it becomes 'safe' (i.e. on disc).

Notes

You should compare the 'your_ref' in this message with the 'my_ref' stored when the original Message_DataSave message was sent. If the messages are the same then the file can be marked as safe, unless the user has edited it in the meantime.

Probably the best implementation is for your program to store the 'my_ref' with the file, but zero it on any modification, as 0 will not be a valid 'your_ref'.

It appears that this message is designed for cases where one program saves a file to another program, and then that second program saves the unmodified file to disc.

[RISC OS Open source page](#)

Message_DeviceClaim

Code &B · WindowManager

This message is broadcast by the task wishing to claim exclusive use of a system device.

Message Block

+20 · Major Device Numbers

+24 · Minor Device Numbers

+28 · Information string (zero terminated)

Notes

The information string is zero terminated.

[RISC OS Open source page](#)

Message_DeviceInUse

Code &C · WindowManager

This message is sent in response to a Message_DeviceClaim if the receiver does not want the claimant to gain exclusive use of a device. i.e. it's already using it.

Message Block

+20 · Major Device Numbers

+24 · Minor Device Numbers

+28 · Information string (zero terminated)

Notes

The information string should be used to give information as to why the claim has failed. e.g. the printer is active.

The information string is zero terminated.

[RISC OS Open source page](#)

Message_DragClaim

Code &12 · WindowManager

This message is sent as a reply to the Message_Dragging message to inform the sender that a task can receive the data.

Message Block

+20 · Flags

+24 · List of filetypes in order of preference (terminated by -1)

Notes

None.

[RISC OS Open source page](#)

Message_DragClaim Flags

WindowManager

Flags

Bits / Meaning when set

0 – Meaning when set: Have changed pointer shape

1 – Meaning when set: Remove Wimp dragbox

3 – Meaning when set: Source should be deleted

Notes

None.

[RISC OS Open source page](#)

Message_Dragging

Code &11 · WindowManager

This message is sent continuously by the task which is carrying out a drag process.

Message Block

+20 · Window handle

+24 · Icon handle

+28 · X

+32 · Y

+36 · Flags

+40 · Bounding box

+56 · List of filetypes in order of preference (terminated by -1)

Notes

This message is initially broadcast to all tasks, but after a task claims the message, it is only sent to that task.

Claiming this message is achieved by Message_DragClaim.

[RISC OS Open source page](#)

Message_Dragging Flags

WindowManager

All other bits are reserved and must be zero.

Flags

Bit / Meaning when set

1 – Meaning when set: Sending data from selection

2 – Meaning when set: Sending data from clipboard

3 – Meaning when set: Source data will be deleted

4 – Meaning when set: Do not claim this message

[RISC OS Open source page](#)

Message_FilerAction

Code &405 · WindowManager

Sending this message to a Filer_Action task both specifies and starts the operation to perform on the list of leafnames sent with Message_FilerAddSelection.

+20 · Operation Values

0 · Copy

1 · Move

2 · Delete

3 · Set access

4 · Set type

5 · Count

6 · Move

7 · Copy local

8 · Stamp files
9 · Find file
+24 · Options
+28 · Operation specific data
[RISC OS Open source page](#)

Message_FilerAddSelection

Code &404 · WindowManager

Sending this message to a Filer_Action task adds the leafnames to the task's list of leafnames to act upon.

Message Block
+20 · List of leafnames (space separated and null terminated)

Notes

The leaf names are for files in the directory which has already been specified with Message_FilerSelectionDirectory
[RISC OS Open source page](#)

Message_FilerCloseDir

Code &401 · WindowManager

This message should be broadcast to instruct the filer to close a directory.

Message Block
+20 · Filing System Numbers
+24 · Flags, bit 0 set to prevent canonicalisation of name before using
+28 · Name of directory (zero terminated)

Notes

All directories starting with given name will be closed.
[RISC OS Open source page](#)

Message_FilerControlAction

Code &406 · WindowManager

Sending this message to a Filer_Action task performs various controls.

Message Block
+20 · Control
0 · Acknowledge control message
1 · Verbose on (show the action window)
2 · Verbose off (hide the action window)

Notes

Control 0 can be used to see if the task is still busy.
[RISC OS Open source page](#)

Message_FilerOpenDir

Code &400 · WindowManager

This message should be broadcast to instruct the filer to open a directory.

Message Block

- +20 · Filing System number, or -1 to refresh all Filer windows
- +24 · Flags, bit 0 set to prevent canonicalisation of name before using
- +28 · Name of directory (zero terminated), or 0 if +20 is -1

Notes

Filing system modules use Service_StartFiler to record the Filer's task handle and hence do not use a broadcast message.

[RISC OS Open source page](#)

Message_FilerOpenDirAt

Code &402 · WindowManager

This message is used to open Filer windows but allows the position and mode of the viewer to be specified.

- +20 · Filing System Numbers
- +24 · 0 (flags, all bits reserved)
- +28 · X position
- +32 · Y position
- +36 · Width
- +40 · Height
- +44 · ViewMode
- +45 · Name of directory (zero terminated)

Notes

Message_FilerOpenDir is a similar call but simply opens a filer window on the screen.

[RISC OS Open source page](#)

Message_FilerOpenDirAt ViewMode

WindowManager

These bits specify how a filer window should be displayed.

View Mode

- 0-1** – Value: 0 · Meaning: Large icons
 - Value: 1 · Meaning: Small icons
 - Value: 2 · Meaning: Full info
 - Value: 3 · Meaning: Undefined
- 2-3** – Value: 0 · Meaning: Sort by name
 - Value: 1 · Meaning: Sort by type
 - Value: 2 · Meaning: Sort by size
 - Value: 3 · Meaning: Sort by date

- 4** – Value: 0 · Meaning: Use default display mode
- Value: 1 · Meaning: Use display as specified by bits 0-1
- 5** – Value: 0 · Meaning: Use default sort mode
- Value: 1 · Meaning: Use sort as specified by bits 2-3

[RISC OS Open source page](#)

Message_FilerSelection

Code &407 · WindowManager

This message is sent from the filer to an application when a selection has been dragged, just before the application receives the multiple Message_DataLoad messages.

- +20 · Selection bounding box x0 (screen co-ordinates)
- +24 · Selection bounding box y0 (screen co-ordinates)
- +28 · Selection bounding box x1 (screen co-ordinates)
- +32 · Selection bounding box y1 (screen co-ordinates)
- +36 · Width (of each selected item)
- +40 · Height
- +44 · ViewMode
- +48 · Start column of selection in window
- +52 · Start row
- +56 · End column
- +60 · End row

Notes

None.

[RISC OS Open source page](#)

Message_FilerSelection ViewMode

WindowManager

These bits specify the settings of a filer window when a selection of items have been dragged into an application.

View Mode

- 0-1** – Value: 0 · Meaning: Large icons
- Value: 1 · Meaning: Small icons
- Value: 2 · Meaning: Full info
- Value: 3 · Meaning: Undefined
- 2-3** – Value: 0 · Meaning: Sort by name
- Value: 1 · Meaning: Sort by type
- Value: 2 · Meaning: Sort by size
- Value: 3 · Meaning: Sort by date

[RISC OS Open source page](#)

Message_FilerSelectionDirectory

Code &403 · WindowManager

Sending this message to a Filer_Action task sets the directory to act on, and clears the list of leafnames.

Message Block

+20 · Directory name (null terminated)

[RISC OS Open source page](#)

Message_FontChanged

Code &400CF · WindowManager

On receiving this broadcast message, the Window Manager re-reads the configured WimpFont setting? and redraws the whole screen.

Notes

If any error is encountered with the configured font, then the system font is used.

[RISC OS Open source page](#)

Message_FSLockStateChangeConfirm

Code &50D85 · WindowManager

This message is broadcast by the main application, and received by the Lock plug-in to confirm the lock state change.

Message Block

+0 · Length: 20

+12 · Your_ref: 0

Notes

Exclusive to the central application and the Lock plug-in. All tasks, except the Lock plug-in, must ignore this Message.

It is never sent if one of the other plug-ins objected to being quit.

[RISC OS Open source page](#)

Message_FSLockStateChanged

Code &50D86 · WindowManager

This message is broadcast by the Lock plug-in to announce that the FSLock state has been changed.

Message Block

+0 · Length: 20

+12 · Your_ref: 0

Notes

Exclusive to the central application and the Lock plug-in. All tasks, except the Lock plug-in, must ignore this Message.

It allows, for example, multiple instantiations of the Lock plug-in to update their windows synchronously.

[RISC OS Open source page](#)

Message_FSLockStateChangeRequest

Code &50D84 · WindowManager

This message is broadcast by the Lock plug-in, for the attention of the central application.

Message Block

+0 · Length: 24

+12 · Your_ref: 0

+20 · New lock status (0, 1 or 2)

Notes

Exclusive to the central application and the Lock plug-in. All tasks, except the Lock plug-in, must ignore this Message.

If R1+20 is 2, the central application will quit all other plug-ins and shade their icons in the main window; the plug-ins will have a chance to object using the Quit protocol.

[RISC OS Open source page](#)

Message_HelpEnable

Code &504 · WindowManager

This message can be sent by any application to configure the interactive !Help application.

Message Block

+20 · Flags words

Bit 0 set · Disable interactive Help (put it into suspend state)

Bit 1 set · Force Help to re-read its configuration

[RISC OS Open source page](#)

Message_HelpReply

Code &503 · WindowManager

This message is sent by an application in response to a Message_HelpRequest message.

Message Block

+20 · Help string (zero terminated)

Notes

Maximum length is 235 characters (256-20).

Escape Codes may be used as part of a string.

[RISC OS Open source page](#)

Message_HelpRequest

Code &502 · WindowManager

This message is broadcast by the !Help application.

Message Block

+20 · Mouse x

+24 · Mouse y

+28 · Button state

+32 · Window handle, or

-1 if no window

+36 · Icon handle, or

-1 for no icon

Notes

Any application wishing to provide help on the given window/icon, should reply with the Message_HelpReply message.

[RISC OS Open source page](#)

Message_Iconize

Code &400CA · WindowManager

This message is broadcast by the Wimp when a window's close icon is shift clicked.

Message Block

+20 · Window handle

+24 · Task handle of owner

+28 · 20 bytes of title string (last part of first word)

Notes

If an iconizer is absent nothing will happen. If one is present, it will acknowledge the message, and send Message_WindowInfo to the window.

[RISC OS Open source page](#)

Message_IconizeAt

Code &400D0 · WindowManager

This message is used to specify the exact location on the Pinboard that an icon should be positioned when a window is iconised.

Message Block

+20 · Window handle

+24 · Task handle of window being iconised

+28 · X co-ordinate

+32 · Y co-ordinate

+36 · Flags

Notes

In the absence of this message, Pinboard will just position the window icon beneath the window's close button.

[RISC OS Open source page](#)

Message_IconizeAt Flags

The Window Manager

With bit 0 set, it allows the 'iconise at button' behaviour to function. h4. See also

Flags

Bit / Meaning

20 – Meaning: Window as iconised with a shift-click on close button

Notes

- Message Codes
- Message_IconizeAt
- Wimp_Poll

[RISC OS Open source page](#)

Message_MenusDeleted

Code &400C9 · WindowManager

This message is sent when an applications menu was closed without a selection having been made.

Message_MenuDeleted

Message Block

+20 · Pointer to menu block (Wimp_CreateMenu)

Notes

None.

[RISC OS Open source page](#)

Message_MenuWarning

Code &400C0 · WindowManager

This message is received when the mouse pointer moves over the sub-menu arrow in a menu entry with the 'generate message' bit set.

Message Block

+20 · 'Sub-menu pointer' from menu item

+24 · X co-ordinate of top left new sub-menu

+28 · Y co-ordinate of top left new sub-menu

+32 · Selection list (same as Menu_Selection)

Notes

To determine which window or menu to open using Wimp_CreateSubMenu, the 'sub-menu pointer' and/or the selection list can be used.

The sub-menu that an application opens does not have to be the one passed via +20. However, if you wish the sub-menu to be re-opened when the Wimp recreates the menu the new handle should be stored at +20.

[RISC OS Open source page](#)

Message_ModeChange

Code &400C1 · WindowManager

This message is received when the screen mode has been changed.

Notes

Calling Wimp_SetMode also causes this message to be broadcast.

Applications should not acknowledge this call.

After sending this message, the Wimp sends an Open_Window_Request message for each window.

No window should be deleted. Any window that needs to be deleted should simply be marked as needing deletion and do the necessary work when you receive the Open_Window_Request.

[RISC OS Open source page](#)

Message_Notify

Code &40040 · WindowManager

This message is broadcast by the NetFiler to enable some task to display a *Notify command.

Message Block

+20 · Sending station number

+21 · Station Network number

+22 · Real time on receipt (5 bytes)

+27 · Message

Notes

If no task acknowledges the message, the NetFiler will display it using Wimp_ReportError.

[RISC OS Open source page](#)

Message_OpenConfigWindow

Code &50D83 · WindowManager

This message is generated by the central application when the user clicks on an icon in the main window and the relevant plug-in is still active.

Message Block

+0 · Length: 20

+12 · Your_ref: 0

Notes

The plug-in that receives the message must re-open its window at the top of the window stack in the same x/y position, but not make any other changes (equivalent to an Adjust-click on "Cancel" or otherwise).

[RISC OS Open source page](#)

Message_PaletteChange

Code &9 · WindowManager

This message is broadcast by the palette utility when the RGB value for one or more physical colours have changed.

Notes

In 256 colour modes, it is not the physical colour which changes, but only the mapping from the logical to physical colour. The palette utility itself forces a redraw of the whole screen.

Message_ModeChange is broadcast instead of this message upon a mode change.

[RISC OS Open source page](#)

Message_Plugin_Abort

Code &4D552 · WindowManager

This message is sent by the browser when the user clicks on the Stop icon (or performs its equivalent).

Message Block

+20 · Flags, reserved (must be zero)

+24 · Plugin instance handle

Notes

The plugin should stop as much of its activity as possible. Specifically, anything that updates the screen, anything that uses significant processor time and anything that accesses the network should be suspended.

This message is sent to each plugin instance individually and should be treated as such.

[RISC OS Open source page](#)

Message_Plugin_Action

Code &4D551 · WindowManager

This message is used for sending specific commands to a plugin. Not all plugins will understand the commands sent.

Message Block

+20 · Flags, reserved (must be zero)

+24 · Plugin instance handle

+28 · Browser instance handle (copied from Message_Plugin_Open)

+32 · New state

New State

0 – Meaning: Stop

1 – Meaning: Play

2 – Meaning: Pause

3 – Meaning: Fast Forward

- 4 – Meaning: Rewind
- 5 – Meaning: Record
- 6 – Meaning: Mute
- 7 – Meaning: Unmute

Notes

The new state sent it the state the plugin should enter. If it is already in that state, then it should ignore the message.

After entering the state, it should send back a `Message_Plugin_Busy` message giving the new state except for the Mute and Unmute actions.

[RISC OS Open source page](#)

Message_Plugin_Busy

Code &4D550 · WindowManager

This message is used for requesting that the parent display some indication of business (e.g. spinning logo, etc.).

Message Block

- +20 · Flags
- +24 · Plugin instance handle
- +28 · Browser instance handle (copied from `Message_Plugin_Open`)
- +32 · State

Flags

Value / Meaning

- 0 – Meaning: Set; Plugin is busy
- Clear; Plugin is not busy
- 1 – Meaning: Set; state at R1+32 has some meaning
- Clear; ignore word at R1+32
- 2 - 31 – Meaning: Reserved (must be zero)

State

Value / Meaning

- 0 – Meaning: Stop
- 1 – Meaning: Play
- 2 – Meaning: Pause
- 3 – Meaning: Fast Forward
- 4 – Meaning: Rewind
- 5 – Meaning: Record

Notes

If the plugin has set the busy bit in its `Message_Plugin_Opening` message then it should send this message with bit 0 clear when it has finished its loading.

This is also used to notify the parent of any state change by the plugin in case it needs to update any user interface.

[RISC OS Open source page](#)

Message_Plugin_Close

Code &4D542 · WindowManager

This message is sent to a plugin application if it wants an instance to be closed down. Plugins should reply with `Message_Plugin_Closed`.

Message Block

+20 · Flags

+24 · Plugin instance handle to close

+28 · Browser instance handle

Flags

Bit / Meaning if set

0 – Meaning if set: Browser would also like Plugin to exit. Usually this means that the browser is attempting to reclaim as much memory as possible

All other are Reserved (must be zero)

Notes

You may open as a helper, even if the browser requested an embeded window.

[RISC OS Open source page](#)

Message_Plugin_Closed

Code &4D543 · WindowManager

This message is sent by a plugin to indicate that an instance has been closed. Usually it is a reply to `Message_Plugin_Close` message.

Message Block

+12 · my_ref field from `Message_Plugin_Close` (if applicable)

+20 · Flags

+24 · Plugin instance handle of the closed instance

+28 · Browser instance handle of the closed instance

+32 · Error number (if error flag is set)

+36 · Error message to be displayed by browser (zero-terminated) (if error flag is set)

Flags

Bit / Meaning if set

0 – Meaning if set: Plugin will exit after this message

1 – Meaning if set: This message is not in reply to `Message_Plugin_Close`

2 – Meaning if set: There is an error block at +32

All other are Reserved (must be zero)

Notes

You may open as a helper, even if the browser requested an embeded window.

[RISC OS Open source page](#)

Message_Plugin_Focus

Code &4D546 · WindowManager

This message is used to transfer the input focus between a plugin and its parent. It can be sent in either direction.

Message Block

+20 · Flags, reserved (must be zero)

+24 · Plugin instance handle

+28 · Browser instance handle

Notes

If the recipient cannot or does not wish to accept the focus then it just ignores the message. Otherwise it should acknowledge the message with message type 19 to prevent it being bounced back to the originator.

[RISC OS Open source page](#)

Message_Plugin_Notify

Code &4D54E · WindowManager

This message is sent to indicate the completion of a Message_Plugin_URLAccess message.

+20 · Flags, reserved (must be zero)

+24 · Plugin instance handle

+28 · Browser instance handle

+32 · URL accessed (string_value)

+36 · Reason for Notify:

0: Stream finished successfully

1: Stream finished due to an error

2: Stream finished due to user intervention

+40 · Notify data

Notes

None.

[RISC OS Open source page](#)

Message_Plugin_Open

Code &4D540 · WindowManager

This message is sent by the browser to create a plugin instance.

+20 · Flags

+24 · Reserved

+28 · Browser instance handle (provided by the browser)

+32 · Parent window handle

+36 · Left co-ordinate

+40 · Bottom co-ordinate

+44 · Right co-ordinate

+48 · Top co-ordinate

+52 · File Type
 +56 · Filename (string_value)

Flags

Bit / Meaning if set

0 – Meaning if set: Open as a helper (otherwise as a plugin)

All other are Reserved (must be zero)

Notes

The file specified contains a series of parameters in the form of name-value pairs. This data is the list of attributes and parameters from the APPLET, OBJECT or EMBED tag – see their respective definitions. This data is used by the plugin to understand what is being requested of it.

If bit 0 of the flags word is set then this is a request to open the file as a helper application. i.e. external to the parent application. In this case the bounding box (offset bytes 35 to 51) are invalid. The parent window handle may be valid or 0 depending on how the file is launched.

Usually a plugin application will respond with Message_Plugin_Opening.

The bounding box of the Plugin (as defined by left, bottom, right and top co-ordinates) is the work area co-ordinates within the parent's window.

[RISC OS Open source page](#)

Message_Plugin_Opening

Code &4D541 · WindowManager

This message is sent by a plugin application in response to a Message_Plugin_Open request.

Message Block

+12 · my_ref field from Message_Plugin_Open
 +20 · Flags
 +24 · Plugin instance handle
 +28 · Browser instance handle (copied from Message_Plugin_Open)

Flags

0 – Meaning if set: Plugin can accept input focus (otherwise it cannot)

1 – Meaning if set: Plugin wants code resources fetched (otherwise it will fetch code itself)

2 – Meaning if set: Plugin wants data resources fetched (otherwise it will fetch data itself)

3 – Meaning if set: Plugin will delete the parameter file itself (otherwise the parent should)

4 – Meaning if set: Plugin has more work to do, keep showing busy indicator (otherwise it is loaded)

5 – Meaning if set: Plugin understands Message_Plugin_Action message beyond the stop reason code (otherwise only stop is understood)

6 – Meaning if set: Plugin task has opened a helper window (otherwise plugin has embedded itself)

All other are Reserved (must be zero)

Notes

You may open as a helper, even if the browser requested an embeded window.

[RISC OS Open source page](#)

Message_Plugin_Reshape

Code &4D544 · WindowManager

This message is sent by the browser to notify the plugin that it has been re-sized.

+12 · my_ref field from Message_Plugin_Reshape_Request (if applicable)
+20 · Flags (must be zero)
+24 · Plugin instance handle
+28 · Browser instance handle
+32 · Parent window handle
+36 · Left co-ordinate
+40 · Bottom co-ordinate
+44 · Right co-ordinate
+48 · Top co-ordinate

Notes

If a plugin needs resizing it should not just do so, but it should ask using Message_Plugin_Reshape_Request. The browser will then respond with this message to determine the new location.

The bounding box of the Plugin (as defined by left, bottom, right and top co-ordinates) is the work area co-ordinates within the parent's window.

[RISC OS Open source page](#)

Message_Plugin_Reshape_Request

Code &4D545 · WindowManager

This message is sent by the Plugin to request that it be resized in its parent window.

Message Block

+20 · Flags, reserved (must be zero)
+24 · Plugin instance handle
+28 · Browser instance handle
+32 · Width (in OS units)
+36 · Height

Notes

Plugins should use this to resize their windows rather than just issuing Wimp_OpenWindow every time they need to resize, although you might perform the Wimp_OpenWindow as well as this message. The effect of resizing prior to receiving the reply is something you will have to judge for yourself.

The browser will respond with Message_Plugin_Reshape, but because the windows may be reformatted, this message may not arrive for a period.

[RISC OS Open source page](#)

Message_Plugin_Status

Code &4D54F · WindowManager

This message is sent by the plugin to ask the browser to display some information as a status message.

Message Block

- +20 · Flags, reserved (must be zero)
- +24 · Plugin instance handle
- +28 · Browser instance handle
- +32 · Status message (string_value)

Notes

The message should be reasonably short.

[RISC OS Open source page](#)

Message_Plugin_StreamAsFile

Code &4D54C · WindowManager

This message is sent when a stream is being passed as a temporary file.

- +20 · Flags, reserved (must be zero)
- +24 · Plugin instance handle
- +28 · Browser instance handle
- +32 · Plugin stream instance handle
- +36 · Browser stream instance handle
- +40 · URL of stream source/destination (string_value)
- +44 · End of stream in bytes, or 0 if unknown
- +48 · Last modified date of URL (in UNIX time format)
- +52 · Notify data
- +56 · File name of stream data (string_value)

Notes

None.

[RISC OS Open source page](#)

Message_Plugin_StreamDestroy

Code &4D549 · WindowManager

This message is used to notify that a plugin stream is to be destroyed.

- +20 · Flags, reserved (must be zero)
- +24 · Plugin instance handle
- +28 · Browser instance handle
- +32 · Plugin stream instance handle
- +36 · Browser stream instance handle
- +40 · URL of stream source/destination (string_value)
- +44 · Length of stream in bytes, or 0 if unknown
- +48 · Last modified date of URL (in UNIX time format)

+52 · Notify data

+56 · Reason

Reason

Value / Meaning

0 – Meaning: Stream finished successfully

1 – Meaning: Stream finished due to an error

2 – Meaning: Stream finished due to user intervention

Notes

None.

[RISC OS Open source page](#)

Message_Plugin_StreamNew

Code &4D548 · WindowManager

This message is sent by either the Plugin or the Browser to notify the other that a new stream has been created.

+20 · Flags

+24 · Plugin instance handle

+28 · Browser instance handle

+32 · Plugin stream instance handle

+36 · Browser stream instance handle

+40 · URL of stream source/destination (string_value)

+44 · Length of stream in bytes, or 0 if unknown

+48 · Last modified date of URL (in UNIX time format)

+52 · Notify data

+56 · MIME type of URL (string_value)

+60 · Window target (string_value)

Flags

Bit / Meaning

0 - 3 – Meaning: stream type field:

0 = normal

1 = seek only

2 = as file

3 = as file only

4 = stream is seekable

All other are Reserved (must be zero)

Notes

None.

[RISC OS Open source page](#)

Message_Plugin_StreamWrite

Code &4D54A · WindowManager

This message is used to write data to a stream.

- +20 · Flags
- +24 · Plugin instance handle
- +28 · Browser instance handle
- +32 · Plugin stream instance handle
- +36 · Browser stream instance handle
- +40 · URL of stream source/destination (string_value)
- +44 · Length of stream in bytes, or 0 if unknown
- +48 · Last modified date of URL (in UNIX time format)
- +52 · Notify data
- +56 · Logical offset in stream of data
- +60 · Length of data
- +64 · Data pointer

Flags

Bit / Meaning

0 - 3 – Meaning: Data type field :

0 = data in message

1 = anchor (data pointer + offset)

2 = file handle

All others are Reserved (must be zero)

Notes

You may open as a helper, even if the browser requested an embeded window.

[RISC OS Open source page](#)

Message_Plugin_StreamWritten

Code &4D54B · WindowManager

This message is sent as a reply to Message_Plugin_StreamWrite, to confirm that the data was written.

- +12 · my_ref field from Message_Plugin_StreamWrite
- +20 · Flags, reserved (must be zero)
- +24 · Plugin instance handle
- +28 · Browser instance handle
- +32 · Plugin stream instance handle
- +36 · Browser stream instance handle
- +40 · URL of stream source/destination (string_value)
- +44 · Length of stream in bytes, or 0 if unknown
- +48 · Last modified date of URL (in UNIX time format)
- +52 · Notify data
- +56 · Length of data consumed, <0 if error

Notes

None.

[RISC OS Open source page](#)

Message_Plugin_Unlock

Code &4D547 · WindowManager

This message is is reserved.

Message Block

- +20 · Flags, reserved (must be zero)
- +24 · PlugIn instance handle
- +28 · Browser instance handle
- +32 · URL for which to unlock the cache file (string_value)

Notes

Do not use this Message.

[RISC OS Open source page](#)

Message_Plugin_URLAccess

Code &4D54D · WindowManager

This message is sent to a browser to perform various actions on a URL.

- +20 · Flags
- +24 · PlugIn instance handle
- +28 · Browser instance handle
- +32 · URL to access (string_value)
- +36 · Window to target (string_value), or 0 to return to the plugin
- +40 · Notify data to be returned
- +44 · Length of data to be POSTed
- +48 · Filename (string_value) (if Bit 2 of flag is set)
- Pointer to data (string_value) (if Bit 2 of flag is clear)

Flags

Bit / Meaning

0 – Meaning: Set; return a Message_Plugin_Notify message on completion

1 – Meaning: Set; POST to the URL

Clear; GET from the URL

2 – Meaning: Set; POST a file (only if bit 1 is set)

Clear; POST a block of memory (only if bit 1 is set)

All others are Reserved (must be zero)

Notes

If the window target is "" then the URL is fetched to the window containing the plugin. If the window target is "" then it will be fetched to the given window name. If the target value is actually 0 then a stream is opened and data is sent to the plugin.

[RISC OS Open source page](#)

Message_PlugInQuit

Code &50D80 · WindowManager

This message informs plug-ins that they are about to be quit, unless one of their number aborts the process.

+0 · Length: 24 (or more to indicate additional data is present)

+20 · Your_ref: 0

+20 · Plug-in system descriptor, consisting of:

&50DBF for the main Configure plug-in system

&50DBE for the Boot window plug-in system

&50DBD for the Fonts window / FontMerge dialogue

Other values may be used, but should be kept within the relevant chunk

+24 · The Boot and Fonts plug-in systems have bit 1 of this word set to indicate that the user clicked on "Set" rather than "Cancel"

Notes

The word at R1+20 allows this message to be used in other cases than Configure plug-ins; if a task receives this message, but does not recognise the word, it must ignore the message. Similarly, if the plug-in was invoked without the -openat switch, it must also ignore this message, as it will not be receiving a subsequent Message_Quit.

[RISC OS Open source page](#)

Message_PlugInQuitAbort

Code &50D82 · WindowManager

This message is used between plug-ins so that a plug-in can be aware that the quit process has been aborted by another.

Message Block

+0 · Length: 24

+12 · Your_ref: 0

+20 · Plug-in system descriptor

Notes

It must reset its internal "complain" flag so that subsequent quit requests can also be complained at. The word at R1+20 allows this message to be used in other cases than Configure plug-ins; if a task receives this message, but does not recognise the word, it must ignore the message.

[RISC OS Open source page](#)

Message_PlugInQuitContinue

Code &50D81 · WindowManager

This message is used by plug-ins to tell the central application to resume the quitting process if it has been halted while the plug-in requests confirmation from the user.

Message Block

+0 · Length: 24

+12 · Your_ref: 0

+20 · Plug-in system descriptor

Notes

The word at R1+20 allows this message to be used in other cases than Configure plug-ins; the plug-in must fill it in correctly, or the central application will ignore the message (or worse, take the wrong subsequent action).

[RISC OS Open source page](#)

Message_PreQuit

Code &8 · WindowManager

This message is used to notify a task that it is about to be Quit.

Message Block

+20 · Flag, bit 0 clear if desktop shutdown, otherwise set

Check block length - if < 20 assume a flag word of 0

Notes

This message is sent to a specific task when the user tries to quit it from the task display menu.

This message is also broadcast to all tasks when the computer is being shut-down.

If the task has no objections to be quit, it should simply ignore the message. It will then receive a Message_Quit message.

An application that does not wish to quit, perhaps due to unsaved data, should acknowledge this message using Wimp_SendMessage, then open a dialogue box allowing the data to be first saved.

If bit 0 is unset. i.e. the message has been broadcast due to the system being shutdown. The application should re-start the shutdown process again. This can be easily achieved by emitting a Ctrl-Shift F12 key press using Wimp_ProcessKey.

[RISC OS Open source page](#)

Message_Quit

Code &0 · WindowManager

On receiving this message, a task should tidy up its memory workspace and exit. This is achieved by calling Wimp_CloseDown, then OS_Exit

Notes

An application that does not want to quit should respond to Message_PreQuit. If the task does not respond to the Message_PreQuit message it will receive the Message_Quit message.

All tasks must respond to this message.

[RISC OS Open source page](#)

Message_RAMFetch

Code &6 · WindowManager

Tasks should send this message either in response to a Message_DataSave message, indicating that you want to use memory transfer, or in response to a Message_RAMTransmit message, to request more data.

Message Block

+20 · Pointer to buffer where receiver should put data

+24 · Size of buffer

Notes

This message should always be sent as a User_Message_Recorded, so that if it is unanswered the task can either revert to the file transfer protocol, or abort the data transfer.

Before sending this message, it is important to allocate a buffer for the data-saving task to write data to. You can use the estimated data size from the Message_DataSave message, but you should be prepared for more (or less) data to actually be sent.

If you receive a Message_RAMTransmit message where the data-saving task has filled the buffer (data transferred=buffer size), then you should reply with this message. If you receive a Message_RAMTransmit where the data-saving task has not filled your buffer, then this marks the successful conclusions of the transfer.

If you cannot process the received data, you should report an error and abandon the transfer.

Possible responses from the data-saving tasks:

- It sends Message_RAMTransmit (data successfully written to specified buffer)
- It does not reply (memory transfer not supported/error writing data)

If this message returns unanswered:

- If no reply is received to the first RAMFetch, file transfer should be used instead (send a Message_DataSaveAck).
- If there is no reply to subsequent messages, then the transfer should be abandoned silently (the other end will already have reported an error)

[RISC OS Open source page](#)

Message_RAMTransmit

Code &7 · WindowManager

Tasks should send this message in response to receiving a Message_RAMFetch message, and after subsequently writing the data to the given buffer, using Wimp_TransferBlock.

Message Block

+20 · Pointer to buffer where receiver should put data

+24 · Size of buffer

Notes

Since you cannot guarantee that the buffer offered is large enough for all your data, you should keep a record (across Wimp_Polls) of how much data has been transmitted so far.

If the number of bytes written doesn't fill the buffer, then this will be the last message in the dialogue. You should not expect a reply, sending it as a User_Message.

If the buffer does get filled, then you should send this message as a User_Message_Recorded and expect further Message_RAMFetch messages.

Possible responses from the data-saving tasks:

- It sends Message_RAMTransmit (data successfully written to specified buffer)
- It does not reply (memory transfer not supported/error writing data)

Possible responses from the data-loading task:

- It send another Message_RAMFetch message (request more data)
- It does not reply (error reading data / transfer finished)

If this message returns unanswered as a User_Message_Acknowledge, then you should abort the data transfer and report an error.

[RISC OS Open source page](#)

Message_ReleaseEntity

Code &13 · WindowManager

This message should be broadcast to all tasks when the sender's task has the clipboard and is about to quit.

Message Block

+20 · Flags

Notes

None.

[RISC OS Open source page](#)

Message_ReleaseEntity Flags

WindowManager

All other bits are reserved and must be set to zero.

Flags

Bit / Meaning when set

2 – Meaning when set: Clipboard being released

[RISC OS Open source page](#)

Message_SaveDesktop

Code &A · WindowManager

This message is broadcast by the task manager when the user has selected "Save boot file", or on exit from desktop if the system variable SaveDesk\$File has a value. i.e. a path name.

Message Block

+20 · File handle of desktop file being written as a word

+24 · Flags, at present reserved (must be 0)

[RISC OS Open source page](#)

Message_SetSlot

Code &400C5 · WindowManager

This message is broadcast when the user tries to alter a task's slot size by dragging the 'memory bar' in the Task Manager.

Message Block

+20 · New slot size

+24 · Task handle of owner

Notes

If a task allows for changing the memory workspace via the Task Manager, it should first check the task handle, acknowledge the message and then call Wimp_SlotSize.

This message is also sent by the Task Manager when a task is started. If this or any later message is not acknowledged, the task will get a green 'memory bar' in the Task Manager (instead of a red one) and will not receive this message again; it will not be possible for the user to alter the task's slot size.

[RISC OS Open source page](#)

Message_Shutdown

Code &E · WindowManager

This message is broadcast when TaskManager_Shutdown is called with bit 3 of R0 set, to indicate a forced shutdown.

Message Block

+20 · 0 (all flags reserved)

[RISC OS Open source page](#)

Message_SlotSize

Code &400C4 · WindowManager

This message is broadcast after Wimp_SlotSize.

Message_SlotSize

Message Block

+4 · Task handle which owns slot

+20 · New slot size

+24 · New size of the 'Next' slot

Notes

It is mainly used by the Task Manager to update its display.

This message should not be acknowledged.

[RISC OS Open source page](#)

Message_TaskCloseDown

Code &400C3 · WindowManager

This message is broadcast when a task calls Wimp_CloseDown.

Message_TaskCloseDown

Message Block

+4 · Task handle

Notes

The fields are set so that it appears to originate from the dying task.

[RISC OS Open source page](#)

Message_TaskInitialise

Code &400C2 · WindowManager

This message is broadcast when a task calls Wimp_Initialise.

Message Block

+4 · Task handle

+20 · Currently Active Object Pointer

+24 · Slot size

+28 · Task name given to Wimp_Initialise (control terminated)

Notes

The fields are set so that it appears to originate from the new task.

The Task Manager uses this message to identify the task's name. These names are stored in a table which can be queried using TaskManager_TaskNameFromHandle.

[RISC OS Open source page](#)

Message_TaskNamels

Code &400C7 · WindowManager

This message is sent by the Task Manager in response to a Message_TaskNameRq message.

Message Block

+20 · Handle of task

+24 · Task's slot size

+28 · Task name

Notes

This message returns the name of the task (as shown in the Task Manager's display).

[RISC OS Open source page](#)

Message_TaskNameRq

Code &400C6 · WindowManager

This message can be broadcast to find out the name of another task.

Message_TaskNameRq

Message Block

+20 · Handle of other task

Notes

The Wimp will respond to this message via the Message_TaskNameIs message.

[RISC OS Open source page](#)

Message_TaskStarted

Code &400C8 · WindowManager

This message is sent by the Filer once it has started up all of the desktop filers so that the Task Manager can 'renumber' it.

Notes

This renumbering of filers is carried out so that during the 'SaveDesktop' sequence, the Filer's own commands can be performed after any network logons.

[RISC OS Open source page](#)

Message_WindowClosed

Code &400CB · WindowManager

This message is broadcast when a window is closed, so that the iconiser can remove the icon.

Message Block

+20 · Window handle

Notes

None.

[RISC OS Open source page](#)

Message_WindowInfo

Code &400CC · WindowManager

This message is sent by the iconiser to an application when one of its windows is iconised, to find which sprite and name to give the icon.

Message Block

+20 · Window handle

+24 · 0 (Reserved)

+28 · Sprite name (minus the 'ic_' prefix)

+36 · String to use (10 characters or less and no spaces)

Notes

The sprite specified must be located in the Wimp Sprite Pool. If the sprite cannot be found, it will use the sprite 'ic_?'.

The string specified will be truncated at a space or at the 10th character, whichever is shorter.

[RISC OS Open source page](#)

Reference topics

Architecture, file formats, constants, vectors, data tables, and module introductions.

!Alarm

List of modules

The !Alarm module contains the !Alarm application which is accessed from the Apps folder, , on the left hand side of the iconbar. An analogue or digital clock is displayed on the right hand side of the iconbar. Alarms can be set and saved.

Details

Software information: Alarm

[RISC OS Open source page](#)

!Chars

List of modules

The !Chars module contains the !Chars application which is accessed from the Apps folder on the left hand side of the iconbar.

In practice

The !Chars application is a character viewer showing all the characters available in the selected font. The System Font is shown initially, click menu over the main window to see the full font list. Clicking on a character emulates a key press which allows characters to be entered that are not directly available on the keyboard; for example accented characters or special symbols.

For use with writeable icons hover the mouse pointer over the required character and press the shift key.

Details

Software information: Chars

[RISC OS Open source page](#)

!Draw

List of modules

The !Draw module contains the !Draw application which is accessed from the Apps folder on the left hand side of the iconbar.

In practice

!Draw is a vector drawing application allowing the creation and editing of structured drawings that contain lines and curves. It saves in RISC OS's DrawFile format.

Text and images can be imported from !Edit and !Paint.

Details

Software information: Draw

[RISC OS Open source page](#)

!Edit

List of modules

The !Edit module contains the !Edit application which is accessed from the Apps folder on the left hand side of the iconbar.

In practice

!Edit is a plain text editor.

It can also create and edit BASIC programs.

Details

Software information: Edit

[RISC OS Open source page](#)

!Help

List of modules

The !Help module contains the !Help application which is accessed from the Apps folder on the left hand side of the iconbar.

In practice

The !Help application provides interactive help. Hover the mouse pointer over an object to find out information about that object.

It can be temporarily disabled by clicking on its iconbar icon which will grey out. A further click re-enables it.

Details

Software information: Help

[RISC OS Open source page](#)

!Paint

List of modules

The !Paint module contains the !Paint application which is accessed from the Apps folder on the left hand side of the iconbar.

In practice

The Paint application allows the creation and editing of sprite files. A sprite file contains one or more graphic images, each stored as a bitmap.

Details

Software information: Paint

[RISC OS Open source page](#)

About RISC OS

RISC OS is an Operating System that was originally developed by Acorn Computers Ltd to take advantage of the ARM microprocessor for a range of high end personal computers. In 2003, Castle Technology Ltd purchased the intellectual property to RISC OS, and in 2006 RISC OS Open was formed to facilitate the process of publishing the sources.

In practice

In 2016, RISC OS Developments was formed to provide investment and development in RISC OS and its software, as well as introduce the operating system into new markets. In 2018, RISC OS Developments announced that they had acquired the intellectual property to RISC OS from Castle Technology. RISC OS was also fully open sourced under the Apache 2.0 licence.

RISC OS 5 is the fifth major iteration of the Operating System. The sources and softload images are available [here](#).

RISC OS is pronounced 'risk oh ess', but should be typed as RISC OS, as it is an acronym for Reduced Instruction Set Computer Operating System.

History

Although RISC OS personal computers were first released in 1987, the OS includes code to provide backwards compatibility with an earlier, now obsolete, Operating System called Arthur.

The first version of RISC OS was – strangely – called RISC OS 2, perhaps because the earlier Arthur Operating System was version 1. Over the years, RISC OS was updated to reflect the needs of users, as well as to take advantage of improvements in hardware. A list of all major versions of RISC OS is available [here](#).

RISC OS Developments Ltd currently own the intellectual property of RISC OS, and the sources were made available under the Apache 2.0 licence in October 2018, via the RISC OS Open website. Another stream of RISC OS was developed by a commercial company called RISCOS Ltd until 2009, though this stream branched off from Castle's version around version 4.xx and so the source code to RISC OS 5 is substantially different to that version. Features in one stream may not necessarily exist in the other, and vice-versa.

This Programmers Reference Manual is for RISC OS 5, and includes information for previous versions of the Operating System. It contains only passing references pertaining to RISCOS Ltd's stream where for example a discrepancy is known.

Licences

RISC OS and related software available from the RISC OS Open website are made available under various licence schemes. Please do not assume that all of the software that you can download from the site is published under the same licence terms. More information is available [here](#).

Facilities

RISC OS is a mature and sophisticated Operating System. Broadly speaking, its facilities can be categorised as follows:

- Software Interrupts, or SWIs for short, are a set of powerful interfaces for the developer to interact with the Operating System. You can think of these as low level APIs.
- A range of *Commands that can be issued from the command line and from applications to perform a range of simple tasks.
- Low-level routines, such as interrupt handling.

Hardware Abstraction Layer

RISC OS 5 implemented a Hardware Abstraction Layer (HAL) in order to provide a level of independence from any given hardware platform. Originally, RISC OS was designed to run on a small range of ARM hardware using proprietary chipsets from Acorn Computers Ltd. As the availability of many different ARM processors from a range of semiconductor companies increased, more abstraction became desirable. For more information on the HAL, see [here](#). For more information on supported hardware platforms, see [here](#).

Modules

RISC OS has a kernel that is responsible for the overall control of the Operating System. Extra functionality is provided by various pieces of code called system extension modules or modules for short. Each module is responsible for a specific task, such as font handing, filer windows or sound scheduling. The end result is a modular Operating System that can be easily developed, updated and extended.

Each module conforms to a standard format so that the Operating System can register and integrate each one into the overall system. Because each module is self-contained, developers can easily add new modules to the system or create replacement modules. Modules developed to replace existing modules must provide the same entry points, and return values in exactly the same way as the original to ensure compliance with existing software. More information on modules is available [here](#).

Vectors

Another important aspect of RISC OS is vectors. A vector is a chain of entries that determines which part of the system will perform a specific function. Vectors can be claimed and redirected so replacement code performs the required task. Like replacement modules, any claimed vectors must be handled by the replacement code in the same way. This is achieved by the use of SWIs, as these specify the exact input and output of each vector.

Some vectors are used by just one SWI, while others are used by several SWIs because they perform a similar function. It must be noted, however, that some vectors are not used by SWIs at all, but instead only by the Operating System itself. This means that developers cannot claim these vectors. More information on software vectors can be found in [here](#). Information on hardware vectors can be found [here](#).

Programming Languages and Build Tools

RISC OS was designed to achieve a high level of performance even on slow ARM processors, and this was achieved by writing key parts of the Operating System using ARM assembly language. Much of the kernel and driver software is written in hand-optimized ARM assembler, making it a high performance OS solution for the ARM architecture and allowing it to operate in a minimal memory footprint. Other parts of the Operating System are written in C.

As the performance of ARM processors has increased significantly since the introduction of RISC OS, the preferred language to use when developing the Operating System and new software is now C/C++. To develop and build RISC OS from the sources, the RISC OS development kit is required, which includes the full C/C++ compiler, build environment and other useful tools. This is available for purchase [here](#).

Loading the Operating System

Originally, RISC OS was supplied on ROM chips, rather than the more traditional method of hard disc or floppy. The benefit of this was twofold:

- Loading the Operating System from ROM was faster than loading it into RAM memory via hard disc or floppy
- Having the Operating System stored on ROM meant it could not be lost or damaged by viruses or malicious software

With the introduction of fast and inexpensive memory and hard discs, RISC OS can now be soft-loaded from disc, rather than from ROM. This has the benefit of making it easier to update the Operating System as improvements are made. Soft-loading RISC OS can be achieved by either the ROM-based version of RISC OS, or an earlier boot loader such as U-Boot (the Universal Bootloader). This is useful when porting RISC OS to new ARM hardware platforms.

[RISC OS Open source page](#)

About the Programmer's Reference Manuals

About the Programmer's Reference Manuals

The Programmer's Reference Manuals (PRMs) have been written to provide information on the inner working of the RISC OS operating system. With this information you will be able to understand and develop programs that run on RISC OS.

In practice

The complete PRMs are not yet available online, though the task of adding to them is ongoing. All documentation is editable, and information on how to add or edit content can be found here. Feedback and discussion on the PRMs is welcome. The preferred method to use the General Forum. You will need to register an account at this website (free to do so) before you are able to post.

Numbers

Hexadecimal numbers are preceded by the ampersand sign (&). Where appropriate, the decimal equivalent may also be shown: 15 (&0F).

Code

Code (pseudo or otherwise) is shown within a bounding box with a fixed size typeface:

```
REM This is example source code
PRINT "Looping from 1 to 10."
FOR temp = 1 TO 10
PRINT "Count: " temp
NEXT temp
```

Syntax

Throughout the manual, symbols are used to represent the syntax.

Italics · You must substitute the symbol with an actual variable/object name. For example, filename means that you must supply an actual filename

[optional] · Braces indicates that the item enclosed is optional. For example, [filename] shows that you may omit the filename

on|off · A bar indicates that either value may be used. For example, on|off means that you must supply the value on or off

[RISC OS Open source page](#)

AcornHTTP

List of modules

The AcornHTTP module is the protocol module for handling HTTP requests which sits behind the URL_Fetcher module. It also provides handling for HTTPS requests, relying on AcornSSL for the encryption.

In practice

The module also provides facilities for managing cookies in a global cookie store. The presence of SetCookie in the response header is indicated by setting bit 16 of R0 on return from URL_ReadData. Clients can then work through the incoming cookie queue using HTTP_EnumerateCookies and choose whether to add each one to the global store in memory or to discard by using HTTP_ConsumeCookie. Cookies can also be added directly to the global store using HTTP_AddCookie. Once a cookie has been added to the store, it will remain until it expires, and the appropriate Cookie headers will be added to any request made through URL_GetURL automatically. Incoming cookies which have been queued

but which have not been added to the store are not included in outgoing requests.

An environment variable, `Browse$AcceptAllCookies`, can be set to cause all incoming cookies to be added to the store. If the value is "off" or the variable is unset, then the cookies are queued for evaluation.

On initialisation the module will read any saved cookies from `<Choices$Write>.WWW.AcornHTTP.Cookies` if it exists, or the file named in `Browse$CookieFile` as a fallback. The module writes persistent cookies to `<Choices$Write>.WWW.AcornHTTP.Cookies` whenever an individual cookie is accepted into the store through `HTTP_ConsumeCookie` or whenever cookies are read on an incoming response if `Browse$AcceptAllCookies` is set. The cookie file is also written on calls to `HTTP_AddCookie`, but as that call cannot currently create persistent cookies the file will only change if a cookie is caused to be removed from the store, by reaching its expiry point or being replaced by the newly-added cookie.

The format of the written cookie file is one of two formats, both also compatible with Fresco, an early RISC OS web browser. If `Browse$CookieFileFormat` is set to 1, the format used is similar to the Set-Cookie response header. Normally the module defaults to format version 2, a tab-separated value file, despite the history file describing this as having "no apparent logic".

Details

- SWI Calls

[RISC OS Open source page](#)

AcornHTTP SWI Calls

AcornHTTP

SWI calls for internal use by `URL_Fetcher` to provide support for the HTTP protocol:

In practice

- `HTTP_GetData`
- `HTTP_Status?`
- `HTTP_ReadData?`
- `HTTP_Stop?`

SWI calls for internal use by `URL_Fetcher` to provide support for the HTTPS protocol:

- `HTTP_SecureGetData?`
- `HTTP_SecureStatus?`
- `HTTP_SecureReadData?`
- `HTTP_SecureStop?`

Other SWI calls, not reserved for internal use by `URL_Fetcher`:

- `HTTP_RegisterMethod`
- `HTTP_DeregisterMethod`
- `HTTP_AddCookie`
- `HTTP_ConsumeCookie`
- `HTTP_EnumerateCookies`

[RISC OS Open source page](#)

AcornSSL

List of modules

This page, and the pages under it, are Work In Progress. Assistance and feedback is welcome!

In practice

The AcornSSL module provides a BSD-like socket interface that allows communication through the TCP/IP stack to resources requiring a secure connection.

It uses the mbedTLS library for the majority of its implementation, the module merely calls the corresponding functions and wraps them into a set of SWIs. At the time of writing this supports all current SSL and TLS standards:

- TLS version 1.0
- TLS version 1.1
- TLS version 1.2

The earlier SSL version 2 and 3 standards are now deemed too insecure and are not supported at all, while TLS version 1.0 is likely to be withdrawn. The client SWI interface offers no mechanism to select which is used, so attempting to connect to a server that uses one of the insecure methods will fail.

Details

- SWI Calls
- Service Calls

[RISC OS Open source page](#)

Address Exception Handler

OS_ChangeEnvironment

The address exception handler is entered in the following state:

In practice

- The processor will be in SVC26 mode, with IRQs disabled
- FIQ state will match that at the time of the exception
- R14 will contain the exception address and saved PSR, corresponding to when the exception was raised by the CPU. R14 will be pointing two instructions (i.e. 8 bytes) after the instruction that caused the exception.
- The other core registers will be as per when the exception was raised by the CPU. However if your code examines the registers it must take care to use the correct register bank (e.g. if the exception was from user mode, it should look at R14_usr, not R14_svc).
- R13 will be a valid stack pointer

If the handler is capable of handling the exception (e.g. by emulating the memory access) then it can resume execution by performing a suitable exception return operation.

Notes

Address exceptions are only capable of being generated on MEMC-based hardware; on RISC OS 3.5 and later this environment handler is unused.

The kernel's default handler will save a copy of the ARM integer registers (corresponding to the CPU mode the exception occurred in) to the Exception Registers Block, reset the stack pointers for all the privileged CPU modes, and then call OS_GenerateError to report an Address Exception error.

[RISC OS Open source page](#)

Alphabet Numbers

Alphabet numbers must have bit 7 clear. Numbers larger than 255 require 3rd party support such as Cerilica BigAlphas5.

- 100 · **BFont** – Base Standard3: BBC Master1 · Encoding name: BFont2
- 101 · **Latin1** – Base Standard3: ISO 8859-1 · Encoding name: Latin1
- 102 · **Latin2** – Base Standard3: ISO 8859-2 · Encoding name: Latin2
- 103 · **Latin3** – Base Standard3: ISO 8859-3 · Encoding name: Latin3
- 104 · **Latin4** – Base Standard3: ISO 8859-4 · Encoding name: Latin4
- 105 · **Cyrillic** – Base Standard3: ISO 8859-5 · Encoding name: Cyrillic
- 106 · **Arabic** – Base Standard3: Acorn specific1 4 · Encoding name: Arabic
- 107 · **Greek** – Base Standard3: ISO 8859-7 · Encoding name: Greek
- 108 · **Hebrew** – Base Standard3: ISO 8859-8 · Encoding name: Hebrew
- 109 · **Latin5** – Base Standard3: ISO 8859-9 · Encoding name: Latin5
- 110 · **Welsh** – Base Standard3: ISO IR-182 · Encoding name: Welsh
- 111 · **UTF8 6** – Base Standard3: Unicode · Encoding name: UTF8
- 112 · **Latin9** – Base Standard3: ISO 8859-15 · Encoding name: Latin9
- 113 · **Latin6** – Base Standard3: ISO 8859-10 · Encoding name: Latin6
- 114 · **Latin7** – Base Standard3: ISO 8859-13 · Encoding name: Latin7
- 115 · **Latin8** – Base Standard3: ISO 8859-14 · Encoding name: Latin8
- 116 · **Latin10** – Base Standard3: ISO 8859-16 · Encoding name: Latin10
- 120 · **Cyrillic2** – Base Standard3: DOS code · Encoding name: Cyrillic2
- 121 · **Hebrew2** – Base Standard3: ISO 8859-8+ · Encoding name: Hebrew2

Allocated 3rd Party Alphabet Numbers

- 85 · **ISO8859-6** – Standard: ISO 8859-6 · Encoding name: ISO8859-6 · Allocated to: Cerilica
- 95 · **PDFDocEnc** – Standard: PDF Document Encoding · Encoding name: PDF.DocEncoding · Allocated to: Cerilica
- 96 · **PDFWinAnsi** – Standard: PDF WinAnsi Encoding · Encoding name: PDF.WinAnsiEncoding · Allocated to: Cerilica
- 97 · **PDFStdEnc** – Standard: PDF Standard Encoding · Encoding name: PDF.StandardEncoding · Allocated to: Cerilica
- 98 · **PDFMacRom** – Standard: PDF Mac Roman Encoding · Encoding name: PDF.MacRomanEncoding · Allocated to: Cerilica
- 99 · **CerilicaL1** – Standard: Cerilica Latin1 Encoding · Encoding name: Cerilica.Latin1 · Allocated to: Cerilica
- 117 · **UTF16 6** – Standard: Unicode · Encoding name: n/a (16b, multicode) · Allocated to: Cerilica
- 118 · **Unicode** – Standard: Unicode · Encoding name: n/a (21b) · Allocated to: Cerilica
- 122 · **ShiftJIS 6** – Standard: JIS X 201/208 · Encoding name: n/a (8b, multicode) · Allocated to: Cerilica

Notes

1 Are assigned but not available in RISC OS 5's International module.

2 3rd party definition

3 In most cases the RISC OS alphabet is an extension or modification of the base standard. The UTF8 alphabet is 100% equivalent to its base standard.

4 Acorn's Arabic Alphabet was not ISO-compatible so was not carried over to Arthur from the Master 128. ISO 8859-6 has been placed at 85 for compatibility with ISO6. Cerilica ArabicMod implements both Arabic Alphabets.

5 For example, CP437 is &437 and Macrons is &AF00.

6 3rd Party implementations may support multicode sequences. See Service_International 332 for character code mask and multicode flags.

[RISC OS Open source page](#)

Application Space Handler

OS_ChangeEnvironment

The application space environment handler is not a code entry point. Instead, the value specified in R1 to OS_ChangeEnvironment corresponds to the real size of application space, disregarding any soft limit set via the Memory Limit Handler.

Notes

The value in this handler is an absolute address, i.e. &8000 corresponds to 0 bytes of application space.

Software must not use this environment handler to set the size of application space – setting a value to this handler does not actually cause any memory to be moved, and will cause the size of the dynamic area to become inconsistent with the actual memory map. Instead, OS_ChangeDynamicArea (for single-tasking programs) or Wimp_SlotSize (for multi-tasking programs) should be used to resize application space.

[RISC OS Open source page](#)

Archimedes Hardware

ARM Hardware

This section details the intricacies of the early hardware used by RISC OS. A number of different models of RISC OS computers are classified as Archimedes. Each Archimedes model of computer had an ARM 2 or ARM 3 as the CPU. Several different chips supported the ARM processor. These chips were designed by Acorn Computers Ltd, and were proprietary chips.

In practice

RISC OS 2 and RISC OS 3.11 were used with this class of hardware. The information is no longer valid or useful with RISC OS 5, but is available here for completeness.

Chips

The list of chips used within the original Acorn Archimedes range of computers is as follows:

- ARM processor is the CPU
- VIDC chip which handled both video and sound output. VIDC is short for Video Controller
- IOC chip handles the peripherals and interrupts. IOC is short for Input/Output Controller
- MEMC chip is used as the interface between the ARM, VIDC and IOC and memory. MEMC is short for Memory Controller

ARM Processor

The ARM processor chips used with the Archimedes range of computers were the ARM 2 and ARM 3 which are based on the ARM v2(as) architecture. Newer versions of processors introduced new processor modes, and other important features. An important limitation of the ARM v2(as) architecture is its 26-bit address. This results in only 64 MB of addressable memory ($2^{26} = 64 \text{ MB}$). The address bus was 26-bits in size because R15 used to store the address, was also used to store the status register.

Word Size

The ARM range of processors uses 32-bit words. Each instruction fits into a single word. It also has a 32-bit data bus so that complete instructions can be fetched in a single step.

Processor Modes

ARM 2 and ARM 3 processors can operate in four different modes. They are as follows:

Mode / Use for mode

User Mode – Use for mode: Mode normally used by applications

Supervisor Mode (SVC) – Use for mode: Mainly used by SWI instruction

Interrupt Mode (IRQ) – Use for mode: Used to handle peripherals when they issue interrupt requests

Fast Interrupt Mode (FIQ) – Use for mode: Used to handle peripherals that issue fast interrupt requests

Supervisor Mode, Interrupt Mode and Fast Interrupt Mode are all privileged modes that allow extra control over the computer.

Changing Processor Mode

If the ARM processor is forced to change mode, you must follow this with a no-op command. This is to avoid contention, by allowing the processor time to complete writing to the registers, before switching to the other mode.

Overview of Registers

Twenty seven 32-bit registers are available on the ARM 2 and ARM 3 processor. Of those, only 16 are available within RISC OS. Some registers are shared across different modes, while others are not. The table below shows this:

R0

R1

R2

R3

R4

R5

R6

R7

R8 · R8_fiq

R9 · R9_fiq

R10 · R10_fiq

R11 · R11_fiq

R12 · R12_fiq

R13 · R13_svc – IRQ Mode: R13_irq · FIQ Mode: R13_fiq

R14 · R14_svc – IRQ Mode: R14_irq · FIQ Mode: R14_fiq

R15 (PC/PSR)

Two registers have special functions. They are:

- R15 is used as the Program Counter (PC_) and the Processor Status Register (_PSR)
- R14 (R14_svc, R14_R14_irq and R14_fiq) are used as subroutine link registers

In addition, RISC OS has a convention of using R13 (R13_svc and R13_irq) as private stack pointers for each Processor Mode.

All other registers are deemed as general purpose.

R15 – Program Counter and Processor Status Register

As mentioned above, R15 is used as the Program Counter (PC_) and the Processor Status Register (_PSR). 24-bits is used for the PC, and 8-bits for the PSR.

The contents of R15 is as follows:

0 - 1 – Contents: Processor Mode Flag1

00 = User Mode

01 = FIQ Mode

10 = IRQ Mode

11 = SVC Mode

2 - 25 – Contents: Program Counter (PC)

26 – Contents: FIQ disable flag (set for disable)

27 – Contents: IRQ disable flag (set for disable)

28 – Contents: Conditional Flag - OverFlow (V)

29 – Contents: Conditional Flag - Carry (C)

30 – Contents: Conditional Flag - Zero (Z)

31 – Contents: Conditional Flag - Negative (N)

1 The Processor Mode Flags are not stored in R15, but simply appended to the Program Counter when fetching instructions, thus forming a 26-bit address. The Program Counter must always be word-aligned, and the bottom two bits of the address must always be zero.

R14 - Subroutine Link Registers

R14 (R14_svc, R14_R14_irq and R14_fiq) are used as subroutine link registers. R14 receives a copy of the PC and PSR when a Branch and Link instruction is executed. R14_svc, R14_R14_irq and R14_fiq are used to store the values of R15 when any of the following events occur:

- Interrupts and exceptions arise
- Branch and Link instructions are executed in Supervisor or interrupt routines
- When a SWI instruction is used

R13 - Private Stack Pointers

RISC OS has a convention of using R13 (R13_svc and R13_irq) as private stack pointers for each Processor Mode.

Any routines written that are called from User Mode, but run in SVC or IRQ mode will need to use some of the shared register R0 to R12. Therefore, you must preserve the registers on a stack before you later the registers, and subsequently restore them before returning from your routine.

SVC and IRQ Mode stacks must be full descending stacks and ending at a megabyte boundary. System stack locations should not be changed, however, if they are changed, please note that they are reset to their default locations when errors are generated, and when applications are started.

Routine in FIQ mode require a faster response, so there are seven private registers in FIQ Mode. These should be enough registers to avoid the overhead of having to save them to a stack and using the shared registers. However, if you need to do so, you should use R13_fiq as the stack pointer.

You can use R13 and/or R13_fiq as conventional registers if they are not used as stack pointers.

The VIDC chip

The VIDC chip controls the computer's video and sound. The data is read from RAM into buffers and then processed and converted into the necessary analogue signals to drive the video output displays and sound system. The VIDC can handle many different display and sound formats.

There are three buffers for input data. They are:

- Video data
- Sound data
- Cursor data

Each buffer is of the type FIFO (first-in, first-out). Data is requested in blocks of four 32-bit words at a time. The MEMC chip controls the addressing and fetching of the data under DMA (Direct Memory Access) control.

Video

Data from the video buffer is converted and processed. The are as follows:

- Data is serialised by the VIDC chip into either 1, 2, 4 or 8 bits per pixel
- Passed through a colour look-up palette (16 registers, each is 13-bits wide resulting in 4096 possible colours)
- The output is then converted by three 4-bit digital to analogue converters (DACs). One for red, green and blue colour

- Output is then used to drive the display output device

The registers that control the video system provide a wide range of options. They are as follows:

- Pixel rate can be selected as 1/3, 1/2, 2/3 or 1 times the clock rate of the VIDC chip¹
- Horizontal timing can be controlled (in units of 2 pixels)
- Vertical timing can be controlled (in units of a raster)
- Width of screen border can be altered
- Screen border can be set to any of the 4,096 possible colours
- Interlaced displays
- External synchronisation
- Very high resolution monochrome screen modes (up to 96 MHz pixel rate)

¹ Different versions of the VIDC chip can handle different clock speeds. The original version was 24 MHz, whereas the newer versions could be 25.175 or 36 MHz.

Sound

Up to eight separate channels of mono sounds is provided by the VIDC chip. Each channel can be assigned a stereo position (between left to right) by the use of eight stereo image registers.

Data from the buffer is converted and processed. The following steps taken:

- Data from the buffer is read at a programmable rate
- Data is then passed to an 8-bit DAC
- DAC uses the stereo image registers to convert the digital samples to a stereo analogue sample
- It is then sent to the computer's internal amplifier

Cursor

The cursor data controls a pointer that can be up to 32 pixels wide, and an unlimited number of rasters high, although RISC OS restricts this to a maximum of 32 rasters in height. The pixels can be transparent or can be any of three of the 4,096 possible colours.

The cursor may be positioned anywhere on the screen (within the border).

The IOC chip

The IOC chip manages the peripherals and interrupts throughout the computer. It controls a data bus that peripherals are connected to. In addition, it also provides numerous internal functions that are accessed without any wait states, and a flexible control port.

Peripheral support

The IOC communicates to the rest of the system via the high speed system bus. It is able to handle different speeds of data transfers via the Input/Output or expansion bus automatically.

The specification of the IOC chip is as follows:

- Sixteen interrupt inputs (14 level sensitive and 2 edge-triggered)
- Seven external peripheral select outputs
- Four programmable peripheral timing cycles (slow, medium, fast and 2 MHz synchronous)

RISC OS views the IOC and peripherals as memory-mapped devices. Most peripherals are byte-wide, and word aligned. A single memory instruction can be used to perform any of the following:

- Access IOC control registers
- Select both a peripheral and timing cycle it requires, and access it

A wide range of peripherals are supported by the IOC chip including those that require an interruptible I/O cycle.

Internal functions

The following functions are provided by the IOC chip:

- Full-duplex, bidirectional serial keyboard interface
- Interrupt mask, request and status registers for both normal and fast interrupts
- Six programmable bidirectional control pins
- Four independent 16-bit programmable counters. Two are used as baud rate generators (keyboard and serial port). The third counter (Timer 0_) is used to generate system timing events. The last counter (_Timer 1) is unused by RISC OS, and can be used freely by developers

The MEMC chip

The MEMC chip is used as the interface between the ARM, VIDC and IOC and memory. A single clock input provides all the timing signals by the ARM chip set.

Memory Support

It provides the control signals needed by memory of different speed. The specifications are as follows:

- Provides timing and refresh signals for DRAM
- Provides control signals for several access times of ROM. **The timings are 200ns, 325ns and 450ns when the MEMC is clocked at 8 MHz. The timings are lower (in inverse proportion) when the MEMC is clocked at 10 or 12 MHz
- Can handle up to 32 standard DRAMs can be driven (4 MB of real memory using 1 Mbit devices)
- Can access fast page DRAM to maximise memory bandwidth

Memory Mapping

The MEMC chip maps the physical memory into a 16 MB slot, with the base address at 32 MB. RISC OS does not address this slot directly; instead it addresses another 32 MB logical slot within the 64 MB logical address space.

Each page of the slot that RISC OS addresses can be:

- Unmapped
- Mapped onto one page of physical memory
- Mapped onto numerous pages of physical memory

RISC OS can only read and write from pages that have one-to-one mappings. One-to-many mappings is used to hide pages of applications away when several applications are sharing the same address within the window environment. These addresses start from &8000 upwards, although of course, the pages are not held at &8000.

The physical memory is divided into physical pages. The 32 MB of logical space is divided into logical pages of the same size. The logical page is mapped to the physical page, and the MEMC keeps track of this.

Page size

The MEMC chip has 128 pages to use for its memory mapping. Each page is used to translate a logical address to a physical address. This is achieved by using a content-addressable memory (CAM). Most Archimedes machines use the full 128 pages, but some like the Archimedes 305 only uses 64 pages.

If the MEMC chip does use 128 pages, or any constant number, then:

- The allocated size of each page increases as the computer's physical memory increases. This is because the large amount of memory is being divided into the same number of pages
- The number of logical pages decrease as the size of each page increases. This is because the amount of physical memory is being divided into larger pages

The maximum amount of memory that a single MEMC chip can access is 4MB. If more memory is required, more MEMC chips must be added (in a master > slave scenario).

The table below provides an example of a 'typical' Archimedes computer and its page sizes. The values are only provided for guidance, and must not be relied upon across the whole Archimedes range of computers.

Physical RAM size / Page size / No. of Logical pages

0.5 MB – Page size: 8 kb · No. of Logical pages: 4 K

1 MB – Page size: 8 kb · No. of Logical pages: 4 K

2 MB – Page size: 16 kb · No. of Logical pages: 2 K

4 MB – Page size: 32 kb · No. of Logical pages: 1 K

8 MB – Page size: 32 kb · No. of Logical pages: 1 K

16 MB – Page size: 32 kb · No. of Logical pages: 1 K

To read the computer's page size, you may use the OS_ReadMemMapInfo SWI.

RISC OS programs a minimum of 128 pages, even though it may not use them all. The reasons for this are:

- Random hits in unused pages do not happen
- Extra MEMC chips can be added as slaves to the master MEMC chip allowing for more than 4MB of physical memory to be accessed

Protection Levels & Protection Modes

There are three different protection modes are available to the MEMC chip:

- Supervisor
- Operating System
- User

Each page of memory is set at one of three different protection levels:

- Protection level 0
- Protection level 1
- Protection level 2 (there is a level 3, but it is identical to level 2)

The protection level in conjunction with the protection mode, specifies whether read and/or write access to a page of memory is allowed. This is determined as follows:

/ Protection Level /

Mode · 0 · 1 · 2 · Information

Supervisor · R/W · R/W · R/W · This is the most privileged mode. The MEMC is in this mode whenever the ARM chip is in one of its privileged modes.

Operating System · R/W · R/W · R · The MEMC chip enters this mode by setting a bit in the MEMC's control register (which can only be changed if the ARM is in privileged mode)

User · R/W · R · None · This is the least privileged mode. RISC OS allows it read/write access only to unprotected pages in the logically mapped RAM, and read access to the ROM space

Any attempt to access protected memory from an insufficient protection mode, results in the MEMC trapping the exception and sending an abort signal to the ARM processor.

Memory

The table below show the memory map of a RISC OS Archimedes computer.

IMAGE GOES HERE (UPDATED WIKI REQUIRED)

DMA support

The MEMC chip also provides three programmable address generators to support Direct Memory Access (DMA). They are used for:

- Double buffers for sound data
- Linear buffer for cursor sprite
- Circular buffer for video refresh

[RISC OS Open source page](#)

ARM Hardware Overview

ARM Hardware

RISC OS was developed to take advantage of the ARM microprocessor for a range of high end personal computers. The ARM processor was originally designed by Acorn Computers Ltd, but is now developed by ARM Holdings Ltd.

In practice

RISC OS was designed for the ARM processor which is a RISC chip. i.e It is a processor with a comparatively small set of instructions. This is one of the reasons why ARM chips can run code very efficiently.

Since the original development of the ARM processor back in 1985, many more have been developed. The original processor was the ARM 2. This was followed shortly by the ARM 3, ARM 610, ARM 700 and the StrongARM. Several ARM system-on-chips (SOCs) with integrated memory controller and video controller were also used.

Processor Types

RISC OS supports numerous ARM processors. The table below highlights the some of the processors supported by RISC OS.

ARM 2 · 4 - 5 – ARM architecture: ARM v2 · Extra Notes: -

ARM 250 · 7 – ARM architecture: ARM v2as · Extra Notes: -

ARM 3 · 13.5 – ARM architecture: ARM v2as · Extra Notes: 4Kbyte on-chip cache

ARM 610 · 23 – ARM architecture: ARM v3 · Extra Notes: Address bus and Program counter to use full 32-bit width and extra privileged processor modes

ARM 700 · 35 – ARM architecture: ARM v3 · Extra Notes: -

ARM 7500 · ? – ARM architecture: ARM v3 · Extra Notes: Integrated memory and video controller, plus 4k cache

ARM 7500FE · ? – ARM architecture: ARM v3 · Extra Notes: Integrated memory and video controller, plus 4k cache and includes hardware Floating Point support

StrongARM · 235 – ARM architecture: ARM v4 · Extra Notes: Separate caches for data and instructions

XScale · ? – ARM architecture: ARM v5 · Extra Notes: -

ARM11 · ? – ARM architecture: ARM v6K · Extra Notes: -

Cortex-A5 · ? – ARM architecture: ARM v7 · Extra Notes: -

Cortex-A7 · ? – ARM architecture: ARM v7 · Extra Notes: -

Cortex-A8 · ? – ARM architecture: ARM v7 · Extra Notes: Superscalar processor with extension for handling multimedia and signal processing (NEON technology is 128-bit SIMD extensions)

Cortex-A9 · ? – ARM architecture: ARM v7 · Extra Notes: -

Cortex-A15 · ? – ARM architecture: ARM v7 · Extra Notes: -

Cortex-A17 · ? – ARM architecture: ARM v7 · Extra Notes: -

Cortex-A53 · ? – ARM architecture: ARM v8 · Extra Notes: -

Cortex-A57 · ? – ARM architecture: ARM v8 · Extra Notes: -

Cortex-A72 · ? – ARM architecture: ARM v8 · Extra Notes: -

RISC OS Hardware

The hardware of RISC OS computers can be categorised as follows:

Category / ARM architecture / Models

Archimedes – ARM architecture: ARM v2 & ARM v3 · Models: A305, A310, A310M, A410, A440, BBC A3000, A410/1, A420/1, A440/1, A540, A5000, A4, A3010, A3020, A4000

RiscPC – ARM architecture: ARM v3 & ARM v4 · Models: A7000, A7000+, Risc PC, Kinetic Risc PC

HAL(Hardware Abstraction Layer) – ARM architecture: ARM v3 and above · Models: A7000, A7000+, Risc PC, Kinetic Risc PC, Lyonix, BeagleBoard, PandaBoard, Raspberry Pi, Wandboard, PineA64, OMAP5, Titanium

[RISC OS Open source page](#)

ARMv7 compatibility primer

This guide exists to help programmers update their code to run correctly on ARMv6+ processors. This guide only covers differences between ARMv5 and the newer architectures - i.e. it is assumed that your code is already 32bit compatible and runs correctly on an Iyonix (and without the use of Aemulor!). Also, this guide only covers differences that are relevant to the ARM instruction set - Thumb-only details are not listed. Note that you shouldn't confuse ARMv6 and ARMv7 (current generations of the ARM architecture) with ARM6 and ARM7 (processor families used in the Risc PC and A7000).

A summary of the different ARM architecture versions used by RISC OS machines

- ARMv2 - Used in Archimedes machines.
- ARMv3 - RiscPC 600, RiscPC 700, A7000, A7000+
- ARMv4 - StrongARM RiscPC, A9home, Omega, Virtual RPC, RPCEmu
- ARMv5 - IYONIX pc
- ARMv6 - Raspberry Pi 1, Raspberry Pi Zero
- ARMv7 - BeagleBoard, IGEPv5, iMx6, PandaBoard, Raspberry Pi 2, Titanium, and other Cortex-A7, A8, A9, A15 based machines
- ARMv8 - Raspberry Pi 3, Raspberry Pi 4

Note that whilst StrongARM processors are ARMv4, StrongARM RiscPCs are generally treated as ARMv3 or ARMv3m, due to the RiscPC memory bus not supporting halfword transfers. This is generally true of emulation solutions too.

The Raspberry Pi 2 has an ARMv7 CPU, whereas older models (A, B, B+, Compute) used an ARMv6 CPU. The ARMv6 models are able to take advantage of a compatibility mode (on by default for the standard RISC OS Pi distribution) to allow them to run non-ARMv7 compatible software - see below for more details.

If software is very old (pre-2002), it probably isn't '32 bit compatible'. You might wish to read the Writing 32 bit code guide on how to convert it, in addition to following the details below.

Plain C programs

If a program is written in plain C, chances are that a simple recompilation of the code is all that's needed to ensure your program is compatible.

Norcroft (aka Acorn C/C++)

For users of Norcroft you need to use the `-memaccess` option to disable the use of unaligned loads/stores.

`-memaccess -L22-S22-L41`

The above will produce code which is compatible with all ARM architectures. Norcroft version 5.68 and above uses this setting by default. If you want to upgrade to the latest version, and are already using a ROOL branded copy of the Desktop Development Environment, you can upgrade for half price.

If you don't require RiscPC compatibility, you can change '`-L22-S22`' to '`+L22+S22`' to enable the use of halfword loads/stores (note you will also need to use the appropriate `-cpu` option to enable use of the LDRH/STRH instructions).

GCC

To ensure compatibility with all machines, GCC version 4.7.4 release 3 (or newer) is recommended. Earlier releases of GCC 4.7 have ARMv8 compatibility issues in UnixLib (use of SWP instruction), and versions of GCC prior to 4.1.1 release 2 have issues with unaligned loads/stores.

The latest release of GCC is available from the RISC OS GCCSDK website

Programs containing hand-crafted assembler

There are several differences between ARMv7 and ARMv5 which can cause hand-crafted assembler to fail or malfunction. These issues are listed below.

Unaligned LDM/STM

For ARMv5 and below, the bottom two bits of the source/destination address were always ignored and treated as zero.

For ARMv7 and above, an abort will be raised if the source/destination address is not word-aligned. The abort will occur regardless of the alignment exceptions setting.

For ARMv6, the behaviour is configurable via the system control register – but for maximum compatibility your code should assume ARMv7 behaviour.

Unaligned LDR

For ARMv5 and below, this has the behaviour of a “rotated load”. The data is loaded from a word-aligned address, and then rotated by the number of bytes specified by the bottom two bits of the original address. I.e., for “LDR R1,[R0]”, the behaviour is as follows:

```
BIC temp,R0,#3
LDR R1,[temp]
AND temp,R0,#3
MOV temp,temp,LSL #3
MOV R1,R1,ROR temp
```

For ARMv7 and above, this has the behaviour of a “sequential load” – four bytes are loaded from sequential locations in memory. I.e., for “LDR R1,[R0]”, the behaviour is as follows:

```
LDRB R1,[R0]
LDRB temp,[R0,#1]
ORR R1,R1,temp,LSL #8
LDRB temp,[R0,#2]
ORR R1,R1,temp,LSL #16
LDRB temp,[R0,#3]
ORR R1,R1,temp,LSL #24
```

For ARMv6, the behaviour is again configurable via the system control register.

The important thing to realise is that for any particular unaligned load, the bottom N bytes of the data will be correct, while the top 3-N bytes will be “incorrect”. This has caused problems with C compilers, where halfword loads were typically implemented using the following pseudocode:

```
LDR R1,[R0 EOR #2]
MOV R1,R1,(LSR|ASR) #16
```

This will produce different results in ARMv5 and ARMv7, and could easily lead to unexpected data corruption. This is why alignment exceptions are currently turned on by default for versions of RISC OS running on ARMv6+.

Thus, the recommendation is to avoid the use of unaligned LDRs when running on ARMv6+.

Unaligned STR

On ARMv5 and below, the bottom two bits of the destination address are ignored and treated as zero.

On ARMv7 and above, STR performs a “sequential write” – i.e. writing bytes to sequential memory locations using the same addressing scheme as LDR.

For ARMv6, the behaviour is again configurable via the system control register.

Due to the differences in behaviour, and the use of alignment exceptions by default on ARMv6+, it is recommended that unaligned STRs are avoided when running on ARMv6+.

Unaligned LDRH/STRH

On ARMv6 and below, a non-halfword aligned LDRH/STRH has unpredictable behaviour.

On ARMv7 and above, non-halfword aligned LDRH/STRH performs a sequential load/store, as per LDR/STR.

Due to the differences in behaviour, and the use of alignment exceptions by default on ARMv6+, it is recommended that unaligned LDRH/STRH is avoided when running on all architecture versions.

Other multi-byte memory access instructions

Other, more exotic multi-byte memory access instructions (LDRD, STRD, LDC, STC, LDRT, STRT, etc.) have had their behaviour changed as well. However, since alignment exceptions are likely to be turned on, it is recommended that you avoid using unaligned loads/stores of any kind unless your code is certain that it will work in the intended manner.

MOVS pc, lr (and similar)

On all ARM processors running in 32bit mode, any ALU-based instruction (MOV, ADD, BIC, etc.) that writes to the PC and updates the flags via the 'S' bit will behave in an unpredictable manner if executed in User mode or System mode. So if your code falls foul of the above rule then it clearly isn't as 32bit-compatible as you thought when you read the notice in the Introduction.

Unfortunately, not all unpredictable behaviours are created equal – experience suggests that MOVS pc, lr behaves “correctly” on an IOP321 (i.e. Iyonix), but fails on OMAP3, causing code which was otherwise believed to be 32bit compatible to fail.

LDM reg, {...pc}^

This has the exact same rules as MOVS pc, lr – the combination of loading the PC and using the ^ flag causes the CPU to try and set the PSR to the value of the (mode-specific) SPSR, but in User and System mode there is no SPSR, resulting in the instruction being unpredictable when executed in those modes. This instruction also seems to be another case of an unpredictable instruction that “works” on IOP321 but fails on OMAP3.

MOV pc, lr (and similar)

On ARMv7 and above, any ALU-based instruction (MOV, ADD, BIC, etc.) that writes to the PC performs what is known as interworking. Interworking is a process used to switch between ARM and Thumb mode, and is controlled by the bottom two bits of the target address. This means that unless the bottom two bits of the target address are zero, your code is likely to behave unexpectedly on ARMv7 and above. X1 = switch to Thumb, 00 = switch to ARM, 10 = unpredictable.

On ARMv6 and below, the bottom two bits are ignored and treated as zero.

LDM reg!,{..reg..}

In ARMv7 and above, an LDM with writeback, and with the base register in the register list, is an unpredictable instruction.

In ARMv6 and below, it was merely the final value of reg that would be unpredictable, and not the entire instruction itself.

Undefined instructions

In ARMv7 and above the CPU is allowed to take the undefined instruction vector if an undefined instruction is “executed” but fails the condition code test. For example, code which identifies the CPU architecture and then uses conditional execution to execute the correct MRC/MCR instruction may fail. This code should be restructured to branch over the non-executed paths.

Generally this behaviour of taking the undefined instruction vector is seen on ARMv8 devices rather than on ARMv7.

SWP/SWPB

In ARMv6, SWP/SWPB were deprecated in favour of the new LDREX/STREX instructions. In ARMv7VE support for the SWP/SWPB instructions is optional, and in ARMv8 support for the instructions has been removed completely.

Software which uses SWP/SWPB should be updated to use the newer LDREX/STREX suite instructions. OS_PlatformFeatures 0 can now indicate which instructions are available.

MRS reg,CPSR

In ARMv8, reading the CPSR from user mode will return an “unknown” value for the E, A, I, F, and M fields (essentially the lower 16 bits of the PSR). So far this behaviour has only been observed on ARMv8 CPUs which lack privileged-mode AArch32 support (which RISC OS is unable to directly run on), but the issue could become more significant as time passes (especially if you believe more recent editions of the ARMv7 ARM, which suggest that the same behaviour is also valid on ARMv4-ARMv7!)

It's expected that the above behaviour will mainly impact code which attempts to determine whether the CPU is in a 26bit or 32bit mode; on a new CPU the mode field may be read as zero, which will cause the code to incorrectly believe the CPU is in a 26bit mode.

If you just want to detect 26 vs. 32bit CPU mode, the following code sequence will work on all CPUs and in all CPU modes:

```
TEQ R0, R0 ; sets Z (can be omitted if not in User mode)
```

```
TEQ PC, PC ; EQ if in a 32-bit mode, NE if 26-bit
```

Alignment Exceptions

Alignment exceptions (referred to as alignment faults in the ARM ARM) are a feature available in ARMv6 and above. If alignment faults are enabled, any improperly aligned memory access (i.e. loading a word from a 1, 2 or 3 byte offset, or loading a halfword from a byte offset) will trigger an alignment fault. At present RISC OS will report this with the standard “Abort on data transfer” message, so you may need to use the Debugger module to manually determine the exact cause of the error.

Note that all ARMv6+ ROM images have alignment exceptions enabled by default. This is to help track down code which relies on ARMv6+ -incompatible unaligned LDR/STR behaviour. It is unknown when, or if, alignment exceptions will be switched back to the “off” state, so it is advised that you modify your code accordingly.

For testing purposes only, the below BASIC program can be used to turn alignment exceptions off:

```
10 DIM code% 256
20 P%=code%
30 [ OPT 0
40 SWI "OS_EnterOS"
50 MRC CP15,0,R0,C1,C0,0
60 BIC R0,R0,#2
70 MCR CP15,0,R0,C1,C0,0
80 MSR CPSR_c,#&10
90 MOV R0,R0
100 MOV PC,R14
110 ]
120 CALL code%
```

The same program, but with an ORR instruction instead of BIC (line 60), can be used to turn alignment exceptions back on again.

CPU configuration modes

Raspberry Pi comes with a Configure plugin to set different modes, on a whole-system basis. These are:

- ARMv5 compatibility mode: pretend it's an Iyonix. Emulate rotated loads, alignment exceptions off. Good for running old software that isn't ARMv7 safe.
- ARMv7 strict mode: alignment exceptions on, any unaligned access will cause a program to fail. Use this to test that software is ARMv7 safe
- ARMv7 fast mode: alignment exceptions off, unaligned loads enabled. Allows software written specifically for ARMv7 to run faster – but (currently) you can't run anything that isn't ARMv7 safe.

ARMv5 compatibility mode is the default, so do remember to flip into ARMv7 strict mode when testing.

Note that the ARMv5 compatibility mode is not available on the Raspberry Pi 2 or 3, due to the change to newer CPUs.

Maintenance requirements for the local exclusive monitor

The LDREX/STREX instructions that were introduced with ARMv6 rely on a component of the CPU called the 'local exclusive monitor'. For the load/store exclusive instructions to work correctly, the ARM requires that operating systems manually reset the exclusive monitor to the 'open' state before returning to any pre-empted code.

As far as RISC OS software is concerned, the exclusive monitor must be manually cleared under the following situations:

- On exit from the callback environment handler
- On exit from abort handlers
- On exit from FIQ handlers
- On exit from any code which performs a thread context switch (if returning to a thread which had been pre-empted)

To reset the exclusive monitor, you should call the function returned by `OS_PlatformFeatures` 35. Alternatively, if you are certain about the requirements of the CPU you can either execute a STREX instruction to a dummy location, or use the CLREX instruction (if available - check with `OS_PlatformFeatures` 0).

Failure to correctly clear the exclusive monitor may result in a STREX succeeding in a situation in which it should have failed.

IRQ handlers and vectored callbacks (`OS_AddCallBack`) do not need to clear the exclusive monitor; the OS will do it for you on exit from the handler. The above list comprises the situations where code returns directly to the pre-empted foreground task rather than exiting via the OS.

Exceptions

There are some exceptions to the above rules, for situations where the monitor does not need to be cleared:

- The monitor does not need to be cleared if you are certain the code you were returning to was not in the middle of an exclusive access sequence. However this may be hard to prove, so it's probably best to err on the side of caution.
- The monitor does not need to be cleared if you are not performing a thread context switch and your callback/abort/FIQ handler does not use the exclusive instructions itself.
- The monitor does not need to be cleared if your code does use the exclusive instructions but the last exclusive code sequence is structured in such a way that the exclusive monitor is always left in the 'open' state (i.e. there is no code path where a LDREX is not followed by a STREX).

If any of the above are true then the monitor does not need to be cleared manually.

Note that RISC OS does not guarantee that the exclusive monitor is in the open state on return from SWIs; therefore if your code calls a SWI then it must take action to clear the monitor before it returns.

For full details of the operation of the local exclusive monitor (and the related 'global exclusive monitor'), consult the ARM ARM. Understanding the operation of the exclusive monitors is critical to correctly writing software which uses the exclusive access instructions.

The ARM Architecture Reference Manual (ARM ARM)

For much more detailed information about the differences between ARM architectures, it is recommended that you consult the relevant version of the ARM ARM.

[RISC OS Open source page](#)

AUNMsgs

List of modules

Acorn Universal Networking (AUN) is a network technology to allow Acorn specific protocols to be moved over the widely used protocol; TCP/IP.

In practice

As the modules which implemented these protocols evolved they were released as disc based updates to the copies in ROM, and often needed corresponding updates to the Messages text in ResourceFS. The AUNMsgs module provided a single module which registered several Messages files on behalf of those modules.

The majority of modules now manage their own updates in ResourceFS, meaning AUNMsgs is largely redundant.

[RISC OS Open source page](#)

BASIC

List of modules

The BASIC module provides the BASIC programming language and real-time interpreter. Other variants of the module use 8 bytes (64 bits) when storing floating point numbers, rather than BASIC's more limited 5 bytes.

Details

- *Commands

[RISC OS Open source page](#)

BASICTrans

List of modules

The BASICTrans module provides support to BASIC and BASIC64 modules so that messages can be translated for use in another territory. The BASIC interpreter issues calls to the BASICTrans module which is responsible for providing messages to the particular territory.

In practice

By replacing one BASICTrans module with another, you can change the language used by BASIC for its messages.

Both BASIC and BASIC64 issue the same calls to the same BASICTrans module, thus code and messages are shared between the two modules.

Details

- SWI Calls

[RISC OS Open source page](#)

Battery Management Unit status flags

Portable

The below flags are used by BMU variables 10 and 18, Event 28, and Portable_EnumerateBMU (as a mask of supported flags).

- 0 – Meaning when set: Battery fully charged (threshold 3)
- 1 – Meaning when set: Lid open
- 2 – Meaning when set: Battery flat (threshold 2 reached) [shutdown now]
- 3 – Meaning when set: Battery low (threshold 1 reached) [warn user]
- 4 – Meaning when set: Fault in charging system
- 5 – Meaning when set: Charge state known
- 6 – Meaning when set: Battery present
- 7 – Meaning when set: Charger connected
- 8 – Meaning when set: Battery charging
- 9 – Meaning when set: Battery failing (the battery has been detected as no longer being able to hold a charge well, and should be serviced or replaced)

Notes

Not all BMU status flags are available in all variants of the module. Use Portable_EnumerateBMU to read the available flags, or see the Portable Feature Matrix for the details of versions of the module which don't implement Portable_EnumerateBMU.

[RISC OS Open source page](#)

Battery Management Unit variables

Portable

Variables shown with a grey background are deprecated, and their use should be avoided. 4

- 0 · Version number and memory map of BMU microcode
- 1 – Description: Battery capacity (nominal) · Units: mAh 1
- 2 – Description: Battery capacity (measured) · Units: mAh 1
- 3 – Description: Battery capacity (used) · Units: mAh 1
- 4 – Description: Battery capacity (useable) · Units: mAh 1
- 5 · (reserved)
- 6 – Description: Charge estimate (may be written to) · Units: mAh 1
- 7 – Description: Instantaneous voltage · Units: mV 2
- 8 – Description: Instantaneous current · Units: mA 3
- 9 – Description: Instantaneous temperature · Units: 0.1K 2
- 10 · Status flags
- 11 · Charge rate (in bits 4 – 7)
- 12 · Charge current
- 13 · Discharge current
- 14 · Voltage
- 15 · Temperature 1
- 16 · Temperature 2
- 17 · Charge level
- 18 · Status flags (as variable 10)
- 19 · Input signals

- 20 · Battery capacity
- 21 – Description: Battery time remaining · Units: Seconds
- 22 – Description: Max battery life · Units: Seconds
- 23 – Description: Battery percentage remaining · Units: 0-100
- 24 – Description: Charge time remaining · Units: Seconds
- 25 · Charge cycle count
- 26 – Description: Smoothed voltage · Units: mV
- 27 – Description: Smoothed current · Units: mA
- 28 – Description: Nominal available capacity · Units: mAh
- 29 – Description: Compensated available capacity · Units: mAh

Notes

Not all BMU variables or flags are available in all variants of the module. See the Portable Feature Matrix for more details (BMU variables, flags). `Portable_EnumerateBMU` can (where supported) also be used to read the available flags.

1 Units are 11.7mAh in the A4 version of the Portable module

2 Units are unknown in the A4 version of the Portable module

3 Units are 10.3mA in the A4 version of the Portable module

4 The variables shown with a grey background are deprecated, for the following reasons:

- They correspond to the variables that were introduced with the Stork version of the Portable module
- Apart from variable 14 being supported by the A9 version of the module, it's only the Stork version of the module that supports them
- The Stork version of the module has never been available in an official stable release of the OS
- There is limited documentation about the function or measurement units of the variables
- Many of the variables appear to duplicate the intent or function of A4 variables

[RISC OS Open source page](#)

BBCEconet

List of modules

The BBCEconet module provides emulation of certain obsolete OSBYTE and OSWORD calls that were used by the old 6502-based BBC Computers (pre RISC OS), thus making it easier to port code that used these calls.

In practice

The purpose of this module is to facilitate the porting of old 6502-based programs. You must not use these calls in any new program or application.

OS_Byte 50

OS_Byte 51

OS_Byte 52

OS_Word 16 – Notes: All 8 sub-reason codes are emulated (Transmit, Peek, Poke, JSR, User Procedure Call, Machine type, Halt and Continue)

OS_Word 17 – Notes: Both sub-reason codes are emulated (OpenRx and ReadRx)

OS_Word 19 – Notes: Only these function codes are supported:

0 read file server number

1 write file server number

2 read printer server number

3 write printer server number

4 read protection mask
 5 write protection mask
 8 read local station number
 12 read printer server name
 13 set printer server name
 15 read file server retry delay
 16 set file server retry delay
 17 translate net number

OS_Word 20 – Notes: All 3 sub-reason codes are supported (Do File Server Operation, Notify, and Cause Remote Error)

Correspondence between old and new calls

All the calls listed above use exactly the same parameters as their obsolete 6502-based counterparts. The table below details the correspondence between 6502 parameters and ARM RISC OS registers.

6502 Register / ARM Register

A – ARM Register: R0 (bits 0 – 7)

X – ARM Register: R1 (bits 0 – 7)

Y – ARM Register: R2 (bits 0 – 7)

Note: Bits 8 – 31 of ARM Registers are ignored.

Implementation Process

- The BBCEconet module claims ByteV and WordV vectors
- If it recognises an OS_Byte or OS_Word call that it supports it checks that the necessary modules are present and active to emulate the call
- Translates the calls to the appropriate SWI call(s) to these modules

Restriction

OS_Byte 50 (poll transmission) and OS_Byte 51 (poll receive block) may enable interrupts and should not be called from within interrupt handlers, service code or event routines. During these calls the processor may be put in USR mode with interrupts enabled; this allows CallBacks to occur.

[RISC OS Open source page](#)

BCMSupport

List of modules

The primary purpose of the BCMSupport module is to provide shared access to the mailbox channels that are used for communication between the ARM and the VideoCore GPU that is present in the Raspberry Pi family of computers.

In practice

For the low-level details of this interface, see the Raspberry Pi GitHub wiki

Details

- SWI Calls

[RISC OS Open source page](#)

BootNet

List of modules

The BootNet module provides functionality to enable/disable the AUN (Acorn Universal Networking) client station device booting from an expansion card ROM.

Details

- *Commands

[RISC OS Open source page](#)

Buffer Numbers

BufferManager

The sound buffers (4 - 7) have no meaning for RISC OS, and can be safely ignored.

- 0 – Buffer: Keyboard
- 1 – Buffer: Serial input
- 2 – Buffer: Serial output
- 3 – Buffer: Printer
- 4 – Buffer: Sound channel 0
- 5 – Buffer: Sound channel 1
- 6 – Buffer: Sound channel 2
- 7 – Buffer: Sound channel 3
- 8 – Buffer: Speech
- 9 – Buffer: Mouse

[RISC OS Open source page](#)

Buffers

The Operating System makes extensive use of buffers as temporary holding areas for data. For example, when a character is typed on the keyboard, the character is stored in the keyboard input buffer by the keyboard interrupt handler, and remains there until it is ready to be used.

In practice

All buffers are handled by the Buffer Manager. This is used by DeviceFS to provide buffers for various devices to perform tasks such as:

- Initialising buffers
- Inserting data into a buffer
- Removing data from a buffer
- Deleting a buffer

Using buffers

Because the purpose of buffers is to store data before they are ready for use, the method of processing the data is First In, First Out (FIFO). This ensures that data loaded into the buffer first is removed first.

Some operations are not explicitly stated, such as when sending a character to the printer port, as this actually inserts the character into a buffer. There are several key buffer operations available:

- Inserting a character into a buffer
- Removing a character from the buffer
- Counting the space in a buffer
- Examine the next character in the buffer without removing it
- Clear the buffer contents (the buffer is cleared implicitly when the escape condition is cleared – see the Character Input? section)

Data format

With the exception of the mouse buffer, the data format is byte-orientated ASCII data. The keyboard buffer is provided in more detail within the Character Input? section.

Filing System buffers

RISC OS uses buffers to hold areas of files in memory to increase file access efficiency. The buffers are integrated into the Operating System and there is no direct way of accessing their contents. This section does not cover filing system buffers.

SWI calls

Changing buffer sizes is handled by the Buffer_Create and Buffer_Register. The Buffer Manager section documents other SWI calls. The printer buffer can be configured by the use of *Configure PrinterBufferSize.

Several OS_Byte SWI calls are also provided to control buffers:

- OS_Byte 15 Flush all buffers, or current buffer
- OS_Byte 21 Flush specific buffer
- OS_Byte 128 Get mouse co-ordinates, or number of bytes in a buffer
- OS_Byte 138 Insert a byte into a buffer
- OS_Byte 145 Get a byte from a buffer
- OS_Byte 152 Examine status of a buffer
- OS_Byte 153 Insert a byte into one of two buffers

[RISC OS Open source page](#)

ByteV

Vectors

This vector is called to indirect all calls to OS_Byte.

Before calling

R0 Reason code

R1 First argument

R2 Second argument

Notes

OS_Byte is entered with interrupts disabled, so claimants of this vector should focus on rapid execution, else your routine must enable interrupts and be prepared to be re-entered.

The default action is to call the MOS byte handler.

[RISC OS Open source page](#)

Cache_CleanAll ARMop

OS

The unified cache or data cache are to be globally cleaned (any writeback data updated to memory). Invalidation is not required.

Before calling

--

On return

R0 Corrupt

Notes

Note that any write buffer draining will also be performed by this operation, so that memory is fully updated with respect to any writeback data.

The OS only expects the cleaning to be with respect to data that are not involved in any currently active interrupts.

[RISC OS Open source page](#)

Cache_CleanInvalidateAll ARMop

OS

The cache or caches are to be globally invalidated, with cleaning of any writeback data being properly performed.

Before calling

--

On return

R0 Corrupt

Notes

Note that any write buffer draining should also be performed by this operation, so that memory is fully updated with respect to any writeback data.

The OS only expects the invalidation to be with respect to instructions/data that are not involved in any currently active interrupts.

[RISC OS Open source page](#)

Cache_CleanInvalidateRange ARMop

OS

The cache or caches are to be invalidated for the specified address range, with cleaning of any writeback data being properly performed.

Before calling

R0 Logical address of start of range (inclusive, cache line aligned)

R1 Logical address of end of range (exclusive, cache line aligned)

On return

R0 Corrupt

Notes

Note that any write buffer draining should also be performed by this operation, so that memory is fully updated with respect to any writeback data.

The OS only expects the invalidation to be with respect to instructions/data that are not involved in any currently active interrupts.

[RISC OS Open source page](#)

Cache_InvalidateAll ARMop

OS

The cache or caches are to be globally invalidated. Cleaning of any writeback data is not to be performed.

Before calling

--

On return

R0 Corrupt

Notes

This call is only required for special restart use, since it implies that any writeback data are either irrelevant or not valid. Use with extreme caution.

[RISC OS Open source page](#)

Cache_RangeThreshold ARMop

OS

Return a threshold value for an address range, above which it is advisable to globally clean and/or invalidate caches, for performance reasons. For a range less than or equal to the threshold, a ranged cache operation is recommended.

Before calling

--

On return

R0 Threshold value (bytes)

Notes

This call returns a value that the kernel may use to select between strategies in some cache operations. This threshold may also be of use to some of the ARM operations themselves (although they should typically be able to read the parameter more directly).

The exact value is unlikely to be critical, but a sensible value may depend on both the ARM and external factors such as memory bus speed.

[RISC OS Open source page](#)

Callback Handler

OS_ChangeEnvironment

The Callback handler is used for handling non-transient Callbacks. After calling OS_SetCallback, the handler will be called the next time that RISC OS is threaded out (i.e. execution is returning to a user mode task with the supervisor stack empty).

Before calling

R12 Workspace pointer

R13 SVC/IRQ stack

Other registers: Undefined

SVC or IRQ mode, IRQs disabled, FIQs undefined

In practice

The handler will be called in supervisor or IRQ mode, with R12 pointing to the handler workspace. The handler buffer (as specified to OS_ChangeEnvironment / OS_Callback) will have been filled with the ARM register dump. The buffer has the same size and format as the Exception Registers Block. The register dump reflects the context that would have been restored had the Callback not been triggered. The typical method used to return from a Callback handler is to restore the registers from that block, performing an "exception return" into user mode using the stored PC/PSR. However a popular use of Callback handlers is to implement threading schemes, where you may want to restore a register dump that corresponds to a different thread.

On ARMv6 and above, non-transient Callback handlers will typically need to reset the local exclusive monitor to the open state, as described in the ARMv7 compatibility primer. Failure to do this may result in foreground tasks which use the load/store exclusive instructions to malfunction.

Notes

There are two types of Callback, Transient and Non-transient:

- Transient Callbacks are set up using `OS_AddCallback`. Multiple Callback routines can register via this interface, and they will all be called in turn the next time RISC OS is threaded out (`OS_SetCallback` is not required). Additionally, transient callbacks can occur when the supervisor stack is non-empty.
- Non-transient Callbacks are set up using `OS_SetCallback`. Only one routine can be called in this manner (the Callback environment handler), but this system has the advantage that the callback handler is able to directly control what happens next within the system (e.g. by switching foreground threads).

If both transient and non-transient callbacks are scheduled, the transient callbacks will be called first.

Prior to RISC OS 3.10, non-transient callbacks were called even if the supervisor stack was non-empty.

[RISC OS Open source page](#)

CDFS Control Block

CDFS

CDFS control blocks are used with most CDFS SWI Calls to identify the target drive. Control blocks are a five-word long, word-aligned data structure, and are typically constructed by zero-extending the fields contained in the compacted 32bit 'physical drive number' that is returned by `CDFS_ConvertDriveToDevice`.

Control Block / Physical Drive No. / Contents

Word 0 – Physical Drive No.: Bits 0-2 · Contents: SCSI Device (0 – 7)

Word 1 – Physical Drive No.: Bits 3-4 · Contents: SCSI Card (0 – 3)

Word 2 – Physical Drive No.: Bits 5-7 · Contents: SCSI Logical Unit Number (0 – 7)

Word 3 – Physical Drive No.: Bits 8-15 · Contents: Driver handle (0-255) (see `CD_Identify`)

Word 4 – Physical Drive No.: Bits 16-31 · Contents: Reserved/extension field (0-65535)

Notes

Although the CD driver stack uses SCSI device identifiers to identify drives, the identifiers may not correspond to 'real' SCSI devices that are known to the SCSI stack?. To send SCSI commands to drives, `CD_SCSIUserOp` should be used, rather than attempting to use `SCSI_Op?` directly.

[RISC OS Open source page](#)

CDFS Technical Details

CDFS

See CDFS Control Block

Data Modes

The following table details the different data modes that CDFS currently understands.

Mode / Meaning

0 – Meaning: Audio

1 – Meaning: Data Mode 1

2 – Meaning: Data Mode 2 format 1

3 – Meaning: Data Mode 2 format 2

Addressing Mode

Addresses have different meaning depending on the addressing mode:

Mode / Meaning

- 0 – Meaning: Logical block addressing (Offset in blocks from the start of a CD's data area)
- 1 – Meaning: Red Book addressing (A time offset from a CD's first frame, in the form detailed below this table)
- 2 – Meaning: Physical block addressing (offset in blocks from the first block on the CD)

Red Book Addressing Details

Byte / Label / Meaning

- 0 – Label: FF · Meaning: Frame number (0 - 74) (75 frames in one second)
- 1 – Label: SS · Meaning: Seconds (0-59)
- 2 – Label: MM · Meaning: Minutes (0-99)
- 3 – Meaning: Reserved

CDFSDriver assumes that all discs have a lead-in period of 2 seconds (150 blocks). The CD_ConvertToLBA and CD_ConvertToMSF SWIs know to apply this offset; the conversion is such that LBA 0 and MSF 00:00:00 both correspond to physical block 150. Attempting a mode 0 or mode 1 access to addresses below physical block 150 may result in an error.

CD Drive Status

Value / Meaning

- 1 – Meaning: OK
- 2 – Meaning: Busy
- 4 – Meaning: Not Ready (e.g. Drive empty)
- 8 – Meaning: Unavailable

16 Byte Track Block

The following block is used by CD_SetParameters and CD_GetParameters.

Offset / Contents

- 0 – Contents: Log spin-down delay 0-15 (0 equivalent to 16, 1 = shortest)
- 4 – Contents: Number of retries
- 8 – Contents: Read mode (0, 1 and 3 for data, 2 for audio)
- 12 – Contents: Speed in multiples of 'standard' speed

[RISC OS Open source page](#)

CDFSDriver

List of modules

The CDFSDriver module provides functionality to access low level hardware and data of CDs and other similar forms of optical media.

In practice

CDFSDriver itself does not talk to the hardware; instead it relies on driver backends such as CDFSSoftATAPI and CDFSSoftSCSI?. Driver backends register themselves with CDFSDriver via the CD_Register SWI, which then allows CDFSDriver to route the CD_ SWI calls to the correct driver on a per-drive basis.

Details

- SWI Calls
- Technical Details

[RISC OS Open source page](#)

CDFSDriver Technical Details

CDFSDriver

These are the reason codes passed in R11 when CDFSDriver calls your handler's handler. Note that there is no direct relationship between the handler call number and the CD_SWI number.

Before calling

R0 - R6 As passed to SWI

R7 Pointer to CDFS Control Block

R11 Call number

R12 Workspace pointer

R13 1k stack

R14 Return address

On return

R0 - R6 Updated for Return

R7 - R12 Corrupted

R14 Corrupted

CDFS Driver Information Block

Offset / Contents

0 – Contents: Information word for CDFSDriver module

4 – Contents: Number of times registered (should be set to zero and will be updated by CDFSDriver)

8 – Contents: Number of drive types recognised by this driver

12 – Contents: Reserved (must be zero)

16 – Contents: Reserved (must be zero)

On Exit

12 · A pointer to a routine to convert logical blocks to minutes, seconds, frames

16 · A pointer to a routine to convert minutes, seconds, frames to logical blocks

CDFS Information Word

0 - 5 – Meaning: The number of CD commands recognised, currently 31, maximum 63

6 – Meaning: Use the SCSI 1 read data command

7 – Meaning: Use the SCSI 1 seek command

8 – Meaning: Use old (SCSI I?) Inquiry

9 – Meaning: Use old Capacity

10 – Meaning: Use old Ready

11 – Meaning: Use old Stop/Open

12 – Meaning: Use old Check

13 – Meaning: Use old Status

14 – Meaning: Use old Control

15 – Meaning: Use old Prefetch

16 – Meaning: Use old Reset

17 – Meaning: Use Read data proprietary

18 – Meaning: Use Read data scatter

- 19 – Meaning: Use Seek data proprietary
- 20 – Meaning: Use Seek data scatter
- 21 – Meaning: Call SCSI_Op? to implement CD_SCISUserOp rather than the soft loadable driver
- 22 - 31 – Meaning: Reserved (must be zero)

Driver Call Numbers

- 0 – CD_SWI: CD_ReadData
- 1 – CD_SWI: CD_SeekTo
- 2 – CD_SWI: CD_DriveStatus
- 3 – CD_SWI: CD_DriveReady
- 4 – CD_SWI: CD_GetParameters
- 5 – CD_SWI: CD_SetParameters
- 6 – CD_SWI: CD_OpenDrawer
- 7 – CD_SWI: CD_EjectButton
- 8 – CD_SWI: CD_EnquireAddress
- 9 – CD_SWI: CD_EnquireDataMode
- 10 – CD_SWI: CD_PlayAudio
- 11 – CD_SWI: CD_PlayTrack
- 12 – CD_SWI: CD_AudioPause
- 13 – CD_SWI: CD_EnquireTrack
- 14 – CD_SWI: CD_ReadSubChannel
- 15 – CD_SWI: CD_CheckDrive
- 16 – CD_SWI: CD_DiscChanged
- 17 – CD_SWI: CD_StopDisc
- 18 – CD_SWI: CD_DiscUsed
- 19 – CD_SWI: CD_AudioStatus
- 20 – CD_SWI: CD_Inquiry
- 21 – CD_SWI: CD_DiscHasChanged
- 22 – CD_SWI: CD_Control
- 23 – CD_SWI: CD_Supported
- 24 – CD_SWI: CD_Prefetch
- 25 – CD_SWI: CD_Reset
- 26 – CD_SWI: CD_CloseDrawer
- 27 – CD_SWI: CD_IsDrawerLocked
- 28 – CD_SWI: CD_AudioControl?
- 29 – CD_SWI: CD_AudioLevel
- 30 – CD_SWI: CD_Identify
- 31 – CD_SWI: CD_ReadAudio
- 32 – CD_SWI: CD_ReadUserData
- 33 – CD_SWI: CD_SeekUserData?
- 34 – CD_SWI: CD_GetAudioParms
- 35 – CD_SWI: CD_SetAudioParms
- 36 – CD_SWI: CD_SCISUserOp

[RISC OS Open source page](#)

Cerilica_StyledText

Code &5327F · WindowManager

This message is used to send a small amount of text to the task with the cursor. To avoid ambiguity in the interpretation of the character codes in the text string, a font identifier or Font Encoding Identifier should be included. This allows the recipient to insert text in the desired font, or to remap if necessary to the character set in use.

Message Block

+20 · null terminated string

+?? · null terminated Font Identifier String

In practice

It should be sent recorded to the window containing the cursor. If the owner ignores the message and it bounces, the content can be transferred by some other method, such as Wimp_ProcessKey or OS_Byte 138. Note that in this case the sender is responsible for remapping to the current Alphabet.

Notes

For example, to send the Author's name "Лев Николаевич Толстой" one could use:

+20 / "Õþ ½ØÚþÛÐÕÖØç ÅþÛáâþÛ",0 / (chrs shown in Latin1)

+42 – "Õþ ½ØÚþÛÐÕÖØç ÅþÛáâþÛ",0: "ECyrillic",0

[RISC OS Open source page](#)

Character Codes

WindowManager

Within the Desktop the Window Manager redefines the function, cursor, and special keys to return their keyboard buffer insert values, and redefines TAB to be a function key.

Key · None · Shift · Ctrl · Ctrl Shift

Backspace · &008 · &008 · &008 · &008

Return · &00D · &00D · &00D · &00D

Escape · &01B · &01B · &01B · &01B

Home · &01E · &01E · &01E · &01E

Delete · &07F · &07F · &07F · &07F

Print · &180 · &190 · &1A0 · &1B0

F1 · &181 · &191 · &1A1 · &1B1

F2 · &182 · &192 · &1A2 · &1B2

F3 · &183 · &193 · &1A3 · &1B3

F4 · &184 · &194 · &1A4 · &1B4

F5 · &185 · &195 · &1A5 · &1B5

F6 · &186 · &196 · &1A6 · &1B6

F7 · &187 · &197 · &1A7 · &1B7

F8 · &188 · &198 · &1A8 · &1B8

F9 · &189 · &199 · &1A9 · &1B9

Tab · &18A · &19A · &1AA · &1BA

End · &18B · &19B · &1AB · &1BB

Left · &18C · &19C · &1AC · &1BC

Right · &18D · &19D · &1AD · &1BD
 Down · &18E · &19E · &1AE · &1BE
 Up · &18F · &19F · &1AF · &1BF
 Page down · &19E · &18E · &1BE · &1AE
 Page up · &19F · &18F · &1BF · &1AF
 Left Windows1 · &1C0 · &1D0 · &1E0 · &1F0
 Right Windows1 · &1D0 · &1C0 · &1F0 · &1E0
 Menu1 · &1C1 · &1D1 · &1E1 · &1F1
 Hana2 · &1C2 · &1D2 · &1E2 · &1F2
 Kanji2 · &1C3 · &1D3 · &1E3 · &1F3
 Romaji2 · &1C4 · &1D4 · &1E4 · &1F4
 NoConvert2 · &1C5 · &1D5 · &1E5 · &1F5
 Convert12 · &1C6 · &1D6 · &1E6 · &1F6
 Previous12 · &1D6 · &1C6 · &1F6 · &1E6
 Kana12 · &1C7 · &1D7 · &1E7 · &1F7
 Kata12 · &1D7 · &1C7 · &1F7 · &1E7
 AllKanji2 · &1C8 · &1D8 · &1E8 · &1F8
 KanjiNumber2 · &1C9 · &1D9 · &1E9 · &1F9
 F10 · &1CA · &1DA · &1EA · &1FA
 F11 · &1CB · &1DB · &1EB · &1FB
 F12 · &1CC · &1DC · &1EC · &1FC
 Insert · &1CD · &1DD · &1ED · &1FD
 MouseDown3 · &1CE · &1DE · &1EE · &1FE
 MouseUp3 · &1CF · &1DF · &1EF · &1FF

In practice

1 Note that some keys generate the same code, for example 'Right Windows' and shift + 'Left Windows' both generate Wimp key code &1D0. Unfortunately this is unavoidable due to the lack of spare key codes. It is therefore proposed that the use of Ctrl with the 'Left Windows', 'Right Windows' & 'Menu' keys be used for extra functions, but that Shift + 'Left Windows' and Shift + 'Right Windows' be avoided. Shift with the 'Menu' may still be used however.

2 See <https://web.archive.org/web/20130401185753/http://www.iyonix.com/32bit/2205203-02.htm>

Notes

This is a list of character codes that the Wimp uses for different keys and key combinations.

[RISC OS Open source page](#)

Character Entering Buffer Event

Events

This event is generated when a byte is inserted into a buffer.

Before calling

R0 2

R1 Buffer number

R2 Byte being inserted

[RISC OS Open source page](#)

Character Sets

This chapter aims to document all the alphabet sets available under RISC OS. International Standards Organisation ISO 8859 document forms the basis of most sets.

Loading alphabets

Loading an alphabet overlays the previous alphabet. Most alphabets have a some undefined characters, indicated below in light grey. In these cases, the previous character definition for that code is unchanged.

Character codes 0 – 31 and 127 are not printable characters but have special meanings described in VDU Codes. They are indicated below by a dark grey square.

Alphabets can be loaded using OS_Byte 71 or *Alphabet.

The default alphabet

Booting of the kernel sets up a default alphabet.

This default alphabet always contains all character definitions from the Latin1 alphabet. N.B. This definition has been gradually extended by adding extra characters in the range &80 – &9F (128 – 159).

In the case of characters neither defined in the Latin1 alphabet nor used by the VDU drivers, the kernel's representation varies. RISC OS 2 represents them by the underlined string 'These•characters•are•not•defined'. RISC OS 3 displays the character code's hexadecimal value. Some of these undefined characters may be used in the future to further extend the Latin1 alphabet, or their representation may change. Users are most likely to redefine these characters if necessary. Consequently, their initial representation must not be relied upon.

The configured alphabet

The default alphabet is then overlaid according to the computer's configured territory. This configuration is set by *Configure Territory. Under RISC OS 2, the overlay depends on the computer's configured country; see *Configure Country.

The window manager

Some characters are redefined when The Window Manager starts. RISC OS 2 used these to draw windows' borders, and so have to be present for correct rendering of the desktop. Some of these definitions are retained in later versions of RISC OS for backwards compatibility, but are otherwise unused. The presence of these characters must not be relied upon unless your program is running under the desktop in RISC OS 2.

Keyboard shortcuts

The *Country command explains the relationship between country, alphabet and keyboard. Some useful keyboard shortcuts exist which can be used to access various characters and alphabets. These can be used wherever you can use the keyboard: e.g. at the Command Line, in a text editor, or when entering a filename to save a file. The first two keystroke combinations allow easy switching between keyboard layouts:

Shortcut / Meaning

Alt Ctrl F1 – Meaning: Selects keyboard layout appropriate to the UK country

Alt Ctrl F2 – Meaning: Selects keyboard layout appropriate to the configured country

Alt <x> – Meaning: Enters character corresponding to character code typed

Where <x> denotes a decimal character code typed on the numeric keypad

The keyboard layout is switched by the following sequence:

- Press and hold Alt and Ctrl together
- Press F12
- Release Ctrl
- Still holding Alt, type on the numeric keypad the international telephone dialling code for the country you want (e.g. 49 for Germany, 39 for Italy, 33 for France)

- Release Alt

Character set tables

(Character set tables still to be corrected/completed.)

- Latin1 alphabet (ISO 8859/1 + Acorn extensions)

Latin1 alphabet (ISO 8859/1 + Acorn extensions)

+0 – 48: 0 · 64: @ · 80: P · 96: ` · 112: p · 128: € · 144: ' · 176: ° · 192: À · 208: Ð · 224: à · 240: ð · : 0
 +1 – 32: ! · 48: 1 · 64: A · 80: Q · 96: a · 112: q · 128: Ŵ · 144: ' · 160: j · 176: ± · 192: Á · 208: Ñ · 224: á · 240: ñ · : 1
 +2 – 32: " · 48: 2 · 64: B · 80: R · 96: b · 112: r · 128: ŵ · 144: ¢ · 160: ² · 192: Â · 208: Ò · 224: â · 240: ò · : 2
 +3 – 32: # · 48: 3 · 64: C · 80: S · 96: c · 112: s · 128: ¢ · 144: ¤ · 160: £ · 176: ³ · 192: Ã · 208: Ó · 224: ã · 240: ó · : 3
 +4 – 32: \$ · 48: 4 · 64: D · 80: T · 96: d · 112: t · 128: ¤ · 144: ¨ · 160: ¤ · 176: ´ · 192: Ä · 208: Ö · 224: ä · 240: ö · : 4
 +5 – 32: % · 48: 5 · 64: E · 80: U · 96: e · 112: u · 128: Ŷ · 144: " · 160: ¥ · 176: µ · 192: Å · 208: Ø · 224: å · 240: ø · : 5
 +6 – 32: & · 48: 6 · 64: F · 80: V · 96: f · 112: v · 128: ŷ · 144: „ · 160: | · 176: ¶ · 192: Æ · 208: Ö · 224: æ · 240: ö · : 6
 +7 – 32: ' · 48: 7 · 64: G · 80: W · 96: g · 112: w · 144: − · 160: § · 176: · · 192: Ç · 208: × · 224: ç · 240: ÷ · : 7
 +8 – 32: (· 48: 8 · 64: H · 80: X · 96: h · 112: x · 128: · · 144: − · 160: ¨ · 176: ¸ · 192: È · 208: Ø · 224: è · 240: ø · : 8
 +9 – 32:) · 48: 9 · 64: I · 80: Y · 96: i · 112: y · 128: · · 144: − · 160: © · 176: ¸ · 192: É · 208: Ù · 224: é · 240: ù · : 9
 +10 – 32: * · 48: : · 64: J · 80: Z · 96: j · 112: z · 128: · · 144: ¤ · 160: ¸ · 176: ° · 192: Ê · 208: Ú · 224: ê · 240: ú · : A
 +11 – 32: + · 48: ; · 64: K · 80: [· 96: k · 112: { · 128: · · 144: œ · 160: « · 176: » · 192: Ë · 208: Û · 224: ë · 240: û · : B
 +12 – 32: , · 48: < · 64: L · 80: \ · 96: l · 112: | · 128: ... · 144: † · 160: ¬ · 176: ¼ · 192: Ì · 208: Ü · 224: ì · 240: ü · : C
 +13 – 32: - · 48: = · 64: M · 80:] · 96: m · 112: } · 128: ™ · 144: ‡ · 160: · · 176: ½ · 192: Í · 208: Ý · 224: í · 240: ý · : D
 +14 – 32: . · 48: > · 64: N · 80: ^ · 96: n · 112: ~ · 128: % · 144: fi · 160: ® · 176: ¾ · 192: Î · 208: Þ · 224: î · 240: þ · : E
 +15 – 32: / · 48: ? · 64: O · 80: _ · 96: o · 128: • · 144: fl · 160: ¯ · 176: ¸ · 192: Ï · 208: ß · 224: ï · 240: ÿ · : F
 0 – 16: 1 · 32: 2 · 48: 3 · 64: 4 · 80: 5 · 96: 6 · 112: 7 · 128: 8 · 144: 9 · 160: A · 176: B · 192: C · 208: D · 224: E · 240: F

- RISC OS 2: characters &80 – &9F (128 – 159) undefined.

- RISC OS 3: characters &80 – &8B (128 – 139) undefined.

Characters 128-159 are not part of the ISO Latin 1 character set, and were partially populated by Acorn. The dark grey characters in the above table may be defined by the Wimp when plotting text in System Font.

• Note: characters 128-159 (aka x80 to x9F) aren't part of the Latin 1 character set – refer also to Unicode in RISC OS at riscos.info.

[RISC OS Open source page](#)

Close_Window_Request

Reason Code 3 · WindowManager

This code is returned from Wimp_Poll and indicates that a window needs to be closed.

On return

Block

+0 Window handle

Notes

To close a window, an application should call Wimp_CloseWindow.

Any panes associated with a window will need to be closed individually.

Windows used to edit documents that had the close clicked using the adjust mouse button, should open the filer window where the document is stored. To identify which mouse button was clicked, Wimp_GetPointerInfo should be called.

[RISC OS Open source page](#)

CnpV

Vectors

This vector is called to count the number of entries or spaces in a buffer, or to flush the contents of a buffer.

Before calling

R1 Buffer number

V flag and C flag encode the action:

V flag = 0, C flag = 0: Return number of entries

V flag = 0, C flag = 1: Return amount of free space

V flag = 1: Flush buffer

On return

R0 Corrupted

R1-R2 Preserved (if purge), else;

R1 Count AND &FF

R2 Count >> &FF

Notes

If called via OS_CallAVector then interrupts must be disabled first.

[RISC OS Open source page](#)

Colour Mapping Descriptor

OS_SpriteOp

A colour mapping descriptor is a 2-word structure containing a pointer to a ColourTrans Transfer Function and a workspace pointer.

Word / Contents

0 – Contents: Workspace pointer for transfer function

1 – Contents: ColourTrans Transfer Function pointer

In practice

See ColourTrans Transfer Function for more details on how colour mapping operates.

Colour mapping descriptors are accepted by the following SWIs:

- OS_SpriteOp 52 (Paint sprite scaled)
- OS_SpriteOp 56 (Plot sprite transformed)
- OS_SpriteOp 65 (Tile Sprite Scaled)
- JPEG_PlotScaled
- JPEG_PlotFileScaled
- JPEG_PlotTransformed
- JPEG_PlotFileTransformed

[RISC OS Open source page](#)

ColourPicker

List of modules

The ColourPicker modules provides a dialogue window specifically to select a colour. The dialogue is not a Wimp task. The colour picker makes use of the Wimp filter mechanism to receive events for its dialogues. A colour picker client is an application which use the colour picker. All clients must be Wimp tasks.

Details

- Messages
- Service Calls
- SWI Calls
- *Commands

[RISC OS Open source page](#)

ColourTrans

List of modules

The ColourTrans module provides functionality for choosing the most appropriate colour available, given input values of Red, Green and Blue, and the colour depth of the desired Screen Mode. Because it understands all the pixel formats supported by the system, using ColourTrans is the recommended way of setting colours for rendering or converting RGB colours to pixel values.

In practice

ColourTrans also provides an intelligent way to handle colour with sprites and the Font Manager, and is the best way to set up colours when printing. Colour calibration functionality is also supported. It is recommended that colour routines be shared between screen and printer devices. This ensures that code is not duplicated within tasks and reduces code complexity and footprint size.

Details

- Technical Details
- Service Calls
- SWI Calls
- *Commands

[RISC OS Open source page](#)

ColourTrans_ReadPalette Flags

ColourTrans

Flags

Bit / Value / Meaning

Bit 0 – Value: 0 · Meaning: R1 = pointer to sprite name (see ColourTrans mode identification)

– – Value: 1 · Meaning: R1 = pointer to sprite (see ColourTrans mode identification)

Bit 1 – Value: 0 · Meaning: Do not return flashing colours (return one word per palette entry)

– – Value: 1 · Meaning: Return flashing colours (return two words per palette entry)

All other bits reserved (must be zero)

[RISC OS Open source page](#)

ColourTrans_SelectGCOLTable Flags

ColourTrans

R5 is used to store set flags to determine what other registers contain.

Before calling

R5 0 · R1 = pointer to sprite, otherwise R1 = pointer to sprite name (see ColourTrans mode identification)

1 · Use current palette if sprite doesn't have one (otherwise default palette)

[RISC OS Open source page](#)

ColourTrans_SetGCOL Flags

ColourTrans

GCOL is only of use in 256 colour screen modes.

Bit / Meaning

Bit 7 – Meaning: 1 => Background

– – Meaning: 0 => Foreground

Bit 8 – Meaning: 1 => Use ECFs to give better approximation to the colour

– – Meaning: 0 => Do not use ECFs

[RISC OS Open source page](#)

ColourTrans_SetTextColour Flags

ColourTrans

All other bits should be set to zero for further expansion

Bits / Meaning

7 – Meaning: 1 => Background

– – Meaning: 0 => Foreground

[RISC OS Open source page](#)

ColourTrans_WriteCalibrationToFile Flags

ColourTrans

Flags

Bits / Meaning

Bit 0 – Meaning: 1 => Always write calibration to file

– – Meaning: 0 => Write calibration unless it is the default

[RISC OS Open source page](#)

ColourTrans_WritePalette Flags

ColourTrans

Flags

Bit / Value / Meaning

Bit 0 – Value: 0 · Meaning: R1 = pointer to sprite name

– – Value: 1 · Meaning: R1 = pointer to sprite

Bit 1 – Value: 0 · Meaning: Clear flashing state (set both flash colours to same value)

– – Value: 1 · Meaning: Write flashing state (each source entry is two words giving both flash states)

Bit 2 – Value: 0 · Meaning: If writing to a 256 colour sprite palette, attempt to compact down to a VIDC1 compatible 16 entry palette

– – Value: 1 · Meaning: Disable palette compaction

All other bits reserved (must be zero)

[RISC OS Open source page](#)

ColourV

Vectors

This vector is used to indirect all SWI calls made to the ColourTrans module.

Before calling

R8 ColourTrans SWI number

Notes

The default action is to call the routine in the ColourTrans module that decodes and executes SWIs.

This vector passes all ColourTrans calls.

[RISC OS Open source page](#)

Configure

Configure plugin windows should have “Set” and “Cancel” action buttons, and where appropriate, a “Default” one to restore sensible factory defaults.

Action buttons

Mouse button / Action button / Action to be taken

Adjust – Action button: Set · Action to be taken: Configure the current selections

Select – Action button: Set · Action to be taken: Configure the current selections, then close the window

Adjust – Action button: Cancel · Action to be taken: Reset the selections to the configured values (ie reset the window to the way it was when last Adjust-Set or when it opened, whichever is more recent)

Select – Action button: Cancel · Action to be taken: Just close the window – next time it is opened, it will be displaying the configured values anyway

Adjust – Action button: Default · Action to be taken: Reset the selections to the delete-power-on defaults

Select – Action button: Default · Action to be taken: Selecting the defaults without configuring them, and then closing the window, would just appear to duplicate the Cancel button, so instead this will be the same as an Adjust click on “Default”

Interactive Help

Interactive help messages should also be standardised across all plug-ins. Wherever possible, the interactive help messages shall take one of these forms, where *** should be replaced with whatever is appropriate under the circumstances, and |M is the usual sequence telling !Help to insert a line break:

Plug-in activation icon (in main window) – Help Message: Click SELECT to open the *** configuration window.

Plug-in work area – Help Message: This window allows you to configure various aspects of the ***.

Preceding labels – Help Message: This is the ***.

Numeric ranges (writable type) – Help Message: You can adjust the *** using either the keyboard or the arrows.

Numeric ranges (display type) – Help Message: You can adjust the *** using the arrows.

Writable icons – Help Message: You can enter the *** here.

String sets – Help Message: You can choose a *** from the menu.

Sliders – Help Message: You can adjust the *** using the slider.

Following labels – Help Message: The *** is measured in ***. [Use the non-abbreviated unit name, and follow by any necessary explanation of the unit, within () if fairly brief.]

Option buttons – Help Message: This indicates if ***.|MClick SELECT to select or deselect this option.

Radio buttons – Help Message: This indicates if ***.|MClick SELECT to select this alternative.

Drop zones – Help Message: You can drop a *** here.

“Try” action buttons – Help Message: Click SELECT to sample the selected ***.

“Default” action buttons – Help Message: Click SELECT to reset the selections to the factory defaults.

“Cancel” action buttons – Help Message: Click SELECT to close this window without configuring the selections.|MClick ADJUST to reset the selections to the configured settings.

“Set” action buttons – Help Message: Click SELECT to configure the selections and close this window.|MClick ADJUST to configure the selections without closing this window.

Messages File

The plug-in's Messages file should contain three tokens:

Filename / Description

_ConfigText – Description: the name of the plugin

_ConfigHelp – Description: the help text provided for the icon on the main window

_ConfigSprite – Description: the name of the plug-ins sprite for the main window in the Wimp sprite pool

Displaying Icons

Each icon in the main window has three attributes which the central application has to determine, but which are best known to the plug-in:

- the (brief) textual name
- the sprite to use
- the interactive help string

Method for displaying Icon Sprite for compliant PlugIn

Configure will *IconSprites a file called CoSprite (or CoSprite22 etc. according to the usual convention) within each plug-in to ensure the sprite is available. To avoid name clashes with other Wimp sprites, the sprite names will be prefixed with the three characters "co_".

Each plugin should use specify the sprite name within the Messages file's _ConfigSprite token. The appropriate !Sprites sprite for each plug-in will be different from the CoSprite sprite, because the latter is not constrained by the 68×68 OS units rule. If no better sprite is available, the "!configure" sprite may be re-used.

Method for displaying Icon Sprite for non-compliant PlugIn

In order to support protocol-ignorant applications as plug-ins, the following rules are followed:

- by default the text will be the name of the plug-in application (excluding the ! character)
- the sprite will be the name of the plug-in application (excluding the ! character)
- the help text will be a generalised string

Note: The above rules are a fallback position, and must not be relied upon by applications intended for use as plug-ins.

Information Transfer

In order to support as many existing configuration applications as possible, messaging supported by a plug-in can be limited to Message_Quit, although this will result in inconsistencies in window positioning.

Window positioning is handled via two routes: when the plug-in is started through Wimp_StartTask, the preferred co-ordinates of the top-left corner of the new window (in decimal OS units) are passed as arguments on the command line. A typical command line thus ends:

```
...!InetSetup -openat 120 844
```

(If the plug-in is started without this command line argument – eg if its application directory is double-clicked from a directory display – it must centre itself on screen instead.)

The other communication is necessary to bring the window to the top of the stack when the user clicks on an icon in the main window, and the plug-in window in question is already open. This is handled by Message_OpenConfigWindow.

The other main messaging is involved with allowing plug-ins to object to being quit, for example if complex selections have been made but not yet set. This is handled by the plug-in Quit protocol.

There has been another operation in the past where information was passed between parent and child configuration windows: Adjust-clicking on the close icon of a child would re-open the parent if it had been closed due to an icon in the main window having been clicked using Adjust. This would be difficult to implement across different tasks, and near impossible where the child task predates any release of the new specification of Configure. Instead, we will take advantage of the Style Guide convention that windows having a "Cancel" action button don't have a close icon, and apply that convention to all plug-ins; the main window will no longer close when Adjust is used to click on an icon.

Locking

The central application is responsible for detecting whether the condition of the machine is unsuitable for configuration – a comparatively complex decision based upon whether the machine is remotely or locally booted, the <Choices\$Write> access permission and the condition of FSLock.

Since the plug-in applications are located deep within !Boot, it is assumed that the average user will only ever access them through Configure, which only allows the plug-ins to be loaded when the machine is in a configurable state.

The alternative invocation method involves double-clicking on the plug-in application directly. Rather than requiring each plug-in to duplicate the lock detection code from Configure (which might conceivably be subject to change at a later date anyway), the plug-ins must not complain if a write operation to disc or CMOS fails. However, the settings must still be applied for the current session even if such a failure occurs; this will allow more knowledgeable users to adjust the operation

of even locked machines where an alternative GUI is not available (eg mouse speed, Filer filename width, screen saver textual message).

Data Interchange

With the introduction of modularity into Configure, a limited amount of messaging will be introduced between the main application and the plug-ins. This is designed to provide the best possible results when third party configuration applications which are unaware of the protocol are used as plug-ins. There are three separate message protocols involved.

Quit Protocol

Applicability: all plug-ins where selections would be fiddly or time-consuming to repeat - the boot sequence editors, for example.

The established Message_PreQuit method cannot be used to warn plug-ins that they are about to be killed (which can happen either because of a click on the close icon of the main window, or because of the activation of FSLock). The problem is with the re-initialisation of the procedure if the user confirms that they do indeed want to quit anyway; Message_PreQuit allows only for the quitting of a single task or for the termination of the entire desktop.

When the central application receives a close window request, a Message_PluginQuitContinue, or a Message_FSLockStateChangeRequest for a lock status of 2, it will broadcast Message_PluginQuit with event type 18. If no plug-in responds, the Wimp will return the Message_PluginQuit to the central application, which will then send a Message_Quit to each plug-in individually (except the Lock plug-in, in the case of lock activation).

This protocol is also used between top-level and second-level plug-ins in a similar fashion when necessary. The main difference is that the Message_Quit contains a flag word to indicate whether the quit was generated by a click on "Set" rather than "Cancel", because in the former case the plug-in may assume that the user wants to set the selection (and need not therefore generate a Discard/Cancel/Set dialogue box).

Plug-ins must behave as follows (in addition to handling Message_PreQuit as normal) if they wish to be able to object to being quit:

Initialisation

- Set "complain" flag

Message_PluginQuit received

- If "complain" and "modified" flags are set acknowledge the message so that it isn't returned to the central application; query the user whether they want to "Discard", "Cancel" or "Set" (this can span Wimp_Polls); if "Discard" or "Set" is selected, clear "complain" flag; send Message_PluginQuitContinue to the task that broadcast the Message_PluginQuit else (ie "Cancel" is selected), broadcast Message_PluginQuitAbort

Message_PluginQuitAbort received

- Set "complain" flag, ready for the next time

[RISC OS Open source page](#)

Currently Active Object Handler

OS_ChangeEnvironment

The currently active object environment handler is not a code entry point. Instead, the value specified in R1 to OS_ChangeEnvironment is used to set or read the Currently Active Object Pointer.

[RISC OS Open source page](#)

Data Abort Handler

OS_ChangeEnvironment

The data abort handler is entered in the following state:

In practice

- The processor will be in ABT32 mode, with IRQs disabled
- FIQ state will match that at the time of the exception
- R14 & SPSR_und will contain the exception address and saved PSR, corresponding to when the exception was raised by the CPU. R14 will be pointing two instructions (i.e. 8 bytes) after the instruction that caused the exception.
- The other core registers will be as per when the exception was raised by the CPU. However if your code examines the registers it must take care to use the correct register bank (e.g. if the exception was from user mode, it should look at R14_usr, not R14_abt).
- R13 will be a valid stack pointer
- CP15 registers relating to exception processing will be in the correct state corresponding to that exception

If the handler is capable of handling the exception (e.g. by emulating the memory access) then it can resume execution by performing a suitable exception return operation.

Notes

On 26bit OS versions, the CPU will be in SVC26 mode, with R14 containing the combined LR+PSR.

The kernel's default handler will save a copy of the ARM integer registers (corresponding to the CPU mode the exception occurred in) to the Exception Registers Block, reset the stack pointers for all the privileged CPU modes, and then call OS_GenerateError to report a Data Abort error.

Since abort environment handlers are called before the abort recovery code that exists in the kernel's default environment handler, custom environment handlers should be very careful in how they operate, especially when it comes to calling external code (e.g. SWIs). For example, if the abort was caused by a supervisor stack overflow, calling a SWI from within your abort handler would result in a terminal abort loop.

[RISC OS Open source page](#)

Date Format String Tokens

All numbers may be preceded by Z to suppress leading zeros.

- %0 – Meaning: ASCII zero value
- %DY – Meaning: Date (08)
- %ST – Meaning: st, th or nd by date
- %MN – Meaning: Month as number (04)
- %MO – Meaning: Month as long entry (April)
- %M3 – Meaning: Month as 3 letters (Apr)
- %WK – Meaning: Week of year
- %DN – Meaning: Day within year (098)
- %WN – Meaning: Week day as number (7 = Saturday)
- %W3 – Meaning: Week day as 3 letters (Sat)
- %WE – Meaning: Week day as long (Saturday)
- %CE – Meaning: Century as 2 digits (19)
- %YR – Meaning: Year as 2 digits (95)
- %24 – Meaning: Hour as 24 hour (15)

%12 – Meaning: Hour as 12 hour (03)
%AM – Meaning: am or pm
%PM – Meaning: am or pm
%MI – Meaning: Minute as 2 digits (25)
%SE – Meaning: Second as 2 digits (50)
%CS – Meaning: Centisecond as 2 digits (87)
%TZ – Meaning: Time zone (BST/DST+01:00, etc)
%% – Meaning: A % symbol
[RISC OS Open source page](#)

Debugger

List of modules

The Debugger module provides several *Commands and SWIs intended to aid debugging software. Starting with Debugger 1.92, the module is also able to produce annotated stack dumps when a serious error or exception occurs.

Details

- Exception Dumps
- SWI Calls
- *Commands

[RISC OS Open source page](#)

Debugger Exception Dumps

Debugger

Exception dumps are a new feature introduced with version 1.92 of the Debugger module. The primary aim of an exception dump is to collect an accurate snapshot of the system state at the time of an unhandled exception or other serious error (e.g. data abort). The secondary aim is for the exception dump to contain enough information that a developer who does not have access to the originating machine will be able to easily identify the function call and parameters that caused the error.

Quick start guide

Entering the following two commands will enable output of annotated exception dumps to the file \$.excdump:

```
*Set Debugger$AnnotatedFile $.excdump
*Set Debugger$DumpOptions -file annotated
```

Details

The exception dump implementation offers the following features:

- Capture of the following machine state: Capture of the core ARM registers at the time of the exception (R0-R15 and CPSR, and banked registers for USR, SVC, IRQ, ABT and UND modes) FIQ registers are not currently captured Full capture of the default IRQ, SVC, ABT and UND mode stacks FIQ mode has no default stack and is not captured Auxiliary SVC stacks (e.g. as used by RTSupport) are not currently captured Capture of the error block that's associated with the exception Capture of important kernel workspace values, such as the current IRQsema value
- The ability to output the captured dump in either raw (binary) or annotated (text) format The annotated output makes use of additional machine state in order to produce much richer output than that which is currently available in the binary

format. For example, the annotated output will attempt to label key code addresses by the module and function they belong to, and it will also include a partial USR mode stack dump.

Note that exception dump functionality is considered to be a work-in-progress, and so its function or the method of configuration may change in future OS versions.

Configuration

Exception dump collection and processing currently defaults to disabled on machine startup. The following system variables can be configured in order to enable it.

Debugger\$DumpOptions

This is a code variable which acts as the primary configuration interface. When set to an empty string, the Debugger will not be registered with SeriousErrorV and there will be no effect on the processing of exceptions/serious errors by the kernel. When set to a valid non-empty value, the Debugger will register with SeriousErrorV and the appropriate features will be enabled.

The basic syntax of the variable is that of a space-delimited sequence of options and values, similar to the way command line arguments are specified for programs. The following options are supported:

Option / Use

-file [format[,format[,...]]] – Use: Enable file output in the given format(s)

-hal [format[,format[,...]]] – Use: Enable HAL output in the given format(s)

Valid formats are raw and annotated, for the raw (binary) and annotated (text) forms respectively.

File output will cause the dump files to be saved to files on disc; the filenames must be specified in the Debugger\$RawFile and Debugger\$AnnotatedFile variables as appropriate.

HAL output is sent to the HAL via HAL_DebugTX, and so will typically require a connection to a remote machine via a serial cable. This allows collection of logs in situations where the filesystem is unusable. Also note that raw output via the HAL is currently sent as a hex dump rather than true binary.

Any combination of output destinations and formats can be enabled at the same time. For example, `-file raw,annotated -hal raw,annotated` will enable all output destinations and formats. Raw output is produced before annotated output, so if the annotation crashes then you will still have at least some information to go by.

Debugger\$DumpOptions can also be set to a value of `-none`, which will cause the Debugger to collect exception dumps but not process them. This allows other software such as ZeroPain to request manual processing via the SeriousErrorV_CustomReport reason code.

Debugger\$RawFile

This must be set to the filename to save raw binary dumps to when the `-file raw` option is used.

Debugger\$AnnotatedFile

This must be set to the filename to save annotated text dumps to when the `-file annotated` option is used.

Events that cause exception dumps to be collected

Within the kernel, unhandled data aborts, prefetch aborts, or undefined instructions for which a serious error is about to be raised will result in calls being made to SeriousErrorV. If exception dump processing is enabled in the Debugger then the exception dumps will be collected and output. A branch through zero will also trigger an exception dump via the same mechanism.

Note that if a program deliberately triggers an exception with the intent of catching it via an Error Handler, this will result in a “false-positive” exception dump being produced.

The ZeroPain module also contains logic to allow it to manually trigger the collection and processing of an exception dump whenever a zero page access is caught. This involves manually calling SeriousErrorV from ABT mode, something which is not currently supported by standard OS APIs (using OS_CallAVector would result in a switch to SVC mode, potentially overwriting important SVC stack content or triggering a stack overflow). Developers who are interested in triggering exception dumps from their own code are encouraged to get in touch with ROOL so that official APIs can be devised.

File formats

Please see Raw Exception Dumps and Annotated Exception Dumps for details of the file formats.

[RISC OS Open source page](#)

DeskFS

List of modules

DeskFS is no longer used. The following information is for reference only, and should not be used.

In practice

DeskFS was the ROM based filing system that provided system resources for the Desktop in RISC OS 2. It is no longer used nor available from RISC OS 3 and beyond.

The Desktop used the system variable Wimp\$Path to reference these system resources; by default its value was DeskFS;, although it was possible to change this.

A single *command was provided by the DeskFS filing system: *DeskFS. It is no longer used.

[RISC OS Open source page](#)

DeviceFS

List of modules

The DeviceFS module provides a standardised interface to device drivers within the RISC OS environment. Devices are declared within the system, and are seen as objects within the 'devices:' filing system.

Details

- Technical Details
- Service Calls
- SWI Calls

[RISC OS Open source page](#)

DeviceFS Technical Details

DeviceFS

Special Fields within the DeviceFS are commonly used to specify parameters to the device. ie what buffers to be used, if the device should be flushing when a stream is closed and so on.

Special Fields

A validation string can be specified to parse the special field when the stream is being opened. DeviceFS will parse the string and return a block of data that relates to the string contents. The data will remain intact for as long as the stream remains open. If no validation string is specified then it is the duty of the device to take and manage a copy of the data and filter out any unwanted information.

Validation Strings

The syntax for validation strings is as follows:

keyword [, keyword]/ escape_seq [/ escape_seq]

Keywords

Keywords associate each command with an escape sequence. More than once keyword can be associated with a particular escape field. The reason for this is twofold

- When a different word has the same meaning. e.g. Colour or color
- When defining the various states for a switch

The escape sequence describes how preceding data should be validated and what to do with the rest of the special field string (up to the next separator).

The Characters below are valid within in escape sequences:

/N number /S switch

Parameters

Within the special field string each parameter is separated by a comma or a character which is out of place, ie a non-numeric in a numerical field. Each keyword within the special field string is separated by a semi-colon.

The buffer passed to the device contains 1 word per escape character, set to &DEADDEAD if the corresponding keyword is not present in the special field string.

Numbers are read with OS_ReadUnsigned and stored in the word. Switches store the number of the parameter in the switch's validation string.

Example

Given a device called Serial and a validation string of

baud,rate/Nnohandshake,xon,rts,dtr/Snoparity,even,odd/Sdata/N

the file name serialbaud19200;noparity: would yield

Offset / Value / Explanation

0 – Value: 19200 · Explanation: number

4 – Value: &DEADDEAD · Explanation: handshake keyword not supplied

8 – Value: 0 · Explanation: noparity is first entry in switch

12 – Value: &DEADDEAD · Explanation: data keyword not supplied

and serialrate28800;dtr;even;data7: would yield

Offset / Value / Explanation

0 – Value: 28800 · Explanation: rate is an alias for baud

4 – Value: 3 · Explanation: dtr is fourth entry in switch

8 – Value: 1 · Explanation: even is second entry in switch

12 – Value: 7 · Explanation: number

This yields 0 if 'mike' is present within the string, 1 if 'dennis' is present within the string.

The order of commands within the validation string and the special field string need not match; the commands within the validation string control how the values are returned back to the caller.

Device Description

Devices are specified as a list of 28 byte blocks, terminated by a null word.

0 – Contents: Offset to zero terminated device name from start of this block

4 – Contents: Bit 0 set for buffered

Bit 1 set to create path variable (e.g. Serial\$Path for a device Serial)

8 – Contents: Default RX buffer flags

12 – Contents: Default RX buffer size

16 – Contents: Default TX buffer flags

20 – Contents: Default TX buffer size

24 – Contents: Reserved (must be zero)

The device name is used:

- in the DeviceFS directory structure
- to create a DeviceFS\$Device\$options variable if this does not exist
- a path variable (if the flag bit is set)

[RISC OS Open source page](#)

DHCP

List of modules

The DHCP module provides the Dynamic Host Configuration Protocol functionality to RISC OS.

Details

- Service Calls
- SWI Calls
- *Commands

[RISC OS Open source page](#)

DisplayManager

List of modules

The DisplayManager module provides the functionality to select screen resolution, number of colours and other screen related parameters within the desktop.

In practice

The Display manager utility allows a selection of pre-defined modes to be chosen; custom modes can also be used and defined.

Details

- Service Calls
- *Commands

[RISC OS Open source page](#)

DMB_Read ARMop

OS

Although this call doesn't guarantee that any memory operation completes, it's usually all that's required when interacting with hardware devices which use memory-mapped IO. E.g. after reading a hardware register to detect that a DMA write to RAM has completed, issue a read barrier to ensure that any reads from the data buffer see the final data.

Before calling

--

On return

R0 Corrupt

In practice

- Ensures in-order operation of data load instructions
- Does not stall instruction execution
- Does not guarantee that any preceeding memory operations complete in a timely manner (or at all)

Notes

There is no direct equivalent to this in ARMv7 (barriers are either W or RW). However it's useful to define a read barrier, as (e.g.) on Cortex-A9 a RW barrier would require draining the write buffer of the external PL310 cache, while a R barrier can simply be an ordinary DMB instruction.

[RISC OS Open source page](#)

DMB_ReadWrite ARMop

OS

This call is roughly equivalent to the ARMv7 "DMB SY" instruction:

Before calling

--

On return

R0 Corrupt

In practice

- Ensures in-order operation of data load/store instructions
- Does not stall instruction execution
- Does not guarantee that any preceeding memory operations complete in a timely manner (or at all)

Notes

Although this call doesn't guarantee that any memory operation completes, it's usually all that's required when interacting with hardware devices which use memory-mapped IO. E.g. fill a buffer with data, issue a DMB, then write to a hardware register to start some external DMA. The writes to the buffer will have been guaranteed to complete by the time the write to the hardware register completes.

Although ARMv7 has DSB and DMB instructions which perform write buffer drain-style operations in a cache-agnostic manner, these instructions do not operate on caches which are not fully integrated with the CPU - e.g. the PL310 L2 cache that's typically used on Cortex-A9 systems. Therefore, to get a barrier that ensures the correct ordering of memory accesses across all levels of the cache/memory hierarchy, the memory barrier ARMops must be used rather than a direct ARMv7 barrier instruction.

[RISC OS Open source page](#)

DMB_Write ARMop

OS

This call is roughly equivalent to the ARMv7 "DMB ST" instruction:

Before calling

--

On return

R0 Corrupt

In practice

- Ensures in-order operation of data store instructions
- Does not stall instruction execution
- Does not guarantee that any preceeding memory operations complete in a timely manner (or at all)

Notes

Although this call doesn't guarantee that any memory operation completes, it's usually all that's required when interacting with hardware devices which use memory-mapped IO. E.g. fill a buffer with data, issue a DMB, then write to a hardware register to start some external DMA. The writes to the buffer will have been guaranteed to complete by the time the write to the hardware register completes.

Although ARMv7 has DSB and DMB instructions which perform write buffer drain-style operations in a cache-agnostic manner, these instructions do not operate on caches which are not fully integrated with the CPU - e.g. the PL310 L2 cache that's typically used on Cortex-A9 systems. Therefore, to get a barrier that ensures the correct ordering of memory accesses across all levels of the cache/memory hierarchy, the memory barrier ARMops must be used rather than a direct ARMv7 barrier instruction.

[RISC OS Open source page](#)

DOSFS

List of modules

The DOSFS module provides an image filing system to the Operating System. This filing system allows it to access DOS file formats.

Details

- Technical Details?
- SWI Calls
- *Commands

[RISC OS Open source page](#)

DragASprite

List of modules

The DragASprite module provides functionality to drag a sprite around the screen using the pointer.

In practice

To drag a sprite the following process steps should be followed:

- Choose a sprite that you wish to use to display when dragging
- Call the SWI DragASprite_Start to begin the drag. This SWI takes a copy of the sprite so the original can be removed from memory if required
- When the Wimp sends an indication that the drag has finished via Wimp_Poll, DragASprite_Stop should then be called.

Notes

Because a user may not wish for a sprite to be displayed, for whatever reason, a well written application should always check user preferences to see if a sprite should be displayed, or alternatively just display an outline.

Details

- SWI Calls

[RISC OS Open source page](#)

Draw

List of modules

The Draw Module provides vector-graphics drawing, based on MOVE/DRAW and Bezier curve primitives; much like a subset of Postscript. A sequence of primitive operations is called a path. Objects can be obtained by grouping paths.

In practice

Paths and objects can be manipulated in a number of different ways. A comprehensive list of options is provided below:

- Rotation, scaling and translation of the path
- Thickness of a line
- Description of dots and dashes for a line
- Joins between lines can be mitred, round or bevelled
- Leading or trailing ends of a line can have the following shapes: butt, round, squared or triangled
- Filling of arbitrary shapes
- Fill the interior of a shape

A path can be manipulated directly on screen or in memory using SWI calls.

Details

- Technical Details
- Draw Path Block

- Draw Fill Style
 - Draw Dash Block
 - Draw Join_Cap Block
 - SWI Calls
- [RISC OS Open source page](#)

Draw Dash Block

Draw

Each element gives a length of dash, starting with an on dash, alternating on-off, looping round the list of elements.

Dash Block

Size / Meaning

- 4** – Meaning: Distance into dash pattern to start (user co-ordinates)
- 4** – Meaning: n - number of elements in pattern
- 4n** – Meaning: Dash elements (each is a length in user co-ordinates)

Notes

Hence, if n is odd, the second time round the elements on/off values will be reversed.

[RISC OS Open source page](#)

Draw Fill Style

Draw

A value of 0 for the fill style uses the default fill style (value &30). This is equivalent of boundary centre, non-zero rule.

Fill Styles

- 0,1** – Meaning: 0 Non-zero winding rule
 - 1** Negative winding rule
 - 2** Even-odd winding rule
 - 3** Positive winding rule
 - 2** – Meaning: Plot non-boundary exterior pixels
 - 3** – Meaning: Plot boundary exterior pixels
 - 4** – Meaning: Plot boundary interior pixels
 - 5** – Meaning: Plot non-boundary interior pixels
- All other bits must be 0 (reserved)

[RISC OS Open source page](#)

Draw Join_Cap Block

Draw

The mitre limit at offset 4, must be stored in the form &ffff.ffff. If the distance projected by mitres would be greater than this proportion of the line width then it will be bevelled.

Join/Cap Block

- +0 – Meaning: Join style
- +1 – Meaning: Start cap end style
- +2 – Meaning: End cap end style
- +3 – Meaning: 0 (reserved)
- +4 – Meaning: Mitre limit, in form &ffff.ffff (the same form as Draw Transform Units)
- +8,9 – Meaning: Additional width of start triangular cap,
- +10,11 – Meaning: Extension of start cap from line
- +12,13 – Meaning: Additional width of end triangular cap
- +14,15 – Meaning: Extension of end cap from line

Notes

The values in offsets 8 thru 15, are stored in $1/256^{\text{th}}$ of line width.

If mitres and triangles are not used then the block may be 4 bytes long. If mitres are used, but triangles are not, then the block may be 8 bytes long.

Regrettably, triangles cannot be used on thin lines. It would have been possible to use absolute widths and lengths on thin paths, but alas this is not provided.

[RISC OS Open source page](#)

Draw Module Technical Details

Draw

Great care must be taken when using the SWIs provided by the Draw module because four different units of measurement are used.

Definitions

- A path element is a sequence of words. The first word denotes a command number, called the element type, in the bottom byte. The remaining words in the sequence are parameters for the particular element type.
- A subpath is a sequence of path elements that define a single connected polygon or curve. If the subpath joins the initial path element it is said to be closed. If it does not, it is said to be an open. A subpath can cross itself or other subpaths within the same path.
- A path is a sequence of subpaths, each of which is a sequence of elements.
- A Bezier curve is a type of smooth curve connecting two endpoints, with both the direction and curvature specified by two control points.
- Flattening is the process of converting a Bezier curve into a series of straight lines when outputting. Flatness is used to specify how closely the outputted lines will resemble the original Bezier curve. The flatness is the maximum distance that a line is allowed to be from a Bezier curve when flattening it.
- A transformation matrix is the standard mathematical tool for two-dimensional transformations using a three by three dimensional array. This allows for rotating, scaling and translating.

- Stroke is a term that means to draw a thickened line centred on a particular path.
- A gap is transparent line segment in a subpath. If the subpath is stroked, the piece around the gap will not be plotted. Gaps are used by Draw to implement dashed lines.
- Line caps are placed at ends of open subpaths and at the ends of dashes in a dashed line when they are stroked. They can be butted, round, squared or triangular.
- Joins occur between adjacent lines, and between the start and end of a closed subpath. They can be mitred, round or bevelled.
- Fill is simply to draw everything within a particular path.
- Interior is a term used to specify which pixels are to be filled. Exterior is a term used to specify which pixels must not be filled.
- Winding number rule is the method used to determine what is interior or exterior to a path when filling.
- Boundary pixels are those that would be drawn if the line were stroked with a minimum thickness for the screen.
- Thickening a path is used to convert it to a particular thickness.

Measurement Systems

- OS Units are the measurements used by the Window Manager for specifying output to the screen. This is also used by the Draw module when it strokes a path onto the screen.
- Internal Draw Units are $1/256^{\text{th}}$ of an OS unit. This measurement system is only used for its own internal calculations. A 32-bit number is used to represent the internal Draw unit. The first 24 bits are used to specify the integer part of the number, and the bottom 8 bits are used to represent the fractional part of the OS unit.
- User Units are co-ordinates of a path using any compatible measurement system, as they are converted by the transformation matrix into internal Draw units when generating the output. Because it is a fixed point system, scaling problems can occur if the measurement system used is differs too much from Draw own internal measurement system. To avoid scaling issues, it is recommended not to use a numbering system greater than 8, compared to the Internal Draw unit measurement system.
- Transform Units are only used to specify some numbers in the transformation matrix. They are fixed point numbers, with the decimal point between bits 15 and 16. In other words, this means that the numbers are &XXXX.XXXX. That is, the top two bytes contains the whole part of the number, the bottom two bytes the fractional part.

The Transformation Matrix

The transformation matrix is the method used to perform numerous functions on lines, such as scaling, rotating and transforming. It consists of a three by three array. A full explanation of how it is used can be found [here](#).

Winding Rule

The winding rule is used by the Draw module to determine which pixels are interior or exterior, that is, which pixels should be filled.

Four different methods of determining this are provided.

Even-Odd · Specifies that an area should be filled if it is enclosed by an even number of subpaths. This results in an area never having two adjacent areas of the same state. i.e filled or unfilled

Non-Zero · Determines whether an area should be filled based on the direction of the surrounding subpaths were constructed. If an equal number of subpaths in each direction surround the area, then the area is not filled, otherwise it is

Positive · Fill an area if it is surrounded by more anti-clockwise subpaths than clockwise

Negative · Fill an area if it is surrounded by more clockwise subpaths than anti-clockwise

Note: The even-odd and non-zero winding rules are printer driver compatible, whereas the positive and negative are not.

Stroking and Filling

Flattening is the process of converting a Bezier curve into a series of lines within a specified distance of the curve. The flatness is the term used to specify how closely the outputted lines will resemble the original Bezier curve.

The shorter the distance of line, the more accurate resemblance to the original Bezier curve. A longer line will hence, result in a less accurate approximation.

All moving and drawing is relative to the VDU graphics origin. None of the Draw SWIs will plot outside the boundaries of the VDU graphics window. All calls use the colour (both pixel pattern and dash pattern) set up for the VDU driver. Note that not all such colours are compatible with printer drivers.

Paths

The path is a sequence of subpaths, each of which is a sequence of elements. Each element is from one to seven words in length. The lower byte of the first word specifies the element type and the remaining three bytes are free for use. On output to an input path, the Draw module will leave the three bytes unchanged, but if output to an output path, the three bytes will store zeros. A table showing the path structure is available [here](#).

Open and Closed Subpaths

When stroking, if a subpath ends with a 4 or 5 then it is closed and the ends are joined.

If the subpath is a 2 or a 3, then it leaves the subpath open, and the loose ends are capped. These four path elements explicitly leave a stroked subpath open or closed.

It must be noted, that some operations close open subpaths. This, however, will be stated in the descriptions.

In addition, just because the ends of a subpath have the same co-ordinates does not mean that the subpath is closed.

Output Path

Certain SWIs has the ability to output to an output rather than to the screen. After a path has been written to an output path, it is identical to an input path. When the output path is first passed to a SWI as a parameter, the start of the block should contain an element type zero, followed by the number of available bytes. This is to ensure that it will not accidentally overrun the buffer.

Flatness

Flattening is the process of converting a Bezier curve into a series of straight lines when outputting. Flatness is used to specify how closely the outputted lines will resemble the original Bezier curve. The flatness is the maximum distance that a line is allowed to be from a Bezier curve when flattening it.

To ensure a good approximation of the Bezier curve, it is recommended that the flatness value is between 0.5 and 1.0 pixel. Any less is unnoticeable, but any more and the curve becomes noticeably jaggy.

An easy way to calculate the flatness value is:

$$\text{flatness} = \text{number of user units in x axis} / \text{number of pixels in x axis}$$

A value of zero specifies the default flatness value, which provides a good balance between speed and accuracy (using default scaling).

If outputting a path to a high resolution printer instead of a screen, it is advisable to use a smaller flatness value, that results in a more accurate representation of the curve.

Line Thickness

The thickness of the line is specified in user co-ordinates rather than OS units or Draw' own internal co-ordinate system. The following points must be noted when specifying the thickness.

- To use the default thickness value, a value of zero should be used. This draws the line with the minimum width that can be used given any particular screen resolution. i.e one pixel wide
- A value of 2 draws the line with a thickness of 1 user co-ordinate translated to pixels on either side of the theoretical line position
- A non-zero value, then the cap and join parameter must also be passed

Cap and Join Style

Cap and join styles specify how the cap and joins are displayed. Joins can be mitred, round or bevelled. Whereas caps can be butted, round, square or triangular.

More information on Cap and Join styles can be found [here](#).

Dash Pattern

The dash pattern as the name suggests, specifies the pattern of dashes used to represent a given line.

More information on the dash pattern is provided [here](#).

Floating Point

Draw uses fixed point arithmetic for speed, rather than floating point.

The SWIs used by the Draw module all use integer numbers, however, SWI names and numbers have been allocated to support future versions that support Floating Point. To use the floating point versions, the suffix 'FP' needs to be appended to the current SWI name.

The Floating Point numbers used in the specification are IEEE single precision floating point numbers.

Note: Not yet implemented and if used will result in an error if called.

Printing

If printing, it is important to note that not all features are compatible. As a general rule, the following features should be avoided:

- AND, OR and similar functions on colours when writing to the screen
- Choice of fill style. e.g fill excluding/including boundary, fill exterior etc...
- Positive and negative winding number rules
- Line cap enhancements, particularly differing leading and trailing caps and triangular caps

[RISC OS Open source page](#)

Draw Path Block

Draw

Draw path blocks consist of a block of words; `code(element type)` followed by any parameters. The block is terminated by an 'End of Path' element.

Path Block

- 0 – Parameters: - · Meaning: End of path (when writing a path, a parameter is used to indicate the remaining length of the block, which is updated as data is written and the end of path marker is moved)
- 1 – Parameters: Pointer · Meaning: Continuation of path at pointer
- 2 – Parameters: X, Y · Meaning: Move to x,y starting a new sub-path
- 3 – Parameters: X, Y · Meaning: Move to x,y without starting a new sub-path (internal use)
- 4 – Parameters: - · Meaning: Close sub-path with move
- 5 – Parameters: - · Meaning: Close sub-path with line
- 6 – Parameters: CX1, CY1, CX2, CY2, X, Y · Meaning: Bezier curve to x,y using control points cx1, cy1 and cx2, cy2
- 7 – Parameters: X, Y · Meaning: Move to x,y without starting a new sub-path (mostly used for internal use)
- 8 – Parameters: X, Y · Meaning: Line to x,y

Notes

There are some order constraints on these codes (element types):

- path elements 2 and 3 start new subpaths

- path elements 6,7 and 8 may only appear while there is a current subpath
- path elements 4 and 5 may only appear while there is a current subpath, and end it, leaving no current subpath
- path elements 2 and 3 can also be used to close the current subpath (which is part of starting a new subpath)

[RISC OS Open source page](#)

Draw_ProcessPath Fill Style

Draw

Version 1.17 of the Draw Module onwards, move the flag in bit 31 of the output pointer into R1, to allow the pointer to address the full memory range. Attempts to set the new flags on older versions of the module will cause the error 'Reserved bits non zero'.

Fill Styles

- 0,1 – Meaning: 0 Non-zero winding rule
- 1 Negative winding rule
- 2 Even-odd winding rule
- 3 Positive winding rule
- 2 – Meaning: Plot non-boundary exterior pixels
- 3 – Meaning: Plot boundary exterior pixels
- 4 – Meaning: Plot boundary interior pixels
- 5 – Meaning: Plot non-boundary interior pixels
- 25 – Meaning: Output bounding box instead of path
- 26 – Meaning: Pointer in R7 is 32-bit
- 27 – Meaning: Open sub-paths are to be closed
- 28 – Meaning: Path should be flattened
- 29 – Meaning: Path should be thickened
- 30 – Meaning: Path should be re-flattened after thickening
- 31 – Meaning: Use floating point output (not yet implemented)
- All other bits must be 0 (reserved)

Notes

Unsure if this comment is applicable:

[RISC OS Open source page](#)

Draw_ProcessPath Special Values

Draw

When specifying the output path bounding box, flags bits 25 and 26 should be used instead. This is to ensure 32-bit compatability.

Special Values

Value / Meaning

- 0 – Meaning: Output to the input buffer path
- 1 – Meaning: Fill the path in one go
- 2 – Meaning: Fill the path, doing one sub-path at a time
- 3 – Meaning: Calculate output buffer size

1<<31+x – Meaning: Output path's bounding box to block at x

x – Meaning: Output path to block at x. The first item in this buffer should be 0, followed by a word denoting the size of the buffer.

[RISC OS Open source page](#)

Draw_StrokeClipped Clip Block

Draw

Clip Block

Offset / Contents

+0 – Contents: Pointer to path block to clip against

+4 – Contents: Fill style

+8 – Contents: Pointer to transformation matrix

[RISC OS Open source page](#)

DSB_Read ARMop

OS

There is no direct equivalent to this in ARMv7 (barriers are either W or RW). However it's useful to define a read barrier, as (e.g.) on Cortex-A9 a RW barrier would require draining the write buffer of the external PL310 cache, while a R barrier can simply be an ordinary DSB instruction.

Before calling

--

On return

R0 Corrupt

In practice

- Read barrier – writes may cross the instruction
- Instructions following the barrier will only begin execution once the barrier is passed – but any prefetched instructions are not flushed from the pipeline

[RISC OS Open source page](#)

DSB_ReadWrite ARMop

OS

This call is roughly equivalent to the ARMv7 "DSB SY" instruction:

Before calling

--

On return

R0 Corrupt

In practice

- Writebuffers are drained
- Full read/write barrier - no data load/store will cross the instruction
- Instructions following the barrier will only begin execution once the barrier is passed - but any prefetched instructions are not flushed from the pipeline

Notes

Although ARMv7 has DSB and DMB instructions which perform write buffer drain-style operations in a cache-agnostic manner, these instructions do not operate on caches which are not fully integrated with the CPU - e.g. the PL310 L2 cache that's typically used on Cortex-A9 systems. Therefore, to ensure bufferable data is fully flushed to main memory, the memory barrier ARMops must be used rather than a direct ARMv7 sync instruction.

This ARMop was previously known as WriteBuffer_Drain.

[RISC OS Open source page](#)

DSB_Write ARMop

OS

This call is roughly equivalent to the ARMv7 "DSB ST" instruction:

Before calling

--

On return

R0 Corrupt

In practice

- Writebuffers are drained
- Write barrier - reads may cross the instruction
- Instructions following the barrier will only begin execution once the barrier is passed - but any prefetched instructions are not flushed from the pipeline

Notes

Although ARMv7 has DSB and DMB instructions which perform write buffer drain-style operations in a cache-agnostic manner, these instructions do not operate on caches which are not fully integrated with the CPU - e.g. the PL310 L2 cache that's typically used on Cortex-A9 systems. Therefore, to ensure bufferable data is fully flushed to main memory, the memory barrier ARMops must be used rather than a direct ARMv7 sync instruction.

[RISC OS Open source page](#)

DualSerial

List of modules

The Serial module (also known by its component name, DualSerial) creates and manages a DeviceFS device for each serial port which the HAL exposes via the HAL UART API.

Introduction

The DeviceFS devices that are created can be controlled using two protocols:

- The IOCTL protocol described in this document
- The SerialDeviceSupport protocol, which allows the legacy OS_SerialOp SWI to be used to interact with one of the serial ports. Support for this interface was introduced in DualSerial 0.25.

[RISC OS Open source page](#)

Dynamic Area Numbers

1 Available from RISC OS 3.50 onwards

- 1 – Meaning: Application slot^{1 3}
- 0 – Meaning: System Heap
- 1 – Meaning: RMA
- 2 – Meaning: Screen Area
- 3 – Meaning: Sprite Area
- 4 – Meaning: Font Cache
- 5 – Meaning: RAM Filing System
- 6 – Meaning: Free pool¹
- 7 – Meaning: VRAM rescue²
- 8 – Meaning: Kernel buffers²
- 10 – Meaning: Abort SVC and IRQ stack copy^{4 5}
- 11 – Meaning: Operating system ROM^{4 5}
- 12 – Meaning: SWI dispatch and IRQ workspace^{4 5}
- 13 – Meaning: SVC stack^{4 5}
- 14 – Meaning: IRQ stack^{4 5}
- 15 – Meaning: UND stack^{4 5}
- 16 – Meaning: Level 2 Page Tables^{4 5}
- 17 – Meaning: Operating system page tables (SoftCAM)^{4 5}
- 18 – Meaning: System Initialisation area^{4 5}
- 48 – Meaning: Zero Page^{4 5}
- 49 – Meaning: High CPU vectors^{4 5 6}
- 50 – Meaning: Utility executables^{4 5 6}
- 51 – Meaning: ABT stack^{4 5 6}
- 256+ – Meaning: User dynamic area

In practice

2 Available from RISC OS 3.80 onwards

3 Not supported by all API in Classic RISC OS.

4 Available in some RISC OS Select versions.

5 Not supported by RISC OS 5.

6 Available in RISC OS Pyromaniac versions.

[RISC OS Open source page](#)

Econet Technical Details

Econet

If the address to call is -1 then the address called is the argument buffer – this allows ARM code to be sent. This is not advised due the inherent security risk of passing data over a possible unsecure network.

Before calling

R1 Econet procedure call argument buffer

R2 Address of the code entered

R3 Sending station

R3 Sending net

On return

R0 – R4 May be corrupted

Immediate Operations

1 – Name: Econet_Peek · Description: Read destination machine's memory

2 – Name: Econet_Poke · Description: Write destination machine's memory (!)

3 – Name: Econet_JSR · Description: Calls a subroutine

4 – Name: Econet_UserProcedureCall · Description: Generate Event 8

5 – Name: Econet_OSProcedureCall · Description: Generate Event 16

6 – Name: Econet_Halt · Description: Halt the destination machine (BBC & Master only)

7 – Name: Econet_Continue · Description: BBC & Master only

8 – Name: Econet_MachinePeek · Description: Returns a number identifying the machine type

9 – Name: Econet_GetRegisters · Description: Returns R0-R15, R13_irq, R14_irq, R13_svc, R14_svc (80 bytes)

Argument Buffer

Offset / Meaning

0 – Meaning: Size of buffer

Machine Types

0-15 – Value: - · Meaning: Machine Type:

&0000 · Reserved

&0001 · BBC (OS 1 or 2)

&0002 · Atom

&0003 · System 3 or 4

&0004 · System 5

&0005 · Master 128 (OS 3)

&0006 · Electron (OS 0)

&0007 · Archimedes (OS 6)

&0008 · Reserved

&0009 · Acorn Communicator

&000A · Master 128 Econet Terminal (OS 4)

&000B · Acorn Filestore

&000C · Master 128 Compact (OS 5)
 &000D · Acorn Ecolink card for Personal Computers
 &000E · Acorn Unix box
 &000F · Risc PC architecture
 &0010 · Iyonix
 &0011 · A9
 &1040 · Spectrum
 &1041 · AmstradCPC
 &5050 · InetGateway
 &FFF8 · SJ Research GP server
 &FFF9 · SJ Research 80386 UNIX
 &FFFA · SCSI interface
 &FFFB · SJ Research IBM PC Econet interface
 &FFFC · Nascom 2
 &FFFD · Reseach Machines 480Z
 &FFFE · SJ Research File Server
 &FFFF · Z80 CP/M
16-23 – Value: - · Meaning: Software version number (BCD)
24-31 – Value: - · Meaning: Software release number (BCD)

Econet Protection Bits

0 – Disabled if set: Peek
1 – Disabled if set: Poke
2 – Disabled if set: Remote JSR (BL)
3 – Disabled if set: User procedure call
4 – Disabled if set: OS procedure call
5 – Disabled if set: Halt
6 – Disabled if set: Continue (0 on RISC OS)
7 – Disabled if set: Machine peek (0 on RISC OS)
8 – Disabled if set: Get registers
9 - 30 – Disabled if set: Reserved (must be zero)
31 – Disabled if set: Write value to CMOS (in Econet_SetProtection)

[RISC OS Open source page](#)

Econet Vector Reason Codes

EconetV

Reason Codes

16 · &10 – Name: NetFS_StartLoad · Meaning: Total size of data · NetStatus Calls: Hourglass_On
17 · &11 – Name: NetFS_PartLoad · Meaning: Amount of data transferred · NetStatus Calls: Hourglass_Percentage
18 · &12 – Name: NetFS_FinishLoad · NetStatus Calls: Hourglass_Off
32 · &20 – Name: NetFS_StartSave · Meaning: Total size of data · NetStatus Calls: Hourglass_On
33 · &21 – Name: NetFS_PartSave · Meaning: Amount of data transferred · NetStatus Calls: Hourglass_Percentage
34 · &22 – Name: NetFS_FinishSave · NetStatus Calls: Hourglass_Off
48 · &30 – Name: NetFS_StartCreate · Meaning: Total size of data · NetStatus Calls: Hourglass_On
49 · &31 – Name: NetFS_PartCreate · Meaning: Amount of data transferred · NetStatus Calls: Hourglass_Percentage
50 · &32 – Name: NetFS_FinishCreate · NetStatus Calls: Hourglass_Off
64 · &40 – Name: NetFS_StartGetBytes · Meaning: Total size of data · NetStatus Calls: Hourglass_On
65 · &41 – Name: NetFS_PartGetBytes · Meaning: Amount of data transferred · NetStatus Calls: Hourglass_Percentage
66 · &42 – Name: NetFS_FinishGetBytes · NetStatus Calls: Hourglass_Off

80 · &50 – Name: NetFS_StartPutBytes · Meaning: Total size of data · NetStatus Calls: Hourglass_On
81 · &51 – Name: NetFS_PartPutBytes · Meaning: Amount of data transferred · NetStatus Calls: Hourglass_Percentage
82 · &52 – Name: NetFS_FinishPutBytes · NetStatus Calls: Hourglass_Off
96 · &60 – Name: NetFS_StartWait · Meaning: Total size of data · NetStatus Calls: Hourglass_LEDs (both on)
98 · &62 – Name: NetFS_FinishWait · NetStatus Calls: Hourglass_LEDs (both off)
112 · &70 – Name: NetFS_StartBroadcastLoad · Meaning: Total size of data · NetStatus Calls: Hourglass_On & Hourglass_Colours (green/blue)
113 · &71 – Name: NetFS_PartBroadcastLoad · Meaning: Amount of data transferred · NetStatus Calls: Hourglass_Percentage
114 · &72 – Name: NetFS_FinishBroadcastLoad · NetStatus Calls: Hourglass_Off
128 · &80 – Name: NetFS_StartBroadcastSave · Meaning: Total size of data · NetStatus Calls: Hourglass_On & Hourglass_Colours (red/blue)
129 · &81 – Name: NetFS_PartBroadcastSave · Meaning: Amount of data transferred · NetStatus Calls: Hourglass_Percentage
130 · &82 – Name: NetFS_FinishBroadcastSave · NetStatus Calls: Hourglass_Off
192 · &C0 – Name: NetFS_StartTransmission · Meaning: Commence waiting for end of transmission · NetStatus Calls: Hourglass_LEDs (only top on)
194 · &C2 – Name: NetFS_FinishTransmission · Meaning: Econet_DoTransmit ends · NetStatus Calls: Hourglass_LEDs (both off)
208 · &D0 – Name: NetFS_StartReception · Meaning: Commence waiting for end of reception · NetStatus Calls: Hourglass_LEDs (only bottom on)
210 · &D2 – Name: NetFS_FinishReception · Meaning: Econet_WaitForReception ends · NetStatus Calls: Hourglass_LEDs (both off)
[RISC OS Open source page](#)

EconetV

Vectors

This vector is called when there is Econet activity.

Before calling

R0 Econet reason code

R1 Size of data, or data transferred, or not used

On return

R0 Preserved

R1 Preserved

Notes

The meaning (if any) of R1 depends on the reason code. This vector is usually claimed by the NetStatus module to alter the Hourglass to reflect the Econet status.

[RISC OS Open source page](#)

Error Handler

OS_ChangeEnvironment

This handler is called by the default claimant of ErrorV? when an error occurs. It should restore all handlers to their original values and then call the non-X form of OS_GenerateError to pass the error to the previous handler.

Before calling

R0 Workspace pointer

Other registers: Undefined

User mode, IRQs+FIQs enabled

In practice

The error buffer (R3 from OS_ChangeEnvironment) will contain the following:

+0 / PC at which the error occurred

+4 – PC at which the error occurred: Error number

+8 – PC at which the error occurred: Error message text

Notes

This handler is entered in USER mode with the workspace pointer in R0, as opposed to the conventional R12.

The block should be at least 260 bytes in size: 4 bytes for the erroring program counter plus 256 for a standard RISC OS error block. Note that the printed PRM is vague in specifying a “256 byte error buffer”, currently the kernel truncates any error message text to 252 bytes, totalling 4 + 4 + 252 = 260 in the worst case.

There is no guarantee that R13 will contain a valid stack pointer. Even if the foreground task was using R13 as a stack in the usual manner, there is no guarantee that the pointer is still valid – e.g. an RTSupport routine running in SYS mode, or a non-transient callback, may have been using R13 for some other purpose at the time the error was generated. Therefore if a stack is required, it's recommended that you validate R13, or use a different stack private to the error handler.

It's also possible that R1-R12 and R14 do not correspond to the values they had when the error was generated.

[RISC OS Open source page](#)

Escape Event

Events

This event is generated when an escape condition occurs due to escape being pressed (or received in the serial buffer with OS_Byte 181 enabled).

Before calling

R0 6

Notes

It is not generated by calling OS_Byte 125.

If this event is enabled then keyboard escapes generate this event instead of calling the escape handler.

OS_Byte 125 generated escapes always call the escape handler.

[RISC OS Open source page](#)

Escape Handler

OS_ChangeEnvironment

This handler allows programs to control how Escape events are handled.

Before calling

R11 Bit 6:

0 => Escape condition is being cleared/acknowledged

1 => Escape condition is being set

R12 "workspace" – don't use the value 1

R13 SVC/IRQ stack

SVC or IRQ mode, IRQs+FIQs undefined

On return

R12 1 to set callback

R0-R11 Preserved

Notes

This is called whenever an escape condition is detected and the escape event is disabled (except for calls to OS_Byte 125 which always call the handler).

OS_Byte 124 and OS_Byte 126 will also result in the handler being called with R11 bit 6 clear to clear/acknowledge an escape condition.

If the handler returns with R12=1, a non-transient callback will be queued.

[RISC OS Open source page](#)

Event 0

Events

This event is raised when an output buffer which has event generation enabled is emptied, or an attempt is made to remove another byte when empty.

Before calling

R0 0

R1 Buffer number

[RISC OS Open source page](#)

Event 1

Events

This event is raised when an attempt is made to insert a character into a full input buffer which has event generation enabled.

Before calling

R0 1

R1 Buffer number, bit 31 set for block operation

R2 Byte that could not be inserted (if single byte operation), or
Pointer to data (if block operation)

R3 Number of bytes not inserted (if block operation)

[RISC OS Open source page](#)

Event 10

Events

This event is generated when a mouse button is pressed or released.

Before calling

R0 10 (&0A)

R1 Mouse X co-ordinate

R2 Mouse Y co-ordinate

R3 Button state

R4 4 bytes of monotonic centi-second value

[RISC OS Open source page](#)

Event 11

Events

This event is polled when any key is pressed or released.

Before calling

R0 11 (&0B)

R1 Transition value

R2 Key number

R3 Keyboard driver ID

Notes

Transition value is 0 when released and 1 when pressed.

[RISC OS Open source page](#)

Event 12

Events

This event is generated when the sound beat counter is reset.

Before calling

R0 12 (&0C)

R1 2

R2 0

Notes

In future R2 may contain the invocation number of the task causing the event.

[RISC OS Open source page](#)

Event 14

Events

This event is generated when the Econet completes a reception.

Before calling

R0 14 (&0E)

R1 Receive handle

R2 Status (9)

R3 Port

[RISC OS Open source page](#)

Event 15

Events

This event is generated when the Econet completes a transmission.

Before calling

R0 15 (&0F)

R1 Transmit handle

R2 Status

R3 Port

[RISC OS Open source page](#)

Event 16

Events

These events are generated when the Econet receives an OS Procedure call sent by another station calling Econet_Dolmmediate.

Before calling

R0 16

R1 Argument buffer

R2 RPC number

R3 Sending station

R4 Sending station's network

On return

R0 - R4 May be corrupted

Notes

The format of data is identical to receiving an UserRPC call (Event 8); the value in R0, however, would be 8 rather than 16.

[RISC OS Open source page](#)

Event 17

Events

This event is raised by a MIDI card.

Before calling

R0 17 (&11)

R1 Event reason code

MIDI event reasons

Code / Meaning

1 – Meaning: Byte received into previously empty buffer

2 – Meaning: Background MIDI error occurred

3 – Meaning: Scheduler queue about to empty - you can schedule more data

[RISC OS Open source page](#)

Event 19

Events

This event is raised by the Internet module.

Before calling

R0 19 (&11)

R1 Event reason code

R2 Depends on reason code

R3 Depends on reason code

Internet event reasons

Code / Meaning

1 – Meaning: A socket has bytes waiting to be read (UDP, TCP) or has successfully sent some data, freeing up space in its transmit buffer (TCP), or a new client is attempting to connect to a listen socket (TCP) (R2 = socket, R3 = port)

2 – Meaning: An urgent event, such as OOB data (R2 = socket, R3 = port)

3 – Meaning: Socket connection broken (R2 = socket, R3 = port)

4 – Meaning: RevARP reply (R2 = IPv4 replying server, R3 = IPv4 requesting station)

5 – Meaning: RevARP request pre Internet configuration (R2 = mbuf chain, R3 = interface SWI base)

Notes

Events 1-3 are only dispatched for a particular socket if they have been enabled for that socket via the `Socket_ioctl` `FIOASYNC` setting. As with any event, care must also be taken to ensure the event is enabled globally via `OS_Byte 14`.

Events 1-3 only pass the port number in R3 for Internet 5.04 and later.

[RISC OS Open source page](#)

Event 21

Events

A number of different events are available through sub-reason codes.

Before calling

R0 21 (&15)

R1 Event sub-reason code

[RISC OS Open source page](#)

Event 22

Events

This event is generated when `DeviceFS_ReceivedCharacter` is called for an unbuffered stream where the previously received character has not been read.

Before calling

R0 22 (&16)

R1 Device driver's handle

R2 File handle

R3 0

Notes

The new character overwrites the previous character.

[RISC OS Open source page](#)

Event 28

Event Numbers

This event is generated when an interrupt is received from the battery management unit.

Before calling

R0 28 (&1C)

R1 Battery Management Unit status flags

[RISC OS Open source page](#)

Event 3

Events

This event is raised by a BBC I/O card when an analogue-to-digital conversion is completed.

Before calling

R0 3

R1 Channel that just converted

[RISC OS Open source page](#)

Event 4

Events

This event is raised when the electron beam reaches the bottom of the display area and is about to start displaying border colour. This corresponds to the time that OS_Byte 19 returns. The frequency of this event depends on the scan rate of the monitor.

Before calling

R0 4

In practice

If you wish to synchronise music, you should probably use TickerV vector rather than this event.

[RISC OS Open source page](#)

Event 5

Events

This event is raised when the interval timer crosses zero (see OS_Word 4).

Before calling

R0 5

[RISC OS Open source page](#)

Event 7

Events

This event is generated when a serial error is detected.

Before calling

R0 7

R1 (pseudo 6850 status register) >> 1

R2 Character received

Notes

It is not generated by calling OS_Byte 125.

The error is specified by bits in R1:

Bits / Meaning when set

3 – Meaning when set: Framing error

4 – Meaning when set: Over-run error

5 – Meaning when set: Parity error

[RISC OS Open source page](#)

Event 8

Events

These events are generated when the Econet receives a UserRPC call sent by another station calling Econet_Dolmmediate.

Before calling

R0 8

R1 Argument buffer

R2 RPC number

R3 Sending station

R4 Sending station's network

On return

R0-R4 May be corrupted

Notes

The format of data is identical to receiving an Econet OS remote procedure call (Event 16); the value in R0, however, would be 16(&10) rather than 8.

[RISC OS Open source page](#)

Event 9

Events

This event is generated when the user calls `OS_GenerateEvent` with `R0 = 9`, and is entered in `SVC` mode, not `IRQ`.

Before calling

R0 9

R1 User parameters

[RISC OS Open source page](#)

Event Numbers

Events are used by the Operating System to indicate that something has occurred. Events are usually generated by calling `OS_GenerateEvent` when servicing and interrupt. The following events are available.

- 0 – Description: Output buffer has become empty
- 1 – Description: Input buffer has become full
- 2 – Description: Character placed in input buffer
- 3 – Description: ADC conversion ended in BBC I/O expansion card
- 4 – Description: VSync Event
- 5 – Description: Interval timer has crossed zero
- 6 – Description: Escape condition has been detected
- 7 – Description: Serial RS423 error has been detected
- 8 – Description: Econet user remote procedure called
- 9 – Description: User has generated an event
- 10 – Description: Mouse transition
- 11 – Description: Key transition
- 12 – Description: Sound system reached start of a bar
- 13 – Description: PC Emulator event
- 14 – Description: Econet receive complete
- 15 – Description: Econet transmit complete
- 16 – Description: Econet OS remote procedure called
- 17 – Description: MIDI system event
- 18 – Description: Reserved for external developers
- 19 – Description: Internet event
- 20 – Description: Reserved for external developers
- 21 – Description: Expansion event
- 22 – Description: Device overrun event
- 23 – Description: Reserved for external developers

- 24 – Description: Driver received frame for Internet (DCI 2 only)
 - 25 – Description: Driver completed Internet transmission request (DCI 2 only)
 - 26 – Description: Copy of the Econet_Rx
 - 27 – Description: Copy of the Econet_Tx
 - 28 – Description: Portable Battery Management Unit event
- [RISC OS Open source page](#)

Events

RISC OS uses events to inform tasks that something specific has occurred. Each event is assigned an event number. These are typically generated by `OS_GenerateEvent` when RISC OS services an interrupt.

Enabling events

Events are disabled by default, because of the significant processor overhead involved. Events can be enabled or disabled individually if required.

For each event number, a counter keeps track of how many requests have been made to enable it. As long as the counter remains above zero, RISC OS will always generate the event when appropriate. An event will only be disabled after equal numbers of disable event and enable event calls have been made. This ensures that the system always generates the event if at least one task requires it. On a reset, the event counters are set to zero (although some modules may still need an event and hence increment the appropriate event counter).

In addition to the Operating System generating event numbers, it is also possible for developers to generate their own (code 9), or if required, request their own event code (see *Allocate pages*).

Using events

To use an event, the following steps must be taken:

- The event vector must first be claimed by calling `OS_Claim`
- `OS_Byte 14` is called to enable the event

The purpose of claiming the vector is to assign a new routine that must be called when the event occurs. Because multiple tasks may claim the same event, the routine is added to the list of routines that must be called. If you wish to add multiple 'identical' routines, use `OS_AddToVector`. *Software Vectors* provides more information on the use of vectors.

When an event is no longer required, the follow steps must be taken:

- Disable the appropriate event(s) by calling `OS_Byte 13`
- Release the event vector by calling `OS_Release`

Handling routines

Any routine dealing with events must do so very quickly, so the OS can continue with the previous task as quickly as possible. Ideally events are handled so quickly that users never realise their task has been temporarily halted.

If a routine cannot deal with an event quickly, it should re-enable interrupts (as they are always disabled when entering a routine). They must then be disabled before passing on or intercepting the call. The routine should also ensure that the processing of one event is completed before it starts processing another. The routine is always entered in a non-user processor mode.

Error handling

Routines that handle events must only call the error-returning SWI calls (that is, SWI calls that have their X-bit set).

If an error is returned to the routine, appropriate action must be taken within the routine. It may be useful to store an error indicator, so that the next call to an appropriate SWI (one in the module that provides the routine) will generate an error.

[RISC OS Open source page](#)

Events Introduction

Events

RISC OS like all other Operating System must be able inform tasks that something specific has occurred so appropriate action can be taken, if required. Each specific event is assigned a number; an event number. These are typically generated by OS_GenerateEvent when RISC OS services an interrupt.

In practice

Events are disabled default, which prevents RISC OS from generating events. This is because there is a significant processor overhead involved in generating events. Events can be enabled or disabled individually if required.

For each event number, a counter keeps track of how many requests have been made to enable it. As long as the counter remains above zero, RISC OS will always generate the event when appropriate. An event will only be disabled after equal numbers of disable event and enable event calls have been made (i.e. counter falls to zero). This method ensures that the system always generates the event if at least one task requires it. On reset, the event counters are set to zero, although some modules may need an event and hence increment the appropriate event counter.

In addition to the Operating System generating event numbers, it is also possible for developers to generate their own events too (event code 9), or if required, request their own event code (see Allocate pages).

List of events

A complete list of events available under RISC OS is available [here](#) and provides details on their use. As stated above, Event code 9 is assigned for developer use.

Using events

To use an event the following steps must followed:

- The event vector must first be claimed by calling OS_Claim
- Enable the appropriate event(s) by calling OS_Byte 14

The purpose of claiming the vector is to assign a new routine that must be called when the event occurs. Because multiple tasks may claim the same event, the routine is added to the list of routines that must be called. If you wish to add multiple 'identical' routines use OS_AddToVector. You may wish to read up on Software Vectors for more information on the use of vectors.

When an event is no longer required the follow steps must be taken:

- Disable the appropriate event(s) by calling OS_Byte 13
- Release the event vector by calling OS_Release

Event handling routines

The routine dealing with the events must do so very quickly, so it can continue with the previous task as quickly as possible. Often, events are handled so quickly that users never realise their task was temporarily halted.

If a routine cannot deal with an event quickly, it should re-enable interrupts (as they are always disabled when entering a routine), but it must then be disabled before passing on or intercepting the call and ensure that the processing of one event is complete before they start processing another.

Please note: The routine is always entered in a non-user processor mode.

Error handling

Routines that handle events must only call the error-returning SWI calls. That is SWI calls that have their X bit set. i.e. either add the value &20000 to the SWI number, or add the letter 'X' in front of the SWI name.

If an error is returned to the routine, appropriate action must be taken within the routine. It may be useful to store an error indicator within the routine, so that the next call to an appropriate SWI (one in the module that provides the routine) will generate an error.

[RISC OS Open source page](#)

EventV

Vectors

This vector is called to indirect all calls to OS_GenerateEvent.

Before calling

R0 Event Number

Other registers dependent on call

Notes

Therefore it is vital that you check whether the event number is the same as those which you wish to receive (an example of not doing this are the Coconizer tune modules which speed up on key presses because Key Transition is enabled).

The default action is to call the event handler.

[RISC OS Open source page](#)

Exception Dump Region

OS_ChangeEnvironment

The exception dump region is a pointer to a block of memory containing information about the area around the region where an exception occurred last. It has a maximum size, which may be filled, or may be shorter if memory could not be read at the time of the exception.

Region /

+0 – : PC at the time of the exception. If this does not match the exception dump, the data in the block is not valid.

+4 – : Base address of the memory region which follows.

+8 – : Size in bytes of the memory region which follows.

+12... – : Data from the base address

Notes

This interface is implemented on RISC OS Pyromaniac 7.60 and above.

[RISC OS Open source page](#)

Exception Registers Block

OS_ChangeEnvironment

The exception registers block is used by the default implementation of environment handlers 1 to 5 as the location to save the ARM register dump, prior to the fault being announced via OS_GenerateError. Unlike the other environment handlers it is not a pointer to code; it is a pointer to a block of memory.

In practice

On 26bit systems the block must be 16 words long, and will contain R0-R15.

On 32bit systems the block must be 17 words long, and will contain R0-R15 and the PSR.

The register dump will only contain the registers for the mode in which the exception occurred.

Notes

From time to time other system components might write to the register block, e.g. when a floating point instruction raises a division by zero error.

[RISC OS Open source page](#)

Exit Handler

OS_Exit

This is called in USER mode whenever OS_Exit or OS_ExitAndDie is called. It should restore all handlers to their previous values and then call OS_Exit to pass to the previous handler. However, care must be taken to ensure R0-R2 are valid when calling OS_Exit, as the OS will typically have corrupted them prior to calling the exit handler. If it's desirable to pass through the original return code, that can be read via Sys\$ReturnCode, as it will have been stored prior to the exit handler being invoked.

Before calling

R12 Workspace pointer

Other registers: Undefined

User mode, IRQs+FIQs enabled

Notes

R13 may be undefined, either avoid the use of R13 in the handler or provide your own stack space.

The exit handler will not be invoked if an error is generated due to the return code being out of range (instead, you can expect the Error Handler to be invoked)

[RISC OS Open source page](#)

File Descriptor Blocks

OS_FSControl

File Descriptor Blocks

Offset / Contents for OS_FSControl 26

+0 – Contents for OS_FSControl 26: Address of workspace

+4 – Contents for OS_FSControl 26: Length of workspace

Offset / Contents for OS_FSControl 28

+0 – Contents for OS_FSControl 28: Low word of count

+4 – Contents for OS_FSControl 28: High word of count

[RISC OS Open source page](#)

File Formats

RISC OS utilises many different file formats, many that are not widely used on other platforms. Below is a list of some of the most common file formats that a RISC OS developer will come across.

In practice

- DrawFile
- MusicFile (Used primarily by Maestro)
- SpriteFile
- Squash file
- Module
- Utility

A comprehensive list of files, detailing their filetype, name, use, and their default Application assigned to use the file is here.

[RISC OS Open source page](#)

File formats: DrawFile

File Formats

A DrawFile consists of a File Header block, followed by zero or more Object blocks.

Header Block

0 – Length: 4 · Contents: "Draw"

4 – Length: 4 · Contents: Major version, 201 at present

8 – Length: 4 · Contents: Minor version

12 – Length: 12 · Contents: Name of producing program, padded with spaces

24 – Length: 4 · Contents: X low bounding box

28 – Length: 4 · Contents: Y low bounding box

32 – Length: 4 · Contents: X high bounding box

36 – Length: 4 · Contents: Y high bounding box

Object Block

0 – Length: 4 · Contents: Object Type Identifier:

0 Font table
 1 Text object
 2 Path object
 3 (unimplemented Rectangle object)
 4 (unimplemented Rectangle object)
 5 Sprite object
 6 Group object
 7 Tagged object
 8 Not used
 9 Text area object
 10 Text column object
 11 Options object
 12 Transformed text object
 13 Transformed sprite object
 14 Not used
 15 Not used
 16 JPEG object
 17 Grid options object
 4 – Length: 4 · Contents: Word aligned size, including header
 8 – Length: 16 · Contents: Bounding box (not all types have this)
 n – Length: n · Contents: Object specific data, if any

Font Table Object (Object Type 0)

This object has no bounding box. It should only appear once, and before any text objects.

It consists of:

- a series of data items, where each data item consists of a font number (byte), followed immediately by a null terminated fontname
- Then the table itself is padded to the word boundary with nulls.

Text Object (Object Type 1)

24 – Length: 4 · Contents: Text colour
 28 – Length: 4 · Contents: Background hint
 32 – Length: 4 · Contents: Text style
 36 – Length: 4 · Contents: X-size (in 1/640's of a point)
 40 – Length: 4 · Contents: Y-size (ie draw units)
 44 – Length: 4 · Contents: X coord of start of baseline
 48 – Length: 4 · Contents: Y coord of it
 52 – Length: 4 · Contents: Zero terminated string padding

Path Object (Object Type 2)

Offset / Length / Contents

24 – Length: 4 · Contents: Fill colour (-1 for none)
 28 – Length: 4 · Contents: Outline col (-1 for none)
 32 – Length: 4 · Contents: Outline width
 36 – Length: 4 · Contents: Path style
 40 – Length: n · Contents: Optional dash pattern
 n · Path components (consists of a Tag Identifier)

Tag Identifier

Offset / Length / Value / Meaning

+0 · 4 – Meaning: Tag ID:

0 · End of Path
 2 · Move, one point
 5 · Close sub-path
 6 · Bezier, three points
 8 · Line, one point
 +4 · n – Value: 0 – 3 · Meaning: sets of (x,y) dependant on tag

Sprite Object (Object Type 5)

This is followed by a sprite, which is scaled to fill bounding box.

Group Object (Object Type 6)

Offset / Length / Contents

0 – Length: 12 · Contents: Group object name (padded with spaces)
 and then a sequence of objects. These will all lie within the bounding box of this.

Tagged Object (Object Type 7)

Offset / Length / Contents

0 – Length: 4 · Contents: Tagged identifier
 and then followed by an object and optional word-aligned data.

Text Area Object (Object Type 9)

First, 1 or more text columns and then:

Offset / Length / Contents

0 – Length: 4 · Contents: Zero to mark end of columns
 4 – Length: 4 · Contents: Reserved
 8 – Length: 4 · Contents: Reserved
 12 – Length: 4 · Contents: Initial foreground colour
 16 – Length: 4 · Contents: Initial background hint
 20 – Length: n · Contents: Zero terminated body of text
 padding

Text Column Object (Object Type 10)

No data other than the standard 6 word header – the bounding box in the header gives the location of the column. Text column objects can only occur in Text area objects.

Options Object (Object Type 11)

This object has space for a bounding box, but the 4 words should be set to zero.

0 – Length: 4 · Contents: Paper size
 4 – Length: 4 · Contents: Paper limits
 8 – Length: 8 · Contents: Grid spacing (floating point)
 16 – Length: 4 · Contents: Grid division
 20 – Length: 4 · Contents: Grid type
 24 – Length: 4 · Contents: Grid auto-adjust
 28 – Length: 4 · Contents: Grid visible
 32 – Length: 4 · Contents: Grid units
 36 – Length: 4 · Contents: Zoom multiplier (1 – 8)
 40 – Length: 4 · Contents: Zoom divider (1 – 8)
 44 – Length: 4 · Contents: Zoom locking
 48 – Length: 4 · Contents: Toolbox Presence
 52 – Length: 4 · Contents: Initial entry mode
 56 – Length: 4 · Contents: Undo buffer (size in bytes)

Paper Size

&100 – Meaning: A0

&200 – Meaning: A1

&300 – Meaning: A2

&400 – Meaning: A3

&500 – Meaning: A4

&600 – Meaning: A5

&700 – Meaning: A6

&800 – Meaning: A7

Paper Limits

Bit / Meaning when set

0 – Meaning when set: Show limits

1 - 3 – Meaning when set: Reserved

4 – Meaning when set: Landscape (otherwise Portrait)

5 - 7 – Meaning when set: Reserved

8 – Meaning when set: Default to printer limits

9 - 31 – Meaning when set: Reserved

Grid Type

Value / Meaning

0 – Meaning: Rectangular grid

<> 0 – Meaning: Isometric grid

Grid Auto-adjust

Value / Meaning

0 – Meaning: Auto-adjust is disabled

<> 0 – Meaning: Auto-adjust is enabled

Grid Visible

Value / Meaning

0 – Meaning: Grid is not visible

<> 0 – Meaning: Grid is visible

Grid Units

Value / Meaning

0 – Meaning: Grid units is inches

<> 0 – Meaning: Grid units is cm

Zoom Locking

Value / Meaning

0 – Meaning: Zoom locking is set to none

<> 0 – Meaning: Zoom locking is set to powers of 2

Toolbox Presence

Value / Meaning

0 – Meaning: Toolbox is absent

<> 0 – Meaning: Toolbox is present

Initial Entry Mode

0 – Meaning when set: Line

- 1 – Meaning when set: Closed line
- 2 – Meaning when set: Curve
- 3 – Meaning when set: Closed curve
- 4 – Meaning when set: Rectangle
- 5 – Meaning when set: Ellipse
- 6 – Meaning when set: Text line
- 7 – Meaning when set: Select

Transformed Text Object (Object Type 12)

- 24 – Length: 24 · Contents: Transformation matrix
- 48 – Length: 4 · Contents: Font flags
- 52 – Length: 4 · Contents: Text colour
- 56 – Length: 4 · Contents: Background hint
- 60 – Length: 4 · Contents: Text style
- 64 – Length: 4 · Contents: X-size (in 1/640's of a point)
- 68 – Length: 4 · Contents: Y-size (ie draw units)
- 72 – Length: 4 · Contents: X coord of start of baseline
- 76 – Length: 4 · Contents: Y coord of it
- 80 – Length: 4 · Contents: Zero terminated string padding
padding

Font Flags

Bit / Meaning when set

- 0 – Meaning when set: Kern text
- 1 – Meaning when set: Right to left

Transformed Sprite Object (Object Type 13)

Offset / Length / Contents

- 24 – Length: 24 · Contents: Transformation matrix
 - 48 – Length: - · Contents: Format of sprite
- This is scaled to fill the bounding box.

JPEG Object (Object Type 16)

- 24 – Length: 4 · Contents: Height (draw units)
- 28 – Length: 4 · Contents: Width
- 32 – Length: 4 · Contents: x DPI (90 for a normal square pixel mode)
- 36 – Length: 4 · Contents: Y DPI
- 40 – Length: 24 · Contents: Transformation matrix
- 64 – Length: 4 · Contents: Length of JPEG data
- 68 – Length: n · Contents: Embedded JPEG (strictly JFIF) file
- .. – Length: 0 – 3 · Contents: Padding to next word

Currently it seems that the 6 word transformation matrix is used only to scale the JPEG image and translate it from (0,0),(height,width) to the required position on the page. It is not known whether this object can contain JPEG data in any format other than a JFIF file.

Text Style

First (lowest) byte is fontnumber, and refers to font in font table. The rest of word is reserved, and should be zero.

Path Style

- 0 - 1 – Meaning: Join style:
- 0 · Mitred

1 · Round
 2 · Bevelled
2 - 3 – Meaning: Start cap:
 0 · Butt
 1 · Round
 2 · Square
 3 · Triangle
4 - 5 – Meaning: End cap
6 – Meaning: Winding rule:
 0 · Non-zero
 1 · Even-odd
7 – Meaning: Dash pattern
 0 · None
 1 · Present
16 - 23 – Meaning: Triangle cap width
24 - 31 – Meaning: Triangle cap length

Escape Sequence in Draw Text Area

\! – Meaning: Version must be 1
\A – Meaning: Alignment: L, R, C, D
\B / – Meaning: Set background hint
\C / – Meaning: Set foreground colour
\D / – Meaning: Number of columns
\F [/ – Meaning: Declare a font
**** – Meaning: Select a font
\L / – Meaning: Leading
\M / – Meaning: Sets margins
\P / – Meaning: Paragraph leading
\U / – Meaning: Turn on underline
\V[-] – Meaning: Vertical move
\- – Meaning: Soft hyphen
**** – Meaning: Force line break
**** – Meaning: “\”
\; – Meaning: Comment
[RISC OS Open source page](#)

File formats: MusicFile

File Formats

The MusicFile filetype is primarily used by the Maestro application. MusicFiles allow music to be played by the 8-bit sound system, and are limited to 8 channels.

Header block

Size / Description

8 – Description: ‘Maestro’, followed by linefeed character (&0A)

1 – Description: 2 (Type 2 MusicFile)

The Header block is followed by zero or more of the following blocks. The original spec claimed they could appear in any order, but in fact Maestro will expect blocks 1-6 before any others. Note that block types 7, 8 and 9 are not currently implemented by Maestro.

1. Music data

Size / Description

1 – Description: 1

5 – Description: Number of bytes in the Gates array¹

5×8 – Description: Number of bytes in the queue of notes and rests for each of the 8 channels¹

n – Description: Gate data (see below)

q1...q8 – Description: Data for all notes and rests in channels 1-8

1 Stored as a BASIC integer: &40 followed by 4 bytes of data, most significant first.

2. Stave data

Size / Description

1 – Description: 2

1 – Description: Number of staves (0-3)

1 – Description: Number of percussion staves (0-1)

The staves value is incremented by one to create a range of 1-4 staves, whereas the percussion value controls whether a single percussion stave is added. Earlier versions of Maestro permitted more than one percussion stave.

In Maestro, the distribution of channels to staves is as follows:

1 · 0 – Stave 1: 1-8

1 · 1 – Stave 1: 1-7 · Percussion: 8

2 · 0 – Stave 1: 1-4 · Stave 2: 5-8

2 · 1 – Stave 1: 1-4 · Stave 2: 5-7 · Percussion: 8

3 · 0 – Stave 1: 1 · Stave 2: 2-5 · Stave 3: 6-8

3 · 1 – Stave 1: 1 · Stave 2: 2-5 · Stave 3: 6-7 · Percussion: 8

4 · 0 – Stave 1: 1-2 · Stave 2: 3-4 · Stave 3: 5-6 · Stave 4: 7-8

4 · 1 – Stave 1: 1-2 · Stave 2: 3-4 · Stave 3: 5-6 · Stave 4: 7 · Percussion: 8

3. Instrument data

Instrument names are not recorded, only channel numbers.

Size / Description

1 – Description: 3

This is followed by 8 blocks:

Size / Description

1 – Description: Channel number (1-8, consecutive)

1 – Description: Voice number (0=none attached)

4. Volume data

Size / Description

1 – Description: 4

1×8 – Description: Volume of channel (0-7)

5. Stereo position data

Size / Description

1 – Description: 5

1×8 – Description: Stereo position of channel (0-6, left to right)

6. Tempo data

Size / Description

1 – Description: 6

1×8 – Description: Tempo value (0-14)

Tempo values correspond to the following beats per minute: 40, 50, 60, 65, 70, 80, 90, 100, 115, 130, 145, 160, 175, 190 and 210.

To convert to values suitable for use with Sound_QTempo, use the formula: Sound_QTempo value = beats per minute * 128 * 4096 / 6000

7. Title string

Size / Description

1 – Description: 7

n – Description: Null-terminated string of n characters length

8. Instrument names

Size / Description

1 – Description: 8

n1...n8 – Description: 8 null-terminated strings for each Voice number used in ascending order, corresponding to data in block type 3, above

9. MIDI channels

Size / Description

1 – Description: 9

1x8 – Description: MIDI channel for Instrument (1-16, or 0 for not MIDI)

Gates

A Gate is a point in the music where something is interpreted: a note, time signature, key signature, bar line or clef, for example. A note or a rest takes up 1 byte, whereas all other attributes, such as key signatures, etc, take up 2 bytes, where the first byte is 0 and the second contains data on the attribute.

Note/Rest

Bits / Description

0-7 – Description: Gate mask: bit n set => gate 1 note/rest from queue n.

Time signature

Byte 2 bits / Description

0 – Description: %1

1-4 – Description: Beats per bar (0-15)

5-7 – Description: Beat type (0-7, breve to hemidemisemiquaver)

Key signature

Byte 2 bits / Description

0-1 – Description: %10

2 – Description: Accidental type (0=sharp, 1=flat)

3-5 – Description: Number of accidentals (0-7)

6-7 – Description: Reserved (must be 0)

Clef

Byte 2 bits / Description

0-2 – Description: %100

3-4 – Description: 0=treble, 1=alto, 2=tenor, 3=bass

5 – Description: Reserved (must be 0)

6-7 – Description: Stave (0-3)

Maestro currently only implements treble and bass clefs.

Slur

Byte 2 bits / Description

0-3 – Description: %1000

4 – Description: 1=on, 0=off

5 – Description: Reserved (must be 0)

6-7 – Description: Stave (0-3)

Octave shift

Byte 2 bits / Description

0-4 – Description: %10000

5 – Description: 0=up, 1=down

6-7 – Description: Stave (0-3)

Barline

Byte 2 bits / Description

0-5 – Description: %100000

6 – Description: 0=single barline, 1=double barline

7 – Description: Reserved (must be 0)

Double barlines implemented in Maestro 2.11. Earlier versions will ignore double barlines in MusicFiles, and display single barlines instead.

Reserved

Bits 6 and 7 of the attribute byte are reserved for future expansion. Maestro currently does not implement types 4 and 5 (slurs and octave shifts).

Notes/Rests

Notes/rests are stored in 2-byte blocks.

Notes

Bits / Description

0 – Description: Stem orientation (0=up, 1=down)

1 – Description: Tie orientation (0=over note, 1=under note)²

2 – Description: Tie (0=no tie, 1=tie to next note)

3-7 – Description: Stave line position (1-31; 16=centre)

8-10 – Description: Accidental (0-7)³

11-12 – Description: Number of dots (0-3)

13-15 – Description: Type (0-7, breve to hemidemisemiquaver)

² Tie orientation was implemented in version 2.14 of Maestro. In earlier versions of the file specification, this bit was described as being used to mark the note as having a beam connected to the next note. However, this was never implemented in Maestro, and so has been repurposed.

³ Accidental values: 0=None 1=Natural 2=Sharp 3=Flat 4=Double-sharp 5=Double-flat 6=Natural sharp 7=Natural flat

Rests

Bits / Description

0-7 – Description: Reserved (must be 0)

8-10 – Description: Stave line position (0-7)⁴

11-12 – Description: Number of dots (0-3)

13-15 – Description: Type (0-7, breve to hemidemisemiquaver)

⁴ Stave line positions for rests were introduced in Maestro 2.13. Previously, these bits were all unset. In order to maintain compatibility with older versions, a value of 0 for the stave line position is interpreted as the centre stave line. Values 1, 2 and 3 are below the centre line in ascending height order (ie, 1 is furthest from the centre line), and 5, 6 and 7 are above, also in ascending order. The 'resolution' is doubled, so that a rest line position of 5 is equivalent to a note line position of 18, 6 is equivalent to 20, 7 to 22, etc.

If a rest coincides with a note, its position is determined by the following note on the same channel.

[RISC OS Open source page](#)

File formats: Relocatable Module

Filetypes

Modules (filetype &FFA) lie at the heart of RISC OS. A module extends the operating system, or replaces part of it.

Relocatable Modules

For example, while we are used to periodically updating our ROM image on current hardware, on older machines the ROM was a physical ROM so it was common to load a later version of the SharedCLibrary which would replace the one supplied with RISC OS. Likewise, the "nested Wimp" would also be loaded from disc and would entirely replace the older and less capable Wimp supplied in ROM.

Modules can also be used to extend system functionality. For instance, Fat32FS is an extended and more capable version of DOSFS. SparkFS provides a filing system interface to archive files, where they can be opened as if regular directories. A USB-MIDI module exists to permit simpler communication with MIDI keyboards using a USB interface. The options are nearly endless.

Legacy modules

Modules written for earlier versions of RISC OS will not work on RISC OS 5. This is due to incompatibilities in the ARM programmer's model, namely RISC OS 5 uses a 32 bit Program Counter with a separate Status Register, while older versions use a combined 26 bit PC plus PSR. The two are fundamentally incompatible. It is, however, possible to create a 26/32 agnostic module if you set the flag word to indicate that your module is 32 bit safe, and then you include the expected function exit (preserves flags or does not) according to whether or not the system is using a combined or separate PSR. Information on writing modules for the older 26 bit versions of RISC OS may be found [here](#).

Module header

Modules consist of code and data in a single file that is prefixed by a header. The header provides offsets to various entry points and data. A 26 bit module must have at least the first seven fields. A 32 bit module must have all fields present, as the last one is the offset to a flags word containing the 32 bit flag (Note: Other flags may be used in future at present only the bit for 32 bit status is used). Fields that are not applicable should be zero. Most fields are offsets, some of these must be aligned, and all offsets must point within the length of the module to be valid.

Header entries

- +0 – Hex: &00 · Purpose: Offset to Start code (or Branch instruction)
- +4 – Hex: &04 · Purpose: Offset to Initialisation code (or compression offset)
- +8 – Hex: &08 · Purpose: Offset to Finalisation code (and b31 flag)
- +12 – Hex: &0C · Purpose: Offset to Service Call handler
- +16 – Hex: &10 · Purpose: Unaligned offset to Title string
- +20 – Hex: &14 · Purpose: Unaligned offset to Help string
- +24 – Hex: &18 · Purpose: Unaligned offset to Help and Command Keyword table
- +28 – Hex: &1C · Purpose: SWI Chunk Base Number
- +32 – Hex: &20 · Purpose: Offset to SWI Handler code
- +36 – Hex: &24 · Purpose: Unaligned offset to SWI Name Table
- +40 – Hex: &28 · Purpose: Offset to SWI Decoder code
- +44 – Hex: &2C · Purpose: Offset to Messages file name
- +48 – Hex: &30 · Purpose: Offset to Module feature flags

Legacy versions of RISC OS only required the first seven fields (those prior to the SWI chunk). The rest were optional, depending on necessity. RISC OS 5 requires all fields to be present.

Start code

Zero, aligned offset to code, or ARM machine code instruction if any of bits 31-25 or 1-0 are set

Calling OS_Module with either Run or Enter will enter a module through this offset, as will the *RMRun command.

On entry:

=> / R0 / Pointer to command string (excluding module name)

-- R0: R12 · Pointer to command string (excluding module name): Private word for the currently preferred instantiation

-- R0: R14 · Pointer to command string (excluding module name): Not a return address - the module must exit with OS_Exit

The processor is in User mode with interrupts enabled. This entry point is not re-entrant.

This call does not return.

On exit:

As this was started as the current application, it must exit by calling OS_Exit, OS_ExitAndDie, or implicitly by starting another application.

If treated as an ARM instruction, it can only sensibly be a B branch or possibly BL (in order to point R14 near the start of the module). It is not sensible to continue execution into the Initialisation offset, so some kind of branch must be achieved, and the other potential construction of LDR PC,[PC,#offset] is definitely not recommended. A simple offset is best.

BASIC is an example of a module with a Start offset, as are some common Wimp applications built as modules.

Initialisation code

Zero, or aligned offset to code (if b31 clear); aligned length of compressed module (if b31 set), and flags

When a module is initially loaded, reloaded, or after a tidy (in other words, following the Run, Enter, or Relnit calls) the Initialisation entry is called. This is expected to set up the module so that all of the other entry points are able to operate. This entry point is the first one to be called. Typically you would use this entry point to claim some workspace, set up data tables, hook into vectors and events, register as a filing system, etc.

Note that at this point, the module is not yet on the list of active modules so you cannot call your own SWIs - you must jump directly to your own code if necessary.

You must not generate errors so use the X form of SWIs. If there is a problem that prevents the module from initialising, R0 should point to a standard error block and the module should return with the V flag set. The module will then be removed.

Registers R7-R11 and R13 must be preserved.

On entry:

=> · R0 · Pointer to the environment string (any initialisation parameters given when the module was loaded)

R11 · I/O base or instantiation number (see below)

R12 · Private word for this instantiation (see below)

R13 · SVC stack

R14 · return address

The processor is in SVC mode with interrupts enabled. This entry point is not re-entrant.

On exit:

<= / R0-R6 / can be corrupted

-- R0-R6: R12,14 · can be corrupted: can be corrupted

-- R0-R6: V set · can be corrupted: if R0 is an error block

Should preserve processor mode and interrupt state. Flags can be corrupted - RISC OS only examines the V flag to see if an error occurred.

If this entry point is zero, it means that the module does not require initialisation.

R11, the I/O base or instantiation value, has the following meanings: If it is zero, the module was loaded from ROM, from a filing system, or was already present in memory. If it is > 0 then the module was loaded from an expansion card and R11 is the synchronous base address of the card (RiscPC and Lyonix class hardware). Any other value means that the

module is being reincarnated and there are R11 other instantiations of the module. Modules that cannot usefully be instantiated must check R11 and error if inappropriate.

If the private word (R12) contains a non-zero value, then the module is being reinitialised and the contents are as your module left them. Typically, modules claim workspace and write the address to R12 with STR Rxx,[R12], so that when RISC OS later calls the module via the other entry points and provides the private word in R12, you can extract your workspace address from this by using LDR R12, [R12]. However, the Kernel places no interpretation on the contents of the private word except after finalisation (see below).

The value may contain flags indicating the meaning of the initialisation word. Only 2 flags are currently defined. Any flags which are not understood by RISC OS will cause the module to be rejected as invalid, due to the offset being outside the module.

Bit 31 · Module is compressed; initialisation word contains the length of the module

Bit 30 · Module architecture type is present in the module feature flags, and contains a non-0 (not ARM) value

When the module architecture flag (bit 30) is set, versions of RISC OS which do not understand the flag will not initialise the module. Versions of RISC OS which do understand the flag will interpret the module architecture type and reject modules for architectures they do not understand. Currently only RISC OS Pyromaniac 7.22 and later support this flag.

Finalisation code

Zero, or aligned offset to code; b31 set if contents of private word must not be freed

This is the opposite of the initialisation code, it is called prior to a module being removed from memory. The module should remove itself from vectors, events, etc and release claimed workspace. It is not necessary to free the workspace pointed to by the private word – the Kernel will automatically attempt to free that after finalisation if it is non-zero, unless b31 of the Finalisation offset is set.

The finalisation entry is called by the ReInit, Delete, and Clear calls as well as the associated commands *RMReInit, *RMKill, and *RMClear; and also when a newer version of a module replaces an older version, the older one would be finalised prior to the new one being loaded.

If the module does not want to quit, it should set R0 to point to an error block, and return with the V flag set. The finalisation will be aborted and the module will still be active. This should be a last resort, as it can inconvenience the user.

Registers R7-R11 and R13 must be preserved.

On entry:

- => / R10 / Fatality indication – is '1' as finalisation is always fatal.
- – R10: R11 · Fatality indication – is '1' as finalisation is always fatal.: Instantiation number
- – R10: R12 · Fatality indication – is '1' as finalisation is always fatal.: Private word for this instantiation
- – R10: R13 · Fatality indication – is '1' as finalisation is always fatal.: SVC stack
- – R10: R14 · Fatality indication – is '1' as finalisation is always fatal.: return address

The processor is in SVC mode with interrupts enabled. This entry point is not re-entrant.

On exit:

- <= / R0-R6 / can be corrupted
- – R0-R6: R12,R14 · can be corrupted: can be corrupted
- – R0-R6: V set · can be corrupted: if unable to quit, R0 is error block

Should preserve processor mode and interrupt state. Flags can be corrupted – RISC OS only examines the V flag to see if an error occurred.

If this entry point is zero, it means that the module does not require any special finalisation and the Kernel can terminate the module.

The fatality indication in R10 had meaning a long time ago when it was possible to use *RMTidy to tidy up gaps in the module area. It would do this by temporarily 'soft' finalising all modules, relocating everything, then 'soft' reinitialising the modules afterwards. However *RMTidy has not been feasible for a very long time, so it should be assumed that all finalisation attempts are terminal.

R11 provides the dynamic instantiation number, which may not be the same as the value given in R11 on initialisation.

If the Finalisation offset's b31 is clear and the private word (R12) contains a non-zero value after finalisation, RISC OS will consider the value to be the module's claimed workspace and will attempt to free it. Thus, it is good form to write zero to the module's private word (the original R12 on entry) if it is likely to be confused for allocated memory.

Title string

Unaligned pointer; must not be zero

The title string is an offset to a null terminated module title. The title should be short, descriptive and unique, and should contain plain alphanumeric characters (A-Za-z0-9!_). It must not contain spaces or control characters.

It is recommended that it is formatted in a manner similar to the built-in modules; for example: "SpriteUtils", "Desktop", "Filer", "FileSwitch", and "TaskManager". The matching of module names is not case sensitive.

The module name is used with OS_Module calls that take module names (Enter, ReInit, or Delete) as well as the associated commands (*RMReInit and *RMKill), as well as being listed by *Modules).

It is also used as a de facto SWI prefix if the module has a SWI Base number, but no Name Table or Decoder. In this case it must not start with an X.

Help string

Unaligned pointer; must not be zero

The help string is an offset to a formatted null terminated string that is printed out by *Help prior to any information from the module such as the list of commands available. The help string also provides the module version number that is used by the *RMEnsure command.

The string must not contain any control characters other than Tab (chr 9 aligns to the next eighth column) or character 31 (which behaves like a hard space). The help string may contain spaces, but must not contain digits after the first 16 characters.

In order for *RMEnsure to work correctly you must use the following format (one or two tabs may be required to pad to column 16 in *Help Modules depending on Title length):

module_name<1 or 2 Tab chrs>vv.vv (DD Mmm YYYY)

The module name, which is generally the same as the title string (but can contain spaces), is followed by one or two tab characters to pad to column 16. This is then followed by the version number which is a 4.4 decimal number, eg 1.23 or 1234.5678. Modules typically only use two decimal digits, but up to four are supported. It is possible to follow the version number with letters such as "1.23a", but these are ignored by *RMEnsure. There follows a space and the version date in parentheses: two digit day of the month (with leading zero if necessary), a three character month name (in English), and a four digit year, for example "(12 Apr 2015)".

Sometimes the help string is followed by a short copyright indication, extended version number, or other content, eg " © Author yyyy" or " [beta test version]"

Help and Command Keyword table

Zero, or unaligned pointer

The command table contains *Commands, help available through *Help, and configuration options accessed by *Configure and *Status. It comprises one or more structures of the following form, terminated by a single zero byte (the first structure need not be aligned, but the rest implicitly are):

aligned offset – null terminated string, padded to word boundary: offset from module start to Code, or 0

byte – null terminated string, padded to word boundary: the minimum number of parameters, or 0

byte – null terminated string, padded to word boundary: one bit per parameter if it is to be GSTransed

byte – null terminated string, padded to word boundary: the maximum number of parameters, or 255

byte – null terminated string, padded to word boundary: b7 Filing System Command (low priority)

– – null terminated string, padded to word boundary: b6 Configuration Option

– – null terminated string, padded to word boundary: b5 Help offset is code pointer

– – null terminated string, padded to word boundary: b4 Help and Syntax strings are tokens for the Messages file (see below)

unaligned offset – null terminated string, padded to word boundary: offset from module start to Syntax string

offset – null terminated string, padded to word boundary: unaligned offset from module start to Help string,

– – null terminated string, padded to word boundary: or aligned offset to Help code, if b5 is set

Entries with a zero Code offset provide help only (but that help can be dynamically generated, see later). Otherwise, the Code is entered as follows:

On entry:

=> / R0 / ptr to command parameters or terminator, or if flags b6 was set:

– – ptr to command parameters or terminator, or if flags b6 was set:: 0 = *Configure YourOption with no parameters

– – ptr to command parameters or terminator, or if flags b6 was set:: 1 = *Status YourOption – display current configuration

– – R0: R1 · ptr to command parameters or terminator, or if flags b6 was set:: number of parameters

– – R0: R12 · ptr to command parameters or terminator, or if flags b6 was set:: ptr to private word

– – R0: R13 · ptr to command parameters or terminator, or if flags b6 was set:: SVC stack

– – R0: R14 · ptr to command parameters or terminator, or if flags b6 was set:: return address

On exit:

<= / R0 / error ptr if V set, or for configuration options:

– – error ptr if V set, or for configuration options:: 0 = "Bad *Configure option"

– – error ptr if V set, or for configuration options:: 1 = "Numeric parameter required"

– – error ptr if V set, or for configuration options:: 2 = "Parameter too large"

– – error ptr if V set, or for configuration options:: 3 = "Too many parameters"

– – R0: R1-R6 · error ptr if V set, or for configuration options:: can be corrupted

If flags b5 is set, the Help Code is entered on *Help like this:

On entry:

=> / R0 / ptr to buffer for your convenience

– – R0: R1 · ptr to buffer for your convenience: length of that buffer

– – R0: R12 · ptr to buffer for your convenience: private word

– – R0: R13 · ptr to buffer for your convenience: SVC stack

– – R0: R14 · ptr to buffer for your convenience: return address

Your code can display help text itself, or return a pointer to the provided buffer or your own: On exit:

<= / R0 / passed to OS_PrettyPrint if not zero

– – R0: R1-R6 · passed to OS_PrettyPrint if not zero: can be corrupted

NOTE As this table is terminated by a single zero byte, be careful to align before an adjacent code label. Failing to do so is a very common error.

SWI chunk base number

Zero, or a multiple of 64 less than $1 < 24$; b17 completely ignored, but should be zero

This is the base number of the SWI chunk for the module, if the module provides SWIs. This is used by RISC OS to enable the module to be called when a SWI within its range has been issued.

If you request a SWI number, you will be allocated a "chunk" of 64 SWIs, starting at a Base Number such as &4E440. This is the number that goes into the SWI chunk base number field and together with the SWI Handler offset indicates that the module provides SWIs. The SWI Name Table (below) provides a simple way of translating between SWI names and numbers.

Because many modules provide SWIs and must all be unique, you must never release a module without a properly allocated SWI chunk.

SWI Handler code

Zero, or aligned offset to code

Called when one of your 64 SWIs has been issued. On entry:

=> / R0-R9 / the caller's registers – your parameters

– – R0-R9: R11 · the caller's registers – your parameters: sub-chunk number (SWI index) 0-63, no other values possible

– – R0-R9: R12 · the caller's registers – your parameters: private word

- R0-R9: R13 · the caller's registers – your parameters: SVC stack
- R0-R9: R14 · the caller's registers – your parameters: return address

On exit:

<= / R0-R9 / as appropriate; R0 can be error block if V set

- R0-R9: R10-R12 · as appropriate; R0 can be error block if V set: can be corrupted

- R0-R9: CPSR · as appropriate; R0 can be error block if V set: flags returned to caller; if V set then R0 is error block

You should return error &1E6, "No such SWI" or suitably worded equivalent for any unimplemented SWIs. A typical Handler is:

```
Handler LDR R12, [R12]
CMP R11, #lastindex+1
ADDCC PC, PC, R11, LSL#2
B BadSWI
B SWIzero
B SWIone
...
BadSWI STMFD R13!, {R1-R7,R14}
ADR R0, BadSWIerr
MOV R1, #0
MOV R2, #0
ADR R4, Title
SWI XMessageTrans_ErrorLookup
LDMFD R13!, {R1-R7,PC}
BadSWIerr & &1E6
= "BadSWI",0
```

SWI Name Table

Zero, or unaligned offset to name table

This field is ignored unless the SWI Base Number and SWI Handler offsets are valid. SWI names comprise a prefix; an underscore; and then a name or numeric index such as "OS_WriteC" or "FooBar_63". The prefix is often the same as the module Title (it does not have to be), but it cannot begin with an "X". If your module Title is long, please take pity on your programmers and consider adopting a shorter SWI prefix. Module Titles and full SWI names must be unique, but SWI prefixes do not necessarily have to be – various modules share "Sound_" for example. This is difficult to coordinate and is discouraged.

The SWI Name Table is a list of consecutive null-terminated strings, starting with the SWI prefix (the part before the underscore) followed by the names (without prefix), and ending with a single null byte. If the Name Table field is zero, RISC OS will call the SWI Decoder code (below) if present, and otherwise default to a numeric index attached to the module Title. If there are insufficient names in the table for the requested SWI number, the Kernel will append a numeric index to your SWI prefix.

An example SWI Name Table:

```
switable = "TTXHelper",0
= "AutoDetect",0
= "ScanFlash",0
= "ConvertToISO",0
= "ConvertToANSI",0
= "ExtractLine",0
= 0
```

Note the final empty string to end the table. Also remember to align before using any code labels.

Unnamed SWIs at the end of the range can be omitted (and would be given numeric names automatically), but you cannot have empty strings within the table, so if the SWI corresponding to "ConvertToISO" in the above example were not defined the name should instead be "3" as that is its zero-based index.

The error-returning form of a SWI is indicated by b17 of the SWI number, and by prefixing the full name by "X". However, this is never included in the Name Table, it is automatic.

SWI Decoder code

Zero, or aligned offset to code

If the module has a valid SWI Base Number and SWI Handler offset but no SWI Name Table, then code can be provided to assist OS_SWINumberToString and OS_SWINumberFromString. R0 selects between the two functions, being 0-63 for the former, and negative for the latter. Other values will not occur.

NumberToString On entry:

- => / R0 / 0-63 sub-chunk SWI number for this module (Base Number matched)
- R0: R1 · 0-63 sub-chunk SWI number for this module (Base Number matched): ptr to buffer for the name, combined with R2
- R0: R2 · 0-63 sub-chunk SWI number for this module (Base Number matched): offset into the buffer to start ("X" may already have been written)
- R0: R3 · 0-63 sub-chunk SWI number for this module (Base Number matched): length of the buffer (i.e. offset from R1 to stop at)
- R0: R12 · 0-63 sub-chunk SWI number for this module (Base Number matched): private word
- R0: R13 · 0-63 sub-chunk SWI number for this module (Base Number matched): SVC stack
- R0: R14 · 0-63 sub-chunk SWI number for this module (Base Number matched): return address

On exit:

- <= · R2 · updated to point past the characters inserted into buffer, Kernel will terminate for you
- R4-R6 · can be corrupted
- R9,R10 · can be corrupted
- R12 · can be corrupted
- V set · if buffer overflow - no error block required

This is called by OS_SWINumberToString to provide a name for a given SWI number. Your code will only be called if the requested SWI number is one of the 64 starting from your SWI Base number, which cannot be < &200.

If the full SWI name fits into the buffer between R2 and R3, then write the unterminated name and return the updated R2. If the buffer is too small, return with V set but no error block in R0. There is no way to refuse to name a SWI, so build unused SWI names in the usual fashion - your SWI prefix; underscore; decimal index (R0 on entry).

NumberFromString On entry:

- => / R0 / negative (-1) to distinguish from the above case
- R0: R1 · negative (-1) to distinguish from the above case: ptr to ctrl-terminated string to recognise, no initial X
- R0: R12 · negative (-1) to distinguish from the above case: private word
- R0: R13 · negative (-1) to distinguish from the above case: SVC stack
- R0: R14 · negative (-1) to distinguish from the above case: return address

On exit:

- <= / R0 / 0-63 sub-chunk number if SWI name recognised, else negative
- R0: R1 · 0-63 sub-chunk number if SWI name recognised, else negative: can be corrupted if recognised
- R0: R2-R6 · 0-63 sub-chunk number if SWI name recognised, else negative: can be corrupted
- R0: R9,R12 · 0-63 sub-chunk number if SWI name recognised, else negative: can be corrupted

Unlike NumberToString, your Decoder will be called for all unrecognised SWIs. However, the value you return in R0 is ORRed with the module's SWI Base Number before being returned. Although this is intended to allow the module to only recognise its 64 SWI names, it can be seen that a module could in extremis return names for half of all SWI numbers.

Messages filename

Zero, or aligned offset to null-terminated filename

In order to facilitate internationalising modules, it is possible to provide a null-terminated filename for the Messages file for RISC OS to look up tokens from the Help and Command Table. If the Messages filename is present and bit4 of a flag byte in the above Command Table is set, the help and syntax text are tokens to be looked up.

This Messages file only relates to the Help and Command table. All other internationalisation within the module needs to be dealt with by the module author.

Module feature flags

Zero (26bit only) or aligned offset to flags word

The flags word comprises:

Bit 0 / Module is 32 bit compatible (26 bit only if bit is clear)

Bit 1 – Module is 32 bit compatible (26 bit only if bit is clear): Module is initialised before other modules (early initialisation)

Bits 2-3 – Module is 32 bit compatible (26 bit only if bit is clear): Reserved (should be zero)

Bits 4-7 – Module is 32 bit compatible (26 bit only if bit is clear): Module architecture type

Bits 8-31 – Module is 32 bit compatible (26 bit only if bit is clear): Reserved (should be zero)

If no flags word is present the module feature flags should be assumed to have a value of 0.

Modules not flagged as 32 bit compatible will not be loaded by a RISC OS 5.

If the 32 bit compatible flag is clear, the module is assumed to be 26 bit compatible only. Note that the 32 bit compatibility flag is only a marker, the actual module code is not guaranteed to be 32 bit safe by the presence of this flag. Incorrectly coded modules (with 26 bit only elements) will be loaded when this flag is present. It is the responsibility of the module author to ensure the code is compatible.

Early initialised modules are supported by RISC OS Select 3 and later. Modules with this flag set will be loaded before all other modules. This flag is used to ensure that extension modules provided by the PoduleManager (or other extensions) can be initialised before the main modules, and can correctly mask built in modules. In prior versions of RISC OS (and in RISC OS 5), the PoduleManager is treated specially.

The module architecture type is used to indicate the type of code which is present within the module. RISC OS will reject any architecture type which it does not understand. This flag must be used in conjunction with bit 30 of the module initialisation word in the header.

Currently defined module architecture types are:

0 / ARM 32 bit (A32)

1 – ARM 32 bit (A32): ARM 64 bit (A64)

2 – ARM 32 bit (A32): x86 64 bit (x86-64)

3-14 – ARM 32 bit (A32): Reserved

15 – ARM 32 bit (A32): Python (RISC OS Pyromaniac)

For more complete information on modules, please refer to the chapter on Modules in book 1 of the Programmer's Reference Manuals (it's a very big chapter!) but please note that the information provided there is nearly a quarter century and several major RISC OS versions out of date. The chapter is useful to provide a detailed overview of how modules work and interact, but is superseded by this document.

[RISC OS Open source page](#)

File formats: Squash file

File Formats

The official filetype of a Squash file is &FCA.

Offset / Length / Contents

0 – Length: 4 · Contents: "SQSH"

4 – Length: 4 · Contents: Length of data when uncompressed

8 – Length: 4 · Contents: Load address typically these hold

12 – Length: 4 · Contents: Exec address filetype and datestamp

16 – Length: 4 · Contents: 0 (reserved for future use)

[RISC OS Open source page](#)

File formats: Utility

Filetypes

Transient Utility files (filetype &FFC) contain position independent code which is loaded into the RMA and executed in User Mode. The purpose of a Utility is for when you want to run a small utility and then return back to the program environment that you were in prior to calling the Utility (unlike loading an Absolute program which would normally replace the current program).

Utility files

On entry to a transient Utility, the registers are as follows:

Entry conditions

R0 · Pointer to first character of command the line

R1 · Pointer to first character of the command tail (will be a control character if no parameters given)

R12 · Pointer to 1024 bytes of workspace

R13 · Pointer to end of workspace, for stack

R14 · Return address

A transient Utility is entered in User Mode with interrupts enabled. You should only call X form SWIs, and if there is an error, you should exit by setting the V flag and pointing R0 to an error block.

The operating system provides you with 1024 bytes of workspace, which includes the stack descending downwards from the end of the workspace. If more space is required, it should be claimed from the RMA.

You must exit the transient Utility with MOV PC, R14 (after freeing any RMA claims). As you are never the "current program", you must not call OS_Exit.

Denoting a Utility as being 32 bit safe

Aemulor may trigger problems with some Utilities. If the Utility is known to be 32 bit safe, it should be marked as such by appending the four bytes '3' '2' 'O' (upper case 'o') and 'K' (in that order!) to the end of the file.

This can be achieved by adding this to the end of the Utility source:

DCS "32OK"

Please note, this does not mean that the Utility is 32-bit only; it should be used to indicate that the Utility will operate correctly on both 26- and 32-bit architectures.

RISC OS Ltd (RISC OS 4/6) Utility header

RISC OS Ltd has defined a header for Utility programs. This is documented here for completeness, though please note that RISC OS 5 does not make use of such a header, and Aemulor would still require the "32OK" word at the end of the file.

ROLtd Utility Header /

+0 – : Branch to entry point (to skip over header)

+4 – : First magic value &79766748 ("Hgvy")

+8 – : Second magic value &216C6776 ("vgl!")

+12 – : Read only size (or 0 to mean the size of the file)

+16 – : Read/write size (or 0 to mean the size of the file)

+20 – : Either 26 or 32 depending on whether built for 26 bit or 32 bit systems

The two magic values are ROT13 for "Utility!".

Example

This is a simple but potentially very useful Utility that will list all open files, plus provide information on the file status.

The output looks like this:

*ListOpen

```

255 RW SDFS::RISCOSpi.$!Boot.Loader
254 R U devices#bulk;size16385:$.USB6
253 W! U devices#bulk;size16385:$.USB6
252 R Resources:$.Fonts.Homerton.Medium.Outlines0
251 W! U devices#endpoint1;interface0;alternate0;bulk;size32768:$.USB7
250 R U devices#endpoint2;interface0;alternate0;bulk;size32768:$.USB7
*
```

The status indicators are:

- R - file opened with Read access
- W - file opened with Write access
- ! - file has been written to
- E - file is at EOF
- U - file is unbuffered
- X - data has been lost

```
REM >makeListOpen
```

```
REM
```

```
REM Creates a transient Utility that lists all open files
```

```
REM
```

```
REM Rick Murray, 2014/10/31
```

```
REM This code is licenced as "whatever, dude". I'm not bothered.
```

```
REM It's for a RISC OS Open wiki example. So have fun with it.
```

```
DIM code% 320, L% -1
```

```
FOR I% = 8 TO 10 STEP 2
```

```
P% = code%
```

```
[ OPT I%
```

```
.begin
```

```
; RISC OS 4/6 Utility header
```

```
B start
```

```
EQUd &79766748
```

```
EQUd &216C6776
```

```
EQUd end - begin
```

```
EQUd 0
```

```
EQUd 32
```

```
.start
```

```
MOV R8, #255 ; start at file handle 255 and work backwards
```

```
.checkfile
```

```
; Read info on this handle
```

```
MOV R0, #254 ; OS_Args 254 -> read info on file handle
```

```
MOV R1, R8
```

```
SWI "XOS_Args" ; can only call X form SWIs in Utilities
```

```
MOVVS PC, R14 ; bomb out upon an error here
```

```
; Validate it
```

```
MOV R7, R0 ; stash the stream status word in R7
```

```
TST R7, #2048 ; check this is a valid handle
```

```
BNE nextfile ; if not, try the next one
```

```
; Output first the file handle number
```

```
MOV R0, R8
```

```
MOV R1, R12
```

```
MOV R2, #4 ; range is 0-255 (unsigned) only
```

```
SWI "XOS_ConvertCardinal1"
```

```

MOVVS PC, R14
; Work out how long the string is to right-align it
SUBS R6, R1, R0 ; R6 = terminator - buffer start
SWILE (&20100+32) ; OS_Writel for space padding
SUBS R6, R6, #1
SWILE (&20100+32) ; padding
SUBS R6, R6, #1
SWILE (&20100+32) ; padding
SWI "XOS_Write0" ; write the handle value
SWI (&20100+32) ; separator space
MOVVS PC, R14
; Now we want to write flags (or not) as
; RW!EUX
; |||||
; |||||'- data lost
; |||||'- unbuffered
; |||'--- at EOF
; ||'---- file has been written to
; |'----- file has write access
; '----- file has read access
TST R7, #(1<<6) ; bit 6 = Read access
MOVNE R0, #ASC("R")
MOVEQ R0, #ASC(" ")
SWI "XOS_WriteC"
TST R7, #(1<<7) ; bit 7 = Write access
MOVNE R0, #ASC("W")
MOVEQ R0, #ASC(" ")
SWI "XOS_WriteC"
TST R7, #(1<<8) ; bit 8 = has been written to
MOVNE R0, #ASC("!")
MOVEQ R0, #ASC(" ")
SWI "XOS_WriteC"
TST R7, #(1<<9) ; bit 9 = at EOF
MOVNE R0, #ASC("E")
MOVEQ R0, #ASC(" ")
SWI "XOS_WriteC"
TST R7, #(1<<10) ; bit 10 = unbuffered stream
MOVNE R0, #ASC("U")
MOVEQ R0, #ASC(" ")
SWI "XOS_WriteC"
TST R7, #(1<<13) ; bit 13 (ho! ho!) = data lost
MOVNE R0, #ASC("X")
MOVEQ R0, #ASC(" ")
SWI "XOS_WriteC"
SWI (&20100+32) ; separator space
MOVVS PC, R14
; Now determine the file NAME associated with this handle
MOV R0, #7 ; OS_Args 7 -> read pathname of open file
MOV R1, R8 ; (PRM 2-58)
MOV R2, R12 ; pointer to buffer
MOV R5, #512 ; workspace is 1024 so assume okay to give 512
SWI "XOS_Args"

```

```
MOVVS PC, R14
; poke a null word at the end in case 512 bytes wasn't enough!
MOV R2, #0 ; null
MOV R3, R12 ; point to workspace
ADD R3, R3, #512 ; add 512
STR R2, [R3] ; stick the null there
; Write the pathname, and a newline
MOV R0, R12
SWI "XOS_Write0"
SWI "XOS_NewLine"
MOVVS PC, R14
.nextfile
SUBS R8, R8, #1 ; for as long as there are file handles
BPL checkfile ; check 'em
; otherwise, we're done here
MOV PC, R14
; flag ourselves as 32 bit okay
ALIGN ; just in case(!)
EQUUS "32OK"
.end
]
NEXT
SYS "OS_File", 10, "$.ListOpen", &FFC,, code%, P%
END
RISC OS Open source page
```

File Server Protocol Interface

The File Server Protocol Interface is documented in PRM2. It only supports 24 bit file lengths, so the Advanced Level 4 File Server added extensions to increase file lengths to 32 bits. These extensions are documented, with some question marks on some details, [here](#).

Details

Functions

[RISC OS Open source page](#)

File Types

Filetypes

This is a community-managed list of filetypes, and thus may be incomplete or inaccurate in some areas. Although ROOL do maintain a complete list of allocated file types, the list cannot be publicly released due to the confidential nature of some allocations.

NOTE

For official information about filetype allocations, consult officially published documentation (e.g. the printed Programmer's Reference Manuals books).

Key Information

Key / Description

Status – Description: = OS-defined, = Registered, = Unregistered, = Do not use, Blank = unknown

Type – Description: Three digit hexadecimal code

Name – Description: The short name for the type (File\$Type_xxx)

Description – Description: The kind of data in files of this type

Application – Description: Primary or example application that use files of this type

MIME Type – Description: Common filename extensions and MIME content-types used elsewhere

Filetypes

- **FFF** – Name: Text · Description: ASCII text · Application: !Edit · MIME Type: txt text/plain
- **FFE** – Name: Command · Description: CLI Commands · Application: *Exec
- **FFD** – Name: Data · Description: Arbitrary data · MIME Type: application/octet-stream
- **FFC** – Name: Utility · Description: Transient program · Application: *Run
- **FFB** – Name: BASIC · Description: Tokenised BASIC program · Application: *BASIC · MIME Type: bas
- **FFA** – Name: Module · Description: Relocatable code module · Application: *RMRun
- **FF9** – Name: Sprite · Description: Picture or saved screen · Application: !Paint
- **FF8** – Name: Absolute · Description: Absolute ARM code, (loaded at &8000) · Application: *Run
- **FF7** – Name: BBC font · Description: BBC font, (VDU code sequence) · Application: *Print
- **FF6** – Name: Font · Description: Outline font, 1bpp or 4bpp bitmap · Application: !FontEd
- **FF5** – Name: PoScript · Description: PostScript · Application: !Printers · MIME Type: ps eps application/postscript
- **FF4** – Name: Printout · Description: Dot-matrix printout · Application: !Printers
- **FF3** – Name: LaserJet · Description: HP LaserJet print data (PCL) · MIME Type: pcl
- **FF2** – Name: Config · Description: 240 byte CMOS RAM configuration
- **FF1** – Name: RawData · Description: Raw unprocessed data
- **FF0** – Name: TIFF · Description: Tagged Image File Format picture · Application: !ChangeFSI · MIME Type: tif tiff image/tiff
- **FEF** – Name: Diary · Description: Arthur desktop diary
- **FEE** – Name: NotePad · Description: Arthur desktop notepad
- **FED** – Name: Palette · Description: Desktop palette · Application: *WimpPalette
- **FEC** – Name: Template · Description: Window templates · Application: !FormEd
- **FEB** – Name: Obey · Description: CLI commands · Application: *Obey
- **FEA** – Name: Desktop · Description: Desktop setup · Application: *Desktop
- **FE9** – Name: ViewWord · Description: View word processor
- **FE8** – Name: ViewPS · Description: View word processor
- **FE7** – Name: ViewSht · Description: ViewSheet spreadsheet
- **FE6** – Name: UNIX Ex
- **FE5** – Name: ROM · Description: OS softload image (or EPROM)
- **FE4** – Name: DOS · Description: IBM PC disc file
- **FE3** – Name: Atari · Description: Atari ST disc file (or ST)
- **FE2** – Name: Amiga · Description: Commodore Amiga disc file (or CBM)
- **FE1** – Name: Makefile · Description: DDE Make Makefile · Application: !Make
- **FE0** – Name: Accessry · Description: Arthur desktop accessory (obsolete)
- **FDF** – Name: VTScript · Description: TCP/IP
- **FDE** – Name: VTSetup · Description: TCP/IP
- **FDD** – Name: MasterUt · Description: BBC Master utilities
- **FDC** – Name: SoftLink
- **FDB** – Name: TextCRLF
- **FDA** – Name: MSDOSbat · Description: Windows batch file · MIME Type: bat
- **FD9** – Name: MSDOSexe · Description: Windows executable · MIME Type: exe
- **FD8** – Name: MSDOScom · Description: Windows command file · MIME Type: com
- **FD7** – Name: TaskObey · Description: *Obey in a TaskWindow

- **FD6** – Name: TaskExec · Description: *Exec in a TaskWindow
- **FD5** – Name: Pict · Description: Macintosh Pict metafile · Application: !ImageFS
- **FD4** – Name: MIDI · Description: Musical Instrument Digital Interface · Application: !ReMidi · MIME Type: mid midi audio/x-midi
- **FD3** – Name: DeblImage · Description: DDE Debug file · Application: !DDT
- **FD2** – Name: ScrDiff
- **FD1** – Name: BASICTxt · Description: DeskEdit Basic program in text form · Application: *BASIC
- **FD0** – Name: PCConfig · Description: PC Emulator configuration · Application: !PcEm
- **FCF** – Name: Cache · Description: FontCache · Application: *LoadFontCache
- **FCE** – Name: Floppy · Description: Filecore floppy disc image · Application: !ADFFS · MIME Type: jfd adf apd
- **FCD** – Name: HardDisc · Description: Filecore hard disc image · Application: !HDFFS · MIME Type: hdf
- **FCC** – Name: Device · Description: Device
- **FCA** – Name: Squash · Description: Compressed file · Application: !Squash
- **FC9** – Name: SunRastr · Description: Sun Microsystems raster file · Application: ImageFS · MIME Type: srf
- **FC8** – Name: DOSDisc · Description: DOS Disc
- **FC7** – Name: MacType1 · Description: Mac format PostScript Type1 font
- **FC6** – Name: PrintDfn · Description: Printer definition · Application: !PrintEdit
- **FC3** – Name: Patch · Description: Risc OS 3 Patch · Application: !Patch
- **FC2** – Name: AIFF · Description: Audio IFF · MIME Type: aif aiff aifc audio/x-aiff
- FC0** – Name: Link · Description: Symbolic link · Application: !LinkFS
- **FBE** – Name: PBCards · Description: PocketFS
- **FBD** – Name: PBWrite · Description: PocketFS
- **FBC** – Name: PBAbacus · Description: PocketFS
- **FB5** – Name: NoDisc · Description: Acorn Access
- **FB4** – Name: DiscR · Description: Acorn Access · Application: *ShareFSIcon
- **FB2** – Name: MovieFS · Description: Video · Application: !CineWorks · MIME Type: avi wav mov fli flc video/x-msvideo video/quicktime
- **FB1** – Name: WAV · Description: Sound sample, many sub-formats · Application: DataPlay · MIME Type: wav audio/x-wav
- **FB0** – Name: Allocate · Description: RISC OS name allocation request · Application: !Allocate
- **FAF** – Name: HTML · Description: Hypertext markup language · MIME Type: html htm text/html
- **FAE** – Name: Resource · Description: Toolbox UI definition · Application: !ResEd
- **FAD** – Name: Methods · Description: Toolbox Method definition · Application: !MethodGen
- **FAC** – Name: HCL · Description: HyperCard Language · Application: !Hyper
- FAB** – Name: KnowBase · Description: Expert system
- **FAA** – Name: STImage · Description: Smalltalk image/snapshot file · Application: !Squeak & !Scratch
- **FAA** – Name: UtilLabel · Description: Unused?
- FA8** – Name: EasyData
- FA2** – Name: HomeWrks
- **F9F** – Name: DiscD · Description: Acorn Access · Application: *ShareFSIcon
- **F9E** – Name: DiscDP · Description: Acorn Access · Application: *ShareFSIcon
- **F9D** – Name: DiscCD · Description: Acorn Access · Application: *ShareFSIcon
- **F98** – Name: PhotoShp · Description: Adobe Photoshop picture · Application: !PhotoDesk · MIME Type: psd
- F95** – Name: Code · Description: Parallel of Data (FFD)
- **F91** – Name: URI · Description: Uniform Resource Indicator · Application: !FTPc
- F89** – Name: GZip · Description: Gzip, (GNU Zip) · MIME Type: gz application/x-gzip
- F88** – Name: RealAud · Description: Real compressed audio
- F83** – Name: MNG · Description: Motion Network Graphics animation · MIME Type: mng
- **F81** – Name: JSScript · Description: JavaScript · MIME Type: js
- **F80** – Name: XML · Description: eXtended Markup Language files · MIME Type: xml */xml */+xml
- F7C** – Name: Teletext · Description: Teletext screen data
- F7B** – Name: Obey26 · Description: 26-bit only Obey file · Application: !Aemulor
- F7A** – Name: Basic26 · Description: 26-bit only BASIC file · Application: !Aemulor
- F79** – Name: CSS · Description: Cascading Style Sheet · MIME Type: css
- F78** – Name: JNG · Description: Joint Network Graphics picture · MIME Type: jng

F77 – Name: Squarc · Description: Archive

- **F76** – Name: EDID · Description: Binary EDID dump · Application: *LoadModeFile
- **F75** – Name: JSON · Description: JavaScript Object Notation file · MIME Type: application/json
- **F74** – Name: YAML · Description: YAML Ain't Markup Language file

F1B – Name: 65Host · Description: Usually a Basic program

F19 – Name: BasicII · Description: BBC Basic from BBC Micro

F12 – Name: BasicNt · Description: BBC Basic from MasterET

- **F0E** – Name: EconetUt · Description: BBC Econet Utility

F0E – Name: BasicIV · Description: BBC Basic from Master128

- **F09** – Name: WiniUtil · Description: BBC Winchester Utility

EEB – Name: TinyDraw

EEB – Name: ScrnTurt · Description: ScreenTurtle logo definitions

EEA – Name: TinyLogo

EEA – Name: ScrnTurt · Description: ScreenTurtle compressed screens

ED1 – Name: EdSProg · Description: Script for EdScheme: Scheme + turtle graphics + editor

ED1 – Name: ComSound · Description: Com sound

ED0 – Name: RawSound · Description: Raw sound sample · Application: !DSEdit

ED0 – Name: DSEdSmpl · Description: Raw sound sample · Application: !DSEdit

ECF – Name: ECFData · Description: Pro-Artisan extended colour fill

ECD – Name: TeleSoft

ECB – Name: PCBFile

E92 – Name: Disclndx

E1F – Name: ELF · Description: ELF format executable · Application: elfrun · MIME Type: elf

E13 – Name: Drummer · Description: The Drummer

- **DFF** – Name: Logistix · Description: Logistix data
- **DFE** – Name: CSV · Description: Comma separated value spreadsheet · MIME Type: csv

DFD – Name: DIF

DFC – Name: LogoExt · Description: Logo extensions

DFB – Name: LookUp · Description: Watford Video Digitiser lookup

DFA – Name: Picture · Description: Watford Video Digitiser picture · Application: !Translatr

- **DF9** – Name: Waveform · Description: SoundSynth waveform · Application: SoundSynth/WFS/Studio 24/Arpeggio

DF9 – Name: SndSynth · Description: Sound (not official?) · Application: SoundSynth

DF6 – Name: CDImage · Description: CD ROM image · Application: CDRMF5 · MIME Type: iso application/x-iso9660-image

DF6 – Name: ISOImage · Description: CDROM image · Application: CDFaker · MIME Type: iso application/x-iso9660-image

DF2 – Name: DFSDisc · Description: BBC Micro floppy image

DF0 – Name: Font · Description: AMS Finesse Paint

DEF – Name: Pattern · Description: AMS Finesse Paint

DEE – Name: Canvas · Description: AMS Finesse Paint

DED – Name: Finesse · Description: AMS Finesse Paint

DED – Name: Recycled · Description: Discarded file

DEC – Name: SLD · Description: AutoSketch

DEC – Name: DiscRec · Description: Disc Record

DEB – Name: PLT · Description: AutoSketch

DEA – Name: DXF · Description: Data eXchange Format, AutoSketch, AutoCAD · Application: !Draw · MIME Type: dxf application/dxf

DE9 – Name: SKD · Description: AutoSketch SKD

DE8 – Name: SD+Data · Description: System Delta Plus data

DE7 – Name: SD+Form · Description: System Delta Plus form

DE6 – Name: SG+Graph · Description: System Gamma graph

DE5 – Name: SG+Macro · Description: System Gamma macro

DE4 – Name: Sheet · Description: System Sigma spreadsheet

DE3 – Name: SSMacro · Description: System Sigma macro

DE2 – Name: ProArt · Description: Pro-Artisan compressed screen · Application: !Translatr

DE2 – Name: XarcPic · Description: compressed picture
DE1 – Name: Euclid · Description: Euclid 3D object
DE0 – Name: Sibelius · Description: Sibelius info
DDF – Name: Logistix
DDF – Name: Genesis · Description: Sibelius music
DDF – Name: Disk manager index
 · **DDE** – Name: PDream · Description: PipeDream document · Application: !PipeDream · MIME Type: pd
 · **DDD** – Name: Wilson · Description: unused
DDD – Name: GhostDir · Description: Symbolic link · Application: !GhostDirs
DDC – Name: Archive · Description: Compressed archive · Application: !SparkFS · MIME Type: zip application/zip
ddb – Name: MapData · Description: MapData
DDA – Name: ColSet · Description: Pro-Artisan colour set
DD9 – Name: MagicPen · Description: Pro-Artisan magic brush
DD8 – Name: ProFont · Description: Pro-Artisan font
DD7 – Name: Bezier · Description: Pro-Artisan bezier curves
DD6 – Name: PAX · Description: PAX art package
DD5 – Name: HD Log · Description: Hard disc archiver backup log
DD4 – Name: HDArch · Description: Hard disc archiver
DD3 – Name: Prisma3 · Description: Prisma3
DD2 – Name: PlutoFnt · Description: Pluto font
DD1 – Name: PlutoFIF · Description: Pluto font info
DD0 – Name: BNI_Data · Description: Monitor system
DCF – Name: Song
DCE – Name: Pattern
DCD – Name: Track
DCC – Name: Arrange
DCB – Name: VoiceSet · Description: Studio24+ DrumList/MidiList/ClipList · Application: !Studio24+
DCA – Name: Conductr · Description: Studio24+ graphics · Application: !Studio24+
DC9 – Name: ClipBord · Description: Studio24+ · Application: !Studio24+
DC8 – Name: Song
DC8 – Name: Graphic
DC7 – Name: Rhythm
DC6 – Name: Score · Description: Studio24+ clipboard · Application: !Studio24+
DC5 – Name: ScnMusic · Description: Studio24+ conductor · Application: !Studio24+
DC4 – Name: VidData · Description: Studio24+ voice set · Application: !Studio24+
DC3 – Name: Overlays · Description: Studio24+ arrangement · Application: !Studio24+
DC2 – Name: NMEs · Description: Studio24+ track · Application: !Studio24+
DC1 – Name: Text · Description: Studio24+ pattern · Application: !Studio24+
DC0 – Name: MDMMusic · Description: Studio24+ song · Application: !Studio24+
DB8 – Name: DataKing
DB7 – Name: Squirrel · Description: Squirrel data
DB6 – Name: DToBase · Description: Desktop Office
DB5 – Name: DataVision · Description: Silcon Vision DataVision
DB4 – Name: SuperCalc3
DB3 – Name: DBaseIII · Description: DBase III database
DB2 – Name: DBaseII · Description: DBase II database
DB1 – Name: DBaseInd · Description: DBase index database
DB0 – Name: Lotus123 · Description: Lotus 1-2-3 spreadsheet · MIME Type: wk1
DAF – Name: VPalette
DAF – Name: Sound · Description: Armadillo Sound Sample
DAF – Name: Magpie? · Description: Magpie music
DAE – Name: Fourier? · Description: FourierTrns Fourier transform

DAE – Name: HNCode · Description: Armadillo HighNote module · Application: HighNote
DAD – Name: HNData · Description: Armadillo HighNote sound · Application: HighNote
DAC – Name: HNConfig · Description: Armadillo HighNote config · Application: HighNote
DA0 – Name: VPScript
 · **D98** – Name: PackFile · Description: Archive
 · **D97** – Name: Address · Description: Address book
 · **D96** – Name: CFSIzw · Description: !CFS compressed file · Application: !CFS
 · **D95** – Name: ArtWkMod · Description: Artworks extension · Application: !Artworks
 · **D94** – Name: Artworks · Description: Drawing · Application: !Artworks
 · **D93** – Name: DDF · Description: Impression Document Description Format · Application: !Impression
 · **D92** – Name: Screen · Description: LaserDirect halftone screen
 · **D91** – Name: Equasor · Description: Equasor equations · Application: !Equasor
D90 – Name: DataBase
 · **D90** – Name: FaxExec · Description: unused
 · **D8F** – Name: CCRTF · Description: unused
 · **D8E** – Name: ABIDefs · Description: Application Binary Interface UI definitions
 · **D8D** – Name: LDirect · Description: Laser Direct printer file
 · **D8C** – Name: GreyMap · Description: unused
 · **D8B** – Name: Abbrev · Description: Spell checker abbreviation dictionary
 · **D8A** – Name: Excep'ns · Description: Spell checker exception dictionary
 · **D89** – Name: IModule · Description: Impression Imodule · Application: !Impress
 · **D88** – Name: Stories · Description: Impression stories · Application: !Impress
 · **D87** – Name: DocData · Description: Impression document data · Application: !Impress
 · **D86** – Name: CCcolour · Description: unused
 · **D85** – Name: FaxImage · Description: CC FaxPack picture
 · **D84** – Name: Diction · Description: Impression Junior spelling dictionary
 · **D83** – Name: Ignore · Description: Impression II/Impression Junior ignore list
 · **D82** – Name: IChart · Description: InterChart chart · Application: !IChart
 · **D81** – Name: ISheet · Description: InterSheet spreadsheet · Application: !ISheet
 · **D80** – Name: IWord · Description: InterWord word processor document · Application: !IWord
D7F – Name: ComSprit · Description: Atelier compacted sprite
D7E – Name: MinBrush · Description: Atelier user-defined brush
D7D – Name: MinAnim · Description: Atelier animation sequence
D7C – Name: HomeAcct · Description: Home accounts
D7B – Name: DeltaSet · Description: System Delta Plus MultiStore setup
D7A – Name: Min3DObj · Description: 3-D object
D79 – Name: Min24Bit · Description: 24-bit screen
D78 – Name: FamTree · Description: Family tree
D77 – Name: FamTreeA · Description: Family tree ancillary
D76 – Name: MitDiary · Description: Diary
D76 – Name: MitDiary
D75 – Name: MitProje · Description: Project
D74 – Name: TypeFile
D74 – Name: MitBakup · Description: DiscTree backup
D73 – Name: MitBScpt · Description: DiscTree backup script
D72 – Name: ForthSrc · Description: Forth source
D71 – Name: ForthDat · Description: Forth
D70 – Name: PlutoPal · Description: Pluto palette
D6F – Name: PIBBCFnt · Description: Pluto BBC font
D6E – Name: PIBBCFIF · Description: Pluto BBC font info
D6D – Name: PIBBCPic · Description: Pluto old RLE picture · MIME Type: rle
D6C – Name: PIDesPic · Description: Pluto designer picture

D6B – Name: PComScr · Description: Pluto compressed screen
D6A – Name: AceFilm · Description: Projector/Euclid compressed screen image sequence · Application: !Projector
D69 – Name: M3Spread · Description: Matrix3 spreadsheet
D68 – Name: M3Exec · Description: Matrix3 executable
D67 – Name: M3Conf · Description: Matrix3 configuration
D66 – Name: Viewdata · Description: Hearsay Viewdata frame
D66 – Name: ViewData
D66 – Name: PAX picture
D65 – Name: HSStatus · Description: Hearsay status
D64 – Name: HSTelDir · Description: Hearsay telephone directory
D63 – Name: HSSetup · Description: Hearsay setup
D62 – Name: BNI_sp · Description: ElectroPhysical Workstation single-precision data
D61 – Name: BNI_dp · Description: ElectroPhysical Workstation double-precision data
D60 – Name: BNI_sd · Description: ElectroPhysical Workstation standard deviation data
D5F – Name: BNI_pd · Description: ElectroPhysical Workstation patient data
D5E – Name: BNI_mes · Description: ElectroPhysical Workstation derived measures
D5D – Name: BNI_map · Description: ElectroPhysical Workstation topographic map
D5C – Name: BNI_plot · Description: ElectroPhysical Workstation plotter defaults
D5B – Name: BNI_defm · Description: ElectroPhysical Workstation default measures
D5A – Name: BNI_grap · Description: ElectroPhysical Workstation stored graph
D59 – Name: RendScen · Description: RenderBender scene description
D58 – Name: RendPic · Description: RenderBender compressed pictures · Application: !Converter
D57 – Name: RendAnim · Description: RenderBender/!Translatr animation
D56 – Name: ING · Description: Interactive Network Graphics
D55 – Name: HyperTxt · Description: Hypertext
D54 – Name: TimeTabl · Description: Timetable control
D53 – Name: TTDData · Description: Timetable support
D52 – Name: ISOCObj · Description: ISO C AOF object code
D51 – Name: ISOCLib · Description: ISO C library
D50 – Name: ISOCDbug · Description: ISO C debug
D4F – Name: Colours · Description: Virtual Palettes
D4E – Name: Archive · Description: Tape Archiver
D4D – Name: Archway · Description: Archway DIY programming suite
D4C – Name: SprScan
D4B – Name: DigiSim
D4A – Name: APL · Description: I-APL
D49 – Name: Book · Description: HyperBook book
D48 – Name: Notes · Description: HyperBook notes
D47 – Name: Index · Description: HyperBook indexes
D46 – Name: Glossary · Description: HyperBook glossaries
D45 – Name: Vocablry · Description: HyperBook/Numerator vocabularies
D44 – Name: Numrator · Description: Numerator workspaces
D43 – Name: Modules · Description: Numerator modules
D42 – Name: InData · Description: Numerator input
D41 – Name: WordList
D41 – Name: OutData · Description: Numerator output
D41 – Name: List · Description: PenDown word list
D40 – Name: PDDoc · Description: PenDown
D40 – Name: Help · Description: Numerator help
D3F – Name: FSPaths · Description: Instigator
D3E – Name: FNKeys · Description: Instigator
D3D – Name: OSVars · Description: Instigator

D3C – Name: Sample · Description: !Armadeus, (registered as armSamps")
D3C – Name: Armadeus · Description: Sound sample · Application: !Armadeus
D3C – Name: LinSignd
D3B – Name: Song · Description: Music track containing sample + loop points · Application: !Armadeus
D3A – Name: Patterns · Description: Atelier patterns
D39 – Name: Formulae · Description: Atelier formulae
D38 – Name: Screen
D37 – Name: Compress
D36 – Name: Captured
D35 – Name: Sheet
D34 – Name: MacroLib
D33 – Name: Profile3 · Description: HearSay/ArcComm Videotex profile 3, (Prestel) frame store
D32 – Name: Profile2 · Description: HearSay/ArcComm Videotex profile 2, (Tltel) frame store
D31 – Name: Profile1 · Description: ArcComm Videotex profile 1, (Bildschirmtext) frame store
D30 – Name: DToComms · Description: DesktopOffice
D30 – Name: CommProc · Description: ArcComm procedure,(text)
D2F – Name: CommMail · Description: ArcComm mailbox text,(binary)
D2E – Name: Presentr · Description: Presenter
D2D – Name: DOSFile
D2C – Name: ARMIInst
D2B – Name: AtelSeq · Description: Atelier sequence
D2A – Name: Magazine
D29 – Name: Proman
D28 – Name: GIPAL
D27 – Name: GIData
D26 – Name: PremDoc · Description: Premier document
 · **D26** – Name: KML · Description: Route data e.g. GPS · Application: !RiscOSM
D25 – Name: PremCmd · Description: Premier command
D24 – Name: PremData · Description: Premier
D23 – Name: Lisp
D22 – Name: LispText
 · **D21** – Name: PDCmdFil · Description: PipeDream macro (commands) · Application: !PipeDream
D20 – Name: APECImag · Description: APEC ROM image
D1F – Name: PALData · Description: PAL programmer fuse image
D1E – Name: DiscDump · Description: Investigator2 dump
D1D – Name: Labels · Description: LabelMaster+
D1C – Name: ArcScan · Description: ArcScan bibliographic database
D1B – Name: Frame
D1A – Name: Tformat · Description: !Titler
D1A – Name: Grid
D19 – Name: Menu · Description: !Menu Waiter
D18 – Name: Quest · Description: Quest
D17 – Name: CmunITel · Description: CommuniTelFrame
D16 – Name: Mode7 · Description: 1K mode 7 screen image
D15 – Name: MitLnkDf · Description: Front linker driver
D14 – Name: MitMak · Description: CFront MakeFile
D13 – Name: Batch
D12 – Name: DBRiscOS
D11 – Name: Pointers
D10 – Name: SatPict
D0C – Name: 1WPDict · Description: First Word Plus spelling dictionary
D0B – Name: 1WPCfg · Description: First Word Plus printer configuration

D09 – Name: Action · Description: Tween action
D08 – Name: Genesis · Description: Genesis script
D07 – Name: Genesis · Description: Genesis button
D01 – Name: EasiDoc · Description: EasiWriter
D00 – Name: EasiPad · Description: EasiWriter
 · **CFF** – Name: CFF Font · Description: Compact Font Format font · MIME Type: cff
CFF – Name: EasiDict · Description: EasiWriter
CF9 – Name: TWScript · Description: TwEdit
CF8 – Name: Alps · Description: ALPS (Alpine Software)
CF7 – Name: Schema · Description: CadMust PCB Schematic
CF6 – Name: PCB · Description: CadMust PCB
CF3 – Name: Standings · Description: Chequered Flag
CF2 – Name: Circuit · Description: Chequered Flag
CF1 – Name: Champ's · Description: Chequered Flag
CF0 – Name: Standings · Description: Chequered Flag
CED – Name: WCADFile · Description: Worra CAD
CEB – Name: KiddiCAD · Description: Kiddi CAD
CEA – Name: WorraCAD · Description: Worra CAD
CE9 – Name: KeyAuth · Description: Key Plus
CE8 – Name: KeyFile · Description: Key Plus
CE7 – Name: KeyCalc · Description: Key Plus
CE6 – Name: KeyMap · Description: Key Plus
 · **CE5** – Name: TeX · Description: TeX source · MIME Type: tex application/x-tex
CE4 – Name: DVI · Description: DVI, (TeX) output · MIME Type: dvi application/x-dvi
CE2 – Name: Tripfile
 · **CDD** – Name: Ovation · Description: Ovation DTP document
 · **CDC** – Name: OvnStyle · Description: Ovation DTP style sheet
CD5 – Name: Tabs
CD3 – Name: SolidCAD · Description: Silicon Vision's Solid CAD
CD3 – Name: Quick Art / High Art
CD2 – Name: TripFile
CD2 – Name: FilmPath
CCF – Name: LotusPic · Description: Lotus picture · Application: ImageFS
CCD – Name: WaveLength
CCC – Description: Mah-Jong Patience (Cambridge International Software) save · Application: !MahJong
CCC – Name: DietSprite
CC7 – Name: DToWord · Description: DesktopOffice
CC6 – Name: DToSheet · Description: DesktopOffice wordprocessor
CC5 – Name: TeqMusic · Description: Music track. Same format as &701
 · **CC3** – Name: Poster · Description: Poster File · Application: !Poster, also used by !Hatchback, !Chameleon
CC2 – Name: SnapShot · Description: SnapShot colour digitiser, (enhanced?) image
CC1 – Name: Image · Description: ArcImage
CBF – Name: Or_Setup · Description: !Orrery
CBA – Name: Karma · Description: !Karma saved game
CB9 – Name: Perform
CB8 – Name: CashWide
CB8 – Name: 7Script · Description: ArcTerm7 command script
CB7 – Name: 7BatCfg · Description: ArcTerm7 batch or configuration
CB6 – Name: STModule · Description: Sound Tracker music sequence and samples · Application: !Tracker · MIME Type: mod
CB6 – Name: ArcTrack
CB5 – Name: STSample · Description: STracker SoundTracker sample
CB4 – Name: Cash Nar

CB4 – Name: C64_Musi · Description: C64 music track
CB2 – Name: PDTMovie · Description: Parametric Design Tool movie
CB1 – Name: PDT Snap · Description: Parametric Design Tool picture
CB0 – Name: PDT Cfg · Description: Parametric Design Tool configuration
CAF – Name: IGES · Description: Industry standard graphics
CAE – Name: HPGLPlot · Description: Hewlett-Packard Graphics Language drawing · MIME Type: hpgl
CAD – Name: PDT Dwg · Description: Parametric Design Tool drawing
CAC – Name: Document · Description: Dekstop Folio document
CAB – Name: Paper · Description: Desktop Folio paper · Application: Desktop Folio
CA7 – Name: Concept
CA6 – Name: Vector · Description: Illusionist Vector
CA5 – Name: Sweep · Description: Illusionist Sweep
CA3 – Name: PK · Description: TeX font bitmaps (compressed) · MIME Type: pk
CA2 – Name: GF · Description: TeX font bitmaps (uncompressed) · MIME Type: gf
CA1 – Name: TFM · Description: TeX font metrics · MIME Type: tfm
CA0 – Name: Metafont · Description: Metafont source · MIME Type: mf
C99 – Name: Binder · Description: Magpie Binder
C8B – Name: JoyStick · Description: Joystick
C8A – Name: Palette+ · Description: PCAT's Palette Plus
C89 – Name: Personal accounts
 · **C88** – Name: Prophile · Description: Prophet accounts file · Application: !Prophet
C88 – Name: AldeComm · Description: Aldebaran (Evolution Trading) save · Application: !Aldebaran
C87 – Name: Personal Accounts
 · **C87** – Name: Invoice · Description: Prophet invoice file · Application: !Prophet
C87 – Name: EliteCom · Description: Elite (Hybrid Technology) save · Application: !élite
C86 – Name: OCRData
 · **C85** – Name: JPEG · Description: JPEG compressed picture · Application: !ChangeFSI · MIME Type: jpg jpeg image/jpeg
C84 – Name: Overlay
C83 – Name: CKOver
C82 – Name: Sequence
C7F – Name: Titler
C7E – Name: Database
C7D – Name: SID · Description: Spreadsheet Interchange Data
C7B – Name: PhoneNum · Description: !Grapevine phone number database including terminal type & key definitions · Application: !Grapevine
C7A – Name: LcadDRAW · Description: !LinCAD · Application: !LinCAD
C76 – Name: Squashed
C75 – Name: Sound
C74 – Name: AlpMusic
C74 – Name: Sound · Description: ExSound
C73 – Name: RawMusic · Description: ALPS
C70 – Name: LcadBLOCK · Description: !LinCAD · Application: !LinCAD
C6D – Name: Pinpoint? · Description: Pinpoint · Application: PinPoint
C6C – Name: PinPoint · Description: Pinpoint · Application: PinPoint
 · **C6B** – Name: NootBook · Description: !Noot animation workbook
 · **C6A** – Name: smArt · Application: !smArt
C62 – Name: DeskEdit · Description: DeskEdit printer options
C5A – Name: Master · Description: Modified Replay video and audio
C59 – Name: Log · Description: Text file describing edited video and audio segments
C58 – Name: Ideas
C57 – Name: Notate · Description: Notate music
C56 – Name: VecFile · Description: Vector · Application: !Vector

C55 – Name: DrumSeq · Description: Drum sequence · Application: RhthymBed
C4E – Name: GTi
 · **C4D** – Name: Battle · Description: Dreadnoughts save game · Application: !DNoughts
 · **C4B** – Name: CmprDraw · Description: Compressed draw file (or DrwC) · Application: !Poster, !Hatchback, !Chameleon
C48 – Name: MIDITrck · Description: Related to FD4?
C46 – Name: Tar · Description: Unix tape archive · Application: !SparkFS · MIME Type: tar application/x-tar
C46 – Name: MIDITrak? · Description: MIDIJuke
C43 – Name: ChartOpt · Description: ChartWell ChartOpt File
C3C – Name: Shape
C3B – Name: FST
C37 – Name: EurSheet · Description: Eureka spreadsheet · Application: !Eureka
C36 – Name: SFXSheet
C35 – Name: CorlDraw · Description: Corel Draw drawing
C33 – Name: CADfile
 · **C32** – Name: RTF · Description: Rich Text Format · MIME Type: rtf application/rtf
C2E – Name: EurMacro · Description: !Eureka · Application: !Eureka
C2D – Name: EurChart · Description: !Eureka · Application: !Eureka
C2A – Name: IotaFilm · Description: Video
C2A – Name: SignGfx
C29 – Name: HelpBook
C29 – Name: DP tmplt · Description: DataPower template · Application: !DataPower
C28 – Name: Graph · Description: DataPower graph · Application: !DPGraphs
C27 – Name: Database · Description: DataPower database · Application: !DataPower
C26 – Name: ASheet
C25 – Name: AWord
C24 – Name: Architech
C23 – Name: Plot
C21 – Name: WX-Task · Description: !WeatherDX · Application: !WeatherDX
C1F – Name: HoledOut · Description: Holed Out golf game · Application: Holed Out golf game
 · **C1E** – Name: FwkzCmd · Description: Fireworkz commands · Application: !Fireworkz
 · **C1D** – Name: FwrkzTem · Description: Fireworkz template · Application: !Fireworkz
 · **C1C** – Name: Wordz · Description: Fireworkz (word processing) · Application: !Fireworkz
C1B – Name: KOPoint
C1A – Name: Subject
C19 – Name: Stencil · Description: Talking Pictures, The Spoken Word
C14 – Name: S-Data · Description: !S-base database · Application: !S-base
C13 – Name: ANGLDATA
C10 – Name: APAPos
C0F – Name: BookPos
C0D – Name: Molecule · Description: !Modeller · Application: !Modeller
C0C – Name: SMacro
C05 – Name: Ultra · Description: Ultra Tracker music track · Application: !DigitalCD · MIME Type: ult
C04 – Name: ImpTrack · Description: Impulse Tracker music track · Application: !DigitalCD
C03 – Name: FastTrak · Description: Fast Tracker music track · Application: !DigitalCD
C02 – Name: ScrmTrak · Description: Scream Tracker music track · Application: !DigitalCD · MIME Type: s3m stm
C01 – Name: Central heating calculator
C00 – Name: Rhapsody · Description: Rhapsody score
BFF – Name: Map
BF8 – Name: MPEG · Description: MPEG video · Application: !MPEGPlay · MIME Type: mpg mpeg video/mpeg
BF7 – Name: RIFF · Description: General RIFF – chunked IBM PC multimedia format
BEE – Name: Splosh
BED – Name: FontDisc

BE9 – Name: SWorkSp

BE8 – Name: PhotoCD · Description: PhotoCD multiresolution picture · Application: !PhotoDesk

BE8 – Name: DAsPict

- **BE1** – Name: Resultz · Description: Fireworkz (spreadsheet) · Application: !Fireworkz
- **BE0** – Name: Recordz · Description: Fireworkz Pro database container · Application: Fireworkz Pro
- **BDF** – Name: Firewrkz · Description: Fireworkz document · Application: !Fireworkz · MIME Type: fwk
- **BDA** – Name: Disc · Description: Acorn Access+ · Application: *ShareFSIcon
- **BD9** – Name: DiscP · Description: Acorn Access+ · Application: *ShareFSIcon

BD8 – Name: MacForks · Description: Mac HFS forked file · Application: !MacFS

- **BD7** – Name: HyprDict · Description: Help document · Application: !HyperView
- **BD6** – Name: Audiowrk · Description: Sound sample · Application: AudioManager

BD5 – Name: Envelope

BD1 – Name: Rephorm

BD0 – Name: PhotoBase

- **BCF** – Name: TblMate

BCB – Name: Lexicon

BC9 – Name: PrimeSlv

- **BC5** – Name: ImpDoc · Description: Impression single file document · Application: !Publisher

BC4 – Name: PremGame · Description: Premier Manager (Gremlin Graphics) save · Application: !Premier

BC2 – Name: PresMgrlm · Description: IBM OS2 Presentation Manager image · Application: ImageFS

- **BBD** – Name: RevBAS · Description: Use 1C7 for BBCBasic(86) and BBCBasic(80)

BBD – Name: CheatData

BBD – Name: BBC Data · Description: BBC Micro datafile

- **BBC** – Name: BBC ROM · Description: BBC micro ROM · Application: !6502Em

BB1 – Name: PhotoBase

BB0 – Name: BibleNts · Application: !HolyBible

BAF – Name: MacDisc · Description: Mac HFS disc image · Application: !MacFS

BA6 – Name: Excel · Description: Excel Spreadsheet · MIME Type: xls

- **BA1** – Name: SNA · Description: !Z80Em ZX Spectrum emulator snapshot · Application: !Z80Em · MIME Type: sna
- **BA0** – Name: Z80 · Description: !Z80Em ZX Spectrum emulator snapshot · Application: !Z80Em · MIME Type: z80

B9F – Name: FLI · Description: Video (also 301) · Application: !MovieFS

B9E – Name: Verstile · Description: Tiling drawfile

B9B – Name: DBasic

B9A – Name: DBasLib

B95 – Name: WimpBasic · Description: WimpBasic design files

B8C – Name: Split · Description: Part of a multi-part file · MIME Type: spt

- **B8B** – Name: TextEase · Description: TextEase document · Application: !TextEase
- **B8A** – Name: TxtEaseT · Description: TextEase document template · Application: !TextEase

B80 – Name: Slthzone

B7D – Name: TableCalc

B7C – Name: LifeForm · Description: MacroLife Lifeform

B7B – Name: Life · Description: MacroLife Life plane

B78 – Name: MIDIWrks · Description: MidiWorks score · Application: MidiWorks

B72 – Name: Compo

B72 – Name: Gamma · Description: Gamma correction data

B6F – Name: WTMModule · Description: Wacky-Talky module · Application: WTRun

B6E – Name: AddrList · Description: Desktop Hacker address list

B6A – Name: Illusion · Description: Illusions Disk Magazine

B69 – Name: MMaker · Description: Music Maker 1 music file

B64 – Name: C64_Info · Description: C64 music info

B62 – Name: Gopher · Description: Gopher command · Application: !Fresco

B61 – Name: XBitMap · Description: X window bitmap image · Application: !ImageFS · MIME Type: xbm image/x-xbitmap

B60 – Name: PNG · Description: Portable Network Graphics image · Application: !ImageFS · MIME Type: png image/png
B5C – Name: 3DModel · Description: !TopModel object · Application: !TopModel
B59 – Name: Page · Description: Bookmaker page file · Application: Bookmaker
B57 – Name: MyBook · Description: Bookmaker book file · Application: Bookmaker
B56 – Name: BubStyle · Description: Bubble Help style · Application: !BubbleHlp
B54 – Name: SOS
B50 – Name: RGSCOpts · Description: Choices for Picture Book 2 etc · Application: Picture Book 2
B4F – Name: Alphabet · Description: Picture Book 2 Alphabet · Application: Picture Book 2
B3D – Name: PagesTmp · Description: TextEase
B3C – Name: Pages · Description: TextEase pages
B3B – Name: SpellBnk · Description: !Spellbank user dictionary · Application: !Spellbank
B3A – Name: SpellFrq · Description: Words which have been looked up in !Spellbank · Application: !Spellbank
B31 – Name: FuzzySht? · Description: FuzzySheet · Application: !FuzzySht · MIME Type: fzs
B30 – Name: FuzzySht · Description: FuzzySheet · Application: !FuzzySht · MIME Type: fzs
B2F – Name: WinMeta · Description: WMF · Application: ImageFS · MIME Type: wmf
B2E – Name: CorelXch · Description: Corel Exchange drawing
B2D – Name: AdobeIll · Description: Adobe Illustrator drawing · MIME Type: ai
B2C – Name: OS2Met · Description: OS2 Metafile · Application: ImageFS
B2B – Name: CGMeta · Description: CGM image · Application: ImageFS
B2A – Name: DaVinci
B29 – Name: VRML · Description: Virtual Reality Modeling Language · Application: !TMViewer · MIME Type: vrml
B28 – Name: URL · Description: Uniform Resource Locator
 · **B27** – Name: OvnPro · Description: Ovation Pro document · Application: !OvnPro
 · **B26** – Name: OVPStyle · Description: Ovation Pro style sheet · Application: !OvnPro
 · **B25** – Name: OVDDL · Description: Ovation Pro document description · Application: !OvnPro
 · **B24** – Name: OVPScript · Description: Ovation Pro script · Application: !OvnPro
B23 – Name: X-File · Description: Directory image · Application: !XFiles
 · **B22** – Name: DFSImage · Description: !ImageDFS Image filing system for 5.25 DFS Disc Images" · Application: !ImageDFS, !6502Em
B21 – Name: TBAFSarc · Description: TBA Archive · Application: !TBAFSrd
B1E – Name: PsionPIC · Description: Psion picture
B16 – Name: ProOrder
B12 – Name: TTXpage · Description: Teletext+ page · Application: Teletext+
B11 – Name: TTXscript · Description: Teletext+ script file · Application: Teletext+
B0F – Name: CasetDATA
B09 – Name: ImageMap · Application: !ImageMap
B02 – Name: Meta · Description: MidiWorks Meta Event · Application: MidiWorks
 · **AFF** – Name: DrawFile · Application: !Draw · MIME Type: drw
 · **AFE** – Name: Mouse · Description: Mouse event record
 · **AFD** – Name: GCAL · Description: GCAL source, (text)
 · **AFC** – Name: GCODE · Description: GCODE intermediate
AFB – Name: PhonePad
 · **AFA** – Name: DtpStyle · Description: Acorn DTP style sheet
 · **AF9** – Name: DtpDoc · Description: Acorn DTP document
 · **AF8** – Name: 1stWord+ · Description: First Word Plus document
 · **AF7** – Name: HelpInfo · Description: Interactive help
 · **AF6** – Name: SimTrace · Description: ASim trace
 · **AF5** – Name: PostData · Description: Econet mail setup
 · **AF4** – Name: PostFile · Description: Econet mail filed
 · **AF3** – Name: PostBox · Description: Econet mail postbox
 · **AF2** – Name: PostTray · Description: Econet mail in-tray
 · **AF1** – Name: Music · Description: !Maestro music · Application: !Maestro
AF0 – Name: ARCWrite · Description: ArcWriter

AEF – Name: Sheet · Application: Schema
AEE – Name: SheetWS · Description: Schema macro · Application: Schema
AED – Name: SheetMac · Description: Schema configure · Application: Schema
AEC – Name: DBFile
AEB – Name: DBReport
AEA – Name: DBChart
 · **AE9** – Name: Alarms · Description: Alarm times · Application: !Alarm
AE8 – Name: SaveGame · Description: Saved game · Application: Darryl the Dragon, Flashback, High Risc Racing
 · **AE7** – Name: ARMovie · Description: Video · Application: !ARMovie
AE6 – Name: MSWord · Description: Word processor document · Application: !TechWriter
 · **AE4** – Name: Java · Description: Java application · Application: JavaVM · MIME Type: jar
AE3 – Name: Director · Description: Animation · Application: !MMPlayer
AE1 – Name: IndexFile
 · **ADF** – Name: PDF · Description: Portable Document Format · MIME Type: pdf application/pdf
ADE – Name: Safe · Application: !Safe
ADC – Name: DisCat
 · **ADB** – Name: New Font · Description: Outline Font
ADA – Name: Package
AD8 – Name: PsiMBM · Description: Psion Series 5
AD1 – Name: HQ-Inst
ACF – Name: vCard · Description: Personal information including addresses · Application: !Organizer · MIME Type: vcf
ACF – Name: QTM · Description: Music tracker. Does not seem to be used in latest QTM
ACE – Name: Wavelength
ACC – Name: ISO-9660 · Description: CD image · Application: CD-Blaze
ACA – Name: LaTeX · Description: LaTeX source (industry standard) · MIME Type: tex ltx latex application/x-latex
AC9 – Name: VF · Description: TeX virtual font · MIME Type: vf
AC4 – Name: H2Map · Description: Heroes of Might and Magic 2 (R-Comp Interactive) save · Application: !Heroes
AC3 – Name: H2MapX · Description: Heroes of Might and Magic 2 (R-Comp Interactive) save · Application: !Heroes
ABF – Name: Cabinet · Description: Archive · Application: !SparkFS · MIME Type: cab
 · **ABB** – Name: PlugIn · Description: Cerilica Vantage Extension · Application: !Vantage
ABA – Name: CPIO · Description: Archive · Application: !SparkFS
 · **AB9** – Name: Monitor · Description: Cerilica Monitor Calibration · Application: !Monitor
 · **AB7** – Name: PPD · Description: PostScript Printer Definition · MIME Type: ppd
 · **AAD** – Name: SVG · Description: Scalable Vector Graphics drawing · Application: !SVG · MIME Type: svg
AAC – Name: DialUp · Application: !NetFetch, !DialUp
AAB – Name: DialAcc · Application: !NetFetch, !DialUp
AAA – Name: RiscOS3? · Description: (Acorn)
AAA – Name: Compdraw
 · **AA8** – Name: pls · Description: SHOUTcast internet radio playlist · Application: !DigitalCD · MIME Type: pls audio/x-scpls application/pls
 · **AA7** – Name: M3UFile · Description: WinAmp mp3 URL list · Application: !DigitalCD · MIME Type: m3u vlc audio/mpegurl audio/x-mpegurl application/m3u
 · **AA6** – Name: TaskApp · Description: Absolute in a TaskWindow · Application: !InAWindow
 · **AA5** – Name: TskBasic · Description: BASIC in a TaskWindow · Application: !InAWindow
 · **AA0** – Name: MultiPlt · Description: Graph data · Application: !MultiPlot
 · **A9F** – Name: GraphDrw · Description: Graph data · Application: !GraphDraw
 · **A9E** – Name: ChartDrw · Description: Chart data · Application: !ChartDraw
 · **A9D** – Name: KeyTops · Description: Keyboard definition
A91 – Name: Zip · Description: Zip archive · Application: !SparkFS · MIME Type: zip
 · **A8F** – Name: AC3Audio · Description: AC3 raw audio · Application: !DigitalCD · MIME Type: ac3 audio/ac3
 · **A8D** – Name: VOB · Description: MPEG DVD video · MIME Type: vob video/x-vob
A81 – Name: ODT · Application: !TechWrite

A7F – Name: MSeXcelX · Description: Microsoft Excel document · Application: none · MIME Type: xlsx
A7E – Name: MSWordX · Description: Microsoft Word document · Application: !TechWrite · MIME Type: docx
 · **A79** – Name: CUE · Description: CDROM image CUE file · Application: CDFaker · MIME Type: cue application/x-cue
 · **A78** – Name: FzDraw · Description: Drawfile-packaged Fireworkz document · Application: !Fireworkz
 · **A77** – Name: PDChart · Description: Drawfile-packaged PipeDream chart · Application: !PipeDream
 · **A76** – Name: EPUB · Description: Electronic publication · MIME Type: application/epub+zip
 · **A70** – Name: SQLiteDB · Description: SQLite database
 · **A6F** – Name: AnqExp · Description: Anquet Export Format · Application: !SatNav
 · **A66** – Name: WebP · MIME Type: image/webp
 · **A65** – Name: JPEG2000
 · **A64** – Name: MP4 · MIME Type: application/mp4 video/mp4 audio/mp4
 · **A63** – Name: MKV
 · **A61** – Name: QOI · Description: Quite OK Image · Application: !Qoi2Spr · MIME Type: qoi image/x-qoi
A06 – Name: Perl · Description: Perl script (also 102) · MIME Type: pl
A05 – Name: WHelp
 · **9FB** – Name: Basalt · Description: BASIC program translated to run using Basalt · Application: Basalt module
9F8 – Name: HD_BackJ · Application: !HD_Backup
 · **9F0** – Name: Money · Description: Never circulated · Application: !Money
 · **9EB** – Name: Index · Description: No longer used
999 – Name: Spectrum
907 – Name: Goop · Description: Scores for Goop game
900 – Name: Universal
8FF – Name: DisasMap · Description: Disassembler · Application: !Disas
 · **81C** – Name: ICM · Description: ICC Colour Profile · MIME Type: icc
815 – Name: SKoords · Application: !StarMan
810 – Name: DrawLib · Description: Draw Library
 · **80C** – Name: StatPad · Description: Uniform stationary pad
 · **80B** – Name: VideoTex · Description: Uniform Videotex
 · **80A** – Name: DBForm · Description: Uniform database form
 · **809** – Name: DataBase · Description: Uniform database
 · **808** – Name: UniPS · Description: Uniform PostScript
808 – Name: EPS · Description: !EPSObject · Application: !EPSObject · MIME Type: eps application/postscript
 · **807** – Name: GrphChrt · Description: Uniform graphs & charts
806 – Name: PointEd
 · **806** – Name: Graphics · Description: Uniform graphics
 · **805** – Name: Drawing · Description: Uniform drawing
 · **804** – Name: Picture · Description: Uniform picture
 · **803** – Name: Spreadsh · Description: Uniform spreadsheet
 · **802** – Name: UnifText · Description: Uniform text only
 · **801** – Name: Wordproc · Description: Uniform wordprocessor
 · **800** – Name: Uniform · Description: BBC Uniform File
800 – Name: Inlay · Description: Tape inlay
800 – Description: Demon's Lair save · Application: !DemonLair
 · **7FF** – Name: WOFFfont · Description: WOFF font · MIME Type: woff
7FF – Name: SUVec · Description: !SU Vector file · Application: !SU
7FF – Name: SCRUtils
7FF – Name: Psion · Description: Psion
7FF – Name: FS Options · Description: (Copy options)
7FF – Name: Flare
7FE – Name: UUEncode · Description: !U2 · Application: !U2 · MIME Type: uu uue
7FE – Name: SUCoord · Description: !SU Coordinate file · Application: !SU
7FE – Name: CMOS · Description: Cmos_Edit saved CMOS RAM

7FD – Name: SUNames · Description: !SU cave survey names file · Application: !SU
7FD – Name: Menus
7FD – Name: Encoded
7F6 – Name: PlotFile
7F6 – Name: MultPlot · Application: MultiPlot
7F5 – Name: PwrBase · Application: PowerBase
7F4 – Name: PwrBase · Application: PowerBase
7F3 – Name: GrphFile · Application: GraphDraw
7F3 – Name: PwrBase · Application: PowerBase
7F2 – Name: PowerBase · Description: PowerBase database · Application: PowerBase
7F2 – Name: POSFile · Description: !Betts POS file · Application: !Betts · MIME Type: pos
7F2 – Name: KeyStrip · Description: Key strip
7F1 – Name: SVYFile · Description: !Betts SVY file · Application: !Betts · MIME Type: svy
7F1 – Name: PwrBase · Description: PowerBase valid · Application: PowerBase
7F1 – Name: ChrtFile · Description: ChartDraw
7F0 – Name: PwrBasel · Description: PowerBase index · Application: PowerBase
7F0 – Name: CaveData · Description: !Betts source data file · Application: !Betts
7EF – Name: Morph
7EF – Name: MailFile · Description: Mail browser
7DC – Name: HelpFile · Description: PowerBase help · Application: PowerBase
7B7 – Name: ST Song · Description: Soundtracker
7B4 – Name: IFF Samp · Description: Same as FC2
7B3 – Name: ST Patt · Description: Soundtracker
7AF – Name: Pointer
7A3 – Name: DCatFile · Description: Disc catalogue
7A0 – Name: Timestep · Description: TimeStep satellite image
787 – Name: SysEx
 · **77F** – Name: TTF Font · Description: TrueType font · MIME Type: ttf
77F – Name: HypHelp
77E – Name: CmpHHlp
77D – Name: HHBookM
777 – Name: GraphDat · Description: 3-D Graph
777 – Name: GraphDat · Description: Graph
770 – Name: MIDISysX · Description: Raw MIDI System Exclusive data (first byte = &F0, last byte = &F7) · MIME Type: syx
750 – Name: Diction · Description: Articulate
745 – Name: WireWorld
744 – Name: CoreWars · Description: Core Wars
733 – Name: FracTrace
725 – Name: extASM · Description: Assembler source
721 – Name: HP11_Prog
720 – Name: CardFile · Description: CardIndex database
711 – Name: TaskList · Description: StartTask task list
710 – Name: QTM
710 – Name: !Packer
70A – Name: WeathrYr · Description: WeatherYear
707 – Name: DDF
703 – Name: Speculatr
702 – Name: Spectrum · Description: Spectrum
702 – Name: DissMdle · Description: !Dissi Module file · Application: !Dissi
701 – Name: Speculator
701 – Name: ProTrack · Description: !ProTrack Protracker music · Application: !ProTrack
701 – Name: KeyDefs · Description: Function keys

701 – Name: Atoms · Application: !Mole
701 – Name: Archway2
700 – Name: Speculator
700 – Name: Mole · Application: !Mole
700 – Name: DissiCtr · Description: !Dissi disassembler control file · Application: !Dissi
700 – Name: ChessGame · Description: Saved !Chess Game · Application: !Chess
700 – Name: Archway2
700 – Name: Album · Description: Album
6FF – Name: MbrotPar · Description: Mandel 4
 · **6B4** – Name: TTFile · Description: LTGame (never circulated) · Application: !LT
 · **6B3** – Name: JnyFile · Description: LTGame (never circulated) · Application: !LT
 · **6B2** – Name: ArryFile · Description: LTGame (never circulated) · Application: !LT
 · **6B1** – Name: DataFile · Description: LTGame (never circulated) · Application: !LT
6A6 – Name: WPGraphic · Description: WordPerfect Graphic · Application: ImageFS
 · **6A5** – Name: TABFile
 · **6A5** – Name: ICO · Description: Use 132 instead · Application: !ImageFS · MIME Type: ico
 · **6A5** – Name: CUR · Description: Windows 3.0 cursor definition · Application: !ChangeFSI · MIME Type: cur
6A4 – Name: HawkV9 · Description: Wild Vision Hawk V9 digitiser · Application: !Translatr
6A3 – Name: FITS · Description: Flexible Image Transport System image · Application: !Translatr
6A2 – Name: ColoRIX · Description: RIX Softworks image · Application: !Translatr
6A1 – Name: RLE · Description: Unix RLE image · Application: !Translatr · MIME Type: rle
6A0 – Name: MSX2 · Description: MSX-2 image · Application: !Translatr
69F – Name: ZVDA · Description: Zeridajh video digitiser animation · Application: !Translatr
69E – Name: PBMPlus · Description: Portable Bit Map Plus toolkit/portable pixmap · Application: !Translatr · MIME Type: ppm
 image/x-portable-pixmap
69D – Name: Targa · Description: Truevision Targa · Application: !Translatr
 · **69C** – Name: BMP · Description: MS Windows bitmap · Application: !ChangeFSI · MIME Type: bmp
69B – Name: Irlam · Description: Irlam Instruments scanner image · Application: !Translatr
69A – Name: CadSoft · Description: Prisma · Application: !Translatr
699 – Name: MTV · Description: MTV ray tracer picture · Application: !Translatr · MIME Type: mtv
698 – Name: QRT · Description: uick Ray Tracer picture · Application: !Translatr · MIME Type: qrt
697 – Name: PCX · Description: PCX picture · Application: !Translatr · MIME Type: pcx
696 – Name: Pineappl · Description: Pineapple digitised colour image · Application: !Translatr
695 – Name: GIF · Description: Graphics Interchange Format · Application: !ChangeFSI · MIME Type: gif image/gif
694 – Name: MacPaint · Description: Macintosh MacPaint bitmap · Application: !Translatr
693 – Name: AmigaIFF · Description: Amiga IFF (ILBM) · Application: !Translatr · MIME Type: iff
692 – Name: Img · Description: Atari IMG image · Application: !Translatr · MIME Type: img
691 – Name: Degas · Description: Atari Degas image PI1,PI2,PI3,PC1,PC2,PC3 · Application: !Translatr
690 – Name: Clear · Description: Clear image · Application: !Translatr
68E – Name: PackdDir · Description: Packdir compressed archive · Application: !PackDir
681 – Name: NDX
680 – Name: DBaseIV · Description: Database
666 – Name: ReadFont
666 – Name: Read · Description: CGM
666 – Name: AmazeLev
664 – Name: Song · Description: Music track · Application: !Coconizer
657 – Name: BreederCA
656 – Name: ? · Description: Inspiration
655 – Name: ? · Description: Inspiration
654 – Name: ? · Description: Inspiration
653 – Name: ? · Description: Inspiration
652 – Name: ? · Description: Inspiration

651 – Name: ? · Description: Inspiration
639 – Name: XMFSprite · Description: XMF painting sprite
635 – Name: Harmonica
634 – Name: Effex
633 – Name: RawSampl
632 – Name: SgnSampl
632 – Name: RIFFwave · Description: SoundCon sound sample → FB1
631 – Name: LogSampl
604 – Name: NetWork
601 – Name: CCIR601 · Description: 720×288 or 720×243 YUV picture
600 – Name: Mandel · Description: !MandlePlot compressed sprite · Application: !MandlePlot
600 – Name: LoggaLog
5FF – Name: OnHand · Description: On Hand
5FE – Name: Flood? · Description: Flood game saved levels
5F4 – Name: SparkScr · Description: Spark command script · Application: !SparkPlug
5F4 – Name: Fractal
5C4 – Name: Sculptor?
5C2 – Name: Sculptor?
5C1 – Name: Sculptor?
5C0 – Name: Sculptor
560 – Name: DrghtsGame
556 – Name: Archrome
555 – Name: DiskBase · Description: DiskBase
554 – Name: 3View
553 – Name: DrawLib · Description: Vector object library · Application: !DrawPlus
550 – Name: Musix
54F – Name: Delta · Description: Delta film
515 – Name: AMFEdit16
510 – Name: STracker · Description: Music track · Application: !Stracker
500 – Name: LQFont · Description: LQ500 printer downloadable font
500 – Name: World · Description: WorldScape Save Game · Application: !wscape
4FF – Name: QGraphx · Description: QGraphics
4DD – Name: DAs_Pict
4BD – Name: PolySaw
4AD – Name: PhonePad · Description: PhonePad
474 – Name: Sticky · Description: !Sticky backdrop · Application: !Sticky
450 – Name: Configs
449 – Name: G_Proj · Description: Project for Mark Jones's Haskell dialect · MIME Type: gp
448 – Name: G_Script · Description: Script for Mark Jones's Haskell dialect · MIME Type: gs
444 – Name: STScript
43B – Name: Scroll_W · Description: Archway2
439 – Name: Menu_W · Description: Archway2
437 – Name: View_W · Description: Archway2
435 – Name: Tools_W · Description: Archway2
433 – Name: Dialog_W · Description: Archway2
431 – Name: Form_W · Description: Archway2
42F – Name: DisplayW · Description: Archway2
425 – Name: PDTSetup
422 – Name: Fonts · Description: Archway2
420 – Name: PrintSet · Description: Printer settings
420 – Name: Custom · Description: Archway2
419 – Name: Options · Description: Copy/Wipe Filer options

418 – Name: VStrings · Description: Archway2
416 – Name: Messages · Description: Archway2
414 – Name: WClasses · Description: Archway2
413 – Name: SpoolBuf
412 – Name: Win_Met
412 – Name: TelPhone
412 – Name: IClasses · Description: Archway2
411 – Name: TaskList · Description: StartTask
411 – Name: EncPS
410 – Name: OS2_Met · Description: OS2 MetaFile
410 – Name: GIF · Description: do not use → 695
410 – Name: Code · Description: Archway2
410 – Name: AlphaBase
40E – Name: TempestDTP · Description: Tempest DTP
40E – Name: Leader
40E – Name: CharSets · Description: Archway2
40D – Name: TempestDTP · Description: Tempest DTP
40C – Name: LinePats · Description: Archway2
40C – Name: TempestDTP · Description: Tempest DTP
40B – Name: Archway2 · Description: Archway2
40B – Name: DTPFlow · Description: Tempest DTP
40A – Name: DTPStyle · Description: Tempest DTP
40A – Name: FillPats · Description: Archway2
409 – Name: Backup · Description: Backup3
409 – Name: DTPStylesht Tempest DTP
408 – Name: AOutline
408 – Name: DTPDocument Tempest DTP
408 – Name: Pointers · Description: Archway2
407 – Name: Choices · Description: Archway2
407 – Name: Label
406 – Name: Account
406 – Name: Key · Description: Archway2
405 – Name: MetaFile · Description: Computer Graphics Metafile CGM/GMF · MIME Type: cgm
405 – Name: TempstDTP · Description: Tempest DTP
404 – Name: GWMulti · Description: GraphicWriter
403 – Name: GWVDU · Description: GraphicWriter
402 – Name: GwMaster · Description: GraphicWriter Menus
401 – Name: Accounts
401 – Name: GWGraph · Description: GraphicWriter
400 – Name: GuardMod · Description: !Lifeguard screen-saver effects module
400 – Name: GwText · Description: GraphicWriter
400 – Name: LifeGuard
3FB – Name: ArcFSArc · Description: !ArcFS archive
3F8 – Name: FontMGR
3F7 – Name: GamePos
3F2 – Name: ArmSI · Description: ARM speed indicator
3F1 – Name: MemDisc
3ED – Name: FreVocab
 · 3EB – Name: Liberatr · Description: Liberator talker memory dump · Application: !MTI
 · 3E2 – Name: VistTune · Description: Vistamusic Tune
3E0 – Name: HQ-Data
3DC – Name: ARJArc · Description: ARJview

3D6 – Name: StrngHlp · Description: StrongHelp help image · Application: !StrongHlp
3C7 – Name: FEMSFile
3C1 – Name: TwEffect
3BF – Name: ArcFS · Description: !ArcFS archive · Application: !ArcFS
3B8 – Name: MLCS
380 – Name: HBScript · Description: Backup
371 – Name: RGB · Description: RGB picture
371 – Name: RayShade · Description: ditto
365 – Name: CocoTrak
364 – Name: CocoSong
360 – Name: SlideShow · Description: SlideShow sequence
351 – Name: Sequence
350 – Name: Squished
345 – Name: FaxText
333 – Name: Patience
326 – Name: BackOpts
325 – Name: BackData
325 – Name: PDTSetup
321 – Name: Menus · Description: Menu designer
31B – Name: DXen · Description: Music track
309 – Name: OrdSheet
301 – Name: XPixMap
 · **301** – Name: FLI · Description: use B9F · Application: !XPM
300 – Name: TimeStep · Description: Satellite image
2FF – Name: SciGraph
2F8 – Name: Z80 MOS · Description: !Z80Tube MOS · Application: !Z80Tube · MIME Type: mos
2DF – Name: D2Font · Description: Draw2Font file · Application: !D2Font
2BF – Name: Tile_F
2BE – Name: Embrdr_F
2BD – Name: Stitch_F
2BC – Name: Patt_F
2B8 – Name: Wtpatrn
2B4 – Name: Squeezed
2B3 – Name: Squeezed
2B2 – Name: Jigsaw
2B0 – Name: Jigsaw
297 – Name: Comprssd
28A – Name: Edit+ · Description: !Edit+ · Application: !Edit+
289 – Name: P Z88 · Description: Printed
288 – Name: N Z88 · Description: Normal Z88
280 – Name: Z80 · Description: !Z80Tube general Z80 snapshot · Application: !Z80Tube · MIME Type: z80
259 – Name: Server · Description: Server side image map · Application: !ImageMap
258 – Name: Dancer
234 – Name: Thoughts
233 – Name: Event · Description: The Time Machine (The Fourth Dimension) save · Application: !Time_Mach
230 – Name: Ghostly · Description: Haunted House (The Fourth Dimension) save · Application: !Haunted
226 – Name: CompFile
225 – Name: VGGame
224 – Name: VGStatus
223 – Name: CompHole
222 – Name: DateMarker
214 – Name: WolfData · Description: Wolfenstein 3-D data file · Application: Wolf

- 204** – Name: Typeface · Description: Italicer Typeface
- 202** – Name: Song · Description: Mascom2
- 201** – Name: Z80_COM · Description: !Z80Tube .COM file · Application: !Z80Tube · MIME Type: com
- 200** – Name: CAIFSArc
- 200** – Name: CF_Circ
- 200** – Name: GraphSrc
- 1FF** – Name: OldSong · Description: Mascom2
- 1FF** – Name: TData · Description: Tiger
- 1FF** – Name: Tiger
- **1F2** – Name: Puzzle · Description: Saved Game from Simon Tatham's Portable Puzzle Collection · Application: !Puzzles
 - **1F1** – Name: MakroSet · Description: Makro
- 1EC** – Name: Crypt · Description: Crypt
- **1E4** – Name: SeasonSG · Description: Saved game · Application: Seasonal
 - **1E3** – Name: GPX · Description: Route data e.g. GPS · Application: !RiscOSM
 - **1DF** – Name: DiscCat · Description: Disc catalogue data · Application: !Cat
- 1DF** – Name: Objects
- **1DA** – Name: GPIOConf · Description: GPIO configuration data · Application: !GPIOConfig
 - **1D9** – Name: Syncjob · Description: Synnchronisation details · Application: !SyncDiscs
 - **1D8** – Name: PrintPDF · Description: Bookmark and hotspot data for the PrintPDF application · Application: !PrintPDF
 - **1D5** – Name: vCalendr · Description: Calendar and Diary information · Application: !vCalRead, !Organizer · MIME Type: ical ics ifb icalendar
- 1D5** – Name: ? · Application: Was used by !Organizer for vCard. Now 1D5
- 1CF** – Name: Conjugez
- **1CF** – Name: FLAC · Description: Free Lossless Audio Codec · Application: !DigitalCD · MIME Type: flac audio/flac audio/x-flac
 - **1CA** – Name: CashBook · Description: Filetype for the CashBook personal accounts package · Application: !CashBook
- 1CA** – Name: HotKeys
- 1C8** – Name: Symlink · Description: Symbolic link
- **1C7** – Name: Basic8 · Description: Tokenised BBC BASIC in Z80/80×86/Windows format, also known as "Russell format" · MIME Type: bbc
 - **1C6** – Name: 32016 · Description: 32016 code · Application: !32Tube · MIME Type: n32
 - **1C5** – Name: PDP-11 · Description: PDP-11 code · Application: !PDPTube · MIME Type: pdp
- 1C1** – Name: JSW_Data · Description: Jet Set Willy room, scenery, or animation · Application: !JSW
- 1C0** – Name: JSW_Game · Description: Jet Set Willy Game file · Application: !JSW
- 1BC** – Name: Imagen · Application: !ImageMap
- **1B9** – Name: TZX · Description: Advanced Tape file format for use with spectrum emulators · Application: !Z80Em
 - **1B8** – Name: VOC · Description: Standard .VOC file format · Application: !Z80Em
- 1B7** – Name: ArmBob · Description: Filetype for ArmBob files · Application: ArmBob
- **1B5** – Name: DirSync · Description: DirSync project file · Application: !DirSync
 - **1B1** – Name: PCTk · Description: Generic PC music track · Application: !DigitalCD · MIME Type: afm dbm digi dsm dtm far mtm ptm ult 669
 - **1B0** – Name: ScreamTk · Description: ScreamTracker music track · Application: !DigitalCD · MIME Type: s3m stm
 - **1AF** – Name: FstTk · Description: FastTracker2 music track · Application: !DigitalCD · MIME Type: xm
 - **1AE** – Name: ImpulsTk · Description: ImpulseTracker music track · Application: !DigitalCD · MIME Type: it
 - **1AD** – Name: AMPEG · Description: Audio MPEG · MIME Type: mp2 mpga mp3 audio/mpeg audio/x-mpeg audio/mp2 audio/mp3 audio/x-mp3
- 1AD** – Name: Protector
- 1AC** – Name: Project
- 1AB** – Name: Address · Description: Address book
- 1AA** – Name: CSSC · Description: CSSC
- **1A8** – Name: Vorbis · Description: Ogg Vorbis audio · Application: !DigitalCD · MIME Type: ogg oga audio/ogg audio/x-ogg audio/x-vorbis audio/x-vorbis+ogg application/ogg application/x-ogg
 - **1A1** – Name: Locate · Description: Search definitions/results for Locate file-finder · Application: !Locate
- 1A0** – Name: TSAC

- 199 – Name: Encode4 · Description: Encoder input
- 199 – Name: Encoded1
- 198 – Name: Encode4 · Description: Encoder output
- 198 – Name: Encoded2
 - 18D – Name: TaskLua · Description: lua script or compiled object in a TaskWindow
 - 18C – Name: Lua · Description: lua script or compiled object file · Application: !Lua · MIME Type: lua
- 188 – Name: Flash · Description: Macromedia flash file · Application: !Flash
- 187 – Name: awk · Description: awk scrip
- 180 – Name: Copier180
- 180 – Name: Mail · Description: TransArc
 - 17F – Name: VideoLst · Description: Video database · Application: !VideoList
- 17C – Name: Quetzal · Application: Zip2000 · MIME Type: sav
- 17B – Name: Blorb · Application: Zip2000 · MIME Type: blb blorb
- 172 – Name: Copier172
- 16D – Name: Organizer · Description: Addresses, appointments and alarms · Application: !Organizer
- 16C – Name: DoomWad · Description: Doom lwad/Pwad · Application: !Doom, !Deth · MIME Type: wad
- 164 – Name: C64-Disc · Description: C64 archive · MIME Type: D64
- 164 – Name: Copier164
- 160 – Name: Copier160
- 15C – Name: PsiAgnda · Description: Psion Series 5/7 diary
- 15B – Name: PsiData · Description: Psion Series 5/7 database
- 15A – Name: PsiOPL · Description: Psion Series 5/7 program
- 159 – Name: PsiRecrd · Description: Psion Series 5/7 audio
- 158 – Name: Psion · Description: Psion Series 5/7
- 158 – Name: PlylstCD · Application: !RTracker
- 157 – Name: Playlist · Application: !RTracker
- 157 – Name: PsiSheet · Description: Psion Series 5/7 spreadsheet
- 156 – Name: PsiWord · Description: Psion Series 5/7 document
- 156 – Name: Squished
 - 155 – Name: FileList · Description: Playlist · Application: !DigitalCD
- 154 – Name: SR2000
- 153 – Name: Cabri · Description: !Cabriolet (dynamic geometry) · Application: !Cabriolet
- 152 – Name: PolyData · Description: Polyhedron definitions for !PolyDraw, !PolyNet · Application: !PolyDraw, !PolyNet
- 144 – Name: Copier144
- 13D – Name: TankSave · Description: T.A.N.K.S. (Werewolf Software) save · Application: !Tanks
- 13C – Name: JFPatch · Description: Assembler patch file
- 139 – Name: 3DScene
- 138 – Name: MapprMap · Description: !Mapper_O 2D map with overlays · Application: !Mapper
 - 132 – Name: ICO · Description: Windows 3.0 icon collection · Application: !ImgMaster, !ChangeFSI · MIME Type: ico
- 130 – Name: AntiGrav
- 12B – Name: RasMolSc · Application: !RasMol
- 12A – Name: PDB · Application: !RasMol
- 129 – Name: Alchemy · Application: !RasMol
- 128 – Name: MOL · Application: !RasMol
- 128 – Name: Pifile
- 127 – Name: MOPAC · Application: !RasMol
- 123 – Name: ALPS · Description: (Alpine Software)
- 123 – Name: Creator
- 122 – Name: Genetic · Description: Genetic Algorithm data · Application: !MakeCreat, !Path, !GAAsm, !ViewCreat, !GALoader
- 120 – Name: ANSI · Description: Ansi2Text
 - 11D – Name: TAP · Description: Compressed 1-bit sound for emulators · Application: !Z80Em · MIME Type: tap
- 11A – Name: Z-Code · Description: Infocom adventure game · Application: Zip2000 · MIME Type: z1-z8

118 – Name: Angband · Description: Angband saved game file
116 – Name: DemoMolec · Application: !Modeller
113 – Name: FastSpr
113 – Name: MoleAnim · Application: !Modeller
113 – Name: Proj Man
112 – Name: MatlabFn · Description: MatLab
112 – Name: Stasis
112 – Name: Teletext · Description: Teletext frame dump · Application: !Teletext
111 – Name: MatlabSc · Description: MatLab
111 – Name: MSDOS · Description: !Spark/ArcFS RS DOS/PC PKARC compacted archive contents · Application: !Spark, ArcFS
111 – Name: RSDOS · Description: RS DOS
110 – Name: RLaB · Application: !RLaB
10F – Name: CLIcon · Application: !IconCLI
10C – Name: Keyboard
10B – Name: Symphony
10A – Name: SySample
109 – Name: Glass
109 – Name: Glazier
108 – Name: DataVox · Description: SysBeep sound file with internal chunk structure · Application: DataPlay
107 – Name: OreDBase · Description: Oregon Database – Archivist Pro
103 – Name: UniMode · Description: VIDC regs and ModeVariables · Application: !UniMode2
102 – Name: Perl · Description: Perl script (also A06) · Application: !Perl · MIME Type: pl
102 – Name: SpellDict
101 – Name: Calcsprd · Description: Calc spreadsheet
101 – Name: Phone
100 – Name: 1stsheet · Description: 1st Statistics package
100 – Name: BackDrop
0FF – Name: Diss
0FF – Name: FineArt
0FF – Name: F-Paper
0FF – Name: Used · Description: File marked as used · Application: !Used
0FE – Name: F-Pal
0FE – Name: Psion · Description: Psion Organiser II file
0FD – Name: F-Shapes
0FD – Name: Probase
0FD – Name: Z88Link archives
0FC – Name: Ray3d
0FB – Name: GEK Data · Description: GEK games
0FB – Name: RayLib
0FA – Name: RayMacro
0F8 – Name: HelpFile
0F8 – Name: MPEG · Description: Video
 · **0F5** – Name: EPSI · Description: Encapsulated PostScript with Thumbnail · MIME Type: epsi
 · **0ED** – Name: SprColor · Description: Compact colour palette · Application: !Palette
0E3 – Name: SSSheet · Application: !SSS Spreadsheet
0DE – Name: BackMenu · Description: BackMenu
0DC – Name: Keystrip
0D8 – Name: GBSnap · Description: Gameboy emulator snapshot
0D7 – Name: Gameboy · Description: Gameboy emulator ROM-image
0D1 – Name: AddrList
 · **0CF** – Name: Options
0CE – Name: CDData · Application: AudioFS2

0CD – Name: CDTrack · Application: AudioFS2
0CC – Name: Card · Application: !Card2
0CC – Name: HellFile
 · **0CA** – Name: Calendar · Description: Appointments
0C8 – Name: Pi · Description: Anime picture · Application: !Megumi
0C6 – Name: Pic · Description: Anime picture · Application: !Megumi
0C5 – Name: ECC Anim · Description: Anime picture · Application: !Megumi
0C4 – Name: MAG · Description: Anime picture · Application: !Megumi
0BC – Name: PCRun · Description: PC application startup
0BB – Name: BBCRun · Description: BBC application startup
0BB – Name: MegaBoard
 · **0BA** – Name: TaskBAS · Description: BASIC in a taskwindow (use AA5)
0B2 – Name: SoftList
0AD – Name: Split · Description: Subsection of large file · Application: !Split
0AC – Name: DiscBox
0AB – Name: PPPTrack
0AA – Name: Indexer
099 – Name: Puzzle
084 – Name: PIC16C84 · Description: Mike Cook's PIC Programmer
083 – Name: TauStyle · Description: Tau
082 – Name: TauFile · Description: Tau
 · **07F** – Name: OTF Font · Description: OpenType font · MIME Type: otf
077 – Name: FTPc · Application: !FTPc
073 – Name: Sticky
071 – Name: AVI · Description: Audiovisual (See also FB2) · MIME Type: avi
070 – Name: QuakePAK · Description: Id Quake archive
06C – Name: TADS · Application: TADS · MIME Type: gam
069 – Name: BigMArt · Description: BigMArt15
068 – Name: BigMArt · Description: BigMArt15
068 – Name: Z-Code8 · Description: Infocom adventure game (version 8 Z-Code program) · Application: Zip2000 · MIME Type: z8
067 – Name: Z-Code7 · Description: Infocom adventure game (version 7 Z-Code program) · Application: Zip2000 · MIME Type: z7
066 – Name: Z-Code6 · Description: Infocom adventure game (version 6 Z-Code program) · Application: Zip2000 · MIME Type: z6
 · **065** – Name: XPS · Description: XML Paper Specification document · MIME Type: xps
065 – Name: Z-Code5 · Description: Infocom adventure game (version 5 Z-Code program) · Application: Zip2000 · MIME Type: z5
064 – Name: C64 · Description: C64 executable
064 – Name: Z-Code4 · Description: Infocom adventure game (version 4 Z-Code program) · Application: Zip2000 · MIME Type: z4
063 – Name: SIDSong · Description: Playsid music track
063 – Name: Z-Code3 · Description: Infocom adventure game (version 3 Z-Code program) · Application: Zip2000 · MIME Type: z3
062 – Name: Z-Code2 · Description: Infocom adventure game (version 2 Z-Code program) · Application: Zip2000 · MIME Type: z2
061 – Name: Z-Code1 · Description: Infocom adventure game (version 1 Z-Code program) · Application: Zip2000 · MIME Type: z1
050 – Name: 3DPlanes
050 – Name: Filer
050 – Name: FracDef
046 – Name: ColorMap · Application: !FracPaint
045 – Name: FracPars · Application: !FracPaint
044 – Name: Spray
042 – Name: SimpleDTP
042 – Name: Widget5 · Description: !Widget5 512×512 greyscale image · Application: !Widget5
041 – Name: Widget4 · Description: !Widget5 256×256 greyscale image · Application: !Widget5
041 – Name: Resource
040 – Name: Wordsrch
03C – Name: WordSq · Description: !WordSq creation and solving · Application: !WordSq

03B – Name: GEK Pack · Description: GEK games
022 – Name: Niall II
022 – Name: PGP · Description: PGP signed file
021 – Name: compress
017 – Name: PlanF
016 – Name: PlnPg
012 – Name: AIM · Description: Archimedes Image Manager Configuration
012 – Name: AIM Cfg
011 – Name: AIM · Description: Archimedes Image Manager
011 – Name: HelpText
010 – Name: AIM+ · Description: Archimedes Image Manager Plus Image header
010 – Name: AIMInfo · Description: ditto
010 – Name: Mailabl
00F – Name: Pottery
00F – Name: UntypedS · Description: Sound sample · Application: DataPlay
00E – Name: U2Enc · Description: !U2 see also 7FE · Application: !U2
00D – Name: U2Dec · Description: !U2 (obsolete) · Application: !U2
00C – Name: !Circuit
00B – Name: MegaBoard
00B – Name: !Armony
00B – Name: Matrix · Description: Matrix tracker music track
00A – Name: IcnStorm
00A – Name: Armony
008 – Name: GolfDemo
007 – Name: LCrypted
006 – Name: ScotSave
006 – Name: PathFile
006 – Name: WildV12 · Description: WildVision 512×512×8b grey picture
005 – Name: ScottGame
004 – Name: AOutline · Description: ArcOutline
004 – Name: AIM · Description: Archimedes Image Manager greyscale · Application: !Translatr
003 – Name: ArmSI · Description: Speed indicator · Application: !ARMSI
002 – Name: STRacker · Description: !STRacker tune · Application: !Stracker · MIME Type: mod
002 – Name: SampDump · Description: Vivaldi compressed sound sample file
002 – Name: PlayMod
001 – Name: STRacker · Description: !STRacker tune · Application: !Stracker · MIME Type: mod
001 – Name: Castles · Description: Castles
000 – Name: MusDump · Description: Vivaldi compressed music file
000 – Name: Crunched · Description: Cruncher

Originally, filetypes were separated into ranges, though the commercial/non-commercial distinction is no longer meaningful:

Range / Description

E00-FFF – Description: System

B00-DFF – Description: Commercial

A00-AFF – Description: Acornsoft

400-9FF – Description: Commercial

100-3FF – Description: Non-commercial distributed

000-0FF – Description: Personal undistributed

Other languages documentation

Français

[RISC OS Open source page](#)

FileCore FS Descriptor Block

FileCore

Offset / Contents

- +0** – Contents: Disc flags
- +3** – Contents: Filing system number
- +4** – Contents: Offset of filing system title from module base
- +8** – Contents: Offset of boot text from module base
- +12** – Contents: Offset of low level Filing System DiscOp Entry from module base
- +16** – Contents: Offset of low level Filing System MiscOp Entry from module base

Disc Flags

- 0** – Meaning when set: Hard discs need FIQ
- 1** – Meaning when set: Floppies need FIQ
- 2** – Meaning when set: Reserved must be zero
- 3** – Meaning when set: Use only scratch space when a temporary buffer is needed
- 4** – Meaning when set: Hard discs support mount like floppies do (i.e. they fill in sector size, heads, sectors per track and density)
- 5** – Meaning when set: Hard discs support poll change (i.e. the poll change call works for hard discs and returns a sensible value; also locking them gives a sensible result)
- 6** – Meaning when set: Floppies support power eject
- 7** – Meaning when set: Hard discs support power eject
- 8** – Meaning when set: Read drive flags is supported
- 9** – Meaning when set: Sector addressing is possible (RISC OS 3.60 and later)
- 10** – Meaning when set: FileCore module returns new-style errors
- 11** – Meaning when set: Mount floppies like fixed discs are

[RISC OS Open source page](#)

FileCore Timeline

See big maps below. Now FileCore can handle an idlen of up to 21 bits, meaning even smaller objects are possible, relative to total partition size.

Feature / FileCore version / OS version / Expansion in

Even bigger maps · 3.75 – OS version: 5.24 · Expansion in: FileCore format

2K and 4K sectors · 3.63 – OS version: 5.22 · Expansion in: Disc handling and Filecore format

4GB-1 file size · 3.56 – OS version: 5.20 · Expansion in: FileCore format

64-bit byte addressing for user API · 3.31 – OS version: 5.03 · Expansion in: Disc handling

Big directories and big maps · 3.21 – OS version: 3.80 · Expansion in: FileCore format

Big disc sizes and sector addressing · 2.91 – OS version: 3.60 · Expansion in: Disc handling and FileCore format

Big sectors

Allow sectors to be 2K and 4K bytes (previously could be 256, 512 or 1024 bytes). Since device drivers use 29-bit sector addressing for the largest discs, this means the total addressable space increases from

$2^{29} * 1024 = 512 \text{ GB}$

to

$2^{29} * 4096 = 2 \text{ TB}$

Big files

Previously a file could have a maximum size of $2^{31} - 1$ bytes ("2GB - 1"). Now it can have a size of $2^{32} - 1$ bytes ("4GB - 1")

64-bit byte addressing

The user API introduces the DiscOp command set FileCore_DiscOp64. These take exactly the same parameters as FileCore_DiscOp except rather than a 32-bit disc address (3-bit drive number and 29-bit byte address) it now takes a 32-bit pointer to a three-word structure of drive number and 64-bit byte address. This does not imply that the whole address space can be used in practice, since there has been no change to the internal interface between FileCore and device driver; however using this SWI ensures forward compatibility.

Big directories and big maps

It now becomes possible to exceed the 77 object limit in a directory, and filename length goes from 10 characters to 255 characters.

The maximum possible idlen in the format goes from 16 bits to 19 bits. This means disc objects can be smaller than before, for a given partition size.

[RISC OS Open source page](#)

FileCore_DiscOp64 Address

FileCore

The disc address format used with FileCore_DiscOp64 is different to the disc address format used by FileCore_DiscOp.

Address

Word / Bit / Meaning

0 – Bit: 0 – 7 · Meaning: Drive number

8 – 31 · Reserved, must be zero

1 – Meaning: Byte address (low word)

2 – Meaning: Byte address (high word)

Notes

This is the format used with FileCore_DiscOp64.

[RISC OS Open source page](#)

FileCore_Features Mask

FileCore

Mask

Bits /

0 – : FileCore supports new-style scheme

1 – : FileCore supports mounting floppies like fixed discs

2 – 31 – : Clear (reserved for future expansion)

[RISC OS Open source page](#)

FileCore_FloppyStructure Flags

FileCore

Flags

Bits / Meaning when set

6 – Meaning when set: Old map

7 – Meaning when set: Old directory structure

[RISC OS Open source page](#)

FileCore_MiscOp 1 Result Flags

FileCore

Exactly one of the bits 0-3 must be set. Once bit 6 or 7 is returned set for a given drive, they must always be so.

Results Flags

0 – Meaning when set: Not changed

1 – Meaning when set: Maybe changed

2 – Meaning when set: Changed

3 – Meaning when set: Empty

4 – Meaning when set: Ready

5 – Meaning when set: Drive is 40 track

6 – Meaning when set: Empty works

7 – Meaning when set: Changed works

8 – Meaning when set: Disc in drive is high density

9 – Meaning when set: Density sensing works

10 – Meaning when set: Ready works

11 - 31 – Meaning when set: Reserved, must be zero

[RISC OS Open source page](#)

Filer

List of modules

The Filer module provides the functionality to display a graphical representation of the filing system structure. It uses standard filing system calls and thus will work with any filing system.

In practice

Other specific filing systems (such as ADFSFile) cooperate with the Filer module by issuing the command *Filer_OpenDir when their icon is clicked. This enables the Filer to open the appropriate directory display window. They also provide other non-generic operations such as *Format and *Verify operations are provided by the ADFSFile.

Details

- Service Calls
- *Commands

[RISC OS Open source page](#)

Filer_Action

List of modules

The Filer_Action modules performs file manipulation operations on behalf of the Filer without the desktop freezing.

Details

- SWI Calls
- *Commands

[RISC OS Open source page](#)

FileSwitch

List of modules

FileSwitch is a module that is at the heart of all filing systems in RISC OS. It provides a common set of functions that is used by all filing systems. It only provides the parts of these services that are device-independent.

In practice

Any services that are device-specific are not provided by FileSwitch, but by separate modules.

Purpose of FileSwitch

Using a predefined interface, it notifies filing systems when they must do certain actions. It switches between currently active filing systems and keeps the state of each of them.

When filing systems are initialised they inform FileSwitch of three important pieces of information.

- Their name
- Where to find the routines for controlling the hardware
- Special actions that they can perform

Adding Filing Systems

Filing systems can be added to the Operating System easily by adding the relevant module to the system. As FileSwitch provides much of the common functionality, the filing system modules must provide the remaining functionality.

More detailed information on adding a filing system can be found in the Writing A Filing System? section.

Data Format

The data format of a filing system is not defined by the FileSwitch module. It only specifies what the user interface must look like.

Details

- Technical Details
- Service Calls
- SWI Calls

- *Commands

[RISC OS Open source page](#)

FileSwitch Key Features

FileSwitch

Because FileCore in RISC OS 3.7 and below only supported 10 character filenames, it was often assumed that this was the maximum length. However non-filecore filesystems could (and often did) allow for longer filenames. Since RISC OS 4, FileCore also allows for longer filenames.

Filenames

As such, you should not make any assumptions about the length of a filename.

FileSwitch unlike many other Operating Systems makes no distinction between upper and lower case filenames. i.e A file named 'john' is the same as 'John' or 'JOHN'.

Certain characters have special meanings and cannot be used within a filename. A general rule of thumb is not to use top-bit-set characters, even though FileCore can support them.

Here is a list of characters that cannot be used.

- . – Meaning: Used to separate directories (similar to / on other OSs)
- : – Meaning: Introduces a drive or disc. e.g :0, :HardDisc. It also marks the end of a filing system name. e.g adfs:
- * – Meaning: A wildcard to match zero or more characters
- # – Meaning: A wildcard to match any single character
- \$ – Meaning: Is the name of the root directory of the disc
- & – Meaning: Is the User Root Directory (URD)
- @ – Meaning: Is the currently selected directory (CSD)
- % – Meaning: Is the currently selected library directory (CSL)
- \ – Meaning: Is the previously selected directory (PSD)¹
- ^ – Meaning: Is the parent directory

¹ Available on FileCore-based filing systems, and any other that choose to do so.

Note: In RISC OS 2, wildcard matching would be specific to either file or folders, but not both. Later versions of RISC OS changed this, so that a wildcard match would act on all file objects. i.e it will try and find a match on both files and directories.

Directories

Directories, also known as folders on other Operating Systems, are used to group lots of files, and possible other directories together. Objects within the currently selected directory may be accessed without reference to the directory name. The names of objects must be unique within a directory.

The root directory of a storage medium is accessed via \$ character. As \$ is the root directory, there is no parent, and trying to access its parent will simply result in access to the root directory.

File in directories other than the root directory can be accessed by either prefixing the desired object with a directory path, or by setting the currently selected directory (CSD) and then accessing the object.

The use of the '.' (period) character is used to separate directories in a hierarchal structure.

Path / Meaning

\$Photos.Wedding001 – Meaning: File 'Wedding001' in directory Photos in \$

Holidays.Miami – Meaning: File 'Miami' in directory Holidays within currently selected directory

File Attributes

File attributes are used to store information relating to the read/write access available. They are stored using 4 bytes (32 bits). The top 24 bits are filing system dependent, where as the lower byte has the following standardised interpretation:

- 0 – Meaning if set: Object has read access for user
- 1 – Meaning if set: Object has write access for user
- 2 – Meaning if set: Owner execute only1, or private2
- 3 – Meaning if set: Object is locked against deletion by user
- 4 – Meaning if set: Object has public read access
- 5 – Meaning if set: Object has public write access
- 6 – Meaning if set: Hidden (Filer+)
- 7 – Meaning if set: Application (Filer+)
- 1 Applies to BBC ADFS only
- 2 Applies to SJ Research file servers only

If both bits 6 & 7 are set, then the file is super-hidden. The top 24-bits are filing system dependant. e.g. NetFS uses them to store the date.

Filing systems that have been created to use the features of FileCore, such as ADFS and RamFS, ignore the settings of bits 4 and 5. This is so that files can be moved freely between ADFS, RamFS and NetFS without losing information on their public read and write access.

Bits 2, 6 and 7 should be cleared when creating file attributes for a file, as they may be of use in future versions of RISC OS.

File Type

Unlike many other Operating Systems, RISC OS does not use file extensions as its native method of identifying the type of file. Instead, a three digit hexadecimal code, that is not part of the file name, is used instead.

Applications that can load and/or save a file of a specific type, usually inform the Operating System of this, and can assign a Sprite to the file type. Any files 'seen' by the Filer will then automatically display files of that type with the sprite, and if double-clicked upon, will use the application to load the file.

The file type information is stored as part of the Load and Execution Address fields.

File types are split into three distinct categories:

Value / Meaning

- &000 - &0FF** – Meaning: Personal use only
- &100 - &3FF** – Meaning: Non-commercial software
- &400 - &9FF** – Meaning: Commercial software
- &A00 - &AFF** – Meaning: Commercial software (ex AcornSoft)
- &B00 - &DFF** – Meaning: Commerical software
- &E00 - &FFF** – Meaning: Reserved by Operating System

It is possible to list the file types currently known by the computer at any given time, by using the following command.

`*Show File$Type_*`

For a list of commonly used file types by RISC OS, please see the File Types section.

Load and Execution Addresses

All files have in addition to their name, length, type and attributes, two 32-bit fields that are used to describe the file. These are set when the file is first created, and have two possible meanings.

- Load and Execution Addresses
- File type and Date Stamps

In the case of a simple machine code program, two memory addresses are specified; the load address specifies the memory address where the file should be loaded, and the execution address specifies where the program should commence. The table below shows the format of the memory addresses:

Load address / &XXXXLLLL

Execution address – &XXXXLLLL: &GGGGGGGG

Note: The execution address must fall within the load address and load address + the length of file.

In addition, if the top twelve bits of the load address are all set (XXX = &FFF), then the file is assumed to be date-stamped.

Date Stamp

If the top 12 bits of the load address are all set (see Load and Execution Addresses), then the remaining bits are used to store the date/time stamp of when the file was created or last modified, and the file type.

The date/time stamp uses 5 bytes (unsigned) to store the value. The start value begins at 00:00:00 1st January 1900, and each successive value is in centi-seconds. The most significant byte of the date is actually stored in the least significant byte of the load address field.

The format of how the data is stored using each of the 32 bit fields are shown below:

Load address / &FFFttdd

Execution address – &FFFttdd: &ddddddd

Where d is part of the date, and t is part of the file type.

Other languages documentation

[Français](#)

[RISC OS Open source page](#)

FileSwitch Paths To Search For Files

FileSwitch

When an operation on a file is to be acted upon and the full pathname is not explicitly provided, then the FileSwitch module must first search for the file. Two system variables are used to provide a list of file locations to search.

Default Paths

Variable / Description / Default Value

File\$Path – Description: Used for read operations only (load, open, update etc...) · Default Value: Null string

Run\$Path – Description: Used for execute operations only · Default Value: %, 1

1 This searches the current directory first, followed by the library.

The contents of each variable should expand to a list of prefixes, separated by commas.

HardDisc4.\$!boot.Choices.,Resources:\$.Resources.Filer.

When a read operation is to be performed, then the prefixes as specified in the File\$Path system variable are used to find a matching object. They are scanned in the order as provided by the File\$Path, and the first match, either a file or directory is used.

When an execute operation is to be performed, then the prefixes as specified in the Run\$Path system variable are used to find a matching object. They are scanned in the order as provided by the Run\$Path, and the first match, either a file or directory is used.

Adding Paths to Search

Filing system names can be added to the search path of either variable by using the *Set command. An example of two different paths to search on the network, when loading a file is provided below.

Set File\$Path %,Net:Software.,Net:Modules.,Net:Lib.

This would result in the FileSwitch module looking in the following paths for a given file:

@.file, %.file, Net:Software.file and Net:Modules.file and Net:Lib*.file

Error Checking

Each path in File\$Path and Run\$Path are checked in turn, in the same order as provided. An illegal pathname will not be used, and will simply be discarded and move to the next path.

The illegal path will, however, not be removed from the variable itself. It will simply not be scanned, as it is an illegal path name.

Path variables may expand to have leading and trailing spaces around elements of the path, and is a valid pathname. Any parsing of path variables need to take this into account.

*Set Run\$Path adfs:\$. , net:%. , ! is a valid path to search.

Using Other Path Variables

Other system variables can be used by an application to set up a list of paths to search. In addition, some SWIs can also allow for alternative path strings to be used to search for a file, or to perform the action without any form of look up.

To assign an alternative path variable, simply use the *Set command.

*Set Photos\$Path adfs:\$.Photos.,net:\$.Photos.

The pseudo filing system could then be accessed as Photos:

Special Field use with Path Variables

A system variable FileSwitch\$SpecialField can be used as part of a macro to define a path variable. An example of using this in conjunction with the default definition of Serial\$Path is this macro:

```
devices#<FileSwitch$SpecialField>:$.Serial.
```

Default values for the serial port can be set, by the following:

```
devices#baud=9600,bits=8,<FileSwitch$SpecialField>:$.Serial.
```

Any setting passed to FileSwitch as a special field would then override the defaults in the definition of Serial\$Path.

[RISC OS Open source page](#)

FileSwitch Use Of File Types

FileSwitch

As discussed in the key features section on FileSwitch, a 3-digit hexadecimal value is used to represent the file type. It is the file type that determines the action that must be performed on a file.

Determine the Action to Perform

When a date stamped file of type xxx has an action of *Load or *Run performed on it, FileSwitch carries out several steps to determine what it should actually do.

The steps are:

- Look for the system variables Alias\$@LoadType_xxx or Alias\$@RunType_xxx respectively.
- If variable of string or macro type exists, then it is copied (after macro extension), and the full pathname is used to find the file on either File\$Path or Run\$Path
- Any parameters passed are also appended for *Run commands
- The whole string is passed to the command Line interpreter using OS_CLI

An Example of using LoadType

CSD · adfs::HardDisc.\$.Photos.Wedding

Command · *LOAD Photo001

File type · &FF9, Sprite (bitmap)

Step 1 - Find system variables

Variable · *@LoadType_FF9

Step 2&3 - Use full pathname + parameters

Issues command · `*@LoadType_FF9 adfs::HardDisc$.Photos.Wedding.Photo001`
 Alias\$`@LoadType_FF9` · has a value of `SLoad %*0` by default
 Step 4 - CLI issues command
 Output · `*SLoad adfs::HardDisc$.Photos.Wedding.Photo001`

An Example of using RunType

CSD · `adfs::HardDisc$.Programs`
 Command · `*RUN BasicProg p1 p2`
 File type · `&FFB, BASIC`
 Step 1 - Find system variables
 Variable · `*@RunType_FFB`
 Step 2&3 - Use full pathname + parameters
 Issues command · `*@RunType_FFB adfs::HardDisc$.Programs.BasicProg p1 p2`
 Alias\$`@RunType_FFB` · has a value of `Basic -quit |"%0| %*1` by default
 Step 4 - CLI issues command
 Output · `*Basic - quit "adfs::HardDisc$.Programs.BasicProg" p1 p2`

Default Settings

The filing system manager sets several of these system variables by default; however, they can be changed easily. This is useful say, if a newly installed application wants to be responsible for loading files of a certain type.

Setting a file of a specific file type can easily be achieved by using the `*Set` command.

A simple example of this is command is provided below.

`Set Alias$@LoadType_007 SpyApp %*0 Set Alias$@RunType_007 SpyApp %*0`

[RISC OS Open source page](#)

Filing System Numbers

Controlling sources: `Hdr:FSNumbers` managed by the allocations manager

These are codes used for different filing systems.

- 0 – File System: None 1
- 3 – File System: ROMFS
- 5 – File System: NetFS
- 8 – File System: ADFS
- 12 – File System: NetPrint
- 13 – File System: Null
- 14 – File System: Printer
- 15 – File System: Serial
- 17 – File System: VDU
- 18 – File System: RawVDU
- 19 – File System: Kbd
- 20 – File System: RawKbd
- 21 – File System: DeskFS
- 22 – File System: Computer Concepts' RomFS
- 23 – File System: RamFS
- 24 – File System: RISCIXFS
- 25 – File System: Streamer
- 26 – File System: SCSIIFS?
- 27 – File System: Digitiser
- 28 – File System: Scanner

29 – File System: MultiFS
33 – File System: NetFS
37 – File System: CDFS
38 – File System: Computer Concepts' CFS
43 – File System: DOSFS
46 – File System: ResourceFS
47 – File System: PipeFS
49 – File System: IDEFS
53 – File System: DeviceFS
54 – File System: Parallel
56 – File System: ArcFS
61 – File System: PSCompress
66 – File System: SparkFS
77 – File System: MirrorFS
79 – File System: FTP
91 – File System: Memphis
99 – File System: ShareFS
101 – File System: Computer Concepts' PrintQFS
105 – File System: HostFS (RedSquirrel)
118 – File System: CacheFS
123 – File System: IZipFS
136 – File System: FlashFS
142 – File System: raFS
153 – File System: HostFS (ArcEm/Arculator/RPCEmu)
162 – File System: TFTP
192 – File System: SDFS
195 – File System: Source
200 – File System: NVMeFS
&40000000 – File System: None 1 (OS_Args 0 only, see notes)

Notes

1 A bug present in FileSwitch 1.70 to 2.74 inclusive would return the incorrect value for 'None'. Therefore, when checking for 'None' with OS_Args 0 the two values &0 and &40000000 should be considered equivalent.

Their codes may be returned by Message_FilerOpenDir.

[RISC OS Open source page](#)

FilterManager

List of modules

The Filter Manager provides applications with the facility to register pre and post-filters against WIMP tasks?. Filters are useful when a task needs to intercept events that may or may not be under the control of the task.

In practice

Filters do not need to be applied to a specific task as they can be assigned to respond to all tasks. Each filter is a routine that must have well defined entry and exit conditions.

Pre-Filters

Pre-filters are called when a task calls the Wimp (e.g. via Wimp_Poll). They are called before control passes to the Wimp, allowing the filter to adjust the parameters of the call or override it completely.

Post-Filters

Post-filters are called when the Wimp passes control of execution back to a task. This allows the filter to monitor or adjust the result of a call that the task had made, or perform additional processing such as rendering an image over the top of a window.

Details

- Service Calls
- SWI Calls
- *Commands

[RISC OS Open source page](#)

Flags For File Operations

OS_FSControl

Usages C for Copy W for Wipe N for Count (Number)

Flags

- 0 – Use: CWN · Meaning: ^R^ecurse
- 1 – Use: CW · Meaning: ^F^orce
- 2 – Use: CWN · Meaning: Given times in block
- 3 – Use: CWN · Meaning: ^C^onfirm operation
- 4 – Use: CWN · Meaning: ^V^erbose
- 5 – Use: C · Meaning: ^Q^uick (use application area – not really safe)
- 6 – Use: C · Meaning: ^P^rompt for new disc
- 7 – Use: C · Meaning: ^D^elete after copy
- 8 – Use: CWN · Meaning: Allow printing of progress
- 9 – Use: C · Meaning: ^A^ccess attributes as in source
- 10 – Use: C · Meaning: ^S^tamp files
- 11 – Use: C · Meaning: S^t^ructure copy only
- 12 – Use: C · Meaning: ^N^ewer than destination
- 13 – Use: CN · Meaning: Use descriptor block as workspace
- 14 – Use: C · Meaning: ^L^ook at destination first

Notes

Verbose will list every file it performs an operation on. Allow printing will print a summary of every directory it acts upon. Superscript letters show the equivalent *Command option letters.

[RISC OS Open source page](#)

Floating Point Contexts

FPEmulator

Multiple-context versions of the floating point system (version 4.12 or later of FPEmulator, indicated by an "M" suffix in the help string) allow rapid switching between processes using floating point, preserving the complete state of the system, including any pending asynchronous exceptions. A floating point context is an opaque block of memory used to hold the state of the system - an active context must be in an unpagged area of memory, such as the RMA. SWIs are provided to switch between contexts, and the floating point system provides a single default context which is activated when the module is loaded.

Floating point contexts

In all calls that take or return a context pointer, -1 is used to represent the default context. 0 indicates no context - if no context is active, then floating point instructions will generate undefined instruction exceptions. Where specified, some calls also accept -2 to denote the current context.

The contents of a floating point context should be regarded as a "black box": software outside the floating point system itself should neither assume that it knows how to interpret them, nor alter them directly. However, the system does provide some facilities to allow inspection and modification of a floating point context.

Floating point context switching may be of use for performing floating point during interrupts or callbacks, or as part of threading schemes. But note that use of floating point instructions from SVC mode in versions of RISC OS prior to RISC OS 4 is only possible under the following constraints:

- The instruction must not use R13 or R14 in any way;
- R14 must be treated as a scratch register (ie. it will not necessarily be preserved by a floating point instruction);
- R13 must point correctly to a Full Descending stack with a sufficient amount of free space.

This rules out the use of floating point via the C compiler.

It should also be noted that on all versions of RISC OS floating point instructions will enable interrupts if they are handled by the emulator - this is possible even on systems fitted with hardware floating point.

All the new SWIs corrupt the N Z and C flags.

[RISC OS Open source page](#)

Floppy Disc Formats

F · 1.6 Mb – OS: RiscOS 3 · Extra Information: 77-entry directories, new map
E · 800 Kb – OS: RiscOS · Extra Information: 77-entry directories, new map
D · 800 Kb – OS: Arthur 1.2 · Extra Information: 77-entry directories, old map
L · 640 Kb – OS: All ADFS · Extra Information: 77-entry directories, old map
DOS/Q · 1.44 Mb – OS: MSDOS 3.20 · Extra Information: Double-sided High-density 3.5" disc
DOS/M · 720 Kb – OS: MSDOS 3.20 · Extra Information: Double-sided 3.5" disc
DOS/H · 1.2 Mb – OS: MSDOS 3 · Extra Information: Double-sided High-density 5.25" disc
DOS/N · 360 Kb – OS: MSDOS 2.3 · Extra Information: Double-sided 3.5", 5.25" disc
DOS/P · 180 Kb – OS: MSDOS 2.3 · Extra Information: Single-sided 5.25" disc
DOS/T · 320 Kb – OS: MSDOS 1.2.3 · Extra Information: Double-sided 5.25" disc
DOS/U · 160 Kb – OS: MSDOS 1.2.3 · Extra Information: Single-sided 5.25" disc
Atari/M · 720 Kb – OS: Atari ST · Extra Information: Double-sided 3.5" disc
Atari/N · 360 Kb – OS: Atari ST · Extra Information: Single-sided 3.5" disc

[RISC OS Open source page](#)

Font_DecodeMenu Flags

FontManager

If bit 0 was not set, it would specify that a font menu was to be decoded, rather than an encoding menu.

Bit / Meaning when set

0 – Meaning when set: Decoding a encoding menu

[RISC OS Open source page](#)

Font_DecodeMenu Result

FontManager

Results from the call are of the form:

In practice

`\Font_id[\fterritory font_name][\Eencoding_id[\eterritory encoding_name]]`

Where [] indicate optional sections.

Format

The following table shows the format of the string, dependant on the data contained in the file:

File holds / Format of returned String

Font_id, no name – Format of returned String: `\Font_id`

Font_id, with name – Format of returned String: `\Font_id\fterritory fontname`

Encoding_id, no name – Format of returned String: `\Eencoding_id`

Encoding_id, with name – Format of returned String: `\Efont_id\eterritory encoding_name`

Examples

The following are actual results obtained.

`\FEquasor\f1 Bob \FPembroke.Bold \FTrinity.Medium\ELatin1 \ELatin5\e1 Gleep \EWelsh \FCorpus.Medium\eLatin1\e1 Gleep`

Notes

The prefix for Font names or Encoding names are uppercase (F or E), where as the prefix for territory encoding name or font names are lower case (f or e).

The string will be terminated by a control character.

[RISC OS Open source page](#)

Font_MakeBitmap Flags

FontManager

These flags are used to specify the characteristics of the bitmap font file created.

Bit / Meaning when set

0 – Meaning when set: 1 = f9999x9999 (4 bpp), 0 = b9999x9999 (1bpp)

1 – Meaning when set: Perform horizontal subpixel positioning

2 – Meaning when set: Perform vertical subpixel positioning

3 – Meaning when set: Delete old file (do not replace)

4-31 – Meaning when set: Reserved

[RISC OS Open source page](#)

Font_Paint Block

FontManager

These values are in millipoints if bit 4 of the flags in R2 is 0, and in OS units if bit 4 is set.

+0 – Contents: Additional x offset for spaces

+4 – Contents: Additional y offset for spaces

+8 – Contents: Additional x offset for each letter

+12 – Contents: Additional y offset for each letter

+16 – Contents: X co-ordinate for bottom left of rubout box (inclusive)

+20 – Contents: Y co-ordinate for bottom left of rubout box (inclusive)

+24 – Contents: X co-ordinate for top right of rubout box (exclusive)

+28 – Contents: Y co-ordinate for top right of rubout box (exclusive)

[RISC OS Open source page](#)

Font_Paint Flags

FontManager

1 Applies to FontManager 3.41 onwards

0 – Meaning when set: 1= justify specified by graphics cursor, 0= not justified

1 – Meaning when set: 1= rubout box specified by graphics cursor, 0=do not rub out

2-3 – Meaning when set: Reserved. Must be zero

4 – Meaning when set: 1= x and y are in OS units, 0= millipoints

5 – Meaning when set: Justify & rubout specified by block in R5 (bit 4 must be clear), 0= use flags in bits 0 & 1

6 – Meaning when set: Font Paint Transformation Matrix specified by R6 (bit 4 must be clear)

7 – Meaning when set: Given string length

8 – Meaning when set: Given font

9 – Meaning when set: Kern

10 – Meaning when set: Right to left

11 – Meaning when set: Font Background Blending

12 – Meaning when set: R1 points to 16-bit string forms (bit 13 must be clear)¹

13 – Meaning when set: R1 points to 32-bit string forms (bit 12 must be clear)¹

14 – Meaning when set: Enable alpha/supremacy blending²

15-31 – Meaning when set: Reserved. Must be zero

In practice

2 Applies to FontManager 3.60 onwards. Also requires bit 11 to be set (see Font Background Blending).

Notes

To specify justify using the graphics cursor, first move to right side.

To specify a rubout box using the graphics cursor, first move to lower left and then to upper right of box.

If both, then do justify move last.

[RISC OS Open source page](#)

Font_Paint Special Characters

FontManager

Some control characters have special meaning:

- 9, dx_low, middle, high** – Meaning: Move print position horizontally
- 11, dy_low, middle, high** – Meaning: Move print position vertically
- 17, colour** – Meaning: Set font colour (+&80 for background). Use 19... instead
- 18, background, foreground, offset** – Meaning: See Font_SetFontColours. Use 19... instead
- 19, bg_R, bg_G, bg_B, fg_R, fg_G, fg_B, offset** – Meaning: See ColourTrans_SetFontColours
- 21, comment_string, ctrl_terminated (any)** – Meaning: Skip and ignore the comment string
- 25, underline_pos, thickness** – Meaning: Turn underline on/off
- 26, font_handle** – Meaning: See Font_SetFont
- 27, ALIGN; a, b, c, d** – Meaning: Partial Font Paint Transformation Matrix (with no translation)
- 28, ALIGN; a, b, c, d, e, f** – Meaning: Full Font Paint Transformation Matrix (with translation)

Notes

There are no spaces between control sequence fields. (Spaces have been added to the examples above to make them easier to read.)

With codes 9 and 11, the print-position is given in millipoints.

With code 25, position and thickness is given in $1/256^{\text{th}}$ of current font size. Position is sign extended, thickness is unsigned. Set thickness to 0 to turn off underlining.

With codes 27 and 28, the code must be followed by 0-3 null bytes to word align the Font Paint Transformation Matrix.

When using 16 or 32-bit characters, these take a slightly different form.

[RISC OS Open source page](#)

Font_Paint Special Characters Parameters

FontManager

When control codes are used with 16 or 32-bit forms, the parameters are different. The tables below show how they differ.

- 9,11** – 16-Bit Parameter Forms: dx/dy comprised of 2 16-bit words
- 17** – 16-Bit Parameter Forms: 1 16-bit word
- 18** – 16-Bit Parameter Forms: 3 16-bit words
- 19** – 16-Bit Parameter Forms: 2 x (RR00, BBGG word pairs) + 1 16-bit word
- 21** – 16-Bit Parameter Forms: Sequence of 16-bit words, terminated by a word < &0020

25 – 16-Bit Parameter Forms: 2 16-bit words
26 – 16-Bit Parameter Forms: 1 16-bit word
27 – 16-Bit Parameter Forms: 4 32-bit words (word-aligned)
28 – 16-Bit Parameter Forms: 6 32-bit words (word-aligned)
9,11 – 32-Bit Parameter Forms: 1 32-bit word
17 – 32-Bit Parameter Forms: 1 32-bit word
18 – 32-Bit Parameter Forms: 3 32-bit words
19 – 32-Bit Parameter Forms: 3 32-bit words
21 – 32-Bit Parameter Forms: Sequence of 32-bit words, terminated by a word < &00000020
25 – 32-Bit Parameter Forms: 2 32-bit words
26 – 32-Bit Parameter Forms: 1 32-bit word
27 – 32-Bit Parameter Forms: 4 32-bit words
28 – 32-Bit Parameter Forms: 6 32-bit words
[RISC OS Open source page](#)

Font_ScanString Block

FontManager

+0 – Contents: Additional x offset for spaces
+4 – Contents: Additional y offset for spaces
+8 – Contents: Additional x offset for each letter
+12 – Contents: Additional y offset for each letter
+16 – Contents: Split character, or -1 for none
+20 – Contents: Returns left of bounding box (if bit 18 set)
+24 – Contents: Returns bottom of bounding box (if bit 18 set)
+28 – Contents: Returns right of bounding box (if bit 18 set)
+32 – Contents: Returns top of bounding box (if bit 18 set)

[RISC OS Open source page](#)

Font_ScanString Flags

FontManager

1 Font Manager 3.41 onwards

0 – Meaning if set: 0 (reserved)
1 – Meaning if set: 0 (reserved)
2 – Meaning if set: 0 (reserved)
3 – Meaning if set: 0 (reserved)
4 – Meaning if set: 0 (reserved)
5 – Meaning if set: R5 is pointer to block
6 – Meaning if set: R6 is pointer to Font Paint Transformation Matrix
7 – Meaning if set: R7 is length of string
8 – Meaning if set: R0 is initial font handle
9 – Meaning if set: Perform kerning
10 – Meaning if set: Writing direction is right-to-left
11 – Meaning if set: 0 (reserved)
12 – Meaning if set: R1 points to 16-bit string (bit 13 must be clear)¹
13 – Meaning if set: R1 points to 32-bit string (bit 12 must be clear)¹
14 – Meaning if set: 0 (reserved)

- 15 – Meaning if set: 0 (reserved)
- 16 – Meaning if set: 0 (reserved)
- 17 – Meaning if set: Return nearest caret position (if x offset is less than half way through a character, the character is not included)
- 18 – Meaning if set: Return bounding box in R5 (bit 5 must also be set)
- 19 – Meaning if set: Return updated matrix in R6 (bit 6 must also be set)
- 20 – Meaning if set: Return number of split characters in R7 (bit 7 must also be set)
- 21-31 – Meaning if set: 0 (reserved)

Notes

The flags in R2 determine how this call is used. Registers R0, R5, R6 and R7 are only used if bits 5, 6, 7 and 8 are set in flags in R2.

If bit 8 is clear, then the current font will be assumed. This call does not alter the current font, so if you are scanning several sequential lines and these contain font shifts, you should call `Font_FutureFont` to find which font handle to pass in R0 the next time.

If bit 18 is set, the returned bounding box will only be valid if `Font_ScanString` was able to process some characters. E.g. calling with R7=0 will return an invalid bounding box (left and bottom coordinates greater than the right and top coordinates).

If bit 20 is set, there is no split character, then R7 will be used to return the number of non-control characters (codes 32 or greater and not within a control sequence).

[RISC OS Open source page](#)

Font_SwitchOutputToBuffer Flags

FontManager

Bit / Meaning when set

- 0 – Meaning when set: Update R1, but do not actually store anything
- 1 – Meaning when set: Apply hints
- 4 – Meaning when set: Generate error if bitmapped characters found

All other bits reserved

[RISC OS Open source page](#)

FontManager

List of modules

A font is a set of characters in a given type style. The Font Manager provides facilities for painting characters of various styles and sizes on to the screen. Fonts in RISC OS are scalable and are generally proportionally spaced. Anti-aliasing of up to 16 colours is used to give the illusion of greater resolution, and hints may be used to improve the display in low resolutions.

In practice

As of version 3.41, the Font Manager is capable of handling Unicode fonts. See documentation on the UCS Font Manager for more details.

Details

- Technical Details
- Service Calls

- SWI Calls
- *Commands

[RISC OS Open source page](#)

Format Of Sprite

Format Of Sprite Area

Sprites have a fixed size-header followed by a varying amount of data.

Header format

Sprite headers are 44 bytes long, with the following format.

- 0** – Content: Offset to next sprite in area (see notes)
- 1-3** – Content: Sprite name (12 bytes, padded with nulls)
- 4** – Content: Sprite width in words, minus 1
- 5** – Content: Sprite height in pixels, minus 1
- 6** – Content: 0 for new sprites
- – Content: Historically, the first used bit on left edge of sprite (0-31)
- 7** – Content: Last used bit on right edge of sprite (0-31)
- 8** – Content: Offset to sprite image
- 9** – Content: Offset to sprite mask
- 10** – Content: Sprite Mode Word

All offsets are in bytes, relative to the start of the sprite header. If the sprite has no mask, the mask offset should be set to the same value as the image offset. If the sprite is the last sprite in the area, the offset to the next sprite must be set to the first byte after this sprite (i.e. to the total size of the sprite).

If this is a new format sprite (i.e. the mode word does not contain a mode number), the first bit used (word 6) must be zero. This is recommended for all sprites.

Palette data

If a sprite has a palette, the palette data immediately follows the sprite header. Each palette entry is two words long, allowing for flashing colours to be stored. Colours are stored in the standard &BBGRR00 format.

Because the palette is stored immediately after the header, the presence and size of the palette can be determined by examining words 8 and 9 of the sprite header:

$\text{palette_entries} = (\min(\text{offset_to_image}, \text{offset_to_mask}) - 44) / 8$

If the palette length is not a multiple of 8 and does not contain exactly 16, 64 or 256 entries, the palette should be ignored. This is reserved for future expansion.

Note that 256 colour sprites that are designed to be VIDC1 compatible may only have 64 or 16 entry palettes instead of the full 256 entries.

Although SpriteExtend and ColourTrans do not use palettes attached to 'deep colour' sprites (>256 colours) some third party software can make use of such palettes.

Image data

Each row of image data is word-aligned. The byte length/stride of the rows can be easily determined from the sprite width value of the sprite header. However to correctly calculate the pixel width, the bits per pixel of the sprite mode must be known. Pixel width can be calculated as follows:

$\text{pixel_width} = (\text{word_width_minus_one} * 32 + \text{right_bit} + 1 - \text{left_bit}) / \text{bpp}$

Although each row is word aligned, the first pixel of each row may not start on a word boundary. Use word 6 of the sprite header to determine which bit of the word constitutes the first bit of the image. This is known as 'left-hand wastage'.

For a sprite to be considered valid, the first used bit and last used bit (plus one) must be multiples of the sprite bits per pixel, i.e. sprite data must begin and end on a pixel boundary.

Both image and mask data is stored top-down, left-to-right. I.e. the first row is the topmost row of the sprite, and the first pixel of a row is from the leftmost column of the sprite.

Mask data

For old format sprites (those where the mode specifies a numbered mode), each mask pixel is at the same BPP as the image pixels. However the OS only treats the mask as a binary on/off value (0=invisible pixel, non-zero=visible). Words 6 and 7 of the header must be used to determine where the valid mask pixels start and stop within the mask data.

For new format sprites, the mask width is typically 1bpp (0=invisible, 1=visible). However if the 'wide mask' bit of the mode word is set then an 8bpp mask is used, with the mask treated as an alpha channel to allow for smooth blending with the background (0=0% alpha, 255=100% alpha). Note that wide masks are not supported for sprite type 0 (i.e. mode numbers). The first valid mask pixel will always start at bit 0 of the mask data. The end of the last valid mask pixel (and the length/stride of the mask rows) must be calculated manually, based around the pixel width of the sprite and the BPP of the mask.

As with sprites, mask rows are always a multiple of 4 bytes in length.

Additional notes

Image and mask data may appear in any order.

SpriteOps that modify the byte size of the sprite (such as adding rows or creating a palette) require that the "offset to next sprite" is the actual length of the sprite. This is the recommended format.

[RISC OS Open source page](#)

Format Of Sprite Area

Sprite areas have a fixed size-header followed by an optional extension area, and then a number of sprites.

Header format

Sprite area headers are 16 bytes long, with the following format.

Offset / Content

+0 – Content: Size of sprite area

+4 – Content: Number of sprites in area

+8 – Content: Offset to first sprite (16+)

+12 – Content: Offset to first free word in area (i.e. after last sprite)

+16 – Content: Optional Sprite Extension Area (if offset to first > 16)

The sprite area size is measured in bytes including the 16 byte header, and must be a multiple of four.

All offsets are in bytes, relative to the start of the sprite area, and are always word aligned.

To create a valid empty sprite area, the header should be filled in as follows:

Offset / Content

+0 – Content: Size of sprite area buffer

+4 – Content: 0

+8 – Content: 16

+12 – Content: 16

[RISC OS Open source page](#)

Free

List of modules

The Free module provides an interactive free space display from the desktop.

In practice

Any filing system that wants to display an interactive free space display should register with this module. In doing so, the filing system provides the address of a routine that accepts a variety of reason codes, each of which provides support for this module.

Details

- SWI Calls
- *Commands

[RISC OS Open source page](#)

Freeway

List of modules

The Freeway module provides the protocols used by the Access (an old commercial Acorn Computer networking product) so it knows what shared resources are available and can display windows showing them.

In practice

The interfaces used by Freeway are for internal use only. They must not be used.

Details

- Technical Details
- Service Calls
- SWI Calls
- *Commands

[RISC OS Open source page](#)

Freeway Technical Details

Freeway

This is the name of an object. An object name is unique within the set of objects of the same type within a site. Object names are terminated by a control character (and will return a terminator of zero on output), case independent, alphanumeric strings, created by users or application software.

Maximum length is 64 bytes, including the terminator.

Object Descriptor

This provides extra information about the object. Its format is undefined at this level – it has significance to higher-level application software. Object descriptors are arrays of bytes of any value, maximum length 255.

In Acorn Access+ the descriptor for type Discs indicates whether a disc is shared protected or unprotected, or as a CDROM or subdirectory, and visible or invisible to other desktop users.

The descriptor for type Printers is a 'Printer Type' string.

The descriptor for type Hosts is null.

Object Authentication Value

Objects are either authenticated or unauthenticated. Authenticated objects require an authentication value to be provided before their attributes are made visible. An authentication value is a 32-bit integer. The authentication value of zero is reserved and should not be used.

Refresh Interval

The Freeway software in each computer will periodically re-notify other computers about unauthenticated objects which are held by the local machine, so that they know that the objects continue to be accessible. This permits, for example, other machines to know when a computer holding Freeway objects has been powered down. The renotification period is called the "refresh interval", and is determined by the Freeway software and is typically 30 seconds.

Type Number

This is a 16-bit integer identifying a Freeway type. The following type numbers are allocated to Acorn Access+. (Informal names with no significance in software will also be allocated, for easier human reference)

/ Type

1 – Type: Discs

2 – Type: Printers

5 – Type: Hosts

32 – Type: Users

In Acorn Access, ShareFS is the controlling application for types Discs and Hosts, and !Printers is the controlling application for type Printers.

[RISC OS Open source page](#)

FSLock

List of modules

The FSLock module provides the functionality to protect CMOS RAM and other filing systems such as the hard disc against unwanted modification. This is achieved by intercepting any SWI that alters the filing system contents or CMOS RAM, and returning an error instead.

Details

- Technical Details
- SWI Calls
- *Commands

[RISC OS Open source page](#)

FSLock Technical Details

FSLock

The FSLock module provides the functionality to protect CMOS RAM and other filing systems such as the hard disc against unwanted modification. This is achieved by intercepting any SWI that alters the filing system contents or CMOS RAM, and returning an error instead.

The FSLock Module Overview

Currently, FSLock can only protect drives 4 – 7 on any one filing system. It is important to note that two areas of a protected disc are excluded. They are:

- `$.Public` – This location is commonly used for general file storage by the end user. It cannot be created while the computer is in a locked state.
- `$.Scrap.ScrapDir` – This location needs to remain writeable to allow Scrap transfers of files between applications.

Lock States

The following table details the different Lock states.

State / Description

Fully unlocked – Description: An fully unlocked device has no password allocated to it. The discs or configuration can be modified. This state persists over all device resets. This is the default state after the CMOS RAM has been successfully cleared

Partially unlocked – Description: A partially unlocked device has a password allocated to it. The discs and configuration modified. If the machine is reset, it reverts to being locked

Locked – Description: A locked device has a password allocated to it. The discs or configuration cannot be modified. This state persists over all device resets

Lock Status

The lock states are passed to commands using the lock status.

Status / Meaning

0 – Meaning: Fully unlocked

1 – Meaning: Partially unlocked

2 – Meaning: Locked

Passwords

- Passwords are case sensitive.
- The Configure application restricts the password to at least five non-space characters that are acceptable in a writable icon. Although the SWIs will accept any null terminated string, it is advisable to follow the five non-space character limit, otherwise users will be unable to enter a valid password.

[RISC OS Open source page](#)

Gain_Caret

Reason Code 12 · WindowManager

This code is returned from `Wimp_Poll` and indicates that the window has now received the input focus.

On return

Block

+0 Window handle

+4 Icon handle, or -1 if none
+8 X offset (relative to window origin)
+12 Y offset (relative to window origin)
+16 Caret Flags
+20 Index into string, or -1 if none

Notes

This event usually occurs due to a user clicking on an icon or window that accepts text. It can also happen if an application explicitly calls `Wimp_SetCaretPosition`.

Windows and icons can be set to accept text by setting the button types within the window flag and icon flags respectively.

This event code is not generated if the input focus changes within the same window.

[RISC OS Open source page](#)

GPIO

List of modules

The GPIO module allows access to the general purpose input/output pins of various ARM boards on which RISC OS runs.

Details

- SWI Calls
- *Commands

[RISC OS Open source page](#)

GPIO_ReadEvent

GPIO

Polls the event state of the GPIO pin, the event type is set with flag bits in `GPIO_WriteMode`, or returns an error if the pin number is not valid.

Before calling

R0 Logical GPIO pin number

On return

R0 = 0 (No event occurred), 1 (Event occurred)

[RISC OS Open source page](#)

GraphicsV

Vectors

This vector is used by the RISC OS 5 kernel to communicate with the video driver.

Before calling

R4 Bits 0-15: Reason Code

Bits 16-23: Head or overlay number

Bits 24-31: Driver number

Other registers dependent on call

In practice

User software should always aim to use the appropriate OS SWI instead of calling GraphicsV directly. The information presented here is mainly of use to OS maintainers and video driver developers.

R4 is used by all GraphicsV calls to store both the reason code and to indicate whether the call has been claimed (i.e. acted upon by a driver). When a driver claims a call, it will set R4 to zero. By checking R4 after calling GraphicsV it is possible to determine whether the driver supports that call or not.

Reason Codes

- 0 · Null reason code for when vector has been claimed
- 1 · VSync interrupt occurred · Head · BG · SVC/IRQ
- 2 · Set mode · Head · FG2 · SVC
- 3 · Obsolete3 (was Set interlace) · Head · FG · SVC
- 4 · Set blank · Head · FG/BG · SVC
- 5 · Update pointer · Head · FG/BG · SVC/IRQ
- 6 · Set DAG · Head · FG/BG · SVC/IRQ
- 7 · Vet mode · Head · FG · SVC
- 8 · Features · Head · FG · SVC
- 9 · Framestore information · Head · FG · SVC
- 10 · Write palette entry · Head/Overlay · FG/BG · SVC/IRQ
- 11 · Write palette entries · Head/Overlay · FG/BG · SVC/IRQ
- 12 · Read palette entry · Head/Overlay · FG · SVC
- 13 · Render · Head · FG · SVC
- 14 · IIC op · Head · FG · SVC
- 15 · Select head · -- · FG · SVC
- 16 · Select startup mode · Head · FG · SVC
- 17 · List pixel formats · Head · FG · SVC
- 18 · Read info · -- · FG · SVC
- 19 · Vet mode 2 · Head · FG · SVC
- 20 · Create overlay · -- · FG · SVC
- 21 · Destroy overlay · Overlay · FG · SVC
- 22 · Set overlay position · Overlay · FG · SVC
- 23 · Map overlay buffer · Overlay · FG · SVC
- 24 · Unmap overlay buffer · Overlay · FG · SVC
- 25 · Discard overlay buffer · Overlay · FG · SVC
- 26 · Vet overlay · -- · FG · SVC
- 27 · Set overlay transform · Overlay · FG · SVC
- 28 · Set overlay Z order · -- · FG · SVC

Driver registration and deregistration

GraphicsV based video drivers should perform the following sequence of events on startup:

- Call OS_ScreenMode 64 to request a driver number
- Store the number, for use by their GraphicsV handler
- Call OS_Claim to install their GraphicsV handler
- Start generating VSync calls if possible
- Perform any other initialisation that's required to prepare the driver/hardware
- Call OS_ScreenMode 65 to let the OS know that the driver is ready for use

Once OS_ScreenMode 65 is called, the driver should expect to start receiving GraphicsV calls from the OS and from other software. Before that point no calls should be received, and if they are then it's a bug in someone else's code and not yours.

On shutdown, drivers should use the following sequence:

- Call OS_ScreenMode 66 to indicate that the driver is stopping. Once this call returns, the driver should no longer receive GraphicsV calls.
- Halt generation of VSync calls
- Call OS_Release to release the GraphicsV handler
- Call OS_ScreenMode 67 to release the driver number and allow it to be allocated for use by other drivers.

Note that OS_ScreenMode reasons 64-67 must only be called from the foreground.

See also HAL Video API for details on how the OS translates GraphicsV calls to HAL calls for systems which contain a video driver in their HAL.

1 Calling contexts

GraphicsV drivers can be called under a variety of different situations. Consult the table above, and the list below, to determine the situations in which each entry point may be called and how the implementation is allowed to behave.

- FG - These calls must only be made from the foreground, with interrupts enabled. E.g. usermode applications, callbacks, or RTSupport threads. Calls marked "BG" or "FG/BG" may be called while a foreground call is in progress, but no other foreground calls should occur. Foreground calls are allowed to block for long periods of time, and are allowed to enable/disable interrupts at will.
- BG - These calls must only be made from the background, i.e. from within an interrupt handler. The implementation must execute quickly and in a non-blocking manner. On entry, interrupt state is undefined. Drivers may disable interrupts as required, but should not enable them if they were disabled on entry. The implementation must also be fully re-entrant, as the same (or a different) background call may be made while the first is in progress.
- FG/BG - These calls may be made from the foreground or the background. The same execution rules as background calls apply - the implementation must be fast, non-blocking, must not enable interrupts if they were disabled on entry, and must be re-entrant with other driver calls.
- SVC - Vector handlers are only entered in SVC mode.
- SVC/IRQ - The kernel may call these entries from SVC or IRQ mode. Therefore, care must be taken to preserve R14_svc around any SWI calls.

2 GraphicsV 2

As an addition to the above rules, drivers are free to ignore any background calls they receive while they are in the middle of processing a mode change.

3 GraphicsV 3

This call is obsolete. Interlace settings should be specified via the VIDC list given to GraphicsV 2/7.

Other languages documentation

Français

[RISC OS Open source page](#)

GraphicsV 1

GraphicsV

The display driver should call this vector when a VSync interrupt occurs.

Before calling

R4 Bits 0-15: 1 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R0-R3 Corrupt

R4 0

- All other registers preserved

In practice

If a display driver receives this call, then it should ignore it and pass it on unmodified.

Notes

When the kernel receives this vector, it will claim it (by setting R4 to 0) and perform its usual VSync processing - updating the pointer position by issuing GraphicsV 5, releasing any foreground task waiting on OS_Byte 19, etc.

Note that this call must be made from an interrupt handler, otherwise OS_Byte 19 will never return.

[RISC OS Open source page](#)

GraphicsV 10

GraphicsV

The kernel issues this call to instruct the driver to update the display palette. The appropriate driver should respond by performing the requested action and setting R4 to 0 to claim the call.

Before calling

R0 Entry type

R1 Colour

R2 Palette index

R4 Bits 0-15: 10 (reason code)

Bits 16-23: Head/overlay number

Bits 24-31: Driver number

On return

R4 0

- All other registers preserved

Notes

For details of the colour formats used with the different palettes, see GraphicsV palette colours.

[RISC OS Open source page](#)

GraphicsV 11

GraphicsV

The kernel issues this call to instruct the driver to update the display palette. The appropriate driver should respond by performing the requested action and setting R4 to 0 to claim the call.

Before calling

R0 Entry type

R1 Pointer to array of colours

R2 First palette index

R3 Number of entries

R4 Bits 0-15: 11 (reason code)

Bits 16-23: Head/overlay number

Bits 24-31: Driver number

On return

R4 0

- All other registers preserved

Notes

For details of the colour formats used with the different palettes, see GraphicsV palette colours.

[RISC OS Open source page](#)

GraphicsV 12

GraphicsV

The kernel issues this call to obtain the effective colour of a palette entry after taking into account any hardware restrictions in the video controller. The appropriate driver should respond by updating the colour value in R1 (if necessary) and setting R4 to 0 to claim the call.

Before calling

R0 Entry type

R1 Colour

R2 Palette index

R4 Bits 0-15: 12 (reason code)

Bits 16-23: Head/overlay number

Bits 24-31: Driver number

On return

R1 Updated colour

R4 0

- All other registers preserved

[RISC OS Open source page](#)

GraphicsV 13

GraphicsV

The kernel issues this call to instruct the driver to perform a hardware-accelerated rendering task. The appropriate driver should respond to the call in the following manner:

Before calling

R0 Flags:

Bit 0: Synchronise

Bit 1: Synchronise if unclaimed

R1 Render operation:

0 = No-op

1 = Copy rectangle

2 = Fill rectangle

R2 Pointer to operation parameter block

R4 Bits 0-15: 13 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R4 0

- All other registers preserved

In practice

- If the render operation is supported, perform the requested action and set R4 to 0 to claim the call. If bit 0 of R0 is set, synchronise with the video controller to ensure the operation is complete before returning.
- If the render operation is unsupported, leave the call unclaimed. If bit 1 of R0 is set, synchronise with the video controller to ensure that any previous render operations have completed.
- If no hardware acceleration is supported at all, the driver can safely ignore the call.

Notes

Screen memory will typically be bufferable – GraphicsV driver implementations should make sure to perform the necessary memory barrier operations at the start/end of any given acceleration op (e.g. write barrier at start, read barrier at end). Suitable barrier routines are available via OS_MMUControl 2.

If no driver responds to the request, then there is no generic software fallback in place. Software rendering fallbacks must be handled on a case-by-case basis by the caller. Therefore, any user software which wishes to make use of hardware acceleration should either do so via the VDU interface (e.g. use block copy/fill OS_Plot operations, which will have the appropriate fallbacks in place inside the kernel), or call GraphicsV directly and use their own software fallback for any unsupported operations. Just because one call succeeds, it does not guarantee that the next call will succeed, or that a call made with slightly different parameters will succeed.

[RISC OS Open source page](#)

GraphicsV 13 0

GraphicsV

This is a no-op operation that can be used whenever synchronisation with the video controller is required. Other than the flags in R0, this operation takes no parameters.

Before calling

R0 Flags:

Bit 0: Synchronise

Bit 1: Synchronise if unclaimed

R1 0 (render operation)

R4 Bits 0-23: 13 (GraphicsV reason code)

Bits 24-31: Display number

On return

R4 0

- All other registers preserved

Notes

See GraphicsV 13 for general information on how drivers are expected to respond to GraphicsV 13 calls.

[RISC OS Open source page](#)

GraphicsV 13 1

GraphicsV

This operation is used for rectangle block copies/moves. All coordinates are specified in pixels, with 0,0 being the bottom left corner of the screen, and positive values heading right/up.

Before calling

R0 Flags:

Bit 0: Synchronise

Bit 1: Synchronise if unclaimed

R1 1 (render operation)

R2 Pointer to parameter block

R4 Bits 0-23: 13 (GraphicsV reason code)

Bits 24-31: Display number

On return

R4 0

- All other registers preserved

Parameter block format

Word / Value

0 – Value: Source left edge

- 1 – Value: Source bottom edge
- 2 – Value: Dest left edge
- 3 – Value: Dest bottom edge
- 4 – Value: Width-1
- 5 – Value: Height-1

In practice

The source and destination rectangles are guaranteed by the caller to fit entirely on screen, but they may be overlapping.

Notes

See GraphicsV 13 for general information on how drivers are expected to respond to GraphicsV 13 calls.

[RISC OS Open source page](#)

GraphicsV 13 2

GraphicsV

This operation is used for rectangle fills. All coordinates are specified in pixels, with 0,0 being the bottom left corner of the screen, and positive values heading right/up. The rectangle is guaranteed to fit entirely on screen.

Before calling

R0 Flags:

Bit 0: Synchronise

Bit 1: Synchronise if unclaimed

R1 2 (render operation)

R2 Pointer to parameter block

R4 Bits 0-23: 13 (GraphicsV reason code)

Bits 24-31: Display number

On return

R4 0

- All other registers preserved

Parameter block format

Word / Value

0 – Value: Left edge (inclusive)

1 – Value: Top edge (inclusive)

2 – Value: Right edge (inclusive)

3 – Value: Bottom edge (inclusive)

4 – Value: Pointer to ECF Block containing fill pattern

In practice

The ECF block is aligned on a 64 byte boundary.

Notes

See GraphicsV 13 for general information on how drivers are expected to respond to GraphicsV 13 calls.

[RISC OS Open source page](#)

GraphicsV 14

GraphicsV

This call is used to perform IIC operations on the IIC busses managed by the driver. Typically this will be DDC or EDID data from the attached monitor, for example the IIC base address for reading EDID data is 0xA1.

Before calling

R0 Bits 0-15: Offset within device

Bits 16-23: IIC base address

Bits 24-31: Reserved

R1 Pointer to buffer

R2 Byte count

R4 Bits 0-15: 14 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R0 Return code

R1 Advanced by how many bytes read/written

R2 Number of bytes not transferred

R4 0

- All other registers preserved

Notes

Due to its behaviour, it is not possible for a user of GraphicsV to make use of the EDID segment pointer by performing a write to bus address &60. Instead, it is the drivers responsibility to (where possible) automatically write to the segment pointer when asked to access an EDID address ≥ 256 .

To protect against accidental corruption of EDID, it's recommended that drivers do not allow EDID writes (i.e. operations to bus address &A0).

[RISC OS Open source page](#)

GraphicsV 15

GraphicsV

This call is used to switch the display output from one graphics card head to another.

Before calling

R0 Head number (0-based)

R4 Bits 0-23: 15 (reason code)

Bits 24-31: Driver number

On return

R0 0 or error block pointer

R4 0

- All other registers preserved

[RISC OS Open source page](#)

GraphicsV 16

GraphicsV

This call allows a GraphicsV driver to override the choice of screen mode in certain situations, e.g. on driver startup, or the default mode used by the Wimp. For example, a device with a fixed size LCD may want to select a mode which fills the display.

Before calling

R0 Suggested mode number or Mode Selector Block

R4 Bits 0-15: 16 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R0 Mode number/selector to use

R4 0

- All other registers preserved

Notes

The kernel issues this call from within OS_ReadSysInfo 1. However, it will only issue it under certain circumstances – see the OS_ReadSysInfo page for details.

If the mode returned refers to a mode definition which is not one of the kernel's numbered modes, a suitable mode definition needs to have been loaded, e.g. from a ResourceFS based built in MDF. Service_ModeExtension can then subsequently find it and provide the relevant settings. If there is not a mode definition file, a custom Service_ModeExtension response needs to be provided.

[RISC OS Open source page](#)

GraphicsV 17

GraphicsV

This call is used to read a full list of the pixel formats supported by the device. This call uses the NColour, ModeFlags and Log2BPP Mode Variable Numbers to describe the supported pixel formats. Each entry in the returned list is a 3-word structure with the following format:

Before calling

R4 Bits 0-15: 17 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R0 Pointer to list of pixel formats

R1 Number of formats

R4 0

- All other registers preserved

Word / Value

0 – Value: NColour value

1 – Value: ModeFlags value

2 – Value: Log2BPP value

Notes

See Valid Mode Variable Combinations for a list of all valid pixel formats currently understood by the OS.

This call should be used in preference to using GraphicsV 8 to read the supported pixel formats. However, for backwards compatibility, drivers must ensure that GraphicsV 8 always returns a suitable value; i.e. just a mask of the VIDC20/RISC OS 3.5 compatible pixel formats that the driver supports.

[RISC OS Open source page](#)

GraphicsV 18

GraphicsV

This call is used to read assorted pieces of information about a driver into a user-supplied buffer. On exit from the call, R2 will have been decreased by the required buffer length; therefore to determine the required buffer length you should first call with R2=0.

Before calling

R0 Item

R1 Pointer to word-aligned buffer for result (if R2 > 0)

R2 Buffer length

R4 Bits 0-23: 18 (reason code)

Bits 24-31: Driver number

On return

R2 R2 on entry, minus required buffer length

R4 0

Buffer contents updated

All other registers preserved

Items

The supported items are as follows:

/

0 – : Driver version number as a binary coded decimal integer. If the driver is implemented as a module then this should be the module version number.

1 – : Module name, including instance if necessary (e.g. 'NVidia%1'). This item should not be implemented if the driver is not a module.

2 – : Driver name, for display to the user. Often the module name will suffice

3 – : Hardware name, for display to the user. E.g. the PCI card or USB device name

4 – : List of known VIDC control list items, as a series of words, terminated by -1. Drivers will ignore control list items which are not listed here – it is the clients responsibility to check that any critical control list items are supported by the driver when preparing a VIDC list for the mode set or vet calls.

5 – : Maximum number of overlays supported. Reading this item may return zero or be ignored by drivers which do not support overlays.

Notes

All text items are returned as null-terminated ASCII strings.

If R2 is smaller than the required buffer size, only the first R2 bytes of the value will be copied to the buffer.

[RISC OS Open source page](#)

GraphicsV 19

GraphicsV

This is an enhanced version of the GraphicsV 7 call, designed to allow the driver to provide additional feedback to the OS, mainly focusing on describing the framebuffer memory requirements for the mode.

Before calling

R0 Pointer to VIDC List Type 3

R1 Flags (reserved; specify 0)

R4 Bits 0-15: 19 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R0 Result flags:

Bits 0-1:

- 0 Provided mode is unsupported
- 1 Mode is supported, and will use the system framestore (i.e. kernel-managed DA 2)
- 2 Mode is supported, and will use the external framestore (as defined by R3 & R5)
- 3 Mode is supported, but driver doesn't know where the framestore will be (e.g. framestore is managed by another component, like the GPU firmware in the Raspberry Pi)

Bit 2: The input value of the ExtraBytes control list item was invalid, driver has suggested a new value in R2

Bits 3+: Reserved

R1 Required alignment of start of framebuffer

R2 Required/suggested ExtraBytes value

R3 External framebuffer base (iff bits 0-1 of R0 are 2)

R4 0

R5 External framebuffer size (iff bits 0-1 of R0 are 2)

- All other registers preserved

Implementation within drivers

New drivers are expected to support both GraphicsV 19 and the older GraphicsV 7 call, and any mode that passes GraphicsV 19 vetting must also pass GraphicsV 7 vetting. Since GraphicsV 19 provides a superset of the functionality of GraphicsV 7, the recommended procedure for vetting a mode is that code should call GraphicsV 19 first, and only fall back to GraphicsV 7 if the call wasn't recognised by the driver (R4 nonzero) or if an unhandled result is returned (e.g. R0 has unrecognised flag bits set).

Drivers should handle this call as follows:

- They should perform a mode vet procedure, as per GraphicsV 7. However, instead of being restricted to the value of the ExtraBytes control list item that is specified in the input, the driver is free to use an ExtraBytes value of its own choosing, subject to the rules described below.
- When preparing the result registers, bits 0-1 of R0 should be set to the appropriate value depending on whether the mode is supported, and the type/location of framestore that will be used.
- If the mode is supported (bits 0-1 non-zero): R1 must specify the required alignment of the framebuffer (as per GraphicsV 8) R2 must specify a valid ExtraBytes value. If the mode couldn't be supported using the input ExtraBytes value, bit 2 of R0 must be set to indicate that the new value in R2 should be used instead. If the mode is to use an external framestore (bits 0-1 of R0 are 2), R3 and R5 must describe that framestore, as per GraphicsV 9.
- The driver must indicate that the call has been claimed by setting R4 to zero

Of course, if a caller does not care about the new functionality introduced by GraphicsV 19, and cannot support modifications to the ExtraBytes item, there is no harm in only using GraphicsV 7.

Rules for selecting an ExtraBytes value

Typically, the ExtraBytes value returned in R2 will just be the same value that was provided in the input VIDC list (or the default of 0 if the item was absent). However, if using the input ExtraBytes value would mean that the mode can't be supported, the driver can return a different ExtraBytes value in R2 and set bit 2 of R0 to indicate this condition.

Note that the returned ExtraBytes value must be greater than the input value; if a smaller value is required, then the driver must fail the entire vet call by setting bits 0-1 of R0 to zero.

There's also a third option available: The driver can return a new ExtraBytes value in R2, but leave bit 2 of R0 clear. This allows the driver to hint that a different ExtraBytes value to what was provided in the input may provide better performance. In this case, it's valid for the driver to return an ExtraBytes value that was smaller than the input value.

If the ExtraBytes value can affect the choice of framestore location, then the driver should return its result as follows:

- If bit 2 of R0 is clear, the framestore information returned in R1, R3 & R5 must reflect the framestore that would be used if the input ExtraBytes value is used
- If bit 2 of R0 is set, the framestore information returned in R1, R3 & R5 must reflect the framestore that would be used if the output ExtraBytes value is used

Drivers that support GraphicsV 19 must support vetting of the ExtraBytes control list item (so it must be in the list of supported items returned by GraphicsV 18).

Notes

For future-proofing, the driver must ensure it checks that the provided VIDC list is a type 3 list.

The input flags that are provided in R1 are reserved for future expansion. They are intended to be used to signal that the caller supports extended return results from the driver (e.g. to signal that additional registers may be used to return results). Therefore, drivers should simply ignore any flag bits in R1 which they don't recognise.

Instead of calling GraphicsV directly, user software should use OS_CheckModeValid in order to check if a screen mode is valid.

[RISC OS Open source page](#)

GraphicsV 2

GraphicsV

The kernel issues this call in order to instruct the display driver to change the display mode. The appropriate driver should respond by changing to the requested mode and setting R4 to 0 to claim the call.

Before calling

R0 VIDC List Type 3

R4 Bits 0-15: 2 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R4 0

- All other registers preserved

Notes

Prior to changing the mode, the OS will always call GraphicsV 7 to check that the driver supports the mode. Drivers must implement the vet mode call to allow them to object to unsupported modes, as there is currently no way for a driver to refuse a mode change when GraphicsV 2 is called.

Any Overlays created via GraphicsV 20 should be destroyed before returning from this call.

[RISC OS Open source page](#)

GraphicsV 20

GraphicsV

Creates a new hardware overlay.

Before calling

R0 Pointer to Mode Selector Block describing the required properties of the overlay

R1 Aspect ratio to use when calculating returned min/max scale values

Top 16 bits = width

Bottom 16 bits = height

R2 Flags

Bit 0: Overlay must support scaling

Bits 1-31 reserved, should be zero

R3 Mask of which heads the overlay is to be used with

R4 Bits 0-23: 20 (reason code)

Bits 24-31: Driver number

On return

R0 Overlay number

R1 Flags:

Bits 0-7: Overlay type (0 = Z-Order, 1 = Basic, 2+ reserved)

Bits 8-31: Reserved

R2 Minimum width overlay can be scaled to

R3 Minimum height overlay can be scaled to

R4 0

R5 Maximum width overlay can be scaled to

R6 Maximum height overlay can be scaled to

- All other registers preserved

In practice

For rotated overlays, the aspect ratio and the returned width & height values correspond to the rotated form of the overlay, i.e. as it will appear on screen.

GraphicsV drivers are not required or expected to support dynamic allocation of buffers – if multiple buffers are desired, the necessary number of buffers must be specified in the mode selector via mode variable 13.

Notes

If an overlay supports alpha formats, it's expected that it will be alpha-blended by the hardware. TCK for overlays currently isn't supported (only the desktop should use TCK).

To account for driver restrictions and to allow for more optimal memory management, all overlays for a driver will be destroyed when the driver changes mode.

Because some GraphicsV reason codes can accept either a driver number or a head number in bits 16-23 of R4, drivers should choose overlay numbers such that they don't overlap the range of head numbers.

Overlay types

Overlay hardware falls into two basic categories: "Z-Order" and "Basic". Z-Order hardware allows full control of the Z order of the created overlays. Basic hardware offers no control of the Z order, and it's expected that they're placed somewhere between the desktop and the mouse pointer.

Currently the API restricts Z-Order overlays to be placed underneath the desktop. Alpha-blending or transparency colour key (TCK) must be used to mark out the desktop areas through which the overlay(s) should be visible.

Because Basic overlays are on top of the desktop, software must take care to hide them when necessary, e.g. if an error message appears which the user must be able to see.

Mode Selector Block

The Mode Selector Block is expected to include some additional Mode Variable Numbers. Refer to Extended Framebuffer Format Specification for ModeFlags and NColour values:

Value / Name / Meaning

0 – Name: ModeFlags · Meaning: Data format and colour space information

3 – Name: NColour · Meaning: Maximum logical colour

13 – Name: MinScreenBanks · Meaning: Optional. If omitted, 1 overlay bank should be assumed

[RISC OS Open source page](#)

GraphicsV 21

GraphicsV

Destroys a hardware overlay, including freeing/discarding all banks.

Before calling

R4 Bits 0-15: 21 (reason code)

Bits 16-23: Overlay number

Bits 24-31: Driver number

On return

R4 0

- All other registers preserved

In practice

Specifying an overlay number of zero will destroy all overlays for the indicated driver.

[RISC OS Open source page](#)

GraphicsV 22

GraphicsV

Displays or hides an overlay. The coordinate space is that of logical screen coordinates, i.e. (0,0) is the top-left corner of the screen from the viewer's perspective (if the screen is rotated, (0,0) is still the top-left). Within that coordinate space, the coordinates specify the top-left corner of the overlay, as it appears on screen (e.g. if the overlay is rotated 180 degrees then it would actually correspond to the bottom-right pixel of the buffer).

Before calling

R0 Pointer to coordinate array, where each entry is the following 3-word structure:

Word 0: Head number

Word 1: X position

Word 2: Y position

R1 Number of entries in coordinate array

R2 Bank to display (ignored if R1=0)

R4 Bits 0-15: 22 (reason code)

Bits 16-23: Overlay number

Bits 24-31: Driver number

On return

R4 0

- All other registers preserved

In practice

The overlay will only appear on the heads which are included in the list. Therefore, you can hide the overlay completely by specifying R1=0.

[RISC OS Open source page](#)

GraphicsV 23

GraphicsV

Maps an overlay bank/buffer into logical address space, potentially allocating the memory for it if the driver hasn't already done so.

Before calling

R0 Bank number

R4 Bits 0-15: 23 (reason code)

Bits 16-23: Overlay number

Bits 24-31: Driver number

On return

R0 Pointer to array giving buffer addresses:

Word 0: Plane 0 base logical address

Word 1: Plane 0 stride between image rows (in bytes)

Words 2+: Logical addresses and strides for any additional planes

R4 0

- All other registers preserved

In practice

If this is a freshly-allocated buffer, the contents are undefined. Otherwise, the contents will be preserved.

Drivers may have limited logical address space available for mapping overlays, so in Wimp situations, it's recommended to unmap your overlay(s) before allowing the Wimp to swap out your task.

The array returned in R0 will remain valid until the bank is unmapped.

[RISC OS Open source page](#)

GraphicsV 24

GraphicsV

Unmaps an overlay buffer from the logical address space.

Before calling

R0 Bank number

R4 Bits 0-15: 24 (reason code)

Bits 16-23: Overlay number

Bits 24-31: Driver number

On return

R4 0

- All other registers preserved

[RISC OS Open source page](#)

GraphicsV 25

GraphicsV

Discards a buffer, allowing the driver to potentially free the memory that was allocated to it. If necessary, the buffer will be unmapped first. If the overlay is currently displaying this buffer, then the overlay will be hidden.

Before calling

R0 Bank number

R4 Bits 0-15: 25 (reason code)

Bits 16-23: Overlay number

Bits 24-31: Driver number

On return

R4 0

- All other registers preserved

[RISC OS Open source page](#)

GraphicsV 26

GraphicsV

Allows the user to check whether an overlay with the specified properties can be created, and if so, what capabilities it will have. If the overlay can't be supported, the driver should not claim the call.

Before calling

R0 Pointer to Mode Selector Block describing the required properties of the overlay

R1 Aspect ratio to use when calculating returned min/max scale values

Top 16 bits = width

Bottom 16 bits = height

R2 Flags

Bit 0: Overlay must support scaling

Bits 1-31 reserved, should be zero

R3 Mask of which heads the overlay is to be used with

R4 Bits 0-23: 26 (reason code)

Bits 24-31: Driver number

On return

R0 Overlay number

R1 Flags:

Bits 0-7: Overlay type (0 = Z-Order, 1 = Basic, 2+ reserved)

Bits 8-31: Reserved

R2 Minimum width overlay can be scaled to

R3 Minimum height overlay can be scaled to

R4 0

R5 Maximum width overlay can be scaled to

R6 Maximum height overlay can be scaled to

- All other registers preserved

In practice

When vetting the settings, the driver should take into account any restrictions that result from the current system configuration. E.g. if all overlays are currently in use, no Vet call should succeed. Or conversely, if a Vet call succeeds, so too should a corresponding Create call.

[RISC OS Open source page](#)

GraphicsV 27

GraphicsV

Adjusts the scale and clipping rectangle for an overlay. R5 and R6 specify the size the overlay should be scaled to (in screen pixels). Then, the clipping rectangle defined by R0-R3 will be used to crop the displayed area. This is also specified in units of screen pixels. For rotated overlays, the values correspond to the rotated form, i.e. as it will appear on screen. Rounding errors or other hardware limits may prevent the chosen values from being used, in which case the driver will make the necessary adjustments (while maintaining the indicated aspect ratio) and return the actual values used.

Before calling

R0 Left edge of clipping rectangle (inclusive)

R1 Top edge of clipping rectangle (inclusive)

R2 Right edge of clipping rectangle (exclusive)

R3 Bottom edge of clipping rectangle (exclusive)

R4 Bits 0-15: 27 (reason code)

Bits 16-23: Overlay number

Bits 24-31: Driver number

R5 Scaled width

R6 Scaled height

R7 Aspect ratio to use if image size needs adjusting

Top 16 bits = width

Bottom 16 bits = height

On return

R0 Actual left edge of clipping rectangle

R1 Actual top edge of clipping rectangle

R2 Actual right edge of clipping rectangle

R3 Actual bottom edge of clipping rectangle

R4 0

R5 Actual scaled width

R6 Actual scaled height

- All other registers preserved

[RISC OS Open source page](#)

GraphicsV 28

GraphicsV

Specifies the sorting order for any Z-Order overlays. The first entry in the list corresponds to the frontmost overlay. The effect on the order of any overlays not included in the list is unspecified.

Before calling

R0 Pointer to byte array containing overlay IDs

R1 Length of array

R4 Bits 0-23: 28 (reason code)

Bits 24-31: Driver number

On return

R4 0

- All other registers preserved

[RISC OS Open source page](#)

GraphicsV 3

GraphicsV

The kernel issues this call in order to instruct the display driver to enable or disable interlaced output. The appropriate driver should respond by performing the requested action and setting R4 to 0 to claim the call.

Before calling

R0 0: Interlace off

1: Interlace on

R4 Bits 0-15: 3 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R4 0

- All other registers preserved

Notes

This call is obsolete.

- Drivers which support interlace are not required to implement this call
- Clients which wish to specify interlace settings should do so via the VIDC list given to GraphicsV 2/7.

[RISC OS Open source page](#)

GraphicsV 4

GraphicsV

The kernel issues this call in order to instruct the display driver to enable or disable the display, putting the display into the indicated DPMS state if possible. The appropriate driver should respond by performing the requested action and setting R4 to 0 to claim the call.

Before calling

R0 0: Unblank display

1: Blank display

R1 DPMS State when blanking display

R4 Bits 0-15: 4 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R4 0

- All other registers preserved

Notes

When the display is blanked, it's permissible for drivers to take additional steps to reduce power consumption, providing that they do not conflict with the intended DPMS state of the monitor. E.g. it's reasonable to expect a driver to disable video DMA or hide any visible overlays. For digital interfaces it may be possible to gate the delivery of the pixel clock, and when entering DPMS state 3 it may be possible to power-down the GPU completely.

When the display is blanked, there is no requirement for the driver to continue to deliver VSync interrupts.

[RISC OS Open source page](#)

GraphicsV 5

GraphicsV

The kernel issues this call during VSync when the pointer shape or position changes. The appropriate driver should respond by performing the requested action and setting R4 to 0 to claim the call.

Before calling

R0 Flags

Bit 0 = pointer display enable (0=off, 1=on)

Bit 1 = pointer shape update (0=no change, 1=updated)

Bits 2-31 reserved

R1 X position of pointer

R2 Y position of pointer

R3 shape_t descriptor block

R4 Bits 0-15: 5 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R4 0 if request handled by hardware pointer, else preserved if software pointer required

- All other registers preserved

In practice

The coordinates given in R1 and R2 are measured in pixels, and indicate the top-left corner of the pointer image. 0,0 would place the image in the top-left corner of the screen.

R3 points to a shape_t descriptor block:

```
typedef struct shape_t
{
    uint8 width; /* unpadded width in bytes (see notes) */
    uint8 height; /* in pixels */
    uint8 padding[2]; /* 2 bytes of padding for field alignment */
    void *buffLA; /* logical address of buffer holding pixel data */
    void *buffPA; /* corresponding physical address of buffer */
}
```

Notes

- If R0 bit 0 is 0 (pointer off), R1-R3 are undefined
- The shape data from RISC OS is always padded with transparent pixels on the rhs, to a width of 32 pixels (8 bytes)
- Pointer clipping is the responsibility of the video driver (e.g. may be able to allow display of pointer in border region on some h/w)
- The buffer for the pixel data is aligned to a multiple of 256 bytes or better.

A simple video driver, where hardware permits, can use the shape data in the buffer directly, ignoring the updated flag. The OS guarantees that the buffer data is valid for the whole time it is to be displayed. Other drivers may need to make copies of the data, e.g. for conversion to different pixel formats.

Software pointer support

If a driver does not provide full support for a hardware pointer, then it can indicate this by having GraphicsV 8 return with bit 1 of R0 clear. The OS will then render a software pointer whenever a call to GraphicsV 5 comes back as unclaimed (R4 unchanged). This allows for selective use of the software pointer, e.g. if the hardware pointer can only be used in some portions of the screen. Note that the driver must take care to disable the hardware pointer when a request is received to move it into an area in which it cannot be displayed.

[RISC OS Open source page](#)

GraphicsV 6

GraphicsV

The kernel issues this call when it requires the driver to update the DAG ("DMA address generator") registers of the display hardware. The appropriate driver should respond by performing the requested action and setting R4 to 0 to claim the call.

Before calling

R0 DAG:

0 = Set start address of current video display

1 = Set start address of total video buffer

2 = Set end address (exclusive) of total video buffer

3 = Set start address of VDU buffer (i.e. as set by OS_Byte 112)

All other values reserved

R1 Physical address for given DAG

R4 Bits 0-15: 6 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R4 0

- All other registers preserved

In practice

The OS has a video buffer which is $\geq$ total display size, and may be using bank switching (several display buffers) or hardware scroll within the total video buffer.

- DAG=1 will be start address of current total video buffer
- DAG=2 will be end address (exclusive) of current total video buffer
- DAG=0 will be start address in buffer for current display
- DAG=3 is the start address of the buffer to be used for GraphicsV Render ops (GraphicsV 13)

Video drivers should respond differently depending on whether hardware scroll is supported or not. (The OS will already know this from GraphicsV 8).

No hardware scroll

Only DAG=0 is significant, and the end address of the current display is implied by the size of the current mode. Calls with DAG=1,2 should be ignored.

Hardware scroll

DAG=0 again defines display start. DAG=2 defines the last address (exclusive) that should be displayed before wrapping back (if reached within display size), and DAG=1 defines the address to which accesses should wrap back.

Notes

Prior to RISC OS 5.19, the kernel had a bug where the DAG addresses sent to the vector would be incorrect if an external framestore was in use. Instead of the addresses pointing to the external framestore, they pointed to the internal framestore that the kernel would be using if the external framestore wasn't in use. This internal framestore has the same physical base address of the first page of RAM/VRAM in the system; its address can be read by asking OS_Memory 0 to look up the physical address of page number 0.

[RISC OS Open source page](#)

GraphicsV 7

GraphicsV

The kernel issues this call to check whether a driver can support a proposed mode. The appropriate driver should respond by checking the mode parameters, setting R0 appropriately, and setting R4 to 0 to claim the call.

Before calling

R0 Pointer to VIDC List Type 3

R4 Bits 0-15: 7 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R0 Zero: Mode OK

Non-zero: Mode not OK

R4 0

- All other registers preserved

Notes

GraphicsV 19 is an improved version of this call, which allows the driver to provide additional feedback on the memory requirements of the framebuffer.

For future-proofing, the driver must ensure it checks that the provided VIDC list is a type 3 list.

Instead of calling GraphicsV directly, user software should use OS_CheckModeValid in order to check if a screen mode is valid.

[RISC OS Open source page](#)

GraphicsV 8

GraphicsV

The kernel issues this to determine the available features of a driver. The appropriate driver should respond by setting R4 to 0 to claim the call, and setting R0-R2 to the appropriate values.

Before calling

R4 Bits 0-15: 8 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R0 Feature flags:

Bit 0: Supports hardware scroll

Bit 1: Supports hardware pointer

Bit 2: Supports interlace with progressive framestore

Bit 3: Uses separate framestore (GraphicsV 9 implemented)

Bit 4: No VSyncs generated

Bit 5: Framestore address changes on mode change (bit 3 must be set)

Bit 6: Rectangle copies (via GraphicsV 13 op 1) are faster than RAM to VRAM transfers using CPU

R1 Mask of supported pixel formats:

Bit 0: 1bpp palletised

Bit 1: 2bpp palletised

Bit 2: 4bpp palletised

Bit 3: 8bpp palletised

Bit 4: 16bpp RGB (C32K LTBGR)

Bit 5: 32bpp RGB (C16M LTBGR)

R2 Display buffer alignment requirement in bytes (power of 2)

R4 0

- All other registers preserved

Notes

Bit 2 of R0 indicates whether a full-resolution interlace mode (VIDC List Type 3 sync/pol bit 4 set) or a half-resolution one (i.e. 2x upscaling). If the bit is set, the driver is indicating that full-resolution interlace is supported directly. If the bit is clear, the driver is indicating that full-resolution interlace is only possible with additional software support (e.g. the NCOS Interlace module, which would use the MMU to remap screen memory so that two independent "odd" and "even" physical framebuffers can appear as a single interlaced framebuffer by software).

Using this call to read the supported pixel formats is deprecated. Instead, software should prefer to use GraphicsV 17 to read the supported formats, only falling back to this call if that call is unimplemented.

Additionally, bits 4 and 5 of R1 should reflect the availability of those exact formats indicated – bit 4 should only be set if traditional VIDC20 32K colour modes are supported, and bit 5 should only be set if traditional VIDC20 16M colour modes are supported.

See GraphicsV 5 for details on the handling of bit 1 of R0.

Bit 6 of R0 is used by OS_SpriteOp 65 to determine whether to accelerate rendering using GraphicsV (specifically, OS_Plot rectangle copy) or whether to render all the sprites manually.

[RISC OS Open source page](#)

GraphicsV 9

GraphicsV

If a driver indicates via GraphicsV 8 that it has a separate framestore then the kernel will issue this call to determine the framestore size and location. The appropriate driver should respond by setting R4 to 0 to claim the call, and setting R0-R1 to the correct values.

Before calling

R4 Bits 0-15: 9 (reason code)

Bits 16-23: Head number

Bits 24-31: Driver number

On return

R0 Framestore physical base address

R1 Framestore size

R4 0

- All other registers preserved

Notes

Drivers which don't have separate framestores should ignore this call and leave it unclaimed.

If a driver indicates via GraphicsV 8 that the framestore size/address changes on mode change, then the kernel will make this call after every mode change in order to read back the updated framestore details. Otherwise, the kernel will assume the same value is always returned, even if a mode hasn't been set yet.

[RISC OS Open source page](#)

HAL Hardware

ARM Hardware

The Hardware Abstraction Layer that was introduced to RISC OS 5 means that RISC OS is no longer dependent on a fixed set of core chips. Thus this section of the manual will describe the bare minimum hardware required to support RISC OS, and then provide links to more detailed information about actual hardware platforms that run RISC OS.

ARM Processor

The ARM processor must implement version 3 (or above) of the ARM architecture, and must support the ARM instruction set. Support for the Thumb instruction set is not required. RISC OS will use the processor solely in 32bit mode; support for 26bit mode is not required. Finally, the processor must have an integrated MMU.

TODO - EXPAND WITH MORE DETAILS, SIMILAR TO THE ARCHIMEDES HARDWARE PAGE

Memory

Like previous versions of RISC OS, RISC OS 5 only requires a small amount of memory to be functional. The exact minimum is currently unknown, but it is expected to be around 1MB.

Other hardware

The only other hardware that RISC OS absolutely requires is a timer capable of generating 100Hz interrupts on the processor's IRQ line.

All other hardware (keyboard, mouse, audio, video, disc drives, etc.) is optional and not strictly required for a working ROM image. Drivers built into the machine-specific HAL and module chain are used to talk to this additional hardware, and so by customising the ROM image appropriately almost any type of hardware can be used for I/O devices.

Details of specific machines

At present there are six basic machine types that have successfully run versions of RISC OS 5:

- IYONIX pc
- Various set-top boxes
- BeagleBoard and other similar OMAP3-based devices
- RiscPC and A7000, A7000+
- Raspberry Pi
- Pandaboard

[RISC OS Open source page](#)

HAL Video API

Hardware Abstraction Layer

The HAL only attempts to abstract the hardware controller aspects of the OS video. It does not (yet) consider pixel formats, framestore layout, hardware graphics acceleration. All these would affect a great deal of RISC OS graphics code that forms much of the value of the OS. This means that the envisaged HAL/RISC OS combination makes some specific assumptions about graphics framestore layout as follows:

In practice

- Memory mapped framestore
- Expected to be contiguous physical memory, can be specific memory (eg. VRAM)
- Mapped as contiguous logical memory
- Progressive raster scan in logical memory from top left pixel to bottom right
- Start of each raster row must be word aligned
- Number of pixels in a row should be such that row is a whole number of words
- Spacing between start of each row is a constant number of words, possibly greater than row length (via mode variable, LineLength)
- 1,2,4,8,16 or 32 bits per pixel (bpp)
- Little endian pixel packing for 1,2,4 bpp (least significant bits are leftmost pixels)
- Presence of palette assumed for 1,2,4,8 bpp (8-bits per r,g,b component in each entry)
- 16 bpp format:

Bits / Meaning

0-4 – Meaning: Red

5-9 – Meaning: Green

10-14 – Meaning: Blue

15 – Meaning: Supremacy (0=solid, 1=transparent)

- 32 bpp format:

Bits / Meaning

0-7 – Meaning: Red

8-15 – Meaning: Green

16-23 – Meaning: Blue

24-31 – Meaning: Supremacy (0=solid, 255=transparent)

- Palette words are 32 bits:

Bits / Meaning

0-7 – Meaning: Reserved (0), or Supremacy (0=solid, 255=transparent)

8-15 – Meaning: Red

16-23 – Meaning: Green

24-31 – Meaning: Blue

- Pointer/cursor is assumed supported in hardware, 32×32 pixels, each pixel either transparent or one of 3 paletted colours
- Support for physically interlaced, logically progressive framestore via MMU tricks and use of LineLength mode variable, currently not fully integrated into kernel

Note that it is possible to support hardware where only some pixel depths are available, or only some fit the RISC OS assumptions. Also some hardware has some configurability for 'arbitrary' choices like RGB versus BGR ordering. Hence, the restrictions are typically much less severe than might first be thought.

Supporting a software only pointer/cursor is feasible (much less work than new pixel formats) but not yet considered.

Entry points

Items marked ¶ are internal calls for OS use only; their functionality and availability is subject to change without warning. User programs interested in using these calls should instead use the equivalent RISC OS SWI call (if available).

- HAL_VideoFlybackDevice ¶
- HAL_VideoSetMode ¶
- HAL_VideoWritePaletteEntry ¶
- HAL_VideoWritePaletteEntries ¶
- HAL_VideoReadPaletteEntry ¶
- HAL_VideoSetInterlace ¶
- HAL_VideoSetBlank ¶
- HAL_VideoSetPowerSave ¶
- HAL_VideoUpdatePointer ¶
- HAL_VideoSetDAG ¶
- HAL_VideoVetMode ¶
- HAL_VideoPixelFormat ¶
- HAL_VideoFeatures ¶
- HAL_VideoBufferAlignment ¶
- HAL_VideoOutputFormat ¶
- HAL_VideoRender ¶
- HAL_VideoIICOp ¶
- HAL_VideoFramestoreAddress ¶
- HAL_VideoStartupMode ¶
- HAL_VideoPixelFormatList ¶

Minimal implementation

Entry HAL_VideoPixelFormat will default to 0 if not implemented, and since RISC OS 5.21 that means none of the other entry points will ever be called.

Integration with GraphicsV

Starting with RISC OS 5.21, on startup the OS will issue a call to HAL_VideoPixelFormat to detect if the HAL video API is implemented. If this call returns non-zero then the OS will register a "HAL" driver with GraphicsV and map GraphicsV calls to HAL calls as follows:

- 1 – GraphicsV call: VSync interrupt occurred · HAL call(s): –
- 2 – GraphicsV call: Set mode · HAL call(s): HAL_VideoSetMode
- 3 – GraphicsV call: Set interlace · HAL call(s): HAL_VideoSetInterlace
- 4 – GraphicsV call: Set blank · HAL call(s): HAL_VideoSetBlank
- 5 – GraphicsV call: Update pointer · HAL call(s): HAL_VideoUpdatePointer
- 6 – GraphicsV call: Set DAG · HAL call(s): HAL_VideoSetDAG
- 7 – GraphicsV call: Vet mode · HAL call(s): HAL_VideoVetMode
- 8 – GraphicsV call: Features · HAL call(s): HAL_VideoFeatures (R0)
HAL_VideoPixelFormat (R1)
HAL_VideoBufferAlignment (R2)
- 9 – GraphicsV call: Framestore information · HAL call(s): HAL_VideoFramestoreAddress
- 10 – GraphicsV call: Write palette entry · HAL call(s): HAL_VideoWritePaletteEntry
- 11 – GraphicsV call: Write palette entries · HAL call(s): HAL_VideoWritePaletteEntries

12 – GraphicsV call: Read palette entry · HAL call(s): HAL_VideoReadPaletteEntry

13 – GraphicsV call: Render · HAL call(s): HAL_VideoRender

14 – GraphicsV call: IIC op · HAL call(s): HAL_VideoIICOp

15 – GraphicsV call: Select head · HAL call(s): Unsupported

16 – GraphicsV call: Select startup mode · HAL call(s): HAL_VideoStartupMode

17 – GraphicsV call: List pixel formats · HAL call(s): HAL_VideoPixelFormatList

For VSync events, the OS will use HAL_VideoFlybackDevice to determine the IRQ device number to respond to, and will call GraphicsV 1 as appropriate whenever the interrupt fires.

Note that HAL_VideoSetPowerSave is currently not available via GraphicsV.

Prior to RISC OS 5.21, any calls made to GraphicsV which were for driver zero and were not handled by a driver module would be forwarded onto the HAL. Additionally, some HAL video calls were required to be implemented for module-based drivers to operate correctly.

[RISC OS Open source page](#)

HAL_VideoBufferAlignment

HAL entry #51 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

-

On return

- Returns the required alignment for the framestore buffer, in bytes (Expected to be a power of 2)

uint HAL_VideoBufferAlignment(void)

In practice

Determine the framestore buffer alignment required by the hardware.

[RISC OS Open source page](#)

HAL_VideoFeatures

HAL entry #50 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

-

On return

As per R0 of GraphicsV 8

int HAL_VideoFeatures(void)

[RISC OS Open source page](#)

HAL_VideoFlybackDevice

HAL entry #38 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

-

On return

- Returns the device number of the video flyback interrupt.

int HAL_VideoFlybackDevice(void)

In practice

Returns the device number of the video flyback interrupt.

[RISC OS Open source page](#)

HAL_VideoFramestoreAddress

HAL entry #116 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

-

On return

- Returns the parameters of the framestore

__value_in_regs struct { uint addr; uint size; } HAL_VideoFramestoreAddress(void)

In practice

Determines the physical address and size of the framestore.

[RISC OS Open source page](#)

HAL_VideoIICOp

HAL entry #120 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

location Location to read from:

b0-15 offset/address within addressed device

b16-23 I2C device address (7 bits + RnW bit)

block Memory block to transfer data to/from

size Pointer to number of bytes to transfer

On return

- Return code

size is updated to reflect the number of bytes actually transferred

int HAL_VideoIICOp(uint32_t location, const uint8_t *block, uint32_t *size)

In practice

This call allows the HAL to implement the functionality of GraphicsV 14.

[RISC OS Open source page](#)

HAL_VideoOutputFormat

HAL entry #52 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

-

On return

- Returns the current video output format:

0 = analogue (or 'normal', unspecified)

1 = 16-bit non-multiplexed RGB 5-6-5 (Chrontel 7003 format 0)

.. (Other Chrontel formats?)

10 = Chrontel 7003 format 9

>=11 reserved

uint HAL_VideoOutputFormat(void)

In practice

This call returns the current video output format.

[RISC OS Open source page](#)

HAL_VideoPixelFormatList

HAL entry #119 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

-

On return

a1-a2 PixelFormatList struct describing available formats

__value_in_regs PixelFormatList HAL_VideoPixelFormatList(void)

In practice

The call returns a list of pixel formats supported by the hardware, in the form of a PixelFormatList struct:

```
typedef struct
{
    uint32_t NColour;
    uint32_t ModeFlags;
    uint32_t Log2BPP;
} PixelFormat;
typedef struct
{
    PixelFormat *formats;
    uint32_t count;
} PixelFormatList;
```

Each pixel format is expressed as a tuple of (NColour, ModeFlags, Log2BPP) mode variables, as used by GraphicsV 17. See Valid Mode Variable Combinations for a full list of pixel formats currently understood by RISC OS.

[RISC OS Open source page](#)

HAL_VideoPixelFormats

HAL entry #49 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

-

On return

Bit 0 1bpp is supported

Bit 1 2bpp is supported

Bit 2 4bpp is supported

Bit 3 8bpp is supported

Bit 4 16bpp is supported

Bit 5 32bpp is supported

Other bits reserved (returned as 0)

int HAL_VideoPixelFormats(void)

In practice

Bits are set for true. Bits 0-5 refer to support with standard RISC OS pixel layout. (such as little endian packing for 1,2,4 bpp, 5-5-5 RGB for 16 bpp, etc). See the HAL Video API section for more information. HAL_VideoPixelFormatList can be used to provide a full list of formats supported by the hardware.

[RISC OS Open source page](#)

HAL_VideoReadPaletteEntry

HAL entry #42 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

type 0 for normal palette entry

1 for border colour

2 for pointer colour

>=3 reserved

pcolours Palette to block of palette entry colours in BBGRRSS format (Blue,Green,Red,Supremacy)

index Index of entry

On return

- Returns effective BBGRRSS

uint HAL_VideoReadPaletteEntry(uint type, uint pcolour, uint index)

In practice

Returns the effective palette entry after taking into account any hardware restrictions in the video controller, assuming it was originally programmed with the value pcolour.

Indices are in the range 0..255 for normal, 0 for border, 0..3 for pointer colours. Note that RISC OS only makes calls using 1..3 for the pointer, and pointer colour 0 is assumed to be transparent.

Depending on hardware capabilities, HALs may have to remember current settings (eg. bits per pixel) or keep soft copies of entries. Because this call supplies the original pcolour, this need is minimised (some HALs can just return pcolour or a directly modified pcolour).

[RISC OS Open source page](#)

HAL_VideoRender

HAL entry #117 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

flags GraphicsV 13 flags

op GraphicsV 13 operation

params GraphicsV 13 parameter block

On return

>=0 Render operation was performed

<0 Render operation was not performed

uint HAL_VideoRender(uint flags, uint op, void *params)

In practice

This call allows the HAL to implement the functionality of GraphicsV 13.

[RISC OS Open source page](#)

HAL_VideoSetBlank

HAL entry #44 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

blank 0 or 1 for unblank or blank

(all other values reserved)

DPMS 0..3 as specified by monitor DPMSState (from mode file)

0 for no DPMS power saving

On return

-

void HAL_VideoSetBlank(uint blank, uint DPMS)

In practice

The HAL is expected to attempt to turn syncs off according to DPMS, and to turn video DMA off for blank (and therefore on for unblank) if possible. The HAL is not expected to do anything else, eg. blank all palette entries. Such things are the responsibility of the OS, and also this call is expected to be fast. May be called with interrupts off.

[RISC OS Open source page](#)

HAL_VideoSetDAG

HAL entry #47 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

DAG 0 = Set start address of current video display

1 = Set start address of total video buffer

2 = Set end address (exclusive) of total video buffer

3 = Set start address of VDU buffer (i.e. as set by OS_Byte 112)

All other values reserved

paddr Physical address for given DAG

On return

-

void HAL_VideoSetDAG(uint DAG, uint paddr)

In practice

The OS has a video buffer which is $\geq$ total display size, and may be using bank switching (several display buffers) or hardware scroll within the total video buffer.

- DAG=1 will be start address of current total video buffer
- DAG=2 will be end address (exclusive) of current total video buffer
- DAG=0 will be start address in buffer for current display

HALs should respond differently depending on whether hardware scroll is supported or not. (The OS will already know this from HAL_VideoFeatures).

No hardware scroll

Only DAG=0 is significant, and the end address of the current display is implied by the size of the current mode. Calls with DAG=1,2 should be ignored.

Hardware scroll

DAG=0 again defines display start. DAG=2 defines the last address (exclusive) that should be displayed before wrapping back (if reached within display size), and DAG=1 defines the address to which accesses should wrap back.

[RISC OS Open source page](#)

HAL_VideoSetInterlace

HAL entry #43 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

interlace 0 or 1 for interlace off or on
(all other values reserved)

On return

-

void HAL_VideoSetInterlace(uint interlace)

In practice

Sets the video interlaced sync.

[RISC OS Open source page](#)

HAL_VideoSetMode

HAL entry #39 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

VIDCList3 VIDC List Type 3

On return

-

```
void HAL_VideoSetMode(const void *VIDCList3)
```

In practice

Programs the video controller to initialise a display mode. RISC OS passes a standard VIDC List Type 3 as specified in PRM 5a-125. Note that this is a generic video controller list, and so VIDC in this context does not refer to any specific devices such as Acorn VIDC20.

The HAL is expected to set the video controller timings on this call. Any palette, pixel DMA and hardware cursor settings are controlled via other calls.

[RISC OS Open source page](#)

HAL_VideoSetPowerSave

HAL entry #45 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

blank 0 or 1 for power save off or on

(all other values reserved)

On return

-

```
void HAL_VideoSetBlank(uint powersave)
```

In practice

The HAL is expected to perform any reasonable measures on the video controller to save power (eg. turn off DACs), when the display is assumed not to be required. Blanking is handled by a separate call.

(What does this really mean. What is acceptable and safe for displays?)

[RISC OS Open source page](#)

HAL_VideoStartupMode

HAL entry #118 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

mode Mode number or Mode Selector Block

On return

- Returns the original mode or an alternative mode, as appropriate

```
const void *HAL_VideoStartupMode(const void *mode)
```

In practice

Allows the HAL to override the startup mode, in a similar way to GraphicsV 16.

[RISC OS Open source page](#)

HAL_VideoUpdatePointer

HAL entry #46 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

flags Bit 0 = Pointer display enable (0=off, 1=on)

Bit 1 = Pointer shape update (0=no change, 1=updated)

Bits 2..31 reserved (0)

xpos X position of top left of pointer (xpos = 0 for left of display)

ypos Y position of top left of pointer (ypos = 0 for top of display)

shape shape_t descriptor block (see below)

On return

0 Request was not handled by hardware pointer

Non-zero Request was handled by hardware pointer

int HAL_VideoUpdatePointer(uint flags, int x, int y, const shape_t *shape)

In practice

shape points to shape_t descriptor block:

```
typedef struct shape_t
{
    uint8 width; /* unpadded width in bytes (see notes) */
    uint8 height; /* in pixels */
    uint8 padding[2]; /* 2 bytes of padding for field alignment */
    void *buffLA; /* logical address of buffer holding pixel data */
    void *buffPA; /* corresponding physical address of buffer */
}
```

Notes:

- If flags bit 0 is 0 (pointer off), x, y, shape are undefined
- The shape data from RISC OS is always padded with transparent pixels on the rhs, to a width of 32 pixels (8 bytes)
- Pointer clipping is the responsibility of the HAL (eg. may be able to allow display of pointer in border region on some h/w)
- Buffer for pixel data is aligned to a multiple of 256 bytes or better

The HAL may need to take note of the shape updated flag, and make its own new copies if true. This is to handle cases like dual scan LCD pointer, which typically needs two or more shape buffers for the hardware, or possibly to handle clipping properly. This work should only be done when the updated flag is true.

A simple HAL, where hardware permits, can use the shape data in the buffer directly, ignoring the updated flag. The OS guarantees that the buffer data is valid for the whole time it is to be displayed.

The return value is used to indicate to the OS whether the software pointer is needed or not. If HAL_VideoFeatures indicates that the hardware pointer is not supported, and this function returns zero, then the OS will render a software pointer instead. This allows for situations where the hardware pointer can only be displayed in some portions of the screen (the HAL

must take care to disable the hardware pointer when it moves into an area in which it cannot be displayed).

[RISC OS Open source page](#)

HAL_VideoVetMode

HAL entry #48 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

VIDClist Generic video controller list (VIDC List Type 3)

workspace Mode workspace (if mode number), or 0

On return

- Returns 0 if OK (may be minor adjusts to VIDClist and/or workspace values)

Non-zero if not OK

int HAL_VideoVetMode(const void *VIDClist, const void *workspace)

In practice

Allows HAL to vet a proposed mode.

(What does this really do, and what can HAL do. Are we going to allow changes to VIDClist by HAL, ie. not const. Is mode workspace really ok to pass to HAL ???)

[RISC OS Open source page](#)

HAL_VideoWritePaletteEntries

HAL entry #41 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

type 0 for normal palette entry

1 for border colour

2 for pointer colour

>=3 reserved

pcolours Palette to block of palette entry colours in BBGRRSS format (Blue,Green,Red,Supremacy)

index Start index in palette (for first entry in block)

Nentries Number of entries in block (must be >= 1)

On return

-

void HAL_VideoWritePaletteEntries(uint type, const uint *pcolours,
uint index, uint Nentries)

In practice

Indices are in the range 0..255 for normal, 0 for border, 0..3 for pointer colours. Note that RISC OS only makes calls using 1..3 for the pointer, and pointer colour 0 is assumed to be transparent.

[RISC OS Open source page](#)

HAL_VideoWritePaletteEntry

HAL entry #40 · Hardware Abstraction Layer

This is an internal call for OS use only and should not be used by user software. User software should use GraphicsV instead of interacting with the HAL directly.

Before calling

type 0 for normal palette entry

1 for border colour

2 for pointer colour

>=3 reserved

pcolour Palette entry colour in BBGRRSS format (Blue,Green,Red,Supremacy)

index Index of entry

On return

-

void HAL_VideoWritePaletteEntry(uint type, uint pcolour, uint index)

In practice

Indices are in the range 0..255 for normal, 0 for border, 0..3 for pointer colours. Note that RISC OS only makes calls using 1..3 for the pointer, and pointer colour 0 is assumed to be transparent.

[RISC OS Open source page](#)

Hardware Vectors

Vectors

Hardware vectors, like their software counterparts (Software Vectors), contain an address of a routine that will be called in specific situations. Hardware vectors are called when a privileged mode is entered or when a hardware error occurs. These conditions are known as exceptions.

In practice

Each vector will usually hold an address of a routine that will deal with the exception. Each vector has a different priority which is used to determine the order in which exceptions should be handled (if there are simultaneous exceptions).

The ARM processor handles exceptions by:

- Using the registers to save the processor state; thus PC and PSR are copied to R14 and SPSR
- PC and Processor Mode bits are forced to a value (dependent on exception)
- Interrupt disable flags are set where required (to prevent unmanageable nestings of exceptions)
- Re-enable the interrupt (if it is a re-entrant interrupt handler, R14 should be saved onto a stack in main memory beforehand)

List of hardware vectors

The following table list each of the different hardware vectors.

&00 – Name: Reset

&04 – Name: Undefined instruction

&08 – Name: SWI

&0C – Name: Prefetch abort

&10 – Name: Data abort

&14 – Name: Address Exception/Hypervisor trap

&18 – Name: IRQ

&1C – Name: FIQ

The offset given in the table above is relative to the vector base address. Under RISC OS this will either be &0 or &FFF0000, as indicated by the flag returned in bit 20 of OS_PlatformFeatures 0.

Reset Vector

This vector is used to specify that the computer is reset. The ARM processor can be reset by pulling its RESET pin HIGH. When RESET goes LOW again, the following will occur:

- Save R15 in R14_svc and CPSR in SPSR_svc
- Force mode bits to SVC mode and set F and I bits in the PSR
- Force the Program Counter to fetch next instruction from Address (Vector base + &00)

Undefined Instruction Vector

This vector is called when the ARM processor attempts to execute an instruction that is unknown. If a co-processor (software or hardware) is present on the system such as a floating point emulator, the ARM will pass it onto it (when the co-processor is ready). Any instruction still unknown is passed on, and this vector is called.

The ARM processor will:

- Save R15 in R14_und and CPSR in SPSR_und
- Force mode bits to UND mode and set the I bit in the PSR
- Force the Program Counter to fetch next instruction from Address (Vector base + &04)

SWI vector

This vector is called when a SWI instruction is issued. It contains an address of the routine used by RISC OS to decode the SWI number. Due to the importance of this vector it is strongly recommended not to replace it.

The ARM processor will:

- Save R15 in R14_svc and CPSR in SPSR_svc
- Force mode bits to SVC mode and set I bit in the PSR
- Force the Program Counter to fetch next instruction from Address (Vector base + &08)

Prefetch abort vector

This vector is called when an illegal attempt to prefetch an instruction has been detected. The cause of this could be:

- An attempt to access protected memory from within insufficiently privileged mode
- An attempt to access a non-existent logical page

The ARM processor will:

- Save R15 in R14_abt and CPSR in SPSR_abt
- Force mode bits to ABT mode and set the I bit in the PSR
- Force the Program Counter to fetch next instruction from Address (Vector base + &0C)

Data abort vector

This vector is called when an illegal attempt to fetch data has been detected. The cause of this could be:

- An attempt to access protected memory from within insufficiently privileged mode
- An attempt to access a non-existent logical page

The ARM processor will:

- Save R15 in R14_abt and CPSR in SPSR_abt
- Force mode bits to ABT mode and set the I bit in the PSR
- Force the Program Counter to fetch next instruction from Address (Vector base + &10)

Address exception/Hypervisor trap vector

On 26-bit only CPUs this was the address exception vector, which would be called when an attempt was made to transfer data from an address outside the range supported by the 26-bit address bus.

With the introduction of 32-bit processor modes in ARMv3 this vector was deprecated and became unused.

With the virtualisation extensions introduced with ARMv7, this vector entry is now being re-used for handling hypervisor traps.

Currently RISC OS does not make use of hypervisor mode, and so the vector has no practical use on any version of RISC OS from 3.5 and above.

IRQ vector

This vector is called when an interrupt request is received by the ARM processor.

The ARM processor will:

- Save R15 in R14_irq and CPSR in SPSR_irq
- Force mode bits to IRQ mode and set the I bit in the PSR
- Force the Program Counter to fetch next instruction from Address (Vector base + &18)

FIQ vector

The FIQ vector is called when a Fast Interrupt Request is received by the ARM processor. The FIQ vector is entered in FIQ mode.

The ARM processor will:

- Save R15 in R14_fiq and CPSR in SPSR_fiq
- Force mode bits to FIQ mode and set F and I bits in the PSR
- Force the Program Counter to fetch next instruction from Address (Vector base + &1C)

RISC OS provides the owner of the FIQ vector with 228 bytes of workspace to use as they see fit (from vector base +&1C to +&FF, inclusive). If your FIQ handler is small enough you can fit its code within this space, avoiding the need to start the handler with a branch instruction.

Vector priorities

Each vector has a different priority which is used to determine the order in which exceptions should be handled (if there are simultaneous exceptions). We list the hardware vectors in order of priority.

Vector / Priority

Reset – Priority: 1 (highest priority)

Data abort, Address exception – Priority: 2

FIQ – Priority: 3

IRQ – Priority: 4

Prefetch abort – Priority: 5

Undefined instruction, SWI – Priority: 6 (lowest priority)

Pre-veneers

Since the introduction of the full 32bit address space in ARMv3 / RISC OS 3.5, RISC OS has contained a set of “pre-veneers” which are executed prior to the abort environment handlers. These pre-veneers take on a number of responsibilities:

- On 26bit OS versions, they are responsible for switching the processor from 32bit CPU modes to the corresponding 26bit CPU mode, so that environment handlers written for the ARMv2 machines continue to function correctly
- With the introduction of “lazy task swapping” in RISC OS 3.8, the pre-veneers are responsible for mapping in pages of application space on-demand as the pages are accessed
- On ARMv7 machines it's possible for some cache maintenance operations to trigger data aborts; these aborts can safely be ignored, and the RISC OS 5 data abort pre-veener contains code to do this.
- On ARMv6 and later, the data abort pre-veener will also reset the “local exclusive monitor” (see below) prior to invoking the Data Abort Handler. This is necessary because ARM state that the state of the local exclusive monitor is undefined upon taking a data abort (e.g. it's possible the next LDREX instruction that's executed could erroneously succeed).

Claiming hardware vectors

To claim a vector, you use `OS_ClaimProcessorVector`. You pass it the address of the replacement handler routine, which is installed on the vector. The address of the previous routine will be returned to you, so that your handler routine can use it to pass on a call if necessary.

Note that the FIQ vector cannot be claimed using `OS_ClaimProcessorVector`. Instead, `Service_ClaimFIQ` or `Service_ClaimFIQInBackground` must be used, on all versions of RISC OS.

ARMv6 considerations

The LDREX/STREX instructions that were introduced with ARMv6 rely on a component of the CPU called the ‘local exclusive monitor’. For the load/store exclusive instructions to work correctly, the ARM requires that operating systems manually reset the exclusive monitor to the ‘open’ state before returning to any pre-empted code. Failure to do this can result in the program malfunctioning.

Because taking a hardware vector is often a form of pre-emption, this means that several of the vector handlers may have to reset the exclusive monitor on exit, as outlined below.

For more information on how to reset the exclusive monitor, see the ARMv7 compatibility primer.

Undefined instruction, Prefetch abort

Undefined instruction and prefetch abort handlers which make use of load/store exclusive instructions are required to reset the exclusive monitor on exit – whether that exit is by returning to the program, or by passing on to the previous claimant. If the handler does not exit (i.e. it raises an error instead) then there is no need to reset the exclusive monitor.

Data abort

The requirements for data abort handlers are the same as for undefined instruction/prefetch abort handlers, with one addition: the exclusive monitor must be reset before any exclusive load/store instruction is used. This is necessary because ARM state that the state of the local exclusive monitor is undefined upon taking a data abort.

This means that the full set of rules are:

- If your handler makes use of load/store exclusive instructions (either directly or indirectly, e.g. via a SWI), the exclusive monitor must be reset prior to the instructions.
- If your handler returns from the exception, the exclusive monitor must be reset prior to the return (and after all load/store exclusive operations executed by your handler)
- If your handler passes control to the next claimant, the exclusive monitor can be left in an undefined state
- If your handler raises an error, the exclusive monitor must be reset at some point prior to executing the SWI (see first rule)

SWI and Hypervisor calls

ARM recommend that in order to avoid external factors causing the exclusive monitor to spuriously reset, the processing within an exclusive access sequence is kept as short and simple as possible (e.g. only use compare and ALU instructions). A SWI/hypervisor call is not a simple operation, and so they should not be used from within exclusive access sequences. Because they should not be used, there is no point in making sure the exclusive monitor is reset on exit. Therefore RISC OS defines the exclusive monitor state as being unknown on return from a SWI or HVC.

This has implications for other types of handler - if the handler either directly calls a SWI/HVC or calls some other code (e.g. an external callback function) which may use a SWI/HVC then the exclusive monitor state should now be considered as having been modified, and thus it is in need of resetting before the handler returns.

IRQ

IRQ handlers (as registered via `OS_ClaimDeviceVector`) do not need to worry about clearing the exclusive monitor on exit; the post-IRQ code within the kernel will clear it for you.

If handlers are installed via `OS_ClaimProcessorVector` (and they do not return via the kernel) then they must clear the exclusive monitor themselves, as appropriate.

FIQ

Because FIQ handlers return directly to the interrupted code, any FIQ handler which uses the exclusive access instructions must make sure it resets the exclusive monitor on exit.

Additionally, because FIQs can occur while inside data abort handlers, FIQ code which uses exclusive access instructions must reset the exclusive monitor on entry.

[RISC OS Open source page](#)

Hourglass

List of modules

The Hourglass module allows for the mouse pointer to be replaced with an animated hourglass, to indicate to the user that the computer is busy performing a lengthy operation. Generally use of the hourglass is reserved for situations where the computer is unresponsive to most other forms of input, or where the only input the user can perform would have the effect of cancelling the operation. For situations where the computer is still interactive (e.g. background processing in the Wimp) it's recommended that other methods of display are used (e.g. a progress bar in your applications window).

Details

- SWI Calls
- *Commands

[RISC OS Open source page](#)

IMB_Full ARMop

OS

A global instruction memory barrier (IMB) is to be performed.

Before calling

--

On return

R0 Corrupt

Notes

An IMB is an operation that should be performed after new instructions have been stored and before they are executed. It guarantees correct operation for code modification (eg. something as simple as loading code to be executed).

On some ARMs, this operation may be null. On ARMs with harvard architecture this typically consists of:

- Clean data cache
- Drain write buffer
- Invalidate instruction cache

There may be other considerations such as invalidating branch prediction caches.

Functionally, this call is equivalent to calling OS_SynchroniseCodeAreas with bit 0 of R0 clear.

[RISC OS Open source page](#)

IMB_List ARMop

OS

A variant of IMB_Range that accepts a list of address ranges.

Before calling

R0 Pointer to word-aligned list of (start, end) address pairs

R1 Pointer to end of list (past last valid entry)

R2 Total amount of memory to be synchronised

On return

R0 Corrupt

Notes

If you have several areas to synchronise then using this call may result in significant performance gains, both from reducing the function call overhead and from optimisations in the algorithm itself (e.g. only flushing the instruction cache once for StrongARM).

As with IMB_Range, start & end addresses are inclusive-exclusive and must be cache line aligned. The list must contain at least one entry, and must not contain zero-length entries.

The total length given in R2 is used to decide whether a full IMB (IMB_Full) be performed instead.

[RISC OS Open source page](#)

IMB_Range ARMop

OS

An instruction memory barrier (IMB) is to be performed over a logical address range.

Before calling

R0 Logical address of start of range

R1 Logical address of end of range (exclusive)

On return

R0 Corrupt

Notes

R0 and R1 must be aligned on cache line boundaries.

An IMB is an operation that should be performed after new instructions have been stored and before they are executed. It guarantees correct operation for code modification (eg. something as simple as loading code to be executed).

On some ARMs, this operation may be null. On ARMs with harvard architecture this typically consists of:

- Clean data cache
- Drain write buffer
- Invalidate instruction cache

There may be other considerations such as invalidating branch prediction caches.

Note that the range may be very large. The implementation of this call is typically expected to use a threshold (related to Cache_RangeThreshold) to decide when to perform IMB_Full instead, being faster for large ranges.

Functionally, this call is equivalent to calling OS_SynchroniseCodeAreas with bit 1 of R0 set.

[RISC OS Open source page](#)

Installer

List of modules

The Installer module provides the necessary functionality to manage file configuration settings, Font updates as well as Boot and System updates. The !Configure Application utilises the Installer module to handle updates to Fonts, and merging of files to the !System and !Font folders. In addition it also manages the !Boot folder.

In practice

This will ensure, for example, that proper backups and logs are kept of all changes, that modules are always correctly compared, that font messages files are updated, and that !System's version directories are correctly maintained, and it will make it much simpler to update the PreDesktop and Desktop files.

Installer recognises each of the following objects:

Type / Name / Further Info

- -- Name: Font · Further Info: a compound object, consisting of all the files (of any filetype) in the same directory, provided one of which must have a name beginning with "IntMet". The font is referenced by the directory containing the files, for example Resources:\$.Fonts.Homerton.Medium

&FFA -- Name: Module

&FEB – Name: Obey

&FD7 – Name: TaskObey

&FEA – Name: Desktop

-- Name: DataFile · Further Info: any other typed or untyped file

Details

- *Commands

[RISC OS Open source page](#)

InsV

Vectors

This vector is called to place one or more bytes in a buffer.

Before calling

R1 Buffer number plus flags:

b31 clear for Single Insertion, set for Block Mode

b30 Used by the DeepKeys extension

R0 Code to be inserted

R1 Buffer number, b31 clear

R9 20 (&14) vector number

R1 Buffer number, b31 set

R2 Pointer to data still to insert

R3 Number of codes to insert

R9 20 (&14) vector number

On return

R0 Preserved

R1 Preserved

R2 Corrupted

C flag set if insertion failed

R0 Preserved

R1 Preserved

R2 Pointer to data still to insert

R3 Number of bytes left to insert

C flag set if insertion failed

Notes

All characters inserted into buffers are sent through this vector. Used by OS_Byte 138 and OS_Byte 153. If called via OS_CallAVector then interrupts must be disabled first. Note that Block Mode is not necessarily supported for all buffer numbers. The BufferManager module does support Block Mode for the buffers it manages (eg Serial, Printer, Mouse).

[RISC OS Open source page](#)

Internal Communications

Like all Operating Systems, RISC OS must pass information from one part of the system to another. Internal communication is achieved using SWIs in five categories:

In practice

- Vectors
- Service Calls
- Window Manager
- CallBacks
- UpCalls

Vectors

The OS_CallAVector SWI is used to call the routine(s) on a software vector. More information can be found in the Software Vectors section.

Service calls

The OS_ServiceCall SWI is used to pass a service around modules. It is the responsibility of a module if it wishes to provide the service or pass the call on to other modules. More information can be found in the Modules section.

Window Manager

The Window Manager is used to communicate with applications that run within the window environment. The full list of Wimp SWI calls are provided here.

CallBacks

CallBacks are routines that are to be called at a later date via a function pointer (address). There are two types:

- Transient: These routines are called only once and may be called when RISC OS is threaded, but idle. The OS_AddCallBack SWI is used to set up transient callbacks
- Non-Transient: The OS_SetCallBack SWI is used to set up non-transient callbacks. They are handled by the CallBack handler, and the default one can be replaced by OS_ChangeEnvironment

UpCalls

Used by the OS to notify applications of particular situations. The OS_UpCall SWI performs many different operations (different reason codes). Routines that handle UpCalls must only call the error-returning SWI calls. That is SWI calls that have their X bit set. i.e. either add the value &20000 to the SWI number, or add the letter 'X' in front of the SWI name

[RISC OS Open source page](#)

International

List of modules

The International module provides the functionality to allow the user to modify the Operating System for use in different countries by setting:

In practice

Keyboard - the mapping of keys to character codes Alphabet - the mapping from character codes to characters Country - both of the above mappings

Details

- Service Calls
- *Commands

[RISC OS Open source page](#)

InternationalKeyboard

List of modules

The InternationalKeyboard module provides the mapping of keys to character codes.

Details

- Service Calls

[RISC OS Open source page](#)

Internet

List of modules

The Internet module is one example of a protocol module which exists in Layer 3 of the OSI model. It provides the UDP, TCP, IP and ICMP protocols.

Details

- Service Calls
- SWI Calls
- *Commands

[RISC OS Open source page](#)

Interrupts

When a hardware device sends a signal to the ARM processor indicating that it needs attention, this is called an interrupt. The process of sending an interrupt is known as an interrupt request. Devices handled by interrupts include the keyboard, printer, mouse, serial port, disc drives and expansion cards, as well as built-in timers.

In practice

When an interrupt is received, RISC OS temporarily halts the active task, and enters an interrupt routine. The routine deals with the interrupting device very quickly so it can continue with the previous task as quickly as possible. Often, interrupts are handled so quickly that users never realise their task was temporarily halted.

Interrupts are an efficient method of dealing with hardware devices, as the devices inform the system that it needs attention, rather than the system regularly checking all the devices. Because external hardware, such as expansion cards, can generate new interrupts, it is possible to install new routines to deal with them.

Each device that can generate interrupts has a device number. There are corresponding device vectors, similar to Software Vectors and Hardware Vectors. Installed on each vector is a default device driver that receives only interrupts from that device.

Interrupt types

There are two different types of interrupt requests: IRQs (interrupt requests) and FIQs (fast interrupt requests). As the name suggests, FIQs are generated by devices that require their request be serviced more quickly than the normal IRQ request. The ARM processor has a separate mode, its own hardware vector and range of device numbers for dealing with FIQs.

System Device Numbers

Different RISC OS computers use different interrupt generating hardware. The very early Archimedes range used a variety of different peripheral control chips, whereas later versions used the 82C710 or 82C711 chip. Because different controllers have been used, device numbers differ between models of RISC OS computers. A list of current RISC OS system device numbers is here.

Device Vectors

It is possible to claim device vectors and call a routine of your choosing, instead of the default one. With the exception of the Expansion Card interrupts (8 and 13), all device vectors call the most recent routine that claimed the vector. Previous routines that claimed the vector remain on the vector chain, unless they are released. If the most recent routine is released, then the previous owner of the vector is re-installed as the routine that is called.

To claim and release vectors, two SWI calls are used:

- Claim a vector by calling `OS_ClaimDeviceVector`
- Release a vector by calling `OS_ReleaseDeviceVector`

When a routine claims a vector, RISC OS first checks the vector chain to see if there are any earlier instances of the same routine claiming this vector. If found, it removes them from the chain before claiming the vector.

When releasing a device vector, RISC OS checks the vector chain for any earlier claimants of the vector. If none are found, it will automatically disable interrupts from the corresponding device. No attempt to manually disable interrupts should be made.

Interrupt handling routines

As mentioned earlier, the routine dealing with the interrupting device must do so quickly. If a routine cannot deal with a device quickly, it should re-enable interrupts (as they are disabled automatically when entering a routine). Any routine that re-enables interrupts must be able to cope with further interrupts if they occur, and hence the routine being entered for the second time. This is called re-entrancy.

Error Handling

Routines that handle interrupts must only call the error-returning SWI calls (that is, SWI calls that have their X-bit set). If an error is returned to the routine, appropriate action must be taken within the routine. It may be useful to store an error indicator within the routine, so that the next call to an appropriate SWI (one in the module that provides the routine) will generate an error.

SWI re-entrancy

Some SWIs are defined as being re-entrant: that is, they can safely be called even though they are currently in the process of being used already. Some SWIs are deemed to be not re-entrant (i.e. they cannot handle more than one simultaneous call).

As a result of this, non re-entrant SWIs should not be called from within an interrupt handling routine. An example of why this is the case is shown below:

- A non re-entrant SWI is called from within a task. It stores some values in its own private workspace
- An interrupt occurs. The interrupt routine calls the same non re-entrant SWI
- The previous values in the workspace is overwritten
- When the interrupt has been serviced and control is returned to the original task, the SWI workspace is corrupt and so the routine does not work as expected

Clearing interrupt conditions

Once the routine has serviced the interrupt, it must also clear the interrupt so that it does continue to generate the same interrupt. The interrupt is cleared by accessing the hardware that is generating the interrupt.

FIQ specifics

Devices that issue FIQs require them to be handled more quickly, and unlike IRQs no SWI can be called within the handler routine. The default owner of the FIQ vector is the Econet module (which is a proprietary network), and only one owner of the FIQ is supported, so any task that wishes to claim the FIQ vector must follow the process below:

- Claim the FIQ using `OS_ServiceCall`, with one of the following service calls: To claim from the foreground: A service number of 12 (&0C) (`Service_ClaimFIQ`). This call will always be successful, but may involve a wait if any current background FIQ process must be completed first To claim from the background: A service number of 71 (&47) (`Service_ClaimFIQinBackground`). This routine may fail, but does not signify that this was due to an error; it may be because the FIQs could not be claimed. The following steps should then be taken: You must leave your routine to allow the foreground routine to release the FIQs Schedule another attempt at claiming the FIQ
- The fast interrupt mask register should be set to &00. This will prevent fast interrupts while changing to the FIQ code
- Poke your FIQ routine into addresses &1C to &100
- Enable FIQ generation from the device
- Set the bit corresponding to the device in the fast interrupt mask register
- Start your own FIQ operation
- Upon the FIQ operation completing, it must set the interrupt mask register to zero
- Disable FIQ generation from the device
- Release FIQs using `OS_ServiceCall` with the service number of 11 (&0B) (`Service_ReleaseFIQ`)

Disabling interrupts

When an interrupt routine is called, interrupts are by default disabled until the routine completes. However, there may be other times when it is desirable to manually disable interrupts (i.e. IRQs). Because interrupts are an important to the integrity of a system it is important to minimise the amount of time interrupts are disabled.

Perhaps the easiest method of enabling and disabling interrupts from user mode is to use the `OS_IntOn` and `OS_IntOff` SWIs.

Disabling devices or fast interrupts

Unlike the method of disabling IRQs, you must be in a privileged mode to disable FIQs or specific devices. To disable specific devices, all interrupts must first have been disabled, before clearing the relevant bits of the interrupt mask registers.

[RISC OS Open source page](#)

Introduction To *Commands

Introduction to *Commands

*Commands provide a simple method of accessing key functionality provided by RISC OS. A wide range of *Commands is provided by RISC OS for a range of tasks such as file manipulation, updating date and time, using sound and graphics, networking and system configuration.

In practice

The advantages of *Commands are:

- Simple to use

- Easily typed
- May be call within applications/programs
- Provides powerful features

The disadvantages of *Commands are:

- Not as flexible as SWIs
- Limited scope to return information back to a program (output is normally directed straight to screen)

There will be instances when it is more appropriate to use a SWI rather than a *Command, and vice-versa.

Issuing *Commands

*Commands can be used in the following ways:

- From within Programming Languages, such as BASIC, C/C++, or Assembler
- Typed via the Command Line Interpreter (CLI)
- From within applications

When a *Command is used, RISC OS passes the text to OS_CLI. This determines which routine(s) should be called to perform the required action.

Calling from BASIC

*Commands can be called from within BASIC by issuing them directly (e.g. *Help). Using the OSCLI *Command provides a method of passing variables:

```
cmd$ = "Time"
```

```
OSCLI "Help " + cmd$ :REM Issues the *Help Command for Time
```

The final method available from BASIC is passing the *Command string to the SWI OS_CLI.

```
SYS "OS_CLI", "*Help"
```

Calling from Assembler

*Commands can be called from Assembler by passing the *Command string to the SWI OS_CLI. A null byte terminating the command string is required.

Calling from C

[TO DO] Calling from C

Modular nature of *Commands

Because RISC OS is a modular Operating System, *Commands can be altered, added and even replaced. This can be done by claiming vectors, replacing modules, or adding new modules.

When a vector is claimed, it affects how a SWI that uses it works. The same applies for *Commands.

As well as providing a number of SWIs, modules may also provide *Commands. If a module is to be replaced, the replacement must provide the same commands with the same syntax and accept the same parameters (if any). A replacement module may also provide extra *Commands.

New modules can be added to RISC OS to provide new functionality. There is no limit to how many *Commands can be added to the system.

Changing and adding *Commands

By using system variables, it is possible to create new *Commands from existing ones, or to rename existing *Commands. This can be achieved by using an Alias\$command variable, where command is the name of the existing command. The example below defines a new command *Mode which changes to the screen mode to the number given after the command.

```
*Set Alias$Mode echo |<22>|<%0>
```


Index of *Commands Modules Software Vectors System Variables

[RISC OS Open source page](#)

Introduction To Filing Systems

Introduction to Filing Systems

Like other Operating Systems, RISC OS has the ability to access information stored on storage devices such as hard discs, optical media, RAM discs, USB devices and networked devices.

In practice

As standard, RISC OS comes with several filing systems, some of which are listed below:

ADFS – Description: IDE hard discs and floppy disc drives

SCSIFS – Description: SCSI II and USB (SCSI-over-USB) mass storage devices

SDFS – Description: SD cards connected directly (not via a USB adaptor)

CDFS – Description: ISO9660-formatted CDs and DVDs

RamFS – Description: Accessing an area of memory as if it is a disc

DOSFS – Description: Provides access to Microsoft FAT formatted discs as an image filing system

FAT32FS – Description: Third-party filing system for Microsoft FAT32 formatted discs

ResourceFS – Description: Internal filing system that contains resources needed by other Operating System modules, such as the Window Manager

SystemDevice – Description: A module to access various system devices

LanManFS – Description: Network filing system using the CIFS (SMB) protocol

NetFS – Description: Network filing system using the Acorn Econet protocol

ShareFS – Description: Network filing system using the Acorn Access protocol

DeviceFS – Description: A module to manage various hardware devices

PipeFS – Description: Stream based filing system using RAM as a temporary store

The complete set of filing Systems that RISC OS supports is provided in the Filing System Numbers page.

The RISC OS file system commonly uses two modules called FileSwitch and FileCore. These two modules provide much of the functionality required for filing systems. They are both device independent. i.e. they do not access hardware directly. We will briefly discuss both of these modules next.

FileSwitch is a module that is at the heart of all filing systems in RISC OS. It provides a common set of functions that is used by all filing systems. It only provides the parts of these services that are device independent.

By using a predefined interface, it notifies filing systems when they must do certain actions. It also switches between the currently active filing system and keeping the state of each of them.

Any services that are device specific are not provided by FileSwitch, but by separate modules.

Filing System Management

Because FileSwitch works with more than one filing system it has to ensure it knows what filing systems are active, and switch between them as necessary.

When filing systems are initialised, they inform FileSwitch of three important pieces of information.

- Identify their name
- Identify where to find the routines for controlling the hardware
- Identify any special actions that the filing system can perform

Adding Filing Systems

Filing systems can be added to the Operating System easily by adding a module to the system. As FileSwitch provides only common functionality, the filing system modules must provide all remaining functionality.

More detailed information on adding a filing system can be found in the Writing A Filing System? section.

Data Format

The data format of a filing system is not defined by the FileSwitch module. It only specifies what the user interface must look like.

More information can be found in the FileSwitch section.

In practice

The FileCore module provides a core set of services required for a filing system. The module is device independent. i.e. it does not access hardware directly. It structures the organisation of data on a storage device by providing a disc format. FileCore presents a file-level interface to FileSwitch and accepts a block-level interface to modules that access the hardware such as ADFS.

Unlike FileSwitch however, it creates a new instance of itself for each module it supports.

ADFS and SDFS are two examples of filing systems that work with FileCore. Both the ADFS and SDFS modules provide block-level interfaces to access the hardware. As well as their use by FileCore, these block interfaces (SWI calls such as ADFS_DiscOp) can be used by other filing systems to access discs that do not contain FileCore-format volumes.

More information can be found in the FileCore section.

Naming

Files are named based on both the disc name and the name of the filing system that controls that volume. For example SDFS::Accounts.\$.January.Invoices is a disc named 'Accounts' controlled by SDFS. SDFS is a FileCore filing system, so it also implies that the disc called Accounts is also FileCore formatted.

If a volume is not FileCore formatted, it is named by the filing system that provides the file-level services. For example, a FAT32-formatted SD card still uses SDFS for access to the hardware but the FAT32 interpretation is provided by FAT32FS so it is named FAT32FS::DiscName.\$

In practice

DeviceFS is a filing system module that converts the normal calls that FileSwitch sends to a filing system into a simpler set of calls to modules that control the hardware. Its intended purpose is for stream-based Input/Output. The secondary modules that control the hardware are known as device drivers. Two examples of device drivers are the serial and parallel ports. Only a single instance of DeviceFS is required.

Additional device drivers can be added to RISC OS by creating a new device driver module.

More information can be found in the DeviceFS section.

Image Filing Systems

In addition to supporting standard filing systems, RISC OS also provides support for image filing systems. These are a way of opening a file that contains other files so it looks like a standard folder. An example would be SparkFS which allows a Zip file to be opened as if it were a normal folder, read and written to without knowing that it is a Zip.

Another example would be the Sunfish NFS client, which has a small mount file containing the necessary login details to an NFS server. When this mount file is double-clicked Sunfish connects to the NFS server and shows the 'contents' of the mount file to be the files on the NFS share, essentially 'grafting' the NFS server into your local disc tree.

Image file systems provide three important functions.

- Handle files held within an image file
- Identify the format of foreign media
- Provide assistance in the formatting the media

The image file system uses FileSwitch and the standard file system to manage files within an image file, although the host system does not need to be aware of image filing systems to support this functionality. It is provided in a seamless and transparent way to the end user.

Identifying format of foreign media is managed by using a service call. The host filing system does need to be aware of image filing systems to provide this functionality.

Formatting is provided by the host system, but needs to be aware of image filing systems to support this functionality.

More information can be found in the [Writing A Filing System?](#) section.

In practice

The Filer module is another very important module. It is used to display files and directories on the desktop, and also the functionality to interact with them. It uses a standard set of calls to do this, and hence will work with any filing system, and indeed is used for all filing systems.

However, filing systems can also provide their own filers that co-operate with the more system-wide Filer.

Each filing system that provides an icon on the icon bar has a Filer module to do this. Filing system specific filers can also provide other operations which are not sufficiently generic to be provided by the Filer. e.g. Format and Verify operations provided by the ADFS Filer.

A Filer module can use service calls to interact with image filing system, and add their formats to its menu of those it already supports.

The Filer_Action module provides the functions that perform the operation on files and directories on behalf of the Filer. The module of course performs the actions in the background so other tasks can carry on.

[RISC OS Open source page](#)

Introduction To SWIs

Introduction to SWIs

Software Interrupts, or SWIs for short, are a set of powerful instructions to allow developers to interact with the Operating System, without allowing any direct control in Supervisor Mode. The process of using a SWI is as follows:

In practice

- Program issues a specific SWI instruction
- ARM leaves the program and jumps to a fixed location in memory where there is normally a branch instruction into the RISC OS kernel
- SWI instruction is examined the particular OS routine to be run is determined
- The OS routine is called, as specified by the SWI
- Control returned back to program

A SWI consists of a unique name, number, and usually requires parameters to be passed. Because RISC OS is modular, different modules implement specific SWIs.

SWI Names and Numbers

A SWI can be used by either referring to its unique name or number. The 24-bit number, known as the SWI number, is used by RISC OS to identify the routine. The exact method that RISC OS uses to decode the SWI is unimportant. As long as the correct SWI name or number, along with any parameters (if required), are used, then the correct result will be returned.

It is generally advised to use the SWI names when calling SWIs as it improves readability and helps create code that can be updated more easily. It is also important to note that SWI names are case sensitive. They must be typed exactly as documented, otherwise RISC OS will not identify the SWI, and an error will be returned. The names usually follow 'Camel Case' rules; e.g. "BinaryToDecimal".

SWI Numbers

A SWI instruction is allocated a unique number. 24-bits are used to specify the SWI number (0 - &FFFFFF), although only SWIs from &40000 (bit 19 must be set) upwards are available for 3rd party use, as the others are assigned for internal use by RISC OS (although only a small percentage are currently used).

Modules can also provide their own SWIs, and must provide unique SWI numbers. This feature allows for SWIs that perform related functions to be grouped together as part of a module.

If a program executes a SWI that is not recognised, the OS calls a special routine called the 'Unused SWI Vector', or UKSWIV, for short. This usually returns the generic 'No such SWI' error message. However, it is possible for an application to claim this vector, and if the SWI number is recognised by the program, perform the appropriate task.

24-bit Address Field

The 24-bit SWI number consists of the following:

Bits / Identifier / Information

0 - 5 – Identifier: Chunk Offset Number · Information: Identify individual SWIs in a chunk

6 - 16 – Identifier: Chunk Number · Information: Specifies a block of 64 consecutive SWIs for use by application or module

17 – Identifier: Error Flag (X bit) · Information: Determine the action taken on errors

18 - 19 – Identifier: SWI Group · Information: Specifies which part of the system implements the SWI. Possible values are shown in the table below

20 - 23 – Identifier: OS Flag · Information: Specify which OS the SWI expects to execute the SWI. RISC OS requires this bit to be zero

SWI Groups

Bit 19 / Bit 18 / Meaning

0 – Bit 18: 0 · Meaning: Operating System

0 – Bit 18: 1 · Meaning: Operating System modules

1 – Bit 18: 0 · Meaning: Third party applications

1 – Bit 18: 1 · Meaning: User applications

SWI Names

SWI names are broken into two discrete parts separated by the underscore character '_'. The prefix specifies which part of the system will deal with the SWI. The prefix name usually specifies by the module which implements the SWI (e.g. Wimp, Font, ADFS, FileCore, etc.).

The suffix specifies the function of the SWI, and when added to the prefix forms the chunk name. The chunk name is usually the name of the module that provides the SWI, although it is dependent on the name when registering the SWI. Some examples are shown below:

SWI name / Chunk Name / SWI Function

OS_ReadSysInfo – Chunk Name: OS · SWI Function: ReadSysInfo

FileCore_FreeSpace – Chunk Name: FileCore · SWI Function: FreeSpace

PCI_RAMFree – Chunk Name: PCI · SWI Function: RAMFree

Parameters and Results

The ARM registers R0-R9 are used to pass parameters and results to and from SWI routines. Unlike the APCS? standard that's used with most C code, there is no global standard for how the registers are used by SWIs – you must consult the documentation of each SWI call to determine whether a register is an input or an output. However, it is always the case that a SWI which returns an error will return the error block pointer in R0 (see Error handling)

For the OS SWIs documented on this wiki, it's generally the case that any register which isn't used as an input or an output will be preserved across the SWI call. However for some SWIs certain registers will be corrupted – these cases will be noted in the SWI documentation.

PSR fields as inputs

- On 26bit OS versions, SWI handlers are entered with R14 containing the caller's NZCIF flags.
- On 32bit OS versions, there is no way for a SWI to determine the caller's NZCIF flags.
- For all OS versions, the SWI handler is entered with the V flag clear (in both the PC/PSR and R14, as appropriate).
- For all OS versions, there is no way to determine the caller's processor mode.

The above rules do not apply to kernel SWIs, which are able to fully examine the caller's PSR.

NZC flags

Some SWIs use the NZC flags to return values. Normally this is limited to just returning a value in the C flag, but the N and Z flags can be used as well.

If a value isn't explicitly being returned in an NZC flag, the behaviour differs by OS version:

- For 26bit OS versions, the convention is for modules (including 32bit compatible modules) to preserve the flags. However because this is dependent on the implementation of the module, there are no guarantees as to the behaviour.
- For 32bit OS versions, flags should always be considered to be corrupt (because the SWI handler is not passed the original flags on entry).

V flag

The V flag is used to indicate an error state (see Error handling). If the SWI generates an error the V flag will be set; if the SWI does not generate an error then the V flag will be cleared, even if the flag was set on entry.

Other fields

Only kernel SWIs are able to return to the caller in a different CPU mode or with altered interrupt states. However there are a number of SWIs which will temporarily enable IRQs or FIQs during execution.

Error handling

RISC OS provides error handling facilities for SWIs. There are two possible steps to the error checking process:

- Check if an error occurred during the execution of the SWI routine If an error has been generated by the SWI routine, the V flag is set, and R0 points to an error block on exit, and then proceed to the second step in the error checking process If no error has been generated by the SWI routine, the V flag is clear on exit
- Check if the X bit (bit 17) of the SWI instruction was set prior to entering the SWI routine If the X bit was set before the routine, then RISC OS returns control back to the program, which must then deal with the error. The V flag remains set, and R0 points to an error block on exit (known as error-returning) If the X bit was clear before the routine, then RISC OS deals with the error itself. It does this by passing the error to the error handler used by the program (known as error-generating)

The state of the X bit determines how RISC OS deals with a SWI error. It is sometimes useful for RISC OS to deal with an error, while other times it is best if your own program deals with an error. An error-returning SWI can be specified by two different methods:

- Add the value &20000 to the SWI number
- Add the letter 'X' in front of the SWI name; thus OS_WriteC becomes XOS_WriteC

As the most common method of using SWIs is by name.

A detailed summary of how RISC OS handles Error-generating SWIs is provided below, although it is not usually necessary to understand this low-level information.

- RISC OS informs modules of the error using OS_ServiceCall (service number 6) so that it can tidy up. e.g. close files, and so on, before RISC OS handles the error
- RISC OS then calls the error vector (ErrorV?), so it may tidy up (if required). The vector can be claimed, however, the error handler must deal with the error, and not your program

Error numbers

If an error has been generated, R0 points to an error block that identifies the type of error. The format of the error block is described here

The error number contained as part of the error block specifies the type of error. The error number is a 32-bit number, and is divided into three discrete parts:

- The bottom byte is known as the basic error number
- The middle two bytes identifies the cause of the error
- The top byte contains flags; as detailed below

Bit / Meaning if set

24 - 29 – Meaning if set: Sometimes used as a sub-error indicator if bit 31 is set. e.g. ADFS uses them to identify the type of disc error

30 – Meaning if set: Defined to be clear, so can be safely used by developers to flag internal errors

31 – Meaning if set: Specifies a serious error, such as a hardware exception or floating point exception. In these cases, different error number ranges are used

Generating errors

It is possible to generate an error by calling `OS_GenerateError`. The error will be passed to the current error handler, and the SWI call will not return to the caller.

Writing system extension code

When writing system extension code, such as a module or interrupt handler, it is vital that it does not generate errors. This is because users expect it to 'just work'. When issuing SWIs in system extension code, they must always have their X bit set. i.e. either add the value `&20000` to the SWI number, or add the letter 'X' in front of the SWI name.

The exception to the rule is when reporting exception-type errors (i.e. when bit 31 of the error number is set, indicating a serious error).

Allocating SWIs

For software that is to be released to the general public, SWI names and numbers must be allocated through RISC OS Open using their Allocate tool (RISC OS software only), available [here](#).

When registering a SWI, a Chunk Name must be provided, and this will be used as part of the SWI name. e.g. `ChunkName_Function`.

For software which is for private use only, the author is free to pick any SWI names and chunk numbers, so long as the 'User Applications' SWI group (bits 18-19 set to `2_11`) is used. In this case, it is the author's responsibility to ensure the name and chunk does not clash with any other SWIs on their system.

[RISC OS Open source page](#)

InverseTable

List of modules

The `InverseTable` module is a utility module designed to aid in the task of manipulating colours in palettised modes. Specifically, it will generate lookup tables which can be used to convert palette indices to and from RGB values.

In practice

`ColourTrans` provides similar functionality via the `ColourTrans_ReadPalette` and `ColourTrans_GenerateTable` SWIs. However the differences in the implementations and the APIs make the two modules suitable for different purposes.

One major limitation of `InverseTable` is that it only works with 256 colour palettes. If you require translation tables for 2, 4 or 16 colour palettes then `ColourTrans` should be used.

Details

- SWI Calls

[RISC OS Open source page](#)

IrqV

Vectors

This vector is called when an unknown IRQ is detected. It must not be called with `OS_CallAVector`.

Before calling

None

Notes

It was provided in the Arthur OS to allow hardware expansion to generate IRQs. RISC OS provides a new, more efficient method, and this vector is kept for compatibility.

[RISC OS Open source page](#)

Joystick

List of modules

The Joystick module is a driver for one or more analogue or digital joysticks, typically used with games.

Notes

3rd party implementations of the Joystick API for alternative joystick hardware should version their module as 2.00 or later so that an `*RMEnsure` command will override any Joystick module built into the RISC OS ROM.

Details

- SWI Calls

[RISC OS Open source page](#)

Key_Pressed

Reason Code 8 · WindowManager

This code is returned from `Wimp_Poll` and indicates that a key has been pressed while a window belonging to the current application has the input focus.

On return

Block

+0 Window handle with focus

+4 Icon handle, or -1 if none

+8 X offset of caret (relative to window origin)

+12 Y offset of caret (relative to window origin)

+16 Caret Flags

+20 Index of caret into string (if icon)

+24 Character Code (4 bytes)

Notes

If the application does not wish to act upon a keypress such as a 'hot key', then the keypress can be passed on to other applications by calling `Wimp_ProcessKey`.

If the caret is inside a writeable icon when a key is pressed, `Wimp_Poll` will not generate this code for Wimp Managed Key Presses.

Wide characters are reported using multiple `Key_Pressed` events.

[RISC OS Open source page](#)

Keyboard ID

`OS_InstallKeyHandler`

The keyboard ID indicates the overall mapping between key caps (i.e. physical keys on a keyboard) and the Low-Level Internal Key Numbers that the keyboard driver uses to communicate key presses to the system via `KeyV`.

ID / Type

- 1 – Type: Archimedes style keyboard (old-style)
- 2 – Type: PC-AT keyboard (PS/2, USB) (new-style) ¹
- 3 – Type: RCMM handset (NCOS)
- 4 – Type: Pandora keyboard/controller
- 255 – Type: None

In practice

¹ Also used to specify the keyboard type for the Acorn A4 notebook.

[RISC OS Open source page](#)

Keyboard Scan Codes

These are often referred to as "INKEY codes" because they are used by the `INKEY` keyword in BASIC, and the corresponding `OS_Byte` keyboard scan calls.

· Key · # · Key

0 · Shift · 32 · PrintScreen

1 · Ctrl · 113 · F1

2 · Alt · 114 · F2

3 · Left Shift · 115 · F3

4 · Left Ctrl · 20 · F4

5 · Left Alt · 116 · F5

6 · Right Shift · 117 · F6

7 · Right Ctrl · 22 · F7

8 · Right Alt · 118 · F8

9 · Left mouse · 119 · F9

10 · Center mouse · 30 · F10

11 · Right mouse · 28 · F11

12 · FN · 29 · F12

13 · reserved

14 · reserved

15 · reserved
16 · Q
17 · 3 · 65 · A
18 · 4 · 100 · B
19 · 5 · 82 · C
20 · F4 · 50 · D
21 · 8 · 34 · E
22 · F7 · 67 · F
23 · - · 83 · G
25 · Left · 84 · H
26 · 6 (keypad) · 37 · I
27 · 7 (keypad) · 69 · J
28 · F11 · 70 · K
29 · F12 · 86 · L
30 · F10 · 101 · M
31 · ScrollLock · 85 · N
32 · PrintScreen · 54 · O
33 · W · 55 · P
34 · E · 16 · Q
35 · T · 51 · R
36 · 7 · 81 · S
37 · I · 35 · T
38 · 9 · 53 · U
39 · 0 · 99 · V
40 · _ · 33 · W
41 · Down · 66 · X
42 · 8 (keypad) · 68 · Y
43 · 9 (keypad) · 97 · Z
44 · Break
45 · ~ (` on RiscPC and later) · 39 · 0
46 · £ (¥ on Japanese keyboards. Not present on standard UK/US post-Arc keyboards) · 48 · 1
47 · Backspace · 49 · 2
48 · 1 · 17 · 3
49 · 2 · 18 · 4
50 · D · 19 · 5
51 · R · 52 · 6
52 · 6 · 36 · 7
53 · U · 21 · 8
54 · O · 38 · 9
55 · P · 102 · ,
56 · [· 23 · -
57 · Up · 103 · .
58 · + (keypad) · 104 · /
59 · - (keypad) · 56 · [
60 · Enter (keypad) · 120 · \ (# ~ on RiscPC and later (4th row, between ' @ and Return))
61 · Insert · 88 ·]
62 · Home · 87 · ;
63 · Pg Up · 112 · ESC
64 · Caps · 96 · TAB
65 · A · 64 · Caps
66 · X · 31 · ScrollLock
67 · F · 77 · NumLock

68 · Y · 44 · Break
 69 · J · 45 · ~ (` ~ on RiscPC and later)
 70 · K · 46 · £ (¥ on Japanese keyboards. Not present on standard UK/US post-Arc keyboards)
 71 · @
 72 · :
 73 · Return · 47 · Backspace
 74 · / (keypad) · 61 · Insert
 75 · Delete (keypad)
 76 · . (keypad) · 62 · Home
 77 · NumLock · 63 · Pg Up
 78 · Pg Down · 78 · Pg Down
 79 · " (@ on RiscPC and later) · 79 · " (@ on RiscPC and later)
 80 · Shift Lock
 81 · S · 0 · Shift
 82 · C · 1 · Ctrl
 83 · G · 2 · Alt
 84 · H · 3 · Left Shift
 85 · N · 4 · Left Ctrl
 86 · L · 5 · Left Alt
 87 · ; · 125 · Left Logo
 88 ·] · 6 · Right Shift
 89 · Delete · 7 · Right Ctrl
 90 · # (keypad) · 8 · Right Alt
 91 · * (keypad) · 126 · Right Logo
 92 · , (keypad) · 127 · Menu
 93 · +
 94 · [extra key] (\ | on RiscPC and later) · 98 · Space
 95 · [extra key] (\ | when on left of keyboard)
 96 · TAB · 89 · Delete
 97 · Z · 73 · Return
 98 · Space · 105 · End/Copy
 99 · V · 57 · Up
 100 · B · 121 · Right
 101 · M · 25 · Left
 102 · , · 41 · Down
 103 · . · 106 · 0 (keypad)
 104 · / · 107 · 1 (keypad)
 105 · End/Copy · 124 · 2 (keypad)
 106 · 0 (keypad) · 108 · 3 (keypad)
 107 · 1 (keypad) · 122 · 4 (keypad)
 108 · 3 (keypad) · 123 · 5 (keypad)
 109 · NoConvert (Japanese keyboard) · 26 · 6 (keypad)
 110 · Convert (Japanese keyboard) · 27 · 7 (keypad)
 111 · Kana (Japanese keyboard) · 42 · 8 (keypad)
 112 · ESC · 43 · 9 (keypad)
 113 · F1 · 58 · + (keypad)
 114 · F2 · 59 · - (keypad)
 115 · F3 · 76 · . (keypad)
 116 · F5 · 74 · / (keypad)
 117 · F6 · 90 · # (keypad)
 118 · F8 · 91 · * (keypad)
 119 · F9 · 60 · Enter (keypad)

120 · \ (# ~ on RiscPC and later (4th row, between ' @ and Return))
 121 · Right
 122 · 4 (keypad)
 123 · 5 (keypad) · 9 · Left mouse
 124 · 2 (keypad) · 10 · Center mouse
 125 · Left Logo · 11 · Right mouse
 126 · Right Logo · 94 · [extra key] (\ | on RiscPC and later)
 127 · Menu · 93 · +
[RISC OS Open source page](#)

KeyV

Vectors

This vector has multiple uses, depending on the reason code in R0.

Before calling

R0 Reason Code

Reason Codes

0 – : Keyboard present
 1 – : Key released
 2 – : Key pressed
 3 – : Notify driver of LED state
 4 – : Enable keyboard device drivers
 5-11 – : Reserved for system use
 14 – : Key names 1
 15 – : RISCOS Ltd alternative to KeyV 1 and 2 based on HID usage (Select only) 2
 16 – : Keyboard removed
 1 Cerilica
 2 Not implemented on RISC OS 5.
[RISC OS Open source page](#)

KeyV 0

KeyV

After a successful initialisation by the keyboard device driver, the driver should issue this call in order to notify the kernel that a keyboard is present.

Before calling

R0 0 (reason code)
 R1 Keyboard number
 R2 &4E6F4B64 ('NoKd')1

Keyboard present

1 Available from RISC OS 5.06 onwards, or versions of RISC OS Select/Adjust with Kernel 10.23 or newer

Notes

If R2 contains the magic word 'NoKd', the kernel will disable its de-bouncing allowing, for example, a barcode reader to input characters faster than 10-15 cps. De-bouncing can be re-enabled by issuing this call with R2 set to 0.

Once this call has been issued, the driver should go dormant until it receives a KeyV 4 call, which indicates that the system is ready to start receiving key up/down events.

When the last keyboard that a driver manages is removed from the system, KeyV 16 should be issued in order to allow any other drivers to reconfigure the system to meet their requirements.

[RISC OS Open source page](#)

KeyV 1

KeyV

A device driver must inform the kernel of keys being released or pressed by using these reason codes.

Before calling

R0 1 (reason code)

R1 Low-level internal key number

Notes

The kernel must enable the device driver first (see KeyV 4).

[RISC OS Open source page](#)

KeyV 16

KeyV

Keyboard drivers should make this call when the last keyboard they manage is removed from the system. This allows any other drivers which might want to use a different keyboard number to issue a call to KeyV 0 in order to switch the OS to the correct keyboard layout. For example, the builtin keyboard of the Pandora uses keyboard number 4, but if a USB keyboard is attached then the USB drivers will set the keyboard number to 2. Once the last USB keyboard is removed the USB drivers will issue this service call to allow PandoraKey? to set the keyboard type back to 4, allowing InternationalKeyboard to switch back to the keyboard mapping required for the builtin keyboard to function correctly.

Before calling

R0 16 (reason code)

R1 Keyboard number

Notes

When a keyboard driver receives this service call and wishes to make itself the active driver, it should issue a call to KeyV 0 and then claim the KeyV 16 call.

[RISC OS Open source page](#)

KeyV 2

KeyV

A device driver must inform the kernel of keys being released or pressed by using these reason codes.

Before calling

R0 2 (reason code)

R1 Low-level internal key number

Notes

The kernel must enable the device driver first (see KeyV 4).

[RISC OS Open source page](#)

KeyV 3

KeyV

The kernel calls KeyV with this reason code when the state of the keyboard LEDs changes.

Before calling

R0 3 (reason code)

R1 LED status flags

Notes

Keyboard device drivers must claim KeyV and install a routine to handle these calls by setting the keyboard's LEDs as requested.

[RISC OS Open source page](#)

KeyV 4

KeyV

KeyV 4 is called by the kernel to enable keyboard device drivers.

Before calling

R0 4 (reason code)

Notes

The key released (KeyV 1) and key pressed (KeyV 2) reason codes must not be used by the device driver until it receives this reason code.

This is not a reset call and may be seen many times whilst the device driver is active. It does mean that the kernel has flushed its table of keys that are held down, so the device driver should do the same if appropriate.

[RISC OS Open source page](#)

List of HAL APIs

Hardware Abstraction Layer

The HAL entry table is a list of function pointers; the functions pointed to by these pointers provide implementations of several different APIs. This is a list of those APIs.

In practice

- HAL interrupt handling API
- HAL timer API
- HAL counter API
- HAL NVRAM API
- HAL IIC API
- HAL Video API
- HAL UART API
- HAL PCI API
- HAL keyboard API
- HAL USB API
- HAL Miscellaneous

HAL Device API – The preferred method of extending the functionality of the HAL (as it does not involve constantly extending the length of the entry table)

[RISC OS Open source page](#)

Lose_Caret

Reason Code 11 · WindowManager

This code is returned from `Wimp_Poll` and indicates that the window with input focus has changed.

On return

Block

- +0 Window handle
- +4 Icon handle, or -1 if none
- +8 X offset (relative to window origin)
- +12 Y offset (relative to window origin)
- +16 Caret Flags
- +20 Index into string, or -1 if none

Notes

This event usually occurs due to a user clicking on an icon or window that accepts text. It can also happen if an application explicitly calls `Wimp_SetCaretPosition`.

Windows and icons can be set to accept text by setting the button types within the window flag and icon flags respectively.

This event code is not generated if the input focus changes within the same window.

[RISC OS Open source page](#)

Low-Level Internal Key Numbers

Events

Low-Level Internal Key Numbers are used to identify the individual keys/buttons on keyboards and mice. Generally speaking, the numbering matches the order of the keys on an Archimedes keyboard, from top-left to bottom-right. However, because keyboard layouts have changed over the years, not all keyboards will be capable of generating all low-level codes, and the interpretation of some codes may differ. Where multiple interpretations are possible, the Keyboard ID can be used to determine which should be used.

1st row

0 · &00 · Escape

1 · &01 · F1

2 · &02 · F2

3 · &03 · F3

4 · &04 · F4

5 · &05 · F5

6 · &06 · F6

7 · &07 · F7

8 · &08 · F8

9 · &09 · F9

10 · &0A · F10

11 · &0B · F11

12 · &0C · F12

13 · &0D · Print Screen/Sys Rq

14 · &0E · Scroll Lock

15 · &0F · Pause/Break

2nd row

16 · &10 — Key: ` ~ · Notes: ` ~ on RiscPC and later

17 · &11 · 1

18 · &12 · 2

19 · &13 · 3

20 · &14 · 4

21 · &15 · 5

22 · &16 · 6

23 · &17 · 7

24 · &18 · 8

25 · &19 · 9

26 · &1A · 0

27 · &1B · -

28 · &1C · =

29 · &1D — Key: £ · Notes: ¥ on Japanese keyboards. Not present on standard UK/US post-Arc keyboards.

30 · &1E · Backspace

31 · &1F · Insert

32 · &20 · Home

33 · &21 · Page Up

34 · &22 · Num Lock

35 · &23 · Keypad /

36 · &24 · Keypad *

37 · &25 · Keypad #

3rd row

38 · &26 · Tab

39 · &27 · Q

40 · &28 · W

41 · &29 · E

42 · &2A · R

43 · &2B · T

44 · &2C · Y

45 · &2D · U

46 · &2E · I

47 · &2F · O

48 · &30 · P

49 · &31 · [{

50 · &32 ·] }

51 · &33 – Key: \ | · Notes: # ~ on RiscPC and later (4th row, between ' @ and Return)

52 · &34 · Delete

53 · &35 · Copy/End

54 · &36 · Page down

55 · &37 · Keypad 7

56 · &38 · Keypad 8

57 · &39 · Keypad 9

58 · &3A · Keypad -

4th row

59 · &3B · Left Ctrl

60 · &3C · A

61 · &3D · S

62 · &3E · D

63 · &3F · F

64 · &40 · G

65 · &41 · H

66 · &42 · J

67 · &43 · K

68 · &44 · L

69 · &45 · ; :

70 · &46 – Key: ' " · Notes: ' @ on RiscPC and later

71 · &47 · Return

72 · &48 · Keypad 4

73 · &49 · Keypad 5

74 · &4A · Keypad 6

75 · &4B · Keypad +

5th row

76 · &4C · Left Shift

77 · &4D – Key: (not fitted) · Notes: \ | on RiscPC and later

78 · &4E · Z

79 · &4F · X

80 · &50 · C

81 · &51 · V

82 · &52 · B

83 · &53 · N

84 · &54 · M

85 · &55 · , <

86 · &56 · . >

87 · &57 · / ?
 88 · &58 · Right Shift
 89 · &59 · Up
 90 · &5A · Keypad 1
 91 · &5B · Keypad 2
 92 · &5C · Keypad 3
 6th row
 93 · &5D · Caps Lock
 94 · &5E · Left Alt
 95 · &5F · Space
 96 · &60 · Right Alt
 97 · &61 · Right Ctrl
 98 · &62 · Left
 99 · &63 · Down
 100 · &64 · Right
 101 · &65 · Keypad 0
 102 · &66 · Keypad .
 103 · &67 · Keypad Enter
 Non-Archimedes keys
 104 · &68 – Key: Left Windows key · Notes: Win95, 6th row, between Left Ctrl and Left Alt
 105 · &69 – Key: Right Windows key · Notes: Win95, 6th row, between Right Alt and Menu
 106 · &6A – Key: Menu key · Notes: Win95, 6th row, between Right Windows and Right Ctrl
 107 · &6B – Key: NoConvert · Notes: Japanese, 6th row, between Left Alt and Space
 108 · &6C – Key: Convert · Notes: Japanese, 6th row, between Space and Kana
 109 · &6D – Key: Kana · Notes: Japanese, 6th row, between Convert and Right Alt
 110 · &6E – Key: _ · Notes: Japanese, 5th row, between / ? and Right Shift
 111 · &6F – Key: FN · Notes: For reduced key count keyboards
 Mouse
 112 · &70 · Mouse Select
 113 · &71 · Mouse Menu
 114 · &72 · Mouse Adjust
 115 · &73 – Key: Mouse extra button 1 · Notes: Not on Arc mouse
 116 · &74 – Key: Mouse extra button 2 · Notes: Not on Arc mouse

Other languages documentation

Français

[RISC OS Open source page](#)

Memory Limit Handler

OS_ChangeEnvironment

The memory limit environment handler is not a code entry point. Instead, the value specified in R1 to OS_ChangeEnvironment is used to set a soft limit on how much memory an application can use from application space.

Notes

The limit set here is the same value as the RAM limit that OS_GetEnv returns in R1.

The limit should be specified as an absolute address (i.e. &8000 would correspond to 0 bytes of application space), and must be less than the limit set by the Application Space Handler.

When a program spawns a child, e.g. as a result of `system()` from a C program, the memory limit is often used in order to prevent the child from overwriting the parent (assuming the parent has relocated itself to the top of the allocated application space). However setting a memory limit will usually prevent the child from automatically resizing application space, e.g. `Wimp_SlotSize` does not allow resizing of a tasks wimp slot if the soft memory limit does not match the actual limit.

[RISC OS Open source page](#)

Menu_Selection

Reason Code 9 · WindowManager

This code is returned from `Wimp_Poll` and indicates that an item has been selected from a menu.

On return

Block

+0 Item in main menu

+4 Item in 1st sub-menu

+8 Item in 2nd sub-menu

... and so on...

... ..Terminated by -1 value

Notes

The values returned in the block indicate the index number of the item chosen in each menu. The first menu item having a value of 0. A value of -1 denotes no further items.

If the item selected had further sub-menus, then the application should simply ignore the selection.

As the menu does not return information about which menu each item is from, applications must remember which menu it last opened using `Wimp_CreateMenu`.

The Wimp also provides an easy way of decoding menu selections using `Wimp_DecodeMenu`.

Pressing Return inside a writeable menu item also generates this event, however, pressing Return inside a dialog box opened as a menu does not.

Any mouse button can be used to select an item, with the Select and Menu buttons having the same effect. Selecting with the Adjust button, however, selects the menu item but should also keep the menu open to allow the user to make more selections. Some assistance from the application is required for this to happen. See `Wimp_CreateMenu` for more details.

To identify which mouse button was pressed, applications should call `Wimp_GetPointerInfo`.

[RISC OS Open source page](#)

Message Codes

WindowManager

Miscellaneous Codes

&8 – Message_Quit: Message_PreQuit

&9 – Message_Quit: Message_PaletteChange

&A – Message_Quit: Message_SaveDesktop

&B – Message_Quit: Message_DeviceClaim

&C – Message_Quit: Message_DeviceInUse

&E – Message_Quit: Message_Shutdown

&15 – Message_Quit: Message_AppControl
&400 – Message_Quit: Message_FilerOpenDir
&401 – Message_Quit: Message_FilerCloseDir
&402 – Message_Quit: Message_FilerOpenDirAt
&403 – Message_Quit: Message_FilerSelectionDirectory
&404 – Message_Quit: Message_FilerAddSelection
&405 – Message_Quit: Message_FilerAction
&406 – Message_Quit: Message_FilerControlAction
&407 – Message_Quit: Message_FilerSelection
&500 – Message_Quit: Message_AlarmSet
&501 – Message_Quit: Message_AlarmGoneOff
&502 – Message_Quit: Message_HelpRequest
&503 – Message_Quit: Message_HelpReply
&504 – Message_Quit: Message_HelpEnable
&40040 – Message_Quit: Message_Notify
&400C0 – Message_Quit: Message_MenuWarning
&400C1 – Message_Quit: Message_ModeChange
&400C2 – Message_Quit: Message_TaskInitialise
&400C3 – Message_Quit: Message_TaskCloseDown
&400C4 – Message_Quit: Message_SlotSize
&400C5 – Message_Quit: Message_SetSlot
&400C6 – Message_Quit: Message_TaskNameRq
&400C7 – Message_Quit: Message_TaskNamels
&400C8 – Message_Quit: Message_TaskStarted
&400C9 – Message_Quit: Message_MenusDeleted
&400CA – Message_Quit: Message_Iconize
&400CB – Message_Quit: Message_WindowClosed
&400CC – Message_Quit: Message_WindowInfo
&400CD – Message_Quit: Message_Swap?
&400CE – Message_Quit: Message_ToolsChanged?
&400CF – Message_Quit: Message_FontChanged
&400D0 – Message_Quit: Message_IconizeAt

Block Transfer Model (Clipboard, Drag 'n' Drop)

&F / Message_ClaimEntity
&10 – Message_ClaimEntity: Message_DataRequest
&11 – Message_ClaimEntity: Message_Dragging
&12 – Message_ClaimEntity: Message_DragClaim
&13 – Message_ClaimEntity: Message_ReleaseEntity

Data Transfer Model

&1 / Message_DataSave
&2 – Message_DataSave: Message_DataSaveAck
&3 – Message_DataSave: Message_DataLoad
&4 – Message_DataSave: Message_DataLoadAck
&5 – Message_DataSave: Message_DataOpen
&6 – Message_DataSave: Message_RAMFetch
&7 – Message_DataSave: Message_RAMTransmit
&D – Message_DataSave: Message_DataSaved

ColourPicker Protocol

&47700 / Message_ColourPickerColourChoice

&47701 – Message_ColourPickerColourChoice: Message_ColourPickerColourChanged
&47702 – Message_ColourPickerColourChoice: Message_ColourPickerCloseDialogueRequest
&47703 – Message_ColourPickerColourChoice: Message_ColourPickerOpenParentRequest
&47704 – Message_ColourPickerColourChoice: Message_ColourPickerResetColourRequest

PlugIn Protocol

&4D541 – Message_PlugIn_Open: Message_PlugIn_Opening
&4D542 – Message_PlugIn_Open: Message_PlugIn_Close
&4D543 – Message_PlugIn_Open: Message_PlugIn_Closed
&4D544 – Message_PlugIn_Open: Message_PlugIn_Reshape
&4D545 – Message_PlugIn_Open: Message_PlugIn_Reshape_Request
&4D546 – Message_PlugIn_Open: Message_PlugIn_Focus
&4D547 – Message_PlugIn_Open: Message_PlugIn_Unlock
&4D548 – Message_PlugIn_Open: Message_PlugIn_StreamNew
&4D549 – Message_PlugIn_Open: Message_PlugIn_StreamDestroy
&4D54A – Message_PlugIn_Open: Message_PlugIn_StreamWrite
&4D54B – Message_PlugIn_Open: Message_PlugIn_StreamWritten
&4D54C – Message_PlugIn_Open: Message_PlugIn_StreamAsFile
&4D54D – Message_PlugIn_Open: Message_PlugIn_URLAccess
&4D54E – Message_PlugIn_Open: Message_PlugIn_Notify
&4D54F – Message_PlugIn_Open: Message_PlugIn_Status
&4D550 – Message_PlugIn_Open: Message_PlugIn_Busy
&4D551 – Message_PlugIn_Open: Message_PlugIn_Action
&4D552 – Message_PlugIn_Open: Message_PlugIn_Abort

Configure Plugins

&50D80 / Message_PlugInQuit
&50D81 – Message_PlugInQuit: Message_PlugInQuitContinue
&50D82 – Message_PlugInQuit: Message_PlugInQuitAbort
&50D83 – Message_PlugInQuit: Message_OpenConfigWindow
&50D84 – Message_PlugInQuit: Message_FSLockStateChangeRequest
&50D85 – Message_PlugInQuit: Message_FSLockStateChangeConfirm
&50D86 – Message_PlugInQuit: Message_FSLockStateChanged

Throwback Protocol

&42580 / Message_ThrowbackStart?
&42581 – Message_ThrowbackStart?: Message_ProcessingFile?
&42582 – Message_ThrowbackStart?: Message_ErrorsIn?
&42583 – Message_ThrowbackStart?: Message_ErrorDetails?
&42584 – Message_ThrowbackStart?: Message_ThrowbackEnd?
&42585 – Message_ThrowbackStart?: Message_InfoForFile?
&42586 – Message_ThrowbackStart?: Message_InfoDetails?

TaskWindow Protocol

&808C0 / TaskWindow_Input
&808C1 – TaskWindow_Input: TaskWindow_Output
&808C2 – TaskWindow_Input: TaskWindow_Ego
&808C3 – TaskWindow_Input: TaskWindow_Morio
&808C4 – TaskWindow_Input: TaskWindow_Morite
&808C5 – TaskWindow_Input: TaskWindow_NewTask
&808C6 – TaskWindow_Input: TaskWindow_Suspend
&808C7 – TaskWindow_Input: TaskWindow_Resume

Print Protocol

&80141 – Message_PrintFile?: Message_WillPrint?
&80142 – Message_PrintFile?: Message_PrintSave?
&80143 – Message_PrintFile?: Message_PrintInit?
&80144 – Message_PrintFile?: Message_PrintError?
&80145 – Message_PrintFile?: Message_PrintTypeOdd?
&80146 – Message_PrintFile?: Message_PrintTypeKnown?
&80147 – Message_PrintFile?: Message_SetPrinter?
&8014C – Message_PrintFile?: Message_PSPrinterQuery?
&8014D – Message_PrintFile?: Message_PSPrinterAck?
&8014E – Message_PrintFile?: Message_PSPrinterModified?
&8014F – Message_PrintFile?: Message_PSPrinterDefaults?
&80150 – Message_PrintFile?: Message_PSPrinterDefaulted?
&80151 – Message_PrintFile?: Message_PSPrinterNotPS?
&80152 – Message_PrintFile?: Message_ResetPrinter?
&80153 – Message_PrintFile?: Message_PSIFontPrintRunning?

DDE Make Protocol

&81400 · Message_DDE_ToolInfo?

URI Protocol

&4E380 / Message_URI_MStarted?
&4E381 – Message_URI_MStarted?: Message_URI_MDying?
&4E382 – Message_URI_MStarted?: Message_URI_MProcess?
&4E383 – Message_URI_MStarted?: Message_URI_MReturnResult?
&4E384 – Message_URI_MStarted?: Message_URI_MProcessAck?

External Edit Protocol

&45D80 / Message_EditRq?
&45D81 – Message_EditRq?: Message_EditAck?
&45D82 – Message_EditRq?: Message_EditReturn?
&45D83 – Message_EditRq?: Message_EditAbort?
&45D84 – Message_EditRq?: Message_EditDataSave?
&45D85 – Message_EditRq?: Message_EditCursor?

OLE Protocol

&80E1E / Message_OLE_FileChanged?
&80E21 – Message_OLE_FileChanged?: Message_OLE_OpenSession?
&80E22 – Message_OLE_FileChanged?: Message_OLE_OpenSessionAck?
&80E23 – Message_OLE_FileChanged?: Message_OLE_CloseSession?

NewsBase Mail Protocol

&83580 / Message_NewsBase_Command?
&83581 – Message_NewsBase_Command?: Message_NewsBase_Reply?
&83582 – Message_NewsBase_Command?: Message_NewsBase_Update?

IRC Protocol

&DAB00 / Message_IRC_SendData?
&DAB01 – Message_IRC_SendData?: Message_IRC_SendRawData?

ANT Internet Suite Protocol

&46005 · Message_ANT_SendAction?

Popup Help Module

&4A43B / Message_PopupHelp_SendHelp?

&4A43C – Message_PopupHelp_SendHelp?: Message_PopupHelp_RequestHelp?

Third Party Messages

&51C42 – BMgrSvc/BudgieMgr: VideoList · Rick Murray: Rick Murray

&51C44 – BMgrSvc/BudgieMgr: QuickVoy · Rick Murray: Rick Murray

&51C45 – BMgrSvc/BudgieMgr: Emily · Rick Murray: Rick Murray

&51C46 – BMgrSvc/BudgieMgr: Teletext · Rick Murray: Rick Murray

&53200-2F – BMgrSvc/BudgieMgr: OpenRender · Rick Murray: Cerilica

&53240-78 – BMgrSvc/BudgieMgr: FontX · Rick Murray: Cerilica

&53279 – BMgrSvc/BudgieMgr: MouseChange · Rick Murray: Cerilica

&5327A – BMgrSvc/BudgieMgr: Error · Rick Murray: Cerilica

&5327B – BMgrSvc/BudgieMgr: MakeBarcode · Rick Murray: Cerilica

&5327C – BMgrSvc/BudgieMgr: LongMessage · Rick Murray: Cerilica

&5327D – BMgrSvc/BudgieMgr: ClaimResource · Rick Murray: Cerilica

&5327E – BMgrSvc/BudgieMgr: DigitalCD_Misc · Rick Murray: Cerilica

&5327F – BMgrSvc/BudgieMgr: StyledText · Rick Murray: Cerilica

&53380-BF – BMgrSvc/BudgieMgr: PPDManager · Rick Murray: Cerilica

[RISC OS Open source page](#)

MessageTrans

List of modules

The MessageTrans modules provides the functionality that allows for developers to separate text messages from the main body of code for any application. The natural separation of code and text enables applications to be internationalised and message to be updated with out changing code to the application.

In practice

The messages are held in a plain text file, and can be referred to via the use of tokens.

Details

- Service Calls
- SWI Calls

[RISC OS Open source page](#)

MMU_Changing ARMop

OS

Call this when the global MMU mapping has just changed.

Before calling

--

On return

R0 Corrupt

Notes

The operation must typically perform the following:

- Globally invalidate TLB or TLBs
- Globally clean and invalidate all caches
- Drain write buffer

Note that the implementation is not expected to deal with changes to pages which are in active use by interrupt routines.

This operation should typically be used when a large number of cacheable pages have had their attributes changed in a way which will affect cache behaviour.

[RISC OS Open source page](#)

MMU_ChangingEntries ARMop

OS

Call this when the MMU mapping has just changed for a contiguous range of page entries (multiple of 4k).

Before calling

R0 Logical address of first page entry (page aligned)

R1 Number of page entries (≥ 1)

On return

R0 Corrupt

Notes

The operation must typically perform the following:

- Invalidate TLB or TLBs over the range of the entries
- Clean and invalidate all caches over the range of the pages
- Drain write buffer

Note that the implementation is not expected to deal with changes to pages which are in active use by interrupt routines.

Note that the number of entries may be large. The operation is typically expected to use a reasonable threshold, above which it performs a global operation instead for speed reasons.

This operation should typically be used when cacheable pages have had their attributes changed in a way which will affect cache behaviour.

[RISC OS Open source page](#)

MMU_ChangingEntry ARMop

OS

Call this when the MMU mapping has just changed for a single page entry (4k).

Before calling

R0 Logical address of entry (page aligned)

On return

R0 Corrupt

Notes

The operation must typically perform the following:

- Invalidate TLB or TLBs for the entry
- Clean and invalidate all caches over the 4k range of the page
- Drain write buffer

Note that the implementation is not expected to deal with changes to pages which are in active use by interrupt routines.

This operation should typically be used when a cacheable page has had its attributes changed in a way which will affect cache behaviour.

[RISC OS Open source page](#)

MMU_ChangingUncached ARMop

OS

Call this entry when the MMU mapping has just changed in a way that globally affects uncacheable space.

Before calling

--

On return

R0 Corrupt

Notes

The operation must typically globally invalidate the TLB or TLBs. On some systems the write buffer will also be drained.

[RISC OS Open source page](#)

MMU_ChangingUncachedEntries ARMop

OS

Call this when the MMU mapping has just changed for a contiguous range of uncacheable page entries (multiple of 4k).

Before calling

R0 Logical address of first page entry (page aligned)

R1 Number of page entries (≥ 1)

On return

R0 Corrupt

Notes

The operation must typically invalidate the TLB or TLBs over the range of the entries. On some systems the write buffer will also be drained.

[RISC OS Open source page](#)

MMU_ChangingUncachedEntry ARMop

OS

Call this when the MMU mapping has just changed for a single uncacheable page entry (4k).

Before calling

R0 Logical address of entry (page aligned)

On return

R0 Corrupt

Notes

The operation must typically invalidate the TLB or TLBs for the entry. On some systems the write buffer will also be drained.

[RISC OS Open source page](#)

Monitor Types

These are the monitor types that RISC OS can identify, and support.

- 0 – Monitor Type: Television (50Hz PAL)
- 1 – Monitor Type: Multi-Frequency
- 2 – Monitor Type: Hi-res monochrome
- 3 – Monitor Type: VGA
- 4 – Monitor Type: Super VGA
- 5 – Monitor Type: LCD
- 6 – Monitor Type: Reserved?
- 7 – Monitor Type: Use Monitor Definition File

8 – Monitor Type: Television (60Hz NTSC)

Other languages documentation

Français

[RISC OS Open source page](#)

Mouse_Click

Reason Code 6 · WindowManager

This code is returned from Wimp_Poll and indicates that the value of the mouse buttons have changed, or the mouse operation matches the 'button type' of the window or icon.

On return

Block

+0 Mouse x (screen co-ordinate)

+4 Mouse y (screen co-ordinate)

+8 Button value

+12 Window handle (-1 background, -2 icon bar)

+16 Icon handle (-1 workarea)

Notes

If the mouse button has changed status over a window or icon, Wimp_Poll can only return this code if the Window Button Type or Icon Button Type is not set to 0.

Depending on the mouse operation being executed, a different button value will be returned.

[RISC OS Open source page](#)

NetFiler

List of modules

The NetFiler module provides the Network icons on the Desktop icon bar, and uses the Filer to display network directories and their content.

Details

- Service Calls

- *Commands

[RISC OS Open source page](#)

NetPrint

List of modules

The NetPrint module provides functionality to access and use remote printer server devices, using a given network protocol. Like most other filing systems, it uses the FileSwitch module.

In practice

Many of the commands provided by FileSwitch is also available via NetPrint, although there are some that are not applicable to network printer servers, such as those that read or store data.

NetPrint translates commands either directly or via FileSwitch, into printer server commands. These commands are then sent to the printer server using the standard protocol. The printer server then responds accordingly to the commands and files that it receives. It handles spooling and its locally connected printer.

Much of the above is transparent to the user, and for most commands, knowledge about the underlying printer server protocol is not required.

Details

- SWI Calls
- *Commands

[RISC OS Open source page](#)

NetStatus

List of modules

The NetStatus module provides the functionality to control the display of an hourglass on the screen whenever there is prolonged activity on the Econet network.

In practice

It achieves this by:

- Claims the EconetV software vector, examines the reason for each call that is made to the vector
- In turn, makes an appropriate call to the Hourglass module, so that the appearance of the Hourglass indicates the status of the net

It also displays the percentage figures (when appropriate) which show the percentage of a transfer that has completed.

LED Indicators

The hourglass has two 'LED', one on top, and another on the bottom:

LED on / Meaning

Top only – Meaning: User's station is trying to receive data

Bottom only – Meaning: User's station is trying to transmit data

Both – Meaning: User's station is waiting for a broadcast reply

NetStatus Reason Code Translations

NetStatus translates the reason codes for calls to EconetV into SWI calls that it makes to the Hourglass module. These are listed on the Econet Vector Reason Codes page.

[RISC OS Open source page](#)

Null_Reason

Reason Code 0 · WindowManager

This code is returned from `Wimp_Poll` and indicates that no other codes need to be issued.

Notes

Assuming an application does not need to respond to this reason code, it is highly advisable to mask it out. This will greatly improve the responsiveness of the system as `Wimp_Poll` will never give the application control, for it simply to ignore the reason code.

[RISC OS Open source page](#)

OMAP3 Hardware

ARM Hardware

The “Beagle” ROM image is designed to run on the BeagleBoard and other similar devices. Specifically, the machine must use a Texas Instruments OMAP3 system-on-chip, and a Texas Instruments TPS65950 (or compatible - e.g. TWL4030) companion IC. These two chips provide the majority of the subsystems which the ROM image expects to exist. However not all OMAP3-based boards are identical, and so in order to support a new board type some minor adjustments may need to be made to the ROM image, with regard to its assumptions about RAM parameters, GPIO usage or I2C usage.

The OMAP3 system-on-chip

The OMAP3 range of system-on-chips is designed primarily for use in low-power mobile computing devices - e.g. smartphones, PDAs, tablets and netbooks. It contains a number of subsystems:

MPU Subsystem

The microprocessor unit (MPU) subsystem contains the ARM core, caches, and interrupt controller.

A Cortex-A8 ARM core is used, supporting the ARMv7 architecture, including VFPv3/NEON SIMD. The core has a split L1 cache (16KB instruction, 16KB data), and 256KB L2 cache.

The interrupt controller manages 96 interrupt lines, allowing each to be mapped to either the IRQ or FIQ line on the ARM core, or masked and disabled entirely.

IVA2.2 Subsystem

The IVA2.2 subsystem is only present in certain models of the OMAP3. If present, it contains a second CPU core, based around the Texas Instruments TMS320DMC64x+ VLIW DSP (commonly referred to as the “C64x”). With its own caches, interrupt controller, and DMA controller, the IVA2.2 subsystem is capable of decoding 720p compressed video.

On-Chip Memory

The OMAP3 contains 112KB of ROM, containing the initial bootloader, and 64KB of SRAM. The SRAM is currently ignored by RISC OS.

External Memory Interfaces

There are two external memory interfaces:

- The General-purpose memory controller (GPMC) is used to interface the chip with NOR and NAND flash, as well as some types of peripheral hardware (most commonly Ethernet controllers).
- The SDRAM controller (SDRC) is used to interface the chip with up to 2GB of M-SDR or LPDDR SDRAM.

DMA controllers

There are four DMA controllers in a typical OMAP3 device:

- The system DMA controller (sDMA), a flexible multi-channel DMA controller that RISC OS uses for tasks such as audio DMA, accelerating graphics copies and fills, etc.
- The enhanced DMA controller (EDMA), embedded in the IVA2.2 subsystem (if present)
- The display DMA controller, used by the display hardware to read pixel data.
- The USB HS DMA, used by the USB subsystem.

Multimedia Accelerators

Several multimedia accelerators are present:

- 2D and 3D graphics accelerator (SGX). Only present on certain models, the SGX supports programmable shaders, OpenGL 2.0, and can be used for decoding/encoding of common video formats including H.264, H.263, MPEG4 (SP) and WMV9.
- Camera interface subsystem (ISP). Supports most of the raw image sensors available in the market.
- Display interface subsystem. Supports display sizes up to 2048×2048, hardware overlays, display rotation/scaling, and colour-space conversion. Multiple physical interfaces and protocols are supported, including direct connection with LCD panels, and output of analogue S-Video/composite signals.

Security

Certain models of OMAP3 include hardware implementations of common encryption/hashing algorithms such as DES, SHA1/2, MD5, AES and PKA.

Peripherals

The OMAP3 contains many “peripheral” devices:

- 5 multi-channel buffered serial ports (McBSPs), used to perform reliable serial communication with external devices (e.g. audio codecs)
- 4 multi-channel serial port interfaces (McSPI), allowing interface with SPI devices
- High-speed multi-port USB host controller, configurable for EHCI (USB 2.0) or OHCI (USB 1.0) operation
- High-speed USB OTG controller
- HDQ/1-Wire interface
- 3 UARTs, including IrDA and CIR support
- 3 high-speed I2S interfaces
- MMC/SDIO memory card interface
- 12 general-purpose timers
- 3 watchdog timers
- 1 32kHz clock timer
- Six GPIO controllers addressing a total of 192 GPIO pins (although the number of available pins depends on how the chip is packaged)
- 6 mailboxes for inter-processor communication

The TPS65950 chip

To function correctly, the OMAP3 requires a companion chip. This companion chip performs important tasks related to power management, and communicates with the OMAP3 via the use of a fourth, software-inaccessible I2C connection. In

the case of the TPS65950 (and similar software-compatible chips, e.g. TWL4030), several additional functions are provided. These additional functions are typically accessed via one of the three software-accessible I2C interfaces on the OMAP.

- Audio codec
- Battery charger
- USB 2.0 OTG transceiver
- Real-time clock with support for operation from backup battery
- Additional features including LED driver, GPIO, ADC, PWM, keypad interface, and thermal shutdown protection/hot-die detection.

[RISC OS Open source page](#)

OMAPVideo

The OMAPVideo module, whose sources are located at `RiscOS/Sources/Video/HWSupport/OMAPVideo`, is a driver for the display subsystem of the OMAP 3 and 4 range of SoC's. It uses the GraphicsV API to expose the hardware's functionality to RISC OS, thus allowing RISC OS to use the hardware as a display device. The driver implements most of the functionality supported by the current GraphicsV specification.

Current functionality & limitations

- Supports all current GraphicsV reason codes, except: Only limited support for accelerated rectangle copy/fill operations - sources/destinations must be at least byte aligned, and only solid fills with limited ECF patterns are supported. `GraphicsV_SelectHead` currently isn't supported
- Has limited support for running on devices with fixed, 'dumb' LCD panels
- The driver has basic support for TV-out. Except for on Pandora and OMAP4, if a TV is detected on machine startup then the TV will be used as the main display instead of the DVI output.
- VIDC-style hardware scrolling isn't supported, due to restrictions in the display hardware. In some situations it could be emulated via the use of extra overlays, but this probably isn't worth the hassle, especially since RISC OS will fall back to the hardware-accelerated rectangle copies if full hardware scrolling isn't available.

Known issues

- 16bpp modes rely on using the gamma table to map from RISC OS's 5:5:5 16bpp pixel format to the OMAP's 5:6:5 format. However this results in a mode with only 4 blue bits instead of 5, so a proper solution (i.e. implementing the Extended Framebuffer Format Specification) is still desireable.
- In the old HAL-based driver, the DSI PLL would fail to lock if RISC OS was not loaded by U-Boot. No checks have been made to determine whether this problem also exists with the new driver.
- Mouse pointer updates lag behind the rest of the screen by 1 frame. This is because the display controller updates its internal registers from the shadow registers at the start of the vsync period, during which time RISC OS is probably only just entering the interrupt handler and calling the pointer update code. A better approach may be to program the controller to generate an interrupt a few scanlines before vsync is due, and use that as RISC OS's vsync interrupt. With the right timing this will also allow programs that use double buffering & vsync-timed buffer flips to operate as expected.
- The driver may malfunction if given bad mode timings (e.g. requesting a pixel rate of 3.6kHz resulted in a division by zero error!)
- TV-out will be R/B swapped in 16 & 24bpp modes. The only way to fix this is to add support for extra pixel formats to RISC OS (e.g. the Extended Framebuffer Format Specification)

Future improvements

Most major improvements (YUV overlays, display scaling & rotation, etc.) will come as a result of finalising and implementing the Proposed GraphicsV enhancements. Other potential improvements, not dependent on the GraphicsV improvements, are as follows:

- Improved hardware acceleration Optimising the code and/or converting to assembler may result in worthwhile gains for small render operations Right-to-left rectangle copy operations, which are quite slow, may be sped up by splitting the copy rectangle into non-overlapping vertical strips, or making use of the new DMA list functionality in the AM/DM37x DMA controller.
- Implement the missing GraphicsV reason codes

As part of the development of the driver and the extensions to GraphicsV, some other parts of RISC OS are likely to see improvement as well (e.g. improving MakeModes and the Display Manager)

Debugging

A debug version of the module can be built by adding -options DEBUG=TRUE to the OMAPVideo line in the components file (BuildSys.Components.ROOL.OMAP3). At the moment the default method for debug output is to use HAL_DebugTX to output it all via the serial port – so unless you change it to something else you'll need a serial cable connected to another machine to capture the output.

Once the debug option is enabled, an ordinary ROM recompile (Make ROM, Install ROM, Join ROM) is all that's needed to produce the new ROM image.

Debug commands

The following * commands are available in debug builds of the module:

- VideoRegs – produces a dump of the main hardware registers
- TVMode <mode> <type> <testcard> – for playing with TV-out. <mode> should be 0 for NTSC, or 1 for PAL. <type> should be 0 for Composite, 1 for S-Video, or -1 to disable the output (Note that most boards only support S-Video) <testcard> should be 1 to display the video encoder's built in colour bars image, or 0 to display the graphics/video overlays.
- TVRegs – produces a dump of the VENC registers
- TVOut <mode> – 0 switches the desktop to DVI output, 1 to TV. You'll need to change screen mode after issuing the command for it to have an effect.
- SDMARegs – produces a dump of the SDMA registers used for GraphicsV_Render acceleration

MDF entries

This list of MDF entries is primarily intended for users of 'old' OMAPs (revision 3 and below), which had rather tight restrictions on the range of values supported by the porch & sync timing registers. The restrictions meant that they were typically unable to recreate the standard VESA mode timings and instead had to rely on 'reduced blanking' modes (such as those listed below). Newer OMAPs (revision 3.1 and above) have less restrictive timing registers, and so are likely to work with standard MDFs – however owners of new OMAPs may still find use for some of the modes listed here (e.g. 1280×1024). Note that the OMAPVideo driver currently gives little indication to the user of what OMAP revision is in use, so some trial and error may be required!

Some other notes:

- Some of the standard modes listed below (e.g. 1280×720) exhibit timing issues on the BeagleBoard, where the screen is shifted down one scanline with the bottom scanline replicated at the top of the screen. I've seen this on two monitors and in both RISC OS and Linux, so I'm fairly certain it's a hardware bug (JL)
- The 1920×1080 modes aren't likely to work on many monitors due to their 30Hz (or lower) refresh rate.

file_format:1

monitor_title:Beagleboard modes

```
DPMS_state:3
# 640 x 256 (70Hz)
startmode
mode_name:
x_res:640
y_res:256
pixel_rate:25175
h_timings:92,24,22,640,22,0
v_timings:2,106,0,256,0,85
sync_pol:2
endmode
# 640x480@60Hz reduced blanking VESA CVT 0.31M3-R
startmode
mode_name:640 x 480
x_res:640
y_res:480
pixel_rate:23500
h_timings:32,80,0,640,0,48
v_timings:4,7,0,480,0,3
sync_pol:3
endmode
# 640 x 512 (55Hz)
startmode
mode_name:
x_res:640
y_res:512
pixel_rate:32000
h_timings:76,88,96,640,96,28
v_timings:3,19,16,512,16,2
sync_pol:0
endmode
# 800x600@60Hz reduced blanking VESA CVT 0.48M3-R
startmode
mode_name:800 x 600
x_res:800
y_res:600
pixel_rate:35500
h_timings:32,80,0,800,0,48
v_timings:4,11,0,600,0,3
sync_pol:3
endmode
# 1024x768@60Hz reduced blanking VESA CVT 0.79M3-R
startmode
mode_name:1024 x 768
x_res:1024
y_res:768
pixel_rate:56000
h_timings:32,80,0,1024,0,48
v_timings:4,15,0,768,0,3
sync_pol:3
endmode
# 1280x720@60Hz reduced blanking VESA CVT 0.79M3-R
```



```
startmode
mode_name:1280 x 720
x_res:1280
y_res:720
pixel_rate:64000
h_timings:32,80,0,1280,0,48
v_timings:5,13,0,720,0,3
sync_pol:3
endmode
# 720x480@60Hz CEA-861 Format 3
startmode
mode_name:720 x 480
x_res:720
y_res:480
pixel_rate:27027
h_timings:62,60,0,720,0,16
v_timings:6,30,0,480,0,9
sync_pol:3
endmode
# 720x576@60Hz CEA-861 Format 18
startmode
mode_name:720 x 576
x_res:720
y_res:576
pixel_rate:27000
h_timings:64,68,0,720,0,12
v_timings:5,39,0,576,0,5
sync_pol:3
endmode
# 1280x720@50Hz CEA-861B Format 19
startmode
mode_name:1280 x 720
x_res:1280
y_res:720
pixel_rate:74250
h_timings:40,220,0,1280,0,440
v_timings:5,5,0,720,0,20
sync_pol:3
endmode
# 1280x720@60Hz CEA-861B Format 4
startmode
mode_name:1280 x 720
x_res:1280
y_res:720
pixel_rate:74250
h_timings:40,220,0,1280,0,110
v_timings:5,5,0,720,0,20
sync_pol:3
endmode
# 1920x1080@24Hz CEA-861B Format 32
startmode
mode_name:1920 x 1080
```

```
x_res:1920
y_res:1080
pixel_rate:74250
h_timings:44,638,0,1920,0,148
v_timings:5,4,0,1080,0,36
sync_pol:3
endmode
# 1920x1080@25Hz CEA-861B Format 33
startmode
mode_name:1920 x 1080
x_res:1920
y_res:1080
pixel_rate:74250
h_timings:44,528,0,1920,0,148
v_timings:5,4,0,1080,0,36
sync_pol:3
endmode
# 1920x1080@30Hz CEA-861B Format 34
startmode
mode_name:1920 x 1080
x_res:1920
y_res:1080
pixel_rate:74250
h_timings:44,88,0,1920,0,148
v_timings:5,4,0,1080,0,36
sync_pol:3
endmode
# 1280x1024@57Hz
startmode
mode_name:1280 x 1024
x_res:1280
y_res:1024
pixel_rate:86000
h_timings:32,80,0,1280,0,48
v_timings:5,13,0,1024,0,3
sync_pol:3
endmode
# 1600x1000@45Hz
startmode
mode_name:1600 x 1000
x_res:1600
y_res:1000
pixel_rate:86000
h_timings:32,108,14,1600,0,98
v_timings:5,16,0,1000,0,0
sync_pol:3
endmode
# 1024 x 600 (98Hz)
startmode
mode_name:TouchBook
x_res:1024
y_res:600
```

pixel_rate:64000
h_timings:3,39,0,1024,0,3
v_timings:1,7,0,600,0,2
sync_pol:3
endmode
[RISC OS Open source page](#)

Open_Window_Request

Reason Code 2 · WindowManager

This code is returned from Wimp_Poll and indicates one of the following actions has occurred.

On return

Block

- +0 Window handle
- +4 Visible area minimum x
- +8 Visible area minimum y
- +12 Visible area maximum x
- +16 Visible area maximum y
- +20 Scroll x offset
- +24 Scroll y offset
- +28 Window Location

In practice

- A window's size has been changed using the Adjust Size icon
- A window's scroll bars have been moved to a new position

Notes

Upon receipt of this code, Wimp_OpenWindow should be called using the values returned using this call.

Any panes associated with the window should have their new position calculated by using this block and using Wimp_OpenWindow for them also.

[RISC OS Open source page](#)

OS Codes

The version of the Operating System is specified by the value returned in R1 when using OS_Byte 129 (read OS version).

- &A0 – Version: Arthur 1.20
- &A1 – Version: RISC OS 2.00
- &A2 – Version: RISC OS 2.01
- &A3 – Version: RISC OS 3.00
- &A4 – Version: RISC OS 3.10/3.11
- &A5 – Version: RISC OS 3.50
- &A6 – Version: RISC OS 3.60
- &A7 – Version: RISC OS 3.70

&A8 – Version: RISC OS 3.80/4.0x

&A9 – Version: RISC OS 4 Select

&AA – Version: RISC OS 5.0x, 5.1x, 5.2x

&AE – Version: RISC OS Pyromaniac

[RISC OS Open source page](#)

OS_Byte 128 Reason Codes

OS_Byte

Note that for reading free space, R1 = 255 - buffer number.

Before calling

0 Return switch state and number of last ADC channel converted

1 - 4 ADC channel number for which to return value

7 Mouse x

8 Mouse y

246 Free bytes in mouse buffer

252 Free bytes in printer buffer

253 Free bytes in serial output buffer

254 Free bytes in serial input buffer

255 Free bytes in keyboard buffer

[RISC OS Open source page](#)

OS_Byte 160 VDU Variable Numbers

OS_Byte

OS_ReadVduVariables should be used in preference to this call, as it allows for more types of variable to be read, and is more efficient as it allows multiple variables to be read with a single call. Note that the variable numbers used by this call are different to the VDU Variables used with OS_ReadVduVariables.

VDU Variable Numbers

0 – Value: LSB of graphics window left column (ec)

1 – Value: MSB of graphics window left column (ic)

2 – Value: LSB of graphics window bottom row (ic)

3 – Value: MSB of graphics window bottom row (ic)

4 – Value: LSB of graphics window right column (ic)

5 – Value: MSB of graphics window right column (ic)

6 – Value: LSB of graphics window top row (ic)

7 – Value: MSB of graphics window top row (ic)

8 – Value: text window left column

9 – Value: text window bottom row

10 – Value: text window right column

11 – Value: text window top row

12 – Value: LSB of graphics origin x coordinate (ec)

- 13** – Value: MSB of graphics origin x coordinate (ec)
14 – Value: LSB of graphics origin y coordinate (ec)
15 – Value: MSB of graphics origin y coordinate (ec)
16-127 – Value: supported by 3rd party module

Notes

ic: Internal Co-ordinates. The origin is always the bottom left of the screen. One unit is one pixel wide and one pixel high.

ec: External Co-ordinates. A pixel is (1 << XEigFactor) units wide and (1 << YEigFactor) units high, where XEigFactor and YEigFactor are VDU variables.

Cerilica's VDUVars160 module provides the rest of the MOS VDU variables, and restores the official API for reading the VDU writing direction set with VDU 23,16.

[RISC OS Open source page](#)

OS_Byte 221 - 228 Key Codes

OS_Byte

Function keys generate these codes un-shifted.

Notes

- Print** – Code: &80
F1 - F9 – Code: &81 - &89
End/Copy – Code: &8B
Left – Code: &8C
Right – Code: &8D
Down – Code: &8E
Up – Code: &8F
Page Down – Code: &9E
Page Up – Code: &9F
F10 - F12 – Code: &CA - &CC
Insert – Code: &CD

Shift EORs the code with &10 Control EORs the code with &20

The cursor keys and copy only generate these codes if set up to act as function keys by calling OS_Byte 4 with R1=2.

&80 + Buffer value gives the character codes used by the Wimp. When reset occurs &CA (i.e. F10) is inserted into the input buffer.

Unlisted codes cannot be generated by the main keyboard, but can occur by:

- Using these OS_Byte calls to set keys to generate them
- Changing the numeric keypad base with OS_Byte 238
- Inserting bytes into the keyboard buffer with OS_Byte 138 and OS_Byte 153.
- Receiving bytes from the serial port (see also OS_Byte 181 to control serial input interpretation)

Note that page up and down effectively behave as Up and Down with shift toggled. In fact, it is impossible to distinguish shift/control combinations of page up and down with shift/control combinations of cursor up and down, which can prove frustrating.

[RISC OS Open source page](#)

OS_Byte 242 Baud Rates

OS_Byte

The bits in this block are arranged to extend the block used by the original hardware

Notes

Bit / Meaning

0 - 2 – Meaning: Transmit bits 0-2

3 - 5 – Meaning: Receive bits 0-2

6 – Meaning: Receive bit 3

7 – Meaning: Transmit bit 3

where the 4 bit values encode the baud rates:

0 – Baud rate: 19200

1 – Baud rate: 1200

2 – Baud rate: 4800

3 – Baud rate: 150

4 – Baud rate: 9600

5 – Baud rate: 300

6 – Baud rate: 2400

7 – Baud rate: 75

8 – Baud rate: 7200

9 – Baud rate: 134.5

10 – Baud rate: 1800

11 – Baud rate: 50

12 – Baud rate: 3600

13 – Baud rate: 110

14 – Baud rate: 600

15 – Baud rate: Undefined

This use of OS_Byte 242 is deprecated. You should use OS_SerialOp 5 and OS_SerialOp 6 instead.

[RISC OS Open source page](#)

OS_Byte CMOS Settings

OS_Byte

RISC OS provides 240 bytes of data to be stored in non-volatile memory. Much of the storage is reserved for future use, however, it does provide a method of storing key system configurations such as for expansion cards and so forth.

0 – Hex: &00 · Function: Econet station number

1 – Hex: &01 · Function: Econet fileserver station ID (0 if using name)

2 – Hex: &02 · Function: Econet fileserver network number (or first character of name)

3 – Hex: &03 · Function: Econet printer server station number (0 if using name)

4 – Hex: &04 · Function: Econet printer server network number (or first character of name)

5 – Hex: &05 · Function: Default filing system number

6 – Hex: &06 · Function: Unplugged modules byte 0

7 – Hex: &07 · Function: Unplugged modules byte 1

8 – Hex: &08 · Function: Reserved

9 – Hex: &09 · Function: More Unplugged modules bits

- 10 – Hex: &0A · Function: Misc mode data
- 11 – Hex: &0B · Function: Drive/Caps/Mount
- 12 – Hex: &0C · Function: Keyboard auto-repeat delay
- 13 – Hex: &0D · Function: Keyboard auto-repeat time
- 14 – Hex: &0E · Function: Printer ignore character
- 15 – Hex: &0F · Function: Misc printer/Serial data
- 16 – Hex: &10 · Function: Loud/NoScroll/Boot/Serial format
- 17 – Hex: &11 · Function: NetFiler flags
- 18 – Hex: &12 · Function: Unplugged modules byte 2
- 19 – Hex: &13 · Function: Unplugged modules byte 3
- 20 – Hex: &14 · Function: Unplugged extension modules byte 0
- 21 – Hex: &15 · Function: Unplugged extension modules byte 1
- 22 – Hex: &16 · Function: Wimp double click move
- 23 – Hex: &17 · Function: Wimp auto menu delay
- 24 – Hex: &18 · Function: Territory
- 25 – Hex: &19 · Function: Printer buffer size
- 26 – Hex: &1A · Function: ADFS spindown (in 5 seconds)
- 27 – Hex: &1B · Function: Wimp menu drag delay
- 28 – Hex: &1C · Function: Truncate/DragASprite/FilerAction/Dither
- 29 – Hex: &1D · Function: Mouse type
- 30-45 – Hex: &1E-&2D · Function: Reserved for the user 1
- 46-59 – Hex: &2E-&3B · Function: Reserved for applications 2
- 60-79 – Hex: &3C-&4F · Function: Reserved for expansion cards 3
- 80-106 – Hex: &50-&6A · Function: RISC iX and Acorn Access
- 107-111 – Hex: &6B-&6F · Function: Reserved for RISC iX (Other Unix flavours also use these bytes)
- 112-127 – Hex: &70-&7F · Function: Reserved for expansion cards 3
- 128 – Hex: &80 · Function: Year byte 0
- 129 – Hex: &81 · Function: Year byte 1
- 130 – Hex: &82 · Function: More Unplugged module bits
- 131 – Hex: &83 · Function: More Unplugged module bits
- 132 – Hex: &84 · Function: DumpFormat/Misc data?
- 133 – Hex: &85 · Function: Monitor data?
- 134 – Hex: &86 · Function: Font size in 4k units
- 135 – Hex: &87 · Function: Number of ADFS drives?
- 136 – Hex: &88 · Function: Floppy step rates?
- 137 – Hex: &89 · Function: ADFS buffers
- 138 – Hex: &8A · Function: CDFS flags
- 139 – Hex: &8B · Function: Time zone (15 mins as signed)
- 140 – Hex: &8C · Function: NewLook flags?
- 141 – Hex: &8D · Function: More Unplugged module bits
- 142 – Hex: &8E · Function: More Unplugged module bits
- 143 – Hex: &8F · Function: Screen size
- 144 – Hex: &90 · Function: RAMFS size
- 145 – Hex: &91 · Function: System heap size
- 146 – Hex: &92 · Function: RMA size
- 147 – Hex: &93 · Function: Sprite size
- 148 – Hex: &94 · Function: SoundDefault?
- 149-152 – Hex: &95-&98 · Function: Basic editor flags?
- 153-157 – Hex: &99-&9D · Function: Econet printer server name bytes 2-6
- 158-172 – Hex: &9E-&AC · Function: Econet fileserver name bytes 2 - 16
- 173-176 – Hex: &AD-&B0 · Function: More Unplugged modules bits
- 177-184 – Hex: &B1-&B8 · Function: Unplug bits for expansion cards 0 - 7

- 185 – Hex: &B9 · Function: Language (number of module in ROM entered if boot disabled)
- 186 – Hex: &BA · Function: Country
- 187 – Hex: &BB · Function: Unplug bits for network card modules
- 188 – Hex: &BC · Function: Blanker/Misc data?
- 189 – Hex: &BD · Function: ADFS disc size 4
- 190 – Hex: &BE · Function: ADFS disc size 5
- 191 – Hex: &BF · Function: ADFS disc size 6
- 192 – Hex: &C0 · Function: ADFS disc size 7
- 193 – Hex: &C1 · Function: Protection flags?
- 194 – Hex: &C2 · Function: Mouse step
- 195 – Hex: &C3 · Function: Broadcast/Cache/AutoWimp/Misc data?
- 196 – Hex: &C4 · Function: Wimp mode
- 197 – Hex: &C5 · Function: Wimp flags?
- 198 – Hex: &C6 · Function: Filer flags?
- 199 – Hex: &C7 · Function: ADFS directory cache
- 200 – Hex: &C8 · Function: FontMax
- 201 – Hex: &C9 · Function: FontMax1
- 202 – Hex: &CA · Function: FontMax2
- 203 – Hex: &CB · Function: FontMax3
- 204 – Hex: &CC · Function: FontMax4
- 205 – Hex: &CD · Function: FontMax5
- 206 – Hex: &CE · Function: FontMax6 (not used)
- 207 – Hex: &CF · Function: FontMax7 (not used)
- 208 – Hex: &D0 · Function: SCSIFS drives?
- 209 – Hex: &D1 · Function: SCSIFS buffers
- 210 – Hex: &D2 · Function: SCSIFS directory cache
- 211 – Hex: &D3 · Function: SCSIFS disc size 0
- 212 – Hex: &D4 · Function: SCSIFS disc size 1
- 213 – Hex: &D5 · Function: SCSIFS disc size 2
- 214 – Hex: &D6 · Function: SCSIFS disc size 3
- 215 – Hex: &D7 · Function: Reserved (was Auto load applications?)
- 216 – Hex: &D8 · Function: Reserved (was Auto load applications?)
- 217-219 – Hex: &D9-&DB · Function: More Unplugged modules bytes
- 220 – Hex: &DC · Function: Alarm flags/DST?
- 221 – Hex: &DD · Function: Wimp drag click delay
- 222 – Hex: &DE · Function: Wimp drag move delay and iconise widget?
- 223 – Hex: &DF · Function: Wimp double click delay
- 224-229 – Hex: &E0-&E5 · Function: Reserved
- 230 – Hex: &E6 · Function: LCD panel brightness / contrast
- 231 – Hex: &E7 · Function: Another Unplugged modules byte
- 232 – Hex: &E8 · Function: Reserved
- 233-238 – Hex: &E9-&EE · Function: FSLock?
- 239 – Hex: &EF · Function: Checksum

Notes

CMOS usage is subject to change depending on different versions of RISC OS, and applications should not assume the location of any particular byte address. Software should try to use higher-level APIs to read or write system settings, rather than altering CMOS directly.

1 Locations which are reserved for the user can be freely used by user software. However because use of this region of CMOS is not regulated, in order to avoid clashes software which uses these locations should not be publicly distributed.

2 Locations which are reserved for applications can be used by third-party software, providing an allocation has been made. However with a hard disc and !Boot sequence being standard features on modern machines, it's recommended that

software stores its configuration in the Choices folder instead.

3 Expansion cards can determine the size and number of CMOS bytes available for their use by using the Podule_ReadInfo? and PCI_ReadInfo SWIs. Note that configuration data is not automatically wiped when the user moves expansion cards from one slot to another - and so a newly installed card may be reading old configuration data left behind by a previous card. It is the card driver's responsibility to cope with this, e.g. by correctly detecting and ignoring invalid configuration settings.

[RISC OS Open source page](#)

OS_Byte Introduction

OS_Byte & OS_Word

RISC OS provide numerous SWIs that perform many different operations. Most SWIs functions are specified by their name. e.g. OS_RemoveCursors removes cursors. However, there are two SWIs that perform a wide variety of different operations. They are OS_Byte and OS_Word.

In practice

This document provides information on OS_Byte.

The OS_Byte SWI uses at most three registers. The registers upon entry are detailed below:

Register / Use

R0 – Use: Reason code. Used to specify the operation to be performed. (0 - 255)

R1 – Use: Parameter 2. Dependant on reason code (0 - 255)

R2 – Use: Parameter 3. Dependant on reason code (0 - 255)

The name OS_Byte originates from the use of one byte per parameter (values of 0 to 255). It is important to note that the register actually passes the value as 32-bit parameters, and mask out the top 24-bits.

Any routine written to implement or decode OS_Byte calls, must preserve the top 24-bits, at least in R0. This can be achieved by masking the parameters into temporary registers instead of back into the passed parameters.

Some operations of OS_Byte result in returned values too large to fit into a single byte, and should be treated as whole words.

Reason codes

The value passed by R0 specifies the operation to be performed. The reason code must be a value from 0 to 255. Although a large number of different operations are available, the reason codes are grouped by parameter usage.

Reason codes 0 - 127

Where the reason code is between 0 and 127 (inclusive), only R1 is used to pass information to the SWI routine. However, R2 is sometimes used as a temporary register that is corrupted during the SWI routine.

Reason codes 128 - 165

Where the reason code is between 128 and 165 (inclusive), both R1 and R2 are used to pass values to the SWI routine. They may also be used to return values upon exit.

They are often used to return information, rather than perform a particular action.

Reason codes 166 - 255

Where the reason code is between 166 and 255 (inclusive), both R1 and R2 are used to pass values to the SWI routine. In addition, R0 is also used as an index pointer into the RAM which holds the status variables, which are held in consecutive memory locations. i.e. When R0 = 166, points to the first status variable, R1 = 167 points to the second status variable, and so on.

The contents of R1 and R2 determines what happens to the status variable. The formula below shows how the new value of the status variable is calculated:

$\text{New_Value} = (\text{Old_Value} \text{ AND } R2) \text{ EOR } R1$

On exit:

- R1 = old value of status variable
- R2 = value of status variable in the next adjacent memory location (except when R0 = 255)

To read a value without being altered, the following Register values should be used:

- R1 = &00
- R2 = &FF

To write a specific value, the following Register values should be used:

- R1 = New_Value
- R2 = &00

To alter specific bits, the following Register values should be used:

- R1 = Set the bits to the new value you want them to have
- R2 = Clear the bits that you wish to alter

e.g. If bits 5,6 & 7 are to be changed, to the binary pattern 101, then the following binary values should be used:

- R1 = 10100000 (bits to set)
- R2 = 11100000 (bits to alter)

Interrupt handling

OS_Byte is a vectored SWI, but before its vector is called, interrupts are disabled. Most OS_Byte routines are so short that there is no need to re-enable interrupts, and instead rely on RISC OS to do this when control is returned to the program that issued the SWI.

Because the interrupts are not re-enabled, the routines are also used by interrupt handling routines.

If you wish to replace an OS_Byte routine, you must ensure that the replacement routine:

- does not change the way it alters the interrupt status
- does not take so long that interrupts are disabled for unreasonable period of time

Adding OS_Byte calls

Because the OS_Byte is a vectored SWI, it is possible to add your own OS_Byte calls by installing a routine on the software vector that OS_Byte uses.

For more information, read the Software Vectors section.

*Command

Because OS_Byte performs a lot of very useful functions, RISC OS provides a *Command to call the routine directly. The command is *FX.

Because *FX cannot display any return values, it is not possible to use the command to read the values of status variables.

[RISC OS Open source page](#)

OS_ChangedBox Block

OS_ChangedBox

The data pointed to by R1 on exit at offsets 4 - 16 are only valid if the change box calculations were in an enabled state immediately after the call; otherwise they are undefined.

Offset / Contents

- +0 – Contents: New disable/enable flag
- +4 – Contents: X co-ordinate of left edge of box
- +8 – Contents: Y co-ordinate of bottom edge of box
- +12 – Contents: X co-ordinate of right edge of box
- +16 – Contents: Y co-ordinate of top edge of box

Notes

This call checks which areas of the screen were changed by VDU commands and returns the smallest rectangle which covers all changed pixels.

[RISC OS Open source page](#)

OS_ClaimDeviceVector Flag

OS_ClaimDeviceVector

From RISC OS 5.00, if bit 31 of the device number is set when claiming a device vector, the interrupt will be passed on to earlier claimants of the vector unless your service code claims it. This is necessary on PCI systems where all interrupts can be shared.

Notes

It is up to you to determine whether it is your device that has caused the interrupt. If it has, you should service it, then claim the vector by pulling the return address off the stack. If not, pass the service along by returning to the address in R14 on entry. If no handlers claim a particular interrupt, then RISC OS will disable that line.

If you do not claim the interrupt, you must preserve R0 and R3. R1, R2 and R12 may be corrupted.

It is critical that your choice of whether to claim is purely on the basis of whether your card is interrupting, and is accurate. Not claiming when your card is interrupting, or claiming when it isn't can both cause incorrect system behaviour.

[RISC OS Open source page](#)

OS_ConvertVariform Types

OS_ConvertVariform

- 0 – Description: Convert to hex · Input size restriction: Unlimited
- 1 – Description: Convert to cardinal · Input size restriction: Maximum 8 bytes
- 2 – Description: Convert to integer · Input size restriction: Maximum 8 bytes
- 3 – Description: Convert to binary · Input size restriction: Unlimited
- 4 – Description: Convert to spaced cardinal · Input size restriction: Maximum 8 bytes
- 5 – Description: Convert to spaced integer · Input size restriction: Maximum 8 bytes
- 6 – Description: Convert to punctuated cardinal · Input size restriction: Maximum 8 bytes
- 7 – Description: Convert to punctuated integer · Input size restriction: Maximum 8 bytes
- 8 – Description: Convert to fixed file size · Input size restriction: Maximum 8 bytes

- 9 – Description: Convert to file size · Input size restriction: Maximum 8 bytes
 - 10 – Description: Convert to IPv4 · Input size restriction: Unlimited
 - 11 – Description: Convert to EUI · Input size restriction: Unlimited
 - 12 – Description: Convert to IPv6 · Input size restriction: Multiples of 2 bytes
 - 13 – Description: Convert to shortest IPv6 · Input size restriction: Multiples of 2 bytes
 - 14 – Description: Convert to lowercase hex · Input size restriction: Unlimited
 - 15 – Description: Convert to UUID · Input size restriction: Exactly 16 bytes
- [RISC OS Open source page](#)

OS_Find Path Bits

OS_Find

Path Bits

Bits / Meaning

- 0 – Meaning: 0 = File\$Path
- 1 = String pointed to by R2
- 2 = Variable pointed to by R2
- 3 = None
- 2 – Meaning: 1 => then error if open existing dir
- 3 – Meaning: 1 => then error if file doesn't exist

[RISC OS Open source page](#)

OS_FSCControl 10 Options

OS_FSCControl

Options 2 and 3 relate to the BBC micro, respectively action to take on detecting tape errors, and setting the tape interblock gap.

Options

Option / Set per / Meaning

- 0 – Set per: Whole machine · Meaning: Reset all filing systems to defaults
- 1 – Set per: Filing system · Meaning: Filing system messages on loading, saving and creating
- 4 – Set per: Disc · Meaning: Boot action

[RISC OS Open source page](#)

OS_FSCControl 52 Object Types

OS_FSCControl

Object Types

/ Meaning

- 0 – Meaning: No object found (offset is free/defect/beyond image end)
- 1 – Meaning: No object found (offset used for free space map or other non-file use)
- 2 – Meaning: Object found (cannot share offset with other objects)
- 3 – Meaning: Object found (can share offset with other objects)

[RISC OS Open source page](#)

OS_GBPB 10 Buffer

OS_GBPB

All entries are word aligned.

Buffer

Offset / Contents

0 – Contents: Load address

+4 – Contents: Exec address

+8 – Contents: Length

+12 – Contents: Attributes

+16 – Contents: Object type

+20 – Contents: Name (0 - terminated)

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 11 Buffer

OS_GBPB

All entries are word aligned.

Buffer

0 – Contents: Load address

+4 – Contents: Exec address

+8 – Contents: Length

+12 – Contents: Attributes

+16 – Contents: Object type

+20 – Contents: Internal name

+24 – Contents: Time/Date (0 if not stamped)

+29 – Contents: Name (0 - terminated)

Other languages documentation

Français

[RISC OS Open source page](#)

OS_GBPB 12 Buffer

OS_GBPB

All entries are word aligned.

Buffer

Offset / Contents

0 – Contents: Load address

+4 – Contents: Exec address

+8 – Contents: Length

+12 – Contents: Attributes

+16 – Contents: Object type

+20 – Contents: Filetype

+24 – Contents: Name (0 - terminated)

Other languages documentation

Français

[RISC OS Open source page](#)

OS_Memory 0 Flags

OS_Memory

These flags are specific to OS_Memory 0.

Flags

0-7 – Meaning when set: 0 (reason code)

8 – Meaning when set: Physical page number provided

9 – Meaning when set: Logical address provided

10 – Meaning when set: Physical address provided

11 – Meaning when set: Return physical page number (if bit 8 clear)

12 – Meaning when set: Return logical address (if bit 9 clear)

13 – Meaning when set: Return physical address (if bit 10 clear)

14-15 – Meaning when set: Cacheability control: 1

0,1 = No change

2 = Disable cacheing on all specified pages

3 = Enable cacheing on all specified pages

16-31 – Meaning when set: Reserved, must be clear

Notes

1 Cacheability control

As of RISC OS 5.25, using OS_Memory 0 to alter the cacheability of “shared” pages of memory is deprecated.

In this context, “shared” is being used to refer to any page which your software does not have exclusive access to. E.g. if you allocate a block of memory from the RMA, there's a good chance that the pages at the start and end of that block will contain other blocks which have been allocated by other programs. Therefore those pages are “shared”, and there is no way for you or the OS to predict how or when the other programs will access those other allocations (they could be accessed from foreground code after OS_Memory has disabled caching, or they could be accessed from IRQ or FIQ handlers during the OS_Memory operation).

There are several reasons for this deprecation:

- Most ARM chips ignore “unexpected” cache hits. When a page is made uncacheable, the OS first updates the attributes in the page table entry, then it flushes the cached page table entry from the CPU's TLB, and then it cleans and invalidates the cache lines for that page. After the first step, any data/instruction access which the CPU makes to the page may bypass the cache, ignoring any dirty cache lines which have yet to be written out to memory. This will result in that code reading incorrect values from the page (or writing a value which will later get overwritten by a dirty cache line). The OS contains workarounds for this behaviour, but these workarounds involve disabling IRQs and FIQs, which may have an impact on timing-sensitive device drivers. Note: These workarounds were only introduced in RISC OS 5.24. Software running on earlier OS versions may therefore encounter problems if OS_Memory 0 is used to disable caching for a page which is in use by an interrupt handler (specifically, the issue was seen on ARMv5 and newer CPUs)
- The load/store exclusive instructions introduced in ARMv6 typically only operate on cacheable memory. Attempting to use them on uncacheable memory will result in an abort being raised by the CPU. Since the correct operation of these instructions is important to the OS and its apps, it's dangerous for software to use OS_Memory 0 to make shared pages of memory uncacheable, because the caller of OS_Memory 0 can't predict whether any software is going to be performing exclusive load/store operations to the shared regions of the pages.
- On SMP versions of the OS, extra effort will be needed by the OS to make sure that the other cores aren't using the pages while their cacheability is being modified (due to the unexpected cache hit behaviour). As well as the extra complexity this will introduce to the call, it is also likely to result in performance degradation as the other cores will have to have their execution paused for the duration of the operation.

The primary user of the “disable caching” / “enable caching” functionality was DMA drivers, for which OS_Memory 19 has been introduced to act as a replacement.

[RISC OS Open source page](#)

OS_Memory 10 Flags

OS_Memory

These flags are specific to OS_Memory 10.

Flags

Bits / Meaning when set

0-7 – Meaning when set: 10

8 – Meaning when set: 1 = Call is being made by the Wimp

0 = Call is not being made by the Wimp

9-31 – Meaning when set: Reserved, must be clear

[RISC OS Open source page](#)

OS_Memory 12 Flags

OS_Memory

These flags are specific to OS_Memory 12.

Flags

Bits / Meaning when set

0-7 – Meaning when set: 12 (reason code)

8 – Meaning when set: Memory must be suitable for DMA

9 – Meaning when set: R4-R7 specify a physical address range the memory must lie within (RISC OS 5.29+)

10-31 – Meaning when set: Reserved, must be clear
[RISC OS Open source page](#)

OS_Memory 13 Flags

OS_Memory

Flags are specific to this reason code.

Flags

0-7 – Meaning when set: 13 (&0D)

8 – Meaning when set: map bufferable space (0 is normal, non-bufferable)

9 – Meaning when set: map cacheable space (0 is normal, non-cacheable)

10-12 – Meaning when set: Cache policy

16 – Meaning when set: to doubly map

17 – Meaning when set: access privileges specified

24-27 – Meaning when set: access privileges (if bit 17 set) – specified as a Memory Map Page Access value.

All other bits are reserved, and must be zero

Notes

Note that on ARMv6 architecture CPUs and later the memory will be mapped as non-executable, regardless of any specified access privilege.

[RISC OS Open source page](#)

OS_Memory 14 Flags

OS_Memory

Flags

Bits / Meaning

0-7 – Meaning: 14 (&0E)

8 – Meaning: 0 = map unbufferable space

1 = map bufferable space

All other bits are reserved, and must be zero

[RISC OS Open source page](#)

OS_Memory 16 Flags

OS_Memory

Flags

0-7 – Meaning: 16 (&10)

8-15 – Meaning: 1 = cursor/system/sound

2 = IRQ stack

3 = SVC stack

4 = ABT stack

5 = UND stack

6 = Soft CAM

7 = Level 1 page tables
 8 = Level 2 page tables
 9 = HAL workspace
 10 = Kernel buffers
 11 = HAL uncacheable workspace
 12 = Kernel 'zero page' workspace
 13 = Processor vectors
 14 = Debugger workspace
 15 = Scratch space
 16 = Page zero "compatibility page" (OS_Memory 20)
 17 = Kernel C workspace (read-write/zero-init data sections)

All other bits are reserved, and must be zero

[RISC OS Open source page](#)

OS_Memory 19 Input Function

OS_Memory

OS_Memory 19 calls this function to retrieve details about the next region of the buffer involved in the DMA transfer.

Before calling

R9 R2 from OS_Memory 19 input / R9 output from previous Input Function call

R12 R1 from OS_Memory 19 input

SVC32 mode, interrupts enabled

On return

R0 Start address of region

R1 Length of region (0 if end of transfer)

R2 Flags:

Bit 0: Bounce buffer will be used

Bits 1+: Reserved (zero)

R9 New R9 for next Input call

R12 Corrupt

- All other registers preserved

R0,R1 Start address of region

R2 Flags:

Bit 0: Bounce buffer will be used

Bits 1+: Reserved (zero)

R3 Length of region (0 if end of transfer)

R9 New R9 for next Input call

R12 Corrupt

- All other registers preserved

Notes

Bit 11 of the OS_Memory 19 flags should be used to indicate whether your code is using the 32 bit or 64 bit interface.

The address returned in R0 (or R0/R1) should be either a logical or physical address, as indicated by the value you provided in bit 8 of the OS_Memory 19 flags.

By indicating that a bounce buffer will be used, OS_Memory 19 will avoid performing cache maintenance operations for the indicated region.

[RISC OS Open source page](#)

OS_Memory 19 Output Function

OS_Memory

OS_Memory 19 calls this function to pass the details of processed buffer regions to your code.

Before calling

R0 Logical address of start of region

R1 Physical address of start of region

R2 Length of region

R3 Flags:

Bit 0: Bounce buffer must be used

Bits 1+: Reserved

R9 R4 from OS_Memory 19 input / R9 output from previous Output Function call

R12 R1 from OS_Memory 19 input

SVC32 mode, interrupts enabled

R0 Logical address of start of region

R1,R2 Physical address of start of region

R3 Flags:

Bit 0: Bounce buffer must be used

Bits 1+: Reserved

R4 Length of region

R9 R4 from OS_Memory 19 input / R9 output from previous Output Function call

R12 R1 from OS_Memory 19 input

SVC32 mode, interrupts enabled

On return

R9 New R9 for next Output call

R0-R3, R12 Corrupt

- All other registers preserved

Notes

Bit 11 of the OS_Memory 19 flags should be used to indicate whether your code is using the 32 bit or 64 bit interface.

If OS_Memory 19 indicates that a bounce buffer must be used for a region, then it's imperative to not perform DMA to that region, and only perform CPU access instead.

[RISC OS Open source page](#)

OS_Memory 8 Flags

OS_Memory

The flags are specific to this reason code.

Flags

0-7 – Meaning when set: 8

8-11 – Meaning when set: Type of memory

1 = DRAM

2 = VRAM

3 = ROM

4 = I/O (including all I/O, VIDC and EASI space)

5 = Soft ROM (5.00+)

12-31 – Meaning when set: Reserved, must be clear

[RISC OS Open source page](#)

OS_Memory 9 Controller Numbers

OS_Memory

Controller types 32 and above were added by RISCOS Ltd and are currently only supported by their versions of the OS.

Controller Numbers

0-7 – Meaning when set: Controller sequence number

8-31 – Meaning when set: Controller type

0 = EASI card ECTCR (for internal use only) (3.50+)

1 = EASI space (for internal use only) (3.50+)

2 = VIDC1 (3.50+)

3 = VIDC20 (3.50+)

4 = SSpace (5.00+)

5 = Extension ROMs (5.00+)

6 = Tube ULA (5.16+) / RiscPC OPEN bus (5.21+)

7 = SuperIO/82C710 (5.17+)

8-31 Reserved for ROOL

32 = Primary ROM

33 = IOMD

34 = FDC37C665 or similar

35+ Reserved for RISCOS Ltd

Although controller type 33 is currently unsupported by RISC OS 5, the IOMD address can be determined by reading the SSpace address instead, as IOMD is located at the base of SSpace.

[RISC OS Open source page](#)

OS_Memory Page Blocks

OS_Memory

Several different interfaces use page blocks. They are used to pass round lists of memory addresses and memory pages.

Page Blocks (32 bit)

Offset / Contents

+0 – Contents: Physical page number

+4 – Contents: Logical address

+8 – Contents: Physical address

Notes

Each page block record is 12 bytes in size.

Do not confuse this type of page block list with the list format used by the OS_SetMemMapEntries group of SWIs.

[RISC OS Open source page](#)

OS_Memory Page Blocks (64 bit)

OS_Memory

Several different interfaces use page blocks. They are used to pass round lists of memory addresses and memory pages.

Page Blocks (64 bit)

Offset / Contents

+0 – Contents: Physical page number

+4 – Contents: Logical address (low half)

+8 – Contents: Logical address (high half)

+12 – Contents: Physical address (low half)

+16 – Contents: Physical address (high half)

Notes

Each page block record is 20 bytes in size.

Do not confuse this type of page block list with the list format used by the OS_SetMemMapEntries group of SWIs.

[RISC OS Open source page](#)

OS_MMUControl 1 Flags

OS_MMUControl

Bit 29 is currently unimplemented and ignored in all RISCOS versions.

Flags

Bits / Meaning

0-7 – Meaning: 1

- 28 – Meaning: Drain write buffer
 - 29 – Meaning: Flush one entry (otherwise complete flush)
 - 30 – Meaning: Flush processor TLBs
 - 31 – Meaning: Flush processor caches
- All other bits are reserved, and must be zero

[RISC OS Open source page](#)

OS_PlatformFeatures 0 Flags

OS_PlatformFeatures

Bit 11 is set to indicate that SWP/SWPB definitely aren't available, in which case LDREX/STREX will generally be available instead.

Flags

- 0 – Meaning when set: You must tell the OS when a code area changes (OS_SynchroniseCodeAreas)
- 1 – Meaning when set: Enabling then disabling interrupts does not allow them a chance to occur (Use interrupt trigger routine returned in R1)
- 2 – Meaning when set: Hardware vectors are only readable in 32 bit mode
- 3 – Meaning when set: When storing PC, PC+8 is stored (not PC+12)
- 4 – Meaning when set: Data aborts occur with 'full early' timing (Base restored abort model)
- 5 – Meaning when set: CPU has split instruction & data caches
- 6 – Meaning when set: OS is 32-bit
- 7 – Meaning when set: CPU doesn't support 26-bit modes
- 8 – Meaning when set: CPU has 'M' extensions (long multiply - UMULL, etc.)
- 9 – Meaning when set: CPU supports Thumb mode
- 10 – Meaning when set: CPU has 'E' DSP extensions (QADD, etc.)
- 11 – Meaning when set: SWP/SWPB are not available (see notes)
- 12 – Meaning when set: CPU supports LDR/STREX semaphores
- 13 – Meaning when set: CPU supports CLREX and LDR/STREX[B|H|D] semaphores
- 14 – Meaning when set: D-cache can't be disabled safely and OS_MMUControl 0 won't allow it
- 15 – Meaning when set: CPU supports extended small page L2 descriptors
- 16 – Meaning when set: CPU doesn't have a Drain Write Buffer instruction
- 17 – Meaning when set: Aborts don't correctly follow the documented abort model (e.g. StrongARM pre rev-T; prevents lazy task swapping from being used)
- 18 – Meaning when set: CPU is an XScale
- 19 – Meaning when set: XScale JTAG is connected
- 20 – Meaning when set: High processor vectors are in use (vectors at &FFFF0000 instead of &0). See also OS_PlatformFeatures 32
- 21 – Meaning when set: Some of the RAM has a physical address of 2³² or higher. If set, software should prefer to use APIs which use 64bit physical addresses; older APIs which only support 32bit physical addresses will have limited functionality
- 22 – Meaning when set: Physical pages are not available to the OS (operations on physical page numbers will fail)
- 23-30 – Meaning when set: Reserved
- 31 – Meaning when set: Probing for additional OS_PlatformFeatures reason codes is supported (see notes)

SWP/SWPB availability

When bit 11 is clear, and provided you are on RISC OS 3.5 or later OR CallASWI version 0.12 or later is in use, then SWP/SWPB definitely are available.

If you are on an OS version which supports the Archimedes (i.e. pre-RISC OS 3.5), and CallASWI is older than 0.12, then the information returned by bit 11 will be incorrect for ARM2 machines, as SWP/SWPB were introduced with a later architecture version (ARMv2a - as used by the ARM250 and ARM3 CPUs). If your code is running on an old OS version and needs to manually detect whether SWP/SWPB are available then the technique described in Application Note 276 is the suggested method to use.

[RISC OS Open source page](#)

OS_PlatformFeatures 34 Flags

OS_PlatformFeatures

Unless otherwise noted, instruction-related flags only apply to the use of that instruction from ARM CPU modes; information about Thumb modes is not reported.

Flags

- 0 – Meaning when set: AESE, AESD, AESMC, AESIMC instructions are supported
- 1 – Meaning when set: BFC, BFI, SBFX, UBFX instructions are supported
- 2 – Meaning when set: BKPT instruction is supported
- 3 – Meaning when set: BLX instruction is supported
- 4 – Meaning when set: BX instruction is supported
- 5 – Meaning when set: CLREX, LDREXB, LDREXH, STREXB, STREXH instructions are supported
- 6 – Meaning when set: CLZ instruction is supported
- 7 – Meaning when set: CRC32B, CRC32H, CRC32W, CRC32CB, CRC32CH, CRC32CW instructions are supported
- 8 – Meaning when set: DMB, DSB, ISB instructions are supported
- 9 – Meaning when set: ERET and the MRS/MSR (banked register) instructions are supported
- 10 – Meaning when set: HVC instruction is supported
- 11 – Meaning when set: “MOV pc, xx” class of instructions perform interworking branches
- 12 – Meaning when set: Load-acquire/store-release instructions are supported (LDAB, LDAH, LDA, LDAEXB, LDAEXH, LDAEX, LDAEXD, STLB, STLH, STL, STLEXB, STLEXH, STLEX, STLEXD)
- 13 – Meaning when set: LDM/STM are continuable⁴
- 14 – Meaning when set: LDM/STM are noninterruptible⁴
- 15 – Meaning when set: LDM/STM are restartable⁴
- 16 – Meaning when set: LDRD, STRD instructions are supported
- 17 – Meaning when set: LDREXD, STREXD instructions are supported
- 18 – Meaning when set: LDREX, STREX instructions are supported
- 19 – Meaning when set: LDRHT, LDRSBT, LDRSHT, STRHT instructions are supported
- 20 – Meaning when set: LDRH, LDRSH, STRH instructions are supported¹
- 21 – Meaning when set: LDRSB instruction is supported
- 22 – Meaning when set: LDR/STR with writeback and Rd == Rn is not allowed. Applies to all load/store single instructions.
- 23 – Meaning when set: MLS instruction is supported
- 24 – Meaning when set: MOVW, MOVT instructions are supported
- 25 – Meaning when set: MRS, MSR instructions are supported (except banked register version, see flag 9)
- 26 – Meaning when set: MUL with Rd == Rn is not allowed. Applies to MUL, MLA, SMLAL, SMULL, UMLAL, UMULL
- 27 – Meaning when set: MULS, MLAS corrupt C flag. SMLALS, SMULLS, UMLALS, UMULLS corrupt C & V flags. If this feature flag is clear, the corresponding PSR flags are preserved.
- 28 – Meaning when set: NOP and other NOP-compatible hints are supported. See flag 57 for WFE.2
- 29 – Meaning when set: PKHBT, PKHTB, QADD16, QADD8, QASX, QSUB16, QSUB8, QSAX, SADD16, SADD8, SASX, SEL, SHADD16, SHADD8, SHASX, SHSUB16, SHSUB8, SHSAX, SSAT16, SSUB16, SSUB8, SSAX, UADD16, UADD8, UASX, UHADD16, UHADD8, UHASX, UHSUB16, UHSUB8, UHSAX, UQADD16, UQADD8, UQASX, UQSUB16, UQSUB8, UQSAX, USAD8, USADA8, USAT16, USUB16, USUB8, USAX instructions are supported
- 30 – Meaning when set: PLD instruction is supported
- 31 – Meaning when set: PLDW instruction is supported
- 32 – Meaning when set: PLI instruction is supported
- 33 – Meaning when set: PSR ‘GE’ flags are supported
- 34 – Meaning when set: PSR ‘Q’ flag is supported
- 35 – Meaning when set: QADD, QDADD, QDSUB, QSUB instructions are supported

- 36 – Meaning when set: RBIT instruction is supported
- 37 – Meaning when set: REV, REV16, REVSH instructions are supported
- 38 – Meaning when set: SEVL instruction is supported
- 39 – Meaning when set: SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, SHA1SU1 instructions are supported
- 40 – Meaning when set: SHA256H, SHA256H2, SHA256SU0, SHA256SU1 instructions are supported
- 41 – Meaning when set: SMC instruction is supported
- 42 – Meaning when set: SMLABB, SMLABT, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, SMULWT instructions are supported
- 43 – Meaning when set: SMLAD, SMLADX, SMLALD, SMLALDX, SMLSD, SMLSDX, SMLSLD, SMLSLDX, SMMLA, SMMLAR, SMMLS, SMMLSR, SMMUL, SMMULR, SMUAD, SMUADX, SMUSD, SMUSDX instructions are supported
- 44 – Meaning when set: SMULL, SMLAL instructions are supported
- 45 – Meaning when set: SRS, RFE, CPS instructions are supported
- 46 – Meaning when set: SSAT, USAT instructions are supported
- 47 – Meaning when set: SWP, SWPB instructions are fully supported
- 48 – Meaning when set: SWP, SWPB are supported, but are only guaranteed to be atomic in uniprocessor environments
- 49 – Meaning when set: SXTAB, SXTAH, UXTAB, UXTAH instructions are supported
- 50 – Meaning when set: SXTB16, SXTAB16, UXTB16, UXTAB16 instructions are supported
- 51 – Meaning when set: SXTB, SXTH, UXTB, UXTH instructions are supported
- 52 – Meaning when set: SYS processor mode is supported
- 53 – Meaning when set: TEQP, TSTP, CMPP, CMNP instructions are supported
- 54 – Meaning when set: UDIV, SDIV instructions are supported
- 55 – Meaning when set: UMAAL instruction is supported
- 56 – Meaning when set: UMULL, UMLAL instructions are supported
- 57 – Meaning when set: WFE instruction is supported²
- 58 – Meaning when set: Rotated loads are supported³
- 59 – Meaning when set: Unaligned loads/stores are supported³
- 60 – Meaning when set: CP15 IFAR register implemented
- 61 – Meaning when set: CP15 IFSR register implemented
- 62 – Meaning when set: CP15 AIFSR register implemented
- 63 – Meaning when set: CP15 DFAR & DFSR registers are writable
- 64 – Meaning when set: CP15 ADFSR register implemented

Notes

VFP/NEON instruction availability can be determined via `VFP_Support_Features`.

1 Halfword load/store:

Halfword load/store instructions will only be reported as supported if the machine's bus architecture supports them (e.g. the flag will never be set on a RiscPC)

2 WFE and WFI:

WFE (flag 57) is separated out from the other NOP/hint instructions (28) in order to account for the fact that uniprocessor systems typically support WFE by implementing it as a pure NOP – i.e. they do not enter a low-power state. On such systems it is therefore more favourable to use `Portable_Idle` when idling the CPU is desired. Note also that the WFI instruction should not be used directly (use `Portable_Idle` instead); even if the NOP hints are supported there's no guarantee that the WFI instruction is supported (another instruction may be in use instead), or there may be errata or other machine-specific behaviours which need to be dealt with.

3 Memory access alignment modes:

The load/store alignment mode information returned by flags 58 and 59 indicates the capabilities of the CPU. By itself, the information does not indicate what the currently selected alignment mode is, nor does it guarantee that the OS will function if any given alignment mode is selected. If a program wishes to (e.g.) select between three different variants of a routine (rotated, unaligned, and aligned) then it should use `OS_ReadSysInfo 8` to determine the alignment mode that the OS ROM was built for, and use the routine variant which matches that.

4 LDM/STM interruptibility

Flags 13-15 indicate the behaviour of LDM/STM instructions when the system is running in low interrupt latency mode (as configured in the system control register). Only one of the flags will be set.

[RISC OS Open source page](#)

OS_ReadArgs Examples Page

OS_ReadArgs

Given the Syntax String: "number=times/E,file/K/A,expand/S" then the following Input Strings will all match: "-n 10 -file myfile" "-times 1+7 -file myfile -expand" "-file myfile -e"

Examples

However these Input Strings will not: "myfile -number 4" this is because the keyword "-file" is missing, and "-number 20 -times 4 -file myfile" this is because the first keyword has been specified twice; once by its identifier "-number" and again by its alias "-times".

Worked Example

So let's say you're writing a program "MyProgram" in C, and it accepts 2 parameters on the command line: an input filename and an output filename.

For OS_ReadArgs to process the command line you can set the Syntax String to 3 empty keywords: ";;," which says to OS_ReadArgs expect to process 3 parameters.

Thus the command line: MyProgram FileIn FileOut will be in the Output Buffer as: +0 points to 0-terminated string "MyProgram" +4 points to 0-terminated string "FileIn" +8 points to 0-terminated string "FileOut"

So what OS_ReadArgs is doing is reading the first part of the command line (ie program name "MyProgram") and seeing if it has a match in the Syntax String. It doesn't so it puts it in the first unused location in the Output Buffer (at +0). It reads the next part ("FileIn"), no match and so puts it in the next unused location (at +4). And then the final part ("FileOut"), again no match and so into the next unused location (at +8).

Which is why the command line: MyProgram FileOut FileIn will be in the Output Buffer as: +0 points to 0-terminated string "MyProgram" +4 points to 0-terminated string "FileOut" +8 points to 0-terminated string "FileIn"

And if your program assumes that the input filename will always be at +4 and the output filename will always be at +8 in the Output Buffer then you could be in trouble!

And so what keywords enable you to do is to make sure that the Output Buffer has the parameters in the positions you want them in. So if we want the input filename to always be at +4 and the output filename to always be at +8 then we set the Syntax String to be: ";input,output". The program name is still an empty keyword.

That way the user can then specify MyProgram -input FileIn -output FileOut or MyProgram -output FileOut -input FileIn, it matters not because OS_ReadArgs will put the input filename FileIn at +4 and the output at +8 because that's what the Syntax String says. And it puts the program name at +0 because that just happens to be the first unused location.

The Syntax String is saying to OS_ReadArgs, I want this parameter to be at this location in the Output Buffer.

So if you want the output filename to always be at +0, the input filename to always be at +4 and the program name to always be at +8 then you would set the Syntax String to "output,input,".

And so as ReadArgs processes the command line, it will read the program name, it has no match in the Syntax String and so it would go in the first unused location which in this case is +8 in the Output Buffer. It will then read the -output which matches a keyword and put the next parameter "FileOut" to its location at +0. It will then read the -input which again matches a keyword and puts the next parameter "FileIn" to its location at +4. And the Output Buffer will be: +0 points to 0-terminated string "FileOut" +4 points to 0-terminated string "FileIn" +8 points to 0-terminated string "MyProgram"

The problem will come if you use that Syntax String and the user types on the command line: MyProgram FileIn FileOut because it won't fill in the Output Buffer as above (the keywords -input and -output are currently optional), which is where the keyword qualifiers come in.

To force the user to specify the `-input` and `-output` on the command line you use the `/K` qualifier against the relevant keyword, in this case: `"output/K,input/K,"`. Now if the user types: `MyProgram FileIn FileOut` it will cause an error because they must specify `-input` and `-output`.

But they could still type: `MyProgram -input FileIn` without specifying an output file because whilst the keyword must be specified the parameter (keyword and value) is still optional. So let's say you can take input from the keyboard but the output must be to a file, so to make input optional on the command line but output mandatory you would set the Syntax String to be: `"output/A/K,input/K,"`. That way if they type: `MyProgram -input FileIn` they will get an error, but if they type: `MyProgram -output FileOut` it will work just fine with the Output Buffer being: `+0` points to 0-terminated string `"FileOut"` `+4` = `0` `+8` points to 0-terminated string `"MyProgram"`

As will `MyProgram -input FileIn -output FileOut`, and the Output Buffer will be as you expect: `+0` points to 0-terminated string `"FileOut"` `+4` points to 0-terminated string `"FileIn"` `+8` points to 0-terminated string `"MyProgram"`

If you wanted the output to be GStans'd but not the input, then you would use: `"output/A/K/G,input/K,"`. The Output Buffer will then be: `+0` points to GStans'd string: 2 bytes length (=7) followed by `"FileOut"` (no terminator) `+4` points to 0-terminated string `"FileIn"` `+8` points to 0-terminated string `"MyProgram"`

If just the input then: `"output/A/K,input/K/G,"` or if both, then: `"output/A/K/G,input/K/G,"`.

BASIC

But with BASIC programs the parameters `BASIC -quit` are added to the start of the command line. Without changing the Syntax String this will cause an error, and so `“,quit/S”` is usually added at the start of the Syntax String to ensure that `BASIC -quit` gets handled without causing an error.

But if we have the keyword `“quit/S”` at the start of the Syntax String then it will cause a conflict with any other keyword beginning with `“q”`, such as `“quiet/S”` and if `“-q”` is specified on the command line it will be matched to the `“quit/S”` keyword first rather than `“quiet/S”` which is probably not what you want.

One way round this is to specify `“q”` as an alias to `“quiet”`, ie `“quiet=q/S”`. The other way is to put the `“quit/S”` keyword at the end of the Syntax String ensuring that the `“quiet/S”` comes before it.

And so to keep `FileOut` and `FileIn` at `+0` and `+4` in the Output Buffer, the Syntax String needs to be amended to be: `“output/A/K,input/K,quiet/S,,,quit/S”` The Output Buffer will then be: `+0` points to 0-terminated string `"FileOut"` `+4` points to 0-terminated string `"FileIn"` `+8` = `0` if `-quiet` is not specified on the command or = a non-zero value if it is `+12` points to 0-terminated string `"BASIC"` `+16` points to 0-terminated string `"MyProgram"` `+20` = A non-zero value to match `-quit`

You'll notice that the name of our program `"MyProgram"` has moved to `+16` and that `"BASIC"` is now at `+12`, and the `-quit` switch is at `+20`.

Another possible way to deal with `BASIC -quit` is to not treat `“-quit”` as a switch. Whilst this won't solve the `-quiet/-quit` clash over `-q` but it will reduce the number of locations in the Output Buffer by one.

So the Syntax String: `“output/A/K,input/K,quiet/S,,,quit”` will result in an Output Buffer of: `+0` points to 0-terminated string `"FileOut"` `+4` points to 0-terminated string `"FileIn"` `+8` = `0` if `-quiet` is not specified on the command or = a non-zero value if it is `+12` points to 0-terminated string `"BASIC"` `+16` points to 0-terminated string `"MyProgram"`

Code

The following code puts all this into a BASIC procedure that shows how to read each parameter. This uses `DIM LOCAL` so will only work under RISC OS 5. For it to work on earlier versions replace the `DIM buffer% LOCAL buf_size%-1` with either `DIM buffer% buf_size%-1` or another way of claiming some memory. The global variable `argc%` will = `0` if no parameters specified, or if `-help` was specified, or there was an error parsing the parameters.

You will notice that the Syntax String starts `“,quit/help/S”` rather than have `“quit”` at the end. I have done it this way as it seems more natural to have the parameters in the order they would be on the command line. But as has already been demonstrated above, you can put the keywords in any order you like.

I've included `“help”` as it's a good idea for programs to respond to `“-help”` on the command line. And because the `“quit”` keyword has moved we must change `“quiet/S”` to `“quiet=q/S”` to force `“-q”` to match `“-quiet”` rather than `“-quit”`.

The program will store the program name, input and output file specs in the array `arg$()`, whilst `-quiet` will set a flag `quiet%` and `-number` will store the number in the variable `number%`. Then either a syntax message is displayed or the results of the parameters that have been specified on the command line.

```

REM>MyReadArgs
REM
PROCreadArgs
IF argc% = 0 THEN
REM Either no parameters specified or there was an error in parsing them
REM Either way tell the user what the syntax for this program is
PRINT "Syntax: MyReadArgs [-input <file spec>] -output <file spec> [-quiet] [-number N]"
PRINT " where N is between 1 and 99"
END
ENDIF
PRINT "Program: ";arg$(0);" called with parameters:"
PRINT "-input: ";arg$(1);""
PRINT "-output: ";arg$(2);""
PRINT "-quiet set? ";:IF quiet% THEN PRINT "Yes" ELSE PRINT "No"
PRINT "-number = ";number%
END
DEF PROCreadArgs
REM Read and process command line arguments
LOCAL i%, cmd%, syntax$, flags%, buffer%, buf_size%
REM Set default values
argc% = 0
quiet% = FALSE : REM Will be TRUE if -quiet is specified on the command line
number% = 0 : REM Will store the number N if specified on the command line, else defaults to 0
buf_size% = 1024 : REM Max size of command line
DIM buffer% LOCAL buf_size%-1 : REM RISC OS 5 only
REM Syntax: MyReadArgs [-input <file spec>] -output <file spec> [-quiet] [-number N]
syntax$ = ",quit,help/S,input/K/G,output/A/K/G,quiet=q/S,number/E/K"
REM The Output Buffer will be:
REM +0 = "BASIC" (which we ignore)
REM +4 = "MyReadArgs" - will be GStans'd for us, so no need to specify /G
REM +8 = non-zero if -help specified
REM +12 = input file spec (GStans'd)
REM +16 = output file spec (GStans'd)
REM +20 = non-zero if -quiet if specified
REM +24 = pointer to -number N if specified
SYS "OS_GetEnv" TO cmd%
SYS "XOS_ReadArgs",syntax$,cmd%,buffer%,buf_size% TO ;flags%
IF ((flags% AND 1) = 1) THEN ENDPROC
REM If -help specified then exit with argc% = 0 so we get the syntax message displayed otherwise get the file specs
IF buffer%!8 <> 0 THEN argc% = 0 ELSE argc% = 2
DIM arg$(argc%)
SYS "XOS_GenerateError",buffer%!4 TO arg$(0) : REM Get the program name (zero-terminated string)
REM In theory the program name could be greater than 255 characters - in which case arg$(0) will be of length 0
REM And we know it can't be an empty string as we're in the middle of running the program!
IF LEN(arg$(0)) = 0 THEN ERROR 19, "String too long for program name"
IF argc% THEN
REM GStans'd input file spec stored as 2 byte length followed by non-terminated string
cmd% = buffer%!12 : REM Optional, so if not specified then buffer%!12 = 0 and arg$(1) will = ""
IF cmd% <> 0 THEN
REM Check that an empty string "" hasn't been specified on the command line
REM Also in theory the filename could be longer than 255 characters
i% = ?cmd% + ((cmd%?1)*256)

```

```

IF (i% > 0) AND (i% < 256) THEN
FOR i% = 2 TO (?cmd% + ((cmd%?1)*256) + 1) : arg$(1) += CHR$(cmd%?i%) : NEXT
ELSE
IF i% > 255 THEN ERROR 19, "String too long for input filename"
ENDIF
ENDIF
REM GStans'd output file spec stored as 2 byte length followed by non-terminated string
cmd% = buffer%!16 : REM Mandatory, so if not specified then OS_ReadArgs will error
REM But we do need to check that an empty string "" hasn't been specified on the command line
REM And in theory the filename could be longer than 255 characters
i% = ?cmd% + ((cmd%?1)*256)
IF i% = 0 THEN
argc% = 0
ELSE
IF (i% > 0) AND (i% < 256) THEN
FOR i% = 2 TO (?cmd% + ((cmd%?1)*256) + 1) : arg$(2) += CHR$(cmd%?i%) : NEXT
ELSE
ERROR 19, "String too long for output filename"
ENDIF
ENDIF
quiet% = (buffer%!20 <> 0) : REM TRUE if -quiet
cmd% = buffer%!24 : REM Evaluated expression (-number N) stored as 1 byte type followed by 4 bytes of integer if type = 0
IF cmd% <> 0 THEN
REM -number specified on command line
IF ?cmd% = 0 THEN number% = cmd%!1 ELSE argc% = 0 : REM If type is not a number (<>0) then we wont accept it
IF (number% < 0) OR (number% > 99) THEN argc% = 0 : REM N must be in range 1..99 but we don't flag 0 as an error
ENDIF
ENDIF
ENDPROC

```

[RISC OS Open source page](#)

OS_ReadArgs Input String

OS_ReadArgs

The input string is expected to be a copy of the command line that started the program as the result of calling `OS_GetEnv`. Each parameter is separated by spaces, and so to include spaces in a parameter it must be enclosed in quotes. For example: `COPY "source file" destination`

Input String

A parameter may also consist of a keyword followed by a value, again separated by spaces. A keyword that is a switch will not be followed by a value as the value is the presence or absence of the switch. Keywords are preceded by "-". For example: `COPY -input "source file" -output destination -overwrite`

In this example both `-input` and `-output` are keywords followed by a value, whereas `-overwrite` is a switch.

Because keywords are identified by the preceding "-" character you can not have a value starting with a "-" character, for example: `-3+6`. To work around this enclose the value with brackets: `(-3+6)`. You can also use quotes instead of brackets.

Keywords may also be referred to by their alias, or the first letter of either the keyword or the alias. For example, if the syntax string contains `"name=title"` then all of the following input strings will match: `"-name fred"`, `"-n fred"`, `"-title fred"`, `"-t fred"`

If more than one keyword has the same first letter, then only the first keyword in the syntax string will be matched. Keywords are case insensitive, so `"-FILE"` and `"-file"` are identical.

Keywords may be optional and if they are not specified in the input string then the first unused keyword that is not a switch will be given that value.

[RISC OS Open source page](#)

OS_ReadArgs Output Buffer

OS_ReadArgs

The output buffer contains the results of parsing the input string. The first *n* words of the output buffer correspond to the number of keywords in the syntax string.

Output Buffer

For keywords which are a switch (containing a /S qualifier) then a zero value indicates that the switch was not specified, and a non-zero value indicates that the switch was specified in the input string.

For non-switch keywords a value of zero indicates that the keyword was not present, otherwise it will be a pointer to the actual value for that keyword.

If the keyword is GStans'd (contains a /G qualifier) then it points to a block with the following format: length two bytes for the length of the string string length bytes of string (no terminator)

If the keyword is an evaluated expression (contains a /E qualifier) then it points to a block with the following format: type one byte indicating type of result

The rest of the block depends on the value of type. If type = 0 then next 4 bytes will be the integer value. If type is non-zero then the next 2 bytes will be the length of the string followed by the string (no terminator)

If the keyword contains neither a /G or /E qualifier then it points to a zero-terminated string.

[RISC OS Open source page](#)

OS_ReadArgs Syntax String

OS_ReadArgs

The syntax string is made up of one or more keywords separated by commas. Each keyword consists of an identifier, an optional alias and a qualifier which specifies how the keyword and parameter are to be interpreted in the input string. As both identifier and qualifier are optional, "" is a valid keyword.

Syntax String

String= / keyword[, keyword]

Keyword= – keyword[, keyword]: [identifier [=alias]] [/qualifier]

Qualifier / Meaning

A – Meaning: Keyword must always be given value

K – Meaning: Identifier (or alias) must be used. Ignored if /S used

S – Meaning: Identifier is a switch, the value stored will be the presence or absence of the switch

E – Meaning: Call OS_EvaluateExpression on parameter in the input string

G – Meaning: Call OS_GStans on parameter in the input string

Notes

An identifier can be any string composed of alphanumerics and the “_” character. Case is ignored.

The alias is an optional alternative name for the same keyword. You do not need to specify the first letter of the identifier as an alias.

If /K is specified as a qualifier for a keyword then an identifier or alias must also be specified.

Every other qualifier may be used without an identifier, and more than one qualifier may be specified for a keyword though /S, /E and /G are mutually exclusive.

The /E qualifier whilst intended for converting numbers can also return strings. For example 1+7 will return the number 8, whereas 1234 LEFT 2 will return the string "12".

The /G qualifier is recommended for parameters that are expecting a filename or directory path as any system variables will get expanded for you, though not paths (Use OS_FSControl 37 to convert paths).

The position of each keyword in the syntax string determines the position in the output buffer where the parameter value associated with that keyword will be stored regardless of where it is in the input string.

It is recommended to use identifiers where possible as this allows users to specify the parameters for your program in any order.

When used in a BASIC program you must take into consideration that the command line will have "BASIC -quit" added to the start. For example typing MyProgram at the command line, the command line will become: BASIC -quit "MyProgram". See the examples page for further details.

[RISC OS Open source page](#)

OS_ReadLine Flags

OS_ReadLine

The flags are used with both OS_ReadLine and OS_ReadLine32 SWIs.

Flags

Bit / Meaning when set

30 – Meaning when set: Echo character in R4 (password style)

31 – Meaning when set: Echo only characters entering buffer

[RISC OS Open source page](#)

OS_ReadSysInfo 6 Items

OS_ReadSysInfo

1 Unused in RISC OS 5; returns zero.

Items

0 – OS version: 3.80+ · Value: Address of CamEntriesPtr

1 – OS version: 3.80+ · Value: Address of MaxCamEntry

2 – OS version: 3.80+ · Value: PageFlags_Unavailable

3 – OS version: 3.80+ · Value: Address of PhysRamTable

4 – OS version: 3.80+ · Value: ARMA_Cleaner_flipflop1

5 – OS version: 3.80+ · Value: Address of TickNodeChain

6 – OS version: 3.80+ · Value: Address of ROMModuleChain

7 – OS version: 3.80+ · Value: Address of DAList

8 – OS version: 3.80+ · Value: Address of AppSpaceDANode

9 – OS version: 3.80+ · Value: Address of Module_List

10 – OS version: 3.80+ · Value: Address of ModuleSHT_Entries

11 – OS version: 3.80+ · Value: Address of ModuleSWI_HashTab

12 – OS version: 3.80+ · Value: Address of IOSystemType3

- 13** – OS version: 3.80+ · Value: Level 1 page table pointer (L1PT)
- 14** – OS version: 3.80+ · Value: Level 2 page table pointer (L2PT)
- 15** – OS version: 3.80+ · Value: Address of top of UND stack
- 16** – OS version: 3.80+ · Value: Address of top of SVC stack
- 17** – OS version: 3.80+ · Value: SysHeapStart
- 18** – OS version: ROL2 · Value: Address of kernel MessageTrans block
- 5.00+ · Address of SWI dispatch table
- 19** – OS version: ROL2 · Value: Address of error semaphore
- 5.00+ · Base of IRQ device head nodes
- 20** – OS version: ROL2 · Value: Address of OS_PrettyPrint dictionary
- 5.00+ · End of IRQ device head nodes
- 21** – OS version: ROL2 · Value: Address of timer 0 latch value
- 5.00+ · Address of top of IRQ stack
- 22** – OS version: ROL2 · Value: FastTickerV counts per second
- 5.00+ · SoundWorkSpace
- 23** – OS version: ROL2 · Value: Address of vector claimants table
- 5.00+ · Address of IRQsema
- 24** – OS version: ROL2 · Value: Number of vectors supported
- 5.27+
- 25** – OS version: ROL2 · Value: Address of top of IRQ stack
- 5.27+
- 26** – OS version: ROL2 · Value: Address of SWI dispatch table
- 5.27+
- 27** – OS version: ROL2 · Value: Address of branch back to OS after SWIs
- 5.27+
- 28-63** – OS version: - · Value: Reserved for RISCOS Ltd.
- 64** – OS version: 5.17+ · Value: Address of SWI dispatch table
- 65** – OS version: 5.17+ · Value: Base of IRQ device head nodes
- 66** – OS version: 5.17+ · Value: End of IRQ device head nodes
- 67** – OS version: 5.17+ · Value: Address of top of IRQ stack
- 68** – OS version: 5.17+ · Value: SoundWorkSpace
- 69** – OS version: 5.17+ · Value: Address of IRQsema
- 70** – OS version: 5.17+ · Value: Address of DomainId
- 71** – OS version: 5.17+ · Value: Address of OS byte variables
- 72** – OS version: 5.17+ · Value: Address of Foreground ECF OraEor block
- 73** – OS version: 5.17+ · Value: Address of Background ECF OraEor block
- 74** – OS version: 5.17+ · Value: Base of debugger space
- 75** – OS version: 5.17+ · Value: Size of debugger space
- 76** – OS version: 5.17+ · Value: Address of CannotReset
- 77** – OS version: 5.17+ · Value: Address of MetroGnome (OS_ReadMonotonicTime counter)
- 78** – OS version: 5.17+ · Value: Address of CLibCounter
- 79** – OS version: 5.17+ · Value: Address of RISCOSLibWord
- 80** – OS version: 5.17+ · Value: Address of CLibWord
- 81** – OS version: 5.17+ · Value: Address of FPEAnchor
- 82** – OS version: 5.17+ · Value: Address of ESC_Status
- 83** – OS version: 5.21+ · Value: Address of ECFYOffset
- 84** – OS version: 5.21+ · Value: Address of ECFShift
- 85** – OS version: 5.23+ · Value: Address of vector claimants table
- 86** – OS version: 5.23+ · Value: Number of vectors supported
- 87** – OS version: 5.23+ · Value: CAM format (0 = 8 bytes per entry, 1 = 16 bytes per entry)
- 88** – OS version: 5.29+ · Value: Address of top of ABT stack
- 89** – OS version: 5.29+ · Value: PhysRamTable format (0 = addresses are in byte units, 1 = addresses are in 4KB units)

90-127 – OS version: - · Value: Reserved for ROOL

2 Refers to OS versions released by RISC OS Ltd, i.e. RISC OS 4 & RISC OS 6.

3 Unused in RISC OS 5.23; returns zero. In previous versions of RISC OS 5 the item was supported but the value of the variable pointed to was zero.

Notes

This call was introduced in RISC OS 3.80, which supported items 0-17. Later versions of RISC OS introduced additional items. However lack of communication between different development teams has lead to values 18-23 being used for different purposes in different OS versions.

Starting with RISC OS 5.17, items 18-23 have been duplicated into items 64-69. This allows new software aimed at 5.17+ to read the values without risk of accidentally reading the wrong values when used on versions of RISC OS Ltd's OS.

[RISC OS Open source page](#)

OS_ReadSysInfo 8 Flags

OS_ReadSysInfo

If bits are set, it provides additional information about the hardware platform.

Flags

Bit / Platform class

0 – Platform class: Podules supported

1 – Platform class: PCI supported

2 – Platform class: Multiple processors supported

3 – Platform class: Software control over PSU

4 – Platform class: OS runs from RAM

5 – Platform class: OS uses rotated loads

6 – Platform class: OS uses unaligned loads/stores

Notes

If both bits 5 and 6 are flagged as valid (in R2) but neither are set (in R1) then it indicates that the OS only performs aligned memory accesses. If necessary, OS_PlatformFeatures 34 can be used to determine which alignment modes are supported by the CPU.

[RISC OS Open source page](#)

OS_ReadSysInfo 8 Platform Class

OS_ReadSysInfo

The value returned specifies the type of hardware platform.

Platform class

0 – Platform class: Unknown

1 – Platform class: Acorn RISC PC

2 – Platform class: Acorn A7000

3 – Platform class: ARM7500FE based

4 – Platform class: Phoebe

5 – Platform class: HAL

6 – Platform class: VirtualRPC
 7 – Platform class: A9Home
 10 – Platform class: Pyromaniac
[RISC OS Open source page](#)

OS_ReadUnsigned Flags

OS_ReadUnsigned

1 This flag is only checked for if the “WIDE” API is used.

Flags

Bit / Meaning when set

28 – Meaning when set: Read a 64-bit value1
 29 – Meaning when set: Must be in the range [0,R2]
 30 – Meaning when set: Must be in the range [0,255]
 31 – Meaning when set: Must terminate with control char or space

[RISC OS Open source page](#)

OS_ReadVarVal Variable Types

OS_ReadVarVal

The purpose of this call is to read a variable's value

Variable Types

/ Type / Description

0 – Type: String · Description: Pointer to string, which will be GSTransed now (will not be 0 terminated)
 1 – Type: Number · Description: 4 byte integer
 2 – Type: Macro · Description: Pointer to string, which will be GSTransed each time it is used
 3 – Type: Expanded · Description: Expression which will be sent through OS_EvaluateExpression
 4 – Type: Literal · Description: String that will not be GSTransed
 +16 – Type: Code · Description: Pointer to a block of code

[RISC OS Open source page](#)

OS_SetColour Flags

OS_SetColour

If bit 3 is set, then any pixels within the generated ECF block which would normally use the background colour will instead be made transparent.

Flags

0 - 2 – Meaning: GCOL action
 3 – Value: 0 · Meaning: Do not use transparency
 1 · Use transparency
 4 – Value: 0 · Meaning: Alter foreground colour
 1 · Alter background colour

- 5 – Value: 0 · Meaning: R1 is colour number
 - 1 · R1 is pointer to ECF pattern
- 6 – Value: 0 · Meaning: Set graphics colour
 - 1 · Set text colour1
- 7 – Value: 0 · Meaning: Write colour
 - 1 · Read colour 1
- 6 - 31 – Meaning: Reserved, must be zero

Notes

1 Introduced in RISC OS 3.5

[RISC OS Open source page](#)

OS_SynchroniseCodeAreas Flags

OS_SynchroniseCodeAreas

The purpose of this call is to ensure that the code cache is up to date with memory.

Flags

Bit / Value / Meaning

- 0 – Value: 0 · Meaning: Entire address space to be synchronised
 - 1 · Address range to be synchronised
- 2 - 31 – Meaning: Reserved, must be zero

Notes

R1 & R2 on entry are only used if bit 0 of R0 is set.

1 On RISC OS 5.25 and later, this may only clean application space and the RMA.

[RISC OS Open source page](#)

OS_Word 0 Buffer

OS_Word

OS_Word 0 is deprecated. To read the a line from the input stream, please use OS_ReadLine instead.

Buffer

Offset / Contents

- +0 – Contents: Two byte buffer address
- +2 – Contents: Size of buffer
- +3 – Contents: Lowest ASCII code
- +4 – Contents: Highest ASCII code

[RISC OS Open source page](#)

OS_Word 13 Buffer

OS_Word

The buffer in R1 gets updated with the current and previous graphics cursors co-ordinates.

Buffer

- 0 – Contents: Previous x co-ordinate (LSB)
- 1 – Contents: Previous x co-ordinate (MSB)
- 2 – Contents: Previous y co-ordinate (LSB)
- 3 – Contents: Previous y co-ordinate (MSB)
- 4 – Contents: Current x co-ordinate (LSB)
- 5 – Contents: Current x co-ordinate (MSB)
- 6 – Contents: Current y co-ordinate (LSB)
- 7 – Contents: Current y co-ordinate (MSB)

Notes

OS_Word 13 is deprecated. Please use OS_ReadVduVariables instead.

[RISC OS Open source page](#)

OS_Word 22 Type

OS_Word

Type

Bit / Meaning when set

- 0 – Meaning when set: Set screen base used by VDU
- 1 – Meaning when set: Set screen base used by hardware

[RISC OS Open source page](#)

OS_Word 9 Input Block

OS_Word

OS_Word 9 is deprecated. Please use OS_ReadPoint instead.

Input Block

Offset / Contents /

- +0 · X co-ordinate (two bytes)
- +2 · Y co-ordinate (two bytes)

[RISC OS Open source page](#)

OS_Word 9 Output Block

OS_Word

OS_Word 9 is deprecated. Please use OS_ReadPoint instead.

Output Block

Offset / Contents

+4 – Contents: Contains logical colour of specified pixel

[RISC OS Open source page](#)

OS_Word Introduction

OS_Byte & OS_Word

RISC OS provide numerous SWIs that perform many different operations. Most SWIs functions are specified by their name. e.g. OS_RemoveCursors removes cursors. However, there are two SWIs that perform a wide variety of different operations. They are OS_Byte and OS_Word.

In practice

This document provides information on OS_Word.

OS_Word is used to either read or write a large number of status variables at once.

The OS_Word SWI always uses two registers. The registers upon entry are detailed below:

Register / Use

R0 – Use: Reason code. Used to specify the operation to be performed. (0 - 255)

R1 – Use: Parameter 2. Pointer to parameter block, that can be used to stored the returned data

Where OS_Byte is used to return small amounts of data (usually byte sized - hence the name), OS_Word usually returns larger amounts of data. The value passed in R1 points to an area of memory where the data returned from the SWI can be stored.

The size of parameter block required is dependant on the reason code. In addition, some reason codes, contain a sub-reason code, which specifies the size of parameter block.

Reason codes

The value passed by R0 specifies the operation to be performed. The reason code must be a value from 0 to 255. In some cases, sub-reason codes are used to further specify the operation to be performed.

Interrupt handling

OS_Word is a vectored SWI, but before its vector is called, interrupts are disabled. Most OS_Word routines are so short that there is no need to re-enable interrupts, and instead rely on RISC OS to do this when control is returned to the program that issued the SWI.

Because the interrupts are not re-enabled, the routines are also used by interrupt handling routines.

If you wish to replace an OS_Word routine, you must ensure that the replacement routine:

- does not change the way it alters the interrupt status
- does not take so long that interrupts are disabled for unreasonable period of time

Adding OS_Word calls

Because the OS_Word is a vectored SWI, it is possible to add your own OS_Word calls by installing a routine on the software vector that OS_Word uses.

For more information, read the Software Vectors section.

*Command

There is no *Command that provides access to OS_Word's functionality.

[RISC OS Open source page](#)

PaletteV

Vectors

This vector is called whenever a call is made to read or write the palette. It has multiple uses, depending on the reason code in R4.

Before calling

R4 Reason Code

Other registers dependant on call

Reason Codes

- 1 – : Read palette
- 2 – : Set palette
- 3 – : Set first flash state
- 4 – : Set second flash state
- 5 – : Set default palette
- 6 – : Blank screen
- 7 – : Read palette entries
- 8 – : Write palette entries
- 9 – : Set gamma correction tables
- 10 – : Set LCD colour inversion?
- 11 – : Dim screen
- 12 – : Place video hardware in power saving mode?
- 13 – : Return video hardware from power saving mode?
- 14 – : Read gamma correction tables
- 15 – : Read supremacy transfer table
- 16 – : Set supremacy transfer table
- 256 – : HiResGammaCorrection1
- 1 Cerilica

[RISC OS Open source page](#)

PaletteV 1

PaletteV

This vector is called when the palette is read.

Before calling

R0 Logical colour

R1 type of colour (16, 17, 18, 24, 25)

R4 1 (reason code)

On return

R0 Preserved

R1 Preserved

R2 1st flash colour (&BBGGRRxx) – device colour

R3 2nd flash colour (&BBGGRRxx) – device colour

R4 0 (if operation complete)

[RISC OS Open source page](#)

PaletteV 11

PaletteV

This vector is called the ScreenBlanker module to dim and undim the screen.

Before calling

R0 0 to undim, or 1 to dim

R4 11 (reason code)

[RISC OS Open source page](#)

PaletteV 14

PaletteV

This vector is used to read the current gamma correction tables.

Before calling

R0 Pointer to table for red

R1 Pointer to table for green

R2 Pointer to table for blue

R4 14 (reason code)

On return

R0-R2 Tables filled in with current gamma tables

R4 0 if vector claimed

Notes

Each table is a word aligned array of 256 bytes giving the gamma corrected value when indexed by the normal colour value.

[RISC OS Open source page](#)

PaletteV 15

PaletteV

This vector is called to read the supremacy gamma table.

Before calling

R0 Pointer to memory for holding supremacy gamma table

R4 15 (reason code)

On return

R4 0 if vector claimed

All other registers preserved

Notes

The table is a word aligned array of 256 bytes giving the gamma corrected value when indexed by the supremacy value.

[RISC OS Open source page](#)

PaletteV 16

PaletteV

This vector is called when setting the supremacy gamma correction table.

Before calling

R0 Pointer to supremacy gamma table

R4 16 (reason code)

On return

R4 0 if vector claimed

All other registers preserved

Notes

The table is a word aligned array of 256 bytes giving the gamma corrected value when indexed by the supremacy value.

[RISC OS Open source page](#)

PaletteV 2

PaletteV

This vector is called when the palette is to be set.

Before calling

R0 Logical colour

R1 type of colour (16, 17, 18, 24, 25)

R2 1st flash colour (&BBGGRRxx) – device colour

R3 2nd flash colour (&BBGGRRxx) – device colour

R4 2 (reason code)

On return

R0 Preserved

R1 Preserved

R2 Preserved

R3 Preserved

R4 0 (if operation complete)

[RISC OS Open source page](#)

PaletteV 3

PaletteV

This vector is called when the 1st flash rate is to be set.

Before calling

R4 3 (reason code)

On return

R4 0 (if operation complete)

Notes

The 2nd flash rate vector is PaletteV 4.

[RISC OS Open source page](#)

PaletteV 4

PaletteV

This vector is called when the 2nd flash rate is to be set.

Before calling

R4 4 (reason code)

On return

R4 0 (if operation complete)

Notes

The 1st flash rate vector is PaletteV 3.

[RISC OS Open source page](#)

PaletteV 5

PaletteV

This vector is called when the default palette is set.

Before calling

R4 5 (reason code)

On return

R4 0 (if operation complete)

[RISC OS Open source page](#)

PaletteV 6

PaletteV

This vector is called the ScreenBlanker module to blank and unblank the screen.

Before calling

R0 0 to unblank, or 1 to blank

R4 6 (reason code)

[RISC OS Open source page](#)

PaletteV 7

PaletteV

This vector is called when reading palette entries.

Before calling

R0 Pointer to word aligned list of logical colours (words) or 0

R1 Type and number of colours

R2 Pointer to buffer for first flash colour (&BBGGRRxx)

R3 Pointer to buffer for second flash colour (&BBGGRRxx)

R4 7 (reason code)

[RISC OS Open source page](#)

PaletteV 8

PaletteV

This vector is called when setting palette entries.

Before calling

R0 Pointer to word aligned list of logical colours (words) or 0

R1 Type and number of colours

R2 Pointer to list of device colours

R4 8 (reason code)

[RISC OS Open source page](#)

PaletteV 9

PaletteV

This vector is called when setting the gamma correction tables.

Before calling

R0 Pointer to table for red

R1 Pointer to table for green

R2 Pointer to table for blue

R4 9 (reason code)

On return

R4 0 if vector claimed

All other registers preserved

Notes

Each table is a word aligned array of 256 bytes giving the gamma corrected value when indexed by the normal colour value.

[RISC OS Open source page](#)

PCI

List of modules

RISC OS 5 supports 32-bit and 64-bit PCI cards as well as PCI express (PCIe) cards via the PCI Manager, which provides a number of common services to the device driver modules for specific PCI cards.

In practice

It is envisaged that third parties will supply PCI cards bundled with the necessary RISC OS device drivers and the information supplied here should help developers to write those drivers.

A PCI device driver will normally be supplied as a RISC OS module which provides a set of SWIs and other interfaces to the PCI card. Applications should access the PCI card using SWI calls provided by this device driver module.

Details

- Using PCI Devices
 - Service Calls
 - SWI Calls
 - *Commands
- [RISC OS Open source page](#)

PCI_HardwareAddress Address Index

PCI

Register interface

Before calling

R1 Address Index:

0-5 correspond to first 6 Base Address Registers, or
A device with 3 64-bit addresses would use indexes 0, 2, 4 only, or
&100 corresponds to ROM, or
All other values reserved
[RISC OS Open source page](#)

PCI_HardwareAddress Input Flags

PCI

Input Flags

Bits / Meaning when set

- 0 - 3** – Meaning when set: Access privileges (if bit 9 set)
- 4** – Meaning when set: Bufferable
- 5** – Meaning when set: Cacheable
- 6 - 8** – Meaning when set: Cache policy (see Memory Cache Policies)
- 9** – Meaning when set: Access privileges specified (bits 0 - 3)
- 31** – Meaning when set: Do not map, query PCI address only

[RISC OS Open source page](#)

PCI_HardwareAddress Output Flags

PCI

Output Flags

Bits / Meaning

- 0** – Meaning: 0 = Memory
- 1** = IO
- 3** – Meaning: 1 = Prefetchable

[RISC OS Open source page](#)

PCI_LogicalAddress Flags

PCI

Flags

Bits / Meaning when set

4 – Meaning when set: Bufferable

5 – Meaning when set: Cacheable

6 - 8 – Meaning when set: Cache policy (see Memory Cache Policies)

30 – Meaning when set: 0 = Memory

1 = IO

[RISC OS Open source page](#)

PCI_MemoryRead Flags

PCI

Flags

Bits / Meaning when set

3 – Meaning when set: Prefetchable

[RISC OS Open source page](#)

PCI_ReadInfo Request

PCI

Each set bit, specifies the information that must be returned by PCI_ReadInfo.

Information Request

0 – Meaning when set: Device/Function Number

1 – Meaning when set: Bus number

2 – Meaning when set: Function handle of parent bridge

3 – Meaning when set: Slot number

4 – Meaning when set: Vendor/device ID

5 – Meaning when set: Revision ID

6 – Meaning when set: Subsystem vendor/subsystem ID

7 – Meaning when set: Class code (24 bits)

8 – Meaning when set: CMOS address

9 – Meaning when set: CMOS size in bytes

10 – Meaning when set: Pointer to description (0 for none)

11 – Meaning when set: Device vector number

12 – Meaning when set: Ethernet address (low 32 bits)

13 – Meaning when set: Ethernet address (high 16 bits)

14 – Meaning when set: Logical DMA channel

15 – Meaning when set: Pointer to vendor description (0 for none)

16 – Meaning when set: Number of MSI messages (0 if MSI not available)

17-31 – Meaning when set: Reserved for future expansion

[RISC OS Open source page](#)

Percussion

List of modules

The Percussion module provides a SoundVoice file that can be used by desktop or non-desktop application to play a percussion sound.

Details

- Service Calls

[RISC OS Open source page](#)

Pinboard

List of modules

The Pinboard allows for icons to be 'pinned' to the desktop. Icons can represent files, applications and directories. The standard desktop is usually grey in colour, but can be changed to another colour or made to display an image.

Introduction

All facilities are provided by the Pinboard Module.

Details

- Service Calls
- *Commands

[RISC OS Open source page](#)

Pointer Device Numbers

Controlling sources: Hdr:Pointer managed by the allocations manager

These are codes that represent the different types of pointer devices that can be attached to the hardware.

- 0 – Description: Quadrature mouse · Driver module: Mouse
- 1 – Description: Microsoft mouse · Driver module: SerialMouse
- 2 – Description: Mouse Systems mouse · Driver module: SerialMouse
- 3 – Description: PS/2 mouse · Driver module: PS2Driver
- 5 · RCMM handset (NCOS)
- 7 – Description: USB mouse (RISC OS 5 USB stack) · Driver module: USBDriver
- 10 · STD PS/2 to quadrature converter
- 11 · USB mouse (Castle)
- 15 – Description: Pandora controller · Driver module: PandoraKey?

Notes

The quadrature mouse is the original RISC OS mouse type that was used on Archimedes machines and the RiscPC. IOMD-based machines (including the RiscPC) introduced support for PS/2 mice, while later machines generally only support USB mice as standard.

OS_Pointer 0 is used to return the pointer device that is currently attached.

OS_Pointer 1 is used to set the pointer device type.

[RISC OS Open source page](#)

Pointer_Entering_Window

Reason Code 5 · WindowManager

This code is returned from Wimp_Poll and indicates that the mouse pointer is now directly over the windows visible work area.

On return

Block

+0 Window handle

Notes

This code is usually returned when the mouse pointer has moved over a window's visible work area, however, it can also happen due to operations that result in the pointer finding itself directly over the windows visible work area, such as when a menu that the mouse was over, closed.

If a pointer is over a window and it is deleted just before creating a new window and displaying it, the following will happen:

- Wimp_Poll will not return a Pointer_Leaving_Window for the old window
- Wimp_Poll will not return a Pointer_Entering_Window for the new window

[RISC OS Open source page](#)

Pointer_Leaving_Window

Reason Code 4 · WindowManager

This code is returned from Wimp_Poll and indicates that the mouse pointer is no longer over the windows visible work area.

On return

Block

+0 Window handle

Notes

This code is usually returned when the mouse pointer has moved outside a window's visible work area, however, it can be as a result of the pointer being over a menu that has opened.

[RISC OS Open source page](#)

PointerV

Vectors

This vector is used by the Kernel and the pointer device driver to communicate with each other.

Before calling

R0 Reason Code

Reason Codes

- 0** – : Request pointer device status
- 1** – : Enumerate pointer device types
- 2** – : Pointer device type selected
- 3** – : Report
- 4** – : Extended Request
- 5** – : Touch State
- 9** – : Wheel Change

Other registers dependant on call

[RISC OS Open source page](#)

PointerV 0

PointerV

This vector is called by the Kernel every VSync to obtain the latest pointer movement.

Before calling

R0 0 (reason code)

R1 Pointer device number

Pointer Vector (38)

Exit (relative pointing device)

R0 · Preserved

R1 · Preserved

R2 · X movement since last call

R3 · Y movement since last call

R4 · Preserved

Exit (absolute pointing device)

R0 · Preserved

R1 · Preserved

R2 · X position

R3 · Y position

R4 · "Abso" (&6f736241)

Notes

The driver for the specified device type should intercept the call and return the movement of the device since the previous call of this reason code, otherwise the call should be passed on.

For relative pointing devices, movements are signed 32 bit quantities. The values will be scaled by the kernel (by the configured mouse step) and then added on to the current position.

For absolute pointing devices, R2 and R3 must specify the new position in OS units. This mode of response is supported by RISC OS 5, and must not be used on other OS versions (the caller won't be expecting R4 to be changed).

The kernel updates the pointer position with the returned values. The driver should issue KeyV requests for mouse buttons they provide.

[RISC OS Open source page](#)

PointerV 1

PointerV

This vector is called by the Kernel when enumerate the pointer device types.

Before calling

R0 1 (reason code)

R1 Linked list of driver records, or 0 if none

On return

R0 Preserved

R1 Updated driver record linked list

Notes

A pointer device driver must claim this reason code and when called claim space for its record(s) in RMA. It must set the next record pointer of the last record it added to the value of R1 on entry, and pass on the call with R1 pointing to the first record it added.

The caller is responsible for freeing the list after use. The call must not be intercepted.

[RISC OS Open source page](#)

PointerV 2

PointerV

This vector is called by the Kernel when a pointer device type is selected with OS_Pointer.

Before calling

R0 2 (reason code)

R1 Pointer device number

Notes

A device driver should enable itself if it supports this device type, otherwise disable itself.

The call must not be intercepted.

This particular reason code is only available from RISC OS 3.5 onwards.

[RISC OS Open source page](#)

PointerV 3

PointerV

This vector is called by the pointer driver to notify the kernel when a pointer movement is detected.

Pointer Vector (38)

Entry (relative pointing device)

R0 · 3 (reason code)

R1 · Pointer device number

R2 · Change in X position

R3 · Change in Y position

R4 · 0

Entry (absolute pointing device)

R0 · 3 (reason code)

R1 · Pointer device number

R2 · X position

R3 · Y position

R4 · "Abso" (&6f736241)

Notes

Once this call has been received, the kernel will cease polling the driver via PointerV 0. Therefore if a driver issues PointerV 3 once, it must continue to issue it for the rest of its lifetime, or at least until another device is selected via PointerV 2.

For relative pointing devices, movements are signed 32 bit quantities. The values will be scaled by the kernel (by the configured mouse step) and then added on to the current position. Note that the OS does not require R4 to be zero, but it's recommended to set it to that value to ensure there's no chance of the call being mistaken for an absolute position update (or any other kind of update mode that may be added in the future).

For absolute pointing devices, R2 and R3 must specify the new position in OS units.

This particular reason code is only available from RISC OS 5 onwards.

[RISC OS Open source page](#)

PointerV 4

PointerV

This vector is called by the OSPointer module when it wants to receive an update from a pointer driver. If a driver recognises the pointer type as being its own, then it should claim the call and return with either reason code -1 (for relative devices, e.g. mice) or 5 (for absolute devices, e.g. touch screens).

Before calling

R0 4 (reason code)

R1 Pointer device number

Pointer Vector (38)

R0 · -1

R1 · Signed 32bit change in X position

R2 · Signed 32bit change in Y position

R3 · Signed 32bit change in X position of alternate device (i.e. horizontal scroll wheel)

Positive = Right

R4 · Signed 32bit change in Y position of alternate device (i.e. vertical scroll wheel)

Positive = "Up" / away from user

R5 · Button states:

Bit 0: Right button

Bit 1: Middle button

Bit 2: Left button

Bits 3-7: Extra buttons

Bits 8-31: Reserved, must be zero

Exit (absolute pointing device)

R0 · 5

R1 · Absolute X position, 16bit fraction

R2 · Absolute Y position, 16bit fraction

R3 · 0

R4 · 0

R5 · Button states (same as relative devices)

Notes

A driver which responds to this call should not use KeyV to report button state. It should only report the button state in response to this call. This will require some care when crafting a driver which is intended to be compatible with PointerV 1 or PointerV 3.

For absolute devices, the coordinates provided must be in the range 0-65535 inclusive. The returned value will be scaled by the screen size in order to convert the position to pixels.

This particular reason code is only used by the RISC OS Select OSPointer module.

[RISC OS Open source page](#)

PointerV 9

PointerV

This vector should be called by the current pointer driver to notify third-party software such as the WimpScroll module of any scroll wheel or additional button state changes.

Before calling

R0 9 (reason code)

R1 Scroll wheel Y axis delta

Negative = "Up" / away from user

R2 State of additional buttons (starting with mouse button 4 in bit 0)

R3 Scroll wheel X axis delta

Positive = Right

On return

- All registers preserved

Notes

R2 should contain the state of any additional buttons, i.e. beyond the 3 basic buttons (which should have their state reported via KeyV)

Under RISC OS 5, the current behaviour of USBDriver is to only make this service call if the pointer is not over a Wimp window which accepts scroll requests. If the pointer is over a window which accepts scroll requests, a scroll request Wimp

message will be sent instead.

[RISC OS Open source page](#)

PollWord_NonZero

Reason Code 13 · WindowManager

This code is returned from `Wimp_Poll` and indicates that the poll word is non-zero.

On return

Block

+0 Address of poll word

+4 Contents of poll word

Notes

If bit 23 of the Poll Mask is set, then the poll word may be returned before window redraw requests and certain other messages have been issued by `Wimp_Poll`.

This reason code can be very useful to update the underlying data that is to be displayed, before the request to update the windows or icons are returned by `Wimp_Poll`.

This is used to transfer control to an applications foreground process, where control is currently in an interrupt routine, service call handler or such like.

There is no guarantee of the duration until an application regains control, since other applications can take control away from the Wimp for unspecified amounts of time.

[RISC OS Open source page](#)

Portable

List of modules

The Portable module was first introduced in the version of RISC OS 3 that ran on the A4 laptop. The module had three main functions:

Introduction

- Power management
- Display support
- Responding to the extra keyboard hotkeys

Although the A4 is the only native RISC OS portable machine to be released to date, the legacy of the Portable module lives on. Several key components of RISC OS (e.g. the Wimp) still contain active code that will attempt to interface with the Portable module if it is present. This has enabled developers to use the original Portable APIs as a framework upon which to implement their own versions of the module, to deal with the power management requirements of their own hardware.

Versions

There are at least 8 variants of the Portable module in existence:

- The A4 version, for the A4 portable
- The Stork version, for the ill-fated Stork portable
- 'Portable75', for ARM7500-based machines. The Portable75 source can also be used to build 'PortableCX', for CX24430-based machines.

- 'PortableHAL', for HAL-based machines
- The A9 version, for the A9/A9Home
- The VirtualRPC and RPCEmu emulators each have their own versions of the module

For an overview of which features are available in which version, see the feature matrix

Power Management Features

The combined APIs and implementations of the different Portable modules offer the following power management features:

- Battery charger control and battery monitoring. This includes emulators, where the module may report the battery status of the host machine.
- CPU speed controls, to help reduce power consumption when the CPU is idle.
- Hardware subsystem power controls, to turn off power to devices which aren't currently needed.
- Suspend-to-RAM functionality, to place the machine into a deep sleep state at the request of the user or after a certain period of inactivity.

Display Support Features

The A4 portable featured both a builtin greyscale LCD panel and a VGA connector for the attachment of an external monitor. For the LCD panel to function correctly the correct mode timings have to be used, along with a custom palette. Therefore the original Portable module had the job of ensuring that a suitable video mode is in use whenever the LCD panel is enabled, along with providing a method for the user to control whether the machine used the internal LCD panel or an external VGA monitor.

The version of the Portable module used in the Stork portable fulfilled a similar role.

Hotkey Features

The A4 and Stork both featured extra keys on the machine, which would generate key events when pressed. The purpose of these keys was to control hardware features such as adjusting the screen contrast and backlight; therefore in order for the keys to work the Portable module had to listen out for them, and take the appropriate action when each key was pressed.

Details

- Feature Matrix
- Battery Management Unit status flags
- Battery Management Unit variables
- Service Calls
- SWI Calls
- *Commands

Events

[RISC OS Open source page](#)

Portable Feature Matrix

Portable

The tables on this page provide an overview of which Portable features are available in which variant of the module. Note that the contents of this page should only be used as a guide; programs wishing to use the Portable module should always use the appropriate interfaces (e.g. `Portable_ReadFeatures`) to determine the availability of the features they require.

In practice

- SWIs
- Service calls
- *Commands
- `Portable_Control` bits
- BMU variables
- BMU variable 10
- BMU commands
- `Portable_Stop` wakeup reasons
- `Portable_Status` bits

SWIs

This table lists the supported SWIs. The SWIs which the Stork version “Maybe” support are present in the source code, but are currently disabled.

42FC0 · Portable_Speed – A4: Y · Stork: - · 75/CX: - · HAL: Y · A9: ? · VirtualRPC: ? · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FC1 · Portable_Control – A4: Y · Stork: Y · 75/CX: - · HAL: - · A9: ? · VirtualRPC: ? · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FC2 · Portable_ReadBMUVariable – A4: Y · Stork: Y · 75/CX: - · HAL: Y · A9: Y · VirtualRPC: Y · RPCEmu: - · Pyromaniac (PSUtil/OSX): Y / Y

42FC3 · Portable_WriteBMUVariable – A4: Y · Stork: Y · 75/CX: - · HAL: - · A9: - · VirtualRPC: - · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FC4 · Portable_CommandBMU – A4: Y · Stork: Y · 75/CX: - · HAL: - · A9: ? · VirtualRPC: ? · RPCEmu: - · Pyromaniac (PSUtil/OSX): Y / Y

42FC5 · Portable_ReadFeatures – A4: - · Stork: Y · 75/CX: Y · HAL: Y · A9: Y? · VirtualRPC: Y? · RPCEmu: Y · Pyromaniac (PSUtil/OSX): Y / Y

42FC6 · Portable_Idle – A4: - · Stork: Y · 75/CX: Y · HAL: Y · A9: Y? · VirtualRPC: Y? · RPCEmu: Y · Pyromaniac (PSUtil/OSX): Y / Y

42FC7 · Portable_Stop – A4: - · Stork: Y · 75/CX: - · HAL: - · A9: ? · VirtualRPC: ? · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FC8 · Portable_Status – A4: - · Stork: Y · 75/CX: - · HAL: - · A9: ? · VirtualRPC: ? · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FC9 · Portable_Contrast? – A4: - · Stork: Maybe · 75/CX: - · HAL: - · A9: - · VirtualRPC: ? · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FCA · Portable_Refresh? – A4: - · Stork: Maybe · 75/CX: - · HAL: - · A9: - · VirtualRPC: ? · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FCB · Portable_Halt? – A4: - · Stork: Maybe · 75/CX: - · HAL: - · A9: - · VirtualRPC: ? · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FCC · Portable_SleepTime – A4: - · Stork: Maybe · 75/CX: - · HAL: - · A9: - · VirtualRPC: ? · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FCD · Portable_SMBusOp? – A4: - · Stork: Maybe · 75/CX: - · HAL: - · A9: - · VirtualRPC: ? · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FCE · Portable_Speed2 – A4: - · Stork: - · 75/CX: - · HAL: Y · A9: - · VirtualRPC: - · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FCF · Portable_WakeTime? – A4: - · Stork: - · 75/CX: - · HAL: - · A9: - · VirtualRPC: - · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FD0 · Portable_EnumerateBMU – A4: - · Stork: - · 75/CX: - · HAL: Y · A9: - · VirtualRPC: - · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FD1 · Portable_ReadBMUVariables – A4: - · Stork: - · 75/CX: - · HAL: Y · A9: - · VirtualRPC: - · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

42FD2 · Portable_ReadSensor – A4: - · Stork: - · 75/CX: - · HAL: Y · A9: - · VirtualRPC: - · RPCEmu: - · Pyromaniac (PSUtil/OSX): - / -

Service calls

This table lists the reason codes supported by `Service_Portable`.

/ Name / A4 / Stork / 75/CX / HAL / A9 / VirtualRPC / RPCEmu / Pyromaniac (PSUtil/OSX)

0 · PowerDown – A4: Y · Stork: Y · 75/CX: - · HAL: - · A9: ? · VirtualRPC: ? · RPCEmu: - · Pyromaniac (PSUtil/OSX): -

- 1 · **PowerUp** – A4: Y · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 2 · **TidyUp?** – A4: – · Stork: – · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 3 · **Freeze** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 4 · **Unfreeze** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 5 · **FreezeRequest** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –

*Commands

This table lists the supported *Commands.

Name / A4 / Stork / 75/CX / HAL / A9 / VirtualRPC / RPCEmu / Pyromaniac (PSUtil/OSX)

*FreezeTime – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): –

Portable_Control bits

This table lists the status/control bits supported by Portable_Control.

- 0 · **Econet power** – A4: Y · Stork: – · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 1 · **LCD power** – A4: Y · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 2 · **External display power** – A4: Y · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 3 · **Serial buffer & oscillator** – A4: Y · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 4 · **Dual panel mode** – A4: Y · Stork: – · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 5,6 · **Video clock control** – A4: Y · Stork: – · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 7 · **Invert video clock** – A4: Y · Stork: – · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 8 · **Backlight** – A4: Y · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 9 · **1/2 extra display lines** – A4: Y · Stork: – · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 10 · **1/2 DRAMs** – A4: Y · Stork: – · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 11 · **Sound filters & amplifiers** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 12 · **PCMCIA** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 13 · **Parallel port** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 14 · **FDC oscillator power** – A4: Y · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 15 · **IDE hard disc** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 16 · **Invert LCD palette** – A4: Y · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 17 · **Internal tracker ball in use** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 18 · **External mouse in use** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –
- 19 · **External kb & mouse power** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): –

BMU variables

This table lists the BMU variables supported by the BMU SWIs.

Note that there is no HAL column in the below table; apart from variables 12-20, which are deprecated in PortableHAL, the set of supported variables is entirely device-dependent. Well-written programs should query for supported variables at runtime, by attempting to read from the relevant variables (e.g. using Portable_ReadBMUVariables) and looking at the result codes.

- 0 · **Version & BMU memory map** – A4: Y · Stork: – · 75/CX: – · A9: Y · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
- 1 · **Nominal battery capacity** – A4: Y · Stork: – · 75/CX: – · A9: Y · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / Y
- 2 · **Measured battery capacity** – A4: Y · Stork: – · 75/CX: – · A9: Y · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / Y
- 3 · **Used battery capacity** – A4: Y · Stork: – · 75/CX: – · A9: Y · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
- 4 · **Usable battery capacity** – A4: Y · Stork: – · 75/CX: – · A9: Y · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
- 5 · **Reserved** – A4: Y · Stork: – · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
- 6 · **Charge estimate** – A4: Y · Stork: – · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
- 7 · **Instantaneous voltage** – A4: Y · Stork: Y · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / Y
- 8 · **Instantaneous current** – A4: Y · Stork: – · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / Y
- 9 · **Instantaneous temperature** – A4: Y · Stork: Y · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / Y
- 10 · **Status flags** – A4: Y · Stork: Y · 75/CX: – · A9: Y · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): Y / Y
- 11 · **Charge rate** – A4: Y · Stork: Y · 75/CX: – · A9: Y · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –

12 · **Charge current** – A4: – · Stork: Y · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 13 · **Discharge current** – A4: – · Stork: Y · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 14 · **Voltage** – A4: – · Stork: Y · 75/CX: – · A9: Y · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 15 · **Temperature 1** – A4: – · Stork: Y · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 16 · **Temperature 2** – A4: – · Stork: Y · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 17 · **Charge level** – A4: – · Stork: Y · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 18 · **Status flags (as variable 10)** – A4: – · Stork: Y · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 19 · **Input signals** – A4: – · Stork: Y · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 20 · **Battery capacity** – A4: – · Stork: Y · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 21 · **Battery time remaining, seconds** – A4: – · Stork: – · 75/CX: – · A9: – · VirtualRPC: Y · RPCEmu: – · Pyromaniac (PSUtil/OSX): Y / Y
 22 · **Max battery life in seconds** – A4: – · Stork: – · 75/CX: – · A9: – · VirtualRPC: Y · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 23 · **Battery percentage remaining** – A4: – · Stork: – · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): Y / Y
 24 · **Charge time remaining** – A4: – · Stork: – · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / Y
 25 · **Charge cycle count** – A4: – · Stork: – · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / Y
 26 · **Smoothed voltage** – A4: – · Stork: – · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / Y
 27 · **Smoothed current** – A4: – · Stork: – · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / Y
 28 · **Nominal available capacity** – A4: – · Stork: – · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / Y
 29 · **Compensated available capacity** – A4: – · Stork: – · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –

BMU variable 10

This table lists the supported flags in BMU variable 10, variable 18 (which is identical to 10), and by the Portable BMU event. Note that there is some confusion over bit 0; the RISC OS Select documentation incorrectly lists it as “Battery replaced”¹, while the Stork module describes it as an extra threshold, “Battery charged”. This latter meaning is used in PortableHAL.

Note that there is no HAL column in the below table; well-written programs should query for supported bits at runtime, via `Portable_EnumerateBMU`.

0 · **Battery replaced** – A4: – · Stork: – · 75/CX: – · A9: ? · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 – · **Threshold 3 (battery charged)** – A4: – · Stork: Y · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): Y / Y
 1 · **Lid open** – A4: Y · Stork: Y? · 75/CX: – · A9: ? · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / Y
 2 · **Threshold 2 (battery flat)** – A4: Y · Stork: Y? · 75/CX: – · A9: ? · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 3 · **Threshold 1 (battery low)** – A4: Y · Stork: Y? · 75/CX: – · A9: ? · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 4 · **Charge system fault** – A4: Y · Stork: Y? · 75/CX: – · A9: ? · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): Y / Y
 5 · **Charge state known** – A4: Y · Stork: Y? · 75/CX: – · A9: ? · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): Y / Y
 6 · **Battery present** – A4: Y · Stork: Y? · 75/CX: – · A9: ? · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): Y / Y
 7 · **Charger connected** – A4: Y · Stork: Y? · 75/CX: – · A9: ? · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): Y / Y
 8 · **Battery charging** – A4: – · Stork: – · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): Y / Y
 9 · **Battery failing** – A4: – · Stork: – · 75/CX: – · A9: – · VirtualRPC: – · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / Y

BMU commands

This table lists the supported `Portable_CommandBMU` reason codes. Note that although most documentation lists reason codes 2 and 3 as being reserved, the A4 source suggests that they were used to control battery charging.

1 · **Remove power** – A4: Y · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): Y / Y
 2 · **Set part charge** – A4: Y · Stork: – · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 3 · **Set full charge** – A4: Y · Stork: – · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 4 · **Set autostart** – A4: Y · Stork: – · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 5 · **Turn off AC power** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 6 · **Turn on AC power** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 7 · **Disable BMU event generation** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –
 8 · **Enable BMU event generation** – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –

Portable_Stop wakeup reasons

This table lists the supported `Portable_Stop` wakeup reasons.

/ Name / A4 / Stork / 75/CX / HAL / A9 / VirtualRPC / RPCEmu / Pyromaniac (PSUtil/OSX)

1 · Keyboard – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –

2 · RTC – A4: – · Stork: – · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –

Portable_Status bits

This table lists the various bits returned by Portable_Status.

/ Name / A4 / Stork / 75/CX / HAL / A9 / VirtualRPC / RPCEmu / Pyromaniac (PSUtil/OSX)

0 · Docked – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –

1 · External FDC connected – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –

2 · External trackball connected – A4: – · Stork: – · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –

3 · Internal display in use – A4: – · Stork: Y · 75/CX: – · HAL: – · A9: ? · VirtualRPC: ? · RPCEmu: – · Pyromaniac (PSUtil/OSX): – / –

1 As some point during the writing of the Select documentation, the wording was that bit 0 indicated that the battery had been 'changed' (a typo of 'charged'). During a review this got reworded without reference to the original source to indicate that the 'battery has been replaced', as a synonym for 'changed', probably because 'battery changed' doesn't look very nice as a description. Thus confusion. Sorry.

[RISC OS Open source page](#)

Portable_Control Mask

Portable

Changing bits 0, 3, 14 will generate Service_Portable.

Mask

0 – Meaning when set: Econet power enabled

1 – Meaning when set: LCD display power enabled

2 – Meaning when set: External video display power enabled

3 – Meaning when set: Serial buffer and oscillator power enabled

4 – Meaning when set: Dual panel mode

5 - 6 – Meaning when set: Video clock control:

0 = external clock

1 = crystal / 2

2 = crystal

3 = reserved

7 – Meaning when set: Invert video clock

8 – Meaning when set: Enable back light

9 – Meaning when set: Display has 2 extra lines (else 1 extra)

10 – Meaning when set: 2 DRAMs used for dual panel (else 1)

11 – Meaning when set: Sound filters and amplifiers power enabled

12 – Meaning when set: PCMCIA interface power enabled

13 – Meaning when set: Parallel port power enabled

14 – Meaning when set: FDC oscillator power enabled

15 – Meaning when set: IDE hard disc power enabled

16 – Meaning when set: LCD palette set for inverse video

17 – Meaning when set: Internal trackball in use

18 – Meaning when set: External mouse in use

19 – Meaning when set: External keyboard and mouse power enabled

20 - 31 – Meaning when set: Reserved, must be zero

Notes

The 82C711 in the A4 shares one oscillator – this will only be powered down if bits 3 & 14 are both clear. On some machines the oscillator is used by other parts of the system and can never be powered down.

Bits 17 and 18 are mutually exclusive.

Not all control bits are available in all versions of the module; see the Portable Feature Matrix.

[RISC OS Open source page](#)

Prefetch Abort Handler

OS_ChangeEnvironment

The prefetch abort handler is entered in the following state:

In practice

- The processor will be in ABT32 mode, with IRQs disabled
- FIQ state will match that at the time of the exception
- R14 & SPSR_und will contain the exception address and saved PSR, corresponding to when the exception was raised by the CPU. R14 will be pointing to the instruction after the instruction that caused the exception.
- The other core registers will be as per when the exception was raised by the CPU. However if your code examines the registers it must take care to use the correct register bank (e.g. if the exception was from user mode, it should look at R14_usr, not R14_abt).
- R13 will be a valid stack pointer
- CP15 registers relating to exception processing will be in the correct state corresponding to that exception

If the handler is capable of handling the exception then it can resume execution by performing a suitable exception return operation.

Notes

On 26bit OS versions, the CPU will be in SVC26 mode, with R14 containing the combined LR+PSR.

The kernel's default handler will save a copy of the ARM integer registers (corresponding to the CPU mode the exception occurred in) to the Exception Registers Block, reset the stack pointers for all the privileged CPU modes, and then call OS_GenerateError to report a Prefetch Abort error.

[RISC OS Open source page](#)

Printer Driver Numbers

Controlling sources: Hdr:PaletteV managed by the allocations manager

The table above is perhaps out-of-date. More information is required.

- 0 – Printer: PostScript
- 1 – Printer: Epson FX80 (or compatible)
- 2 – Printer: HP LaserJet (or compatible)
- 3 – Printer: Integrex ColourJet
- 4 – Printer: FAX modem
- 5 – Printer: Direct driver laser
- 6 – Printer: Caspel graphics language
- 7 – Printer: PDumper interface
- 8 – Printer: JP150 (crap Acorn inkjet)
- 9 – Printer: Risc Developments Deskjet

10-19 – Printer: Computer Concepts

20 – Printer: ArcFax

99 – Printer: Ace Computing Epson JX/Star LC10 driver or PaintJet driver

[RISC OS Open source page](#)

Processor Vector Numbers

Vectors

/ Vector

0 – Vector: Branch through zero

1 – Vector: Undefined instruction

2 – Vector: SWI executed

3 – Vector: Prefetch abort

4 – Vector: Data abort

5 – Vector: Address exception (ARMv2), Hypervisor trap (ARMv7VE, not used by RISC OS)

6 – Vector: IRQ

[RISC OS Open source page](#)

Program Environment

There are several different execution contexts which code can find itself running in. These execution contexts can place limits on what actions the code can perform – for example, interacting with the filing system from within an interrupt handler is forbidden. Although developers of simple applications may not need to know the details of the different execution contexts, if you are developing modules or using RISC OS-specific features such as callbacks then knowledge of at least some of the different contexts is necessary.

Execution contexts

At the basic level, there are two types of execution context: foreground and background. Applications are described as running in the foreground, and they have few restrictions on what operations they can perform (e.g. they can interact with the filing system, and there are no limits on execution time). Meanwhile, the background is mainly the domain of interrupt handlers and abort handlers, and there are several different types of restriction which can be in place.

Viewed from the least restrictive (foreground) to the most restrictive (background), the following execution contexts exist in the system:

- The foreground Application context (user-mode code) CallBack handlers Transient callbacks SWIs issued from the foreground, *Commands, and most other supervisor-mode code
- The background RTSupport threads Interrupt handlers (including e.g. OS_CallAfter / OS_CallEvery handlers) Abort handlers FIQ handlers

This nesting also represents the way in which the different contexts can interrupt each other; e.g. an application running in the application context can be interrupted by a transient callback or an RTSupport thread. However the execution of an interrupt handler cannot be interrupted by either of these; the interrupt handler must run to completion and the system must return one or more steps towards the foreground.

Of course there are some exceptions, e.g. if a FIQ handler crashes then the CPU will invoke the abort handler. And some foreground SWIs (e.g. OS_ReadC) may elect to trigger transient callbacks, bypassing the usual “are we returning to user mode?” check that the kernel performs.

[RISC OS Open source page](#)

RAMFSFiler

List of modules

The RAMFSFiler module provides the RAMFS icons on the Desktop icon bar, and uses the Filer to display RAMFSFS directories and their content.

Details

- Service Calls
- *Commands

[RISC OS Open source page](#)

ReadLineV

Vectors

This vector is called to indirect all calls to OS_ReadLine and OS_ReadLine32.

Before calling

R0 Buffer

R1 Size of buffer -1 (bit31 should be ignored)

R2 Lowest char allowed

R3 Highest char allowed

R4 Character to output + flags

R0 Buffer + flags

R1 Size of buffer -1

R2 Lowest char allowed

R3 Highest char allowed

R4 Character to output (if bit 30 set)

On return

R0 Preserve

R1 Chars read, excluding Return

R2 Preserve

R3 Preserve

R4 Preserve

C flag is set if input is terminated by an escape condition

R0 Buffer address, without flags

R1 Chars read, excluding Return

R2 Preserve

R3 Preserve

R4 Preserve

C flag is set if input is terminated by an escape condition

ReadLine Vector (15)

The API of this vector depends on processor mode.

In 32bit processor modes, the API is as for OS_ReadLine32:

In 26bit processor modes, the API is dependent on R1 bit31. If it is set, the API is as above (and bit31 must be masked out). Otherwise it is as OS_ReadLine:

Notes

Claimants must first check the processor mode. If running in a 32bit mode, the OS_ReadLine32 API is used, with flags in R4. Else, if R1 bit31 is set, the OS_ReadLine32 API is used, with flags in R4. Otherwise, the OS_ReadLine API is used, with flags in R0.

As well as the kernel, common implementations can be found in the ReadLine and LineEditor third-party modules.

[RISC OS Open source page](#)

Reason Codes

WindowManager

The following tables provides the Event/Reason codes that may be returned by Wimp_Poll:

- 0 – : Null_Reason
- 1 – : Redraw_Window_Request
- 2 – : Open_Window_Request
- 3 – : Close_Window_Request
- 4 – : Pointer_Leaving_Window
- 5 – : Pointer_Entering_Window
- 6 – : Mouse_Click
- 7 – : User_Drag_Box
- 8 – : Key_Pressed
- 9 – : Menu_Selection
- 10 – : Scroll_Request
- 11 – : Lose_Caret
- 12 – : Gain_Caret
- 13 – : PollWord_NonZero
- 14 – : Reserved
- 15 – : Reserved
- 16 – : Reserved
- 17 – : User_Message
- 18 – : User_Message_Recorded
- 19 – : User_Message_Acknowledge

Priority of Reason Codes

Highest / 17 - 19

... - 17 - 19: 1 - 6, 8, 9

... - 17 - 19: 7, 10 - 13

Lowest - 17 - 19: 0

[RISC OS Open source page](#)

Redraw_Window_Request

Reason Code 1 · WindowManager

This code is returned from Wimp_Poll and indicates that a window is in need of updating and must be redrawn.

On return

Block

+0 Window handle

Notes

To update and redraw a window that has been identified as requiring updating, applications should follow the process below:

- Call Wimp_RedrawWindow
- Start a loop by calling Wimp_GetRectangle to return a rectangular part of the window that needs updating
- Loop back to fetch next rectangular part of window needing updating until done

[RISC OS Open source page](#)

RemV

Vectors

This vector is called to remove one or more bytes from a buffer.

Before calling

R1 Buffer number plus flags:

b31 clear for Single Removal, set for Block Mode

b30 used by the DeepKeys extension

R1 Buffer number, b31 clear

R9 21 (&15) vector number

V flag is set if buffer to be examined only, or clear to remove a value

R1 Buffer number, b31 set

R2 Pointer to block to be filled

R3 Number of bytes to place into block

R9 21 (&15) vector number

V flag is set if buffer to be examined only, or 0 if data should be removed

On return

R0 Next value to be removed (examine option), or "corrupted" (remove option)

R1 Preserved

R2 Value removed (remove option), or "corrupted" (examine operation)

C flag set if buffer was empty on entry

R0 Preserved

R1 Preserved

R2 Pointer to updated buffer position

R3 Number of bytes still to be removed

C flag set if buffer was empty on entry

Single Removal (R1b31 clear)

To work around historical keyboard handling bugs, the Kernel always returns the same value in R0 and R2 for its buffers if C is clear. BufferManager does not match this sensible behaviour.

Notes

All characters removed from buffers are sent through this vector. Used by OS_Byte 145 and OS_Byte 152. If called via OS_CallAVector then interrupts must be disabled first. Note that Block Mode is not supported for all buffer numbers. BufferManager does support it for the buffers it manages (Serial, Printer, Mouse).

[RISC OS Open source page](#)

Resolver

List of modules

The Resolver module provides server name to IP address lookup facilities, looking first for entries in the local InetDBase:Hosts file and if not found there will query the remote Domain Name Service (DNS).

Details

- SWI Calls
- *Commands

[RISC OS Open source page](#)

ResourceFiler

List of modules

The ResourceFiler module provides the ResourceFS icons on the Desktop icon bar, and uses the Filer to display ResourceFS directories and their content.

Details

- Service Calls
- *Commands

[RISC OS Open source page](#)

ResourceFS

List of modules

The ResourceFS module provides the necessary functionality for modules to include files in the Resources: filing system.

In practice

This functionality is useful as it allows the resource files associated with a particular module to be included in the same file as the binary image, which helps with release control.

It also has an important application for expansion card modules, since it allows them to *IconSprites a sprite file which they put into Resources: . This is important as there is no other way to introduce a sprite into the Wimp's sprite pool other than from a file.

Another application is for certain resource files to be replaced on a selective basis, which is an additional technique to the path mechanism already in use (e.g. Wimp\$Path can be set up to reference a resource directory).

Details

- Service Calls
- SWI Calls
- *Commands

[RISC OS Open source page](#)

RiscPC Hardware

ARM Hardware

This section details the intricacies of the early hardware used by RISC OS. The first generation is commonly known as the Archimedes. The second generation is commonly known as the RiscPC range. The RiscPC range of computers were designed to take advantage of the more advanced features of the latest ARM processors, although many supporting chips were still proprietary chips designed by Acorn Computers Ltd.

In practice

RISC OS 3.5 up to RISC OS 4.x was used with this class of hardware. The information is specific to this type of hardware and version of RISC OS. The information is no longer valid or useful with RISC OS 5, but is available here for completeness.

Chips

The complete list of chips used with the more advanced RiscPC range of computers are as follows:

- ARM processor is the CPU
- VIDC20 chip which handled both video and sound output. VIDC is short for Video Controller
- IOMD chip handled the interface between the ARM, VIDC20 and all other supporting hardware in the machine. This chip replaced the earlier IOC and MEMC found in the Archimedes range of computers. IOMD is short for Input/Output Memory Device.

These chips are commonly known as the ARM chip set.

On some ranges of machines (such as the A7000), the functionality of these chips were included in the ARM processor itself (ARM7500 and ARM7500FE).

ARM Processor

The ARM processor used with the RiscPC range of computers was one of the following:

- ARM 600
- ARM 710
- StrongARM
- ARM7500 and ARM7500FE. (integrated system-on-chip)

Previous ARM processors were limited to accessing 64 MB of memory because of its 26-bit address bus. This limitation was removed on the newer ARM 600 processors, although it was possible to configure the processor to provide backwards compatibility. This change results in:

- Both the Program Status Register (PSR) and the Program Counter (PC) having its own register
- The PSR cannot be saved with the PC when changing processor modes. Each privileged mode has an extra register to hold the previous mode's PSR (known as the Saved Program Status Register SPSR)
- Undefined instructions, aborts and supervisor mode code no longer has to share the same mode

ARM7500 & ARM7500FE processor

The ARM7500xx processor is a system-on-chip that integrates processor, video generator and most of the functions of the IOMD chip. The major differences are:

- Provides two PS/2 asynchronous serial keyboard ports for keyboard and mouse, rather than the IOMD's synchronous serial keyboard port and quadrature mouse interface
- Provides a four channel PC joystick interface that is not available with the IOMD chip

Word Size

The ARM range of processors uses 32-bit words. Each instruction fits into a single word. It also has a 32-bit data bus so that complete instructions can be fetched in a single step.

Processor Configuration

As mentioned above, the newer ARM processors were updated to handle 32-bit address and data space. It was however possible to configure the processor to provide the original ARM address functionality. i.e. A 26-bit address bus.

The processor could be configured as any one of following:

- 26-bit program and data space Forces the ARM to use 26-bit address space. Only the four 26-bit modes are available, and it is impossible to select a 32-bit mode. This configuration is set at reset on all ARM 6xx and ARM 7xx processors.
- 26-bit program space and 32-bit data space Same as above, but address exceptions are disabled, to allow data transfer operations to access the full 32-bit address space.
- 32-bit program and data space (RISC OS uses this) Provides full 32-bit access, and enables any of the 26-bit or 32-bit processor modes.

RISC OS configures the control register of the ARM processor on start-up into 32-bit program and data space. This must not be changed.

Although 32-bit configuration is selected, much of RISC OS 3.xx still uses 26-bit modes for operation.

Because the processor is in 32-bit configuration, all exceptions (including Undefined Instructions and Software Interrupt) force the processor to a privileged 32-bit mode appropriate to the exception.

Processor Modes

When configured for 32-bit program and data space the following processor modes are available:

User Mode – Use for mode: Mode normally used by applications

User 26 Mode – Use for mode: 26-bit version of above

Supervisor Mode (SVC) – Use for mode: A Protected mode for the OS

Supervisor26 Mode (SVC26) – Use for mode: 26-bit version of the above

Interrupt Mode (IRQ) – Use for mode: Used to handle peripherals when they issue interrupt requests

Interrupt Mode (IRQ26) – Use for mode: 26-bit version of the above

Fast Interrupt Mode (FIQ) – Use for mode: Used to handle peripherals that issue fast interrupt requests

Fast Interrupt Mode (FIQ26) – Use for mode: 26-bit version of above

Abort Mode (ABT) – Use for mode: Entered after data or instruction prefetch abort

Undefined Mode (UND) – Use for mode: Entered when an undefined instruction is executed

When configured for a 26-bit processor mode, the behaviour is the same as the ARM v2as architecture, with the following exceptions:

- Address exceptions are only generated when it is configured for 26-bit program and data space
- New instructions to transfer data between general registers and the Program Status Registers remain operative. The instructions can be used by the OS to return to a 32-bit mode after executing a binary file for a 26-bit ARM processor

- When the processor is in the 32-bit program and data space configuration, all exceptions return the processor to a 32-bit mode, so the OS must be modified to handle them
- If the processor mode attempts to write to a location between &0F and &1F (inclusive), hardware prevents the write operation and will generate a data abort. This allows the OS to intercept the exception vectors and redirect the vector to some other code. The other code should put the processor in a 26-bit mode before calling the 26-bit exception handler.

Overview of Registers

Twenty seven 32-bit registers are available on the ARM 2 and ARM 3 processor. Of those, only 16 are available within RISC OS. Some registers are shared across different modes, while others are not. The table below shows this:

R0

R1

R2

R3

R4

R5

R6

R7

R8 · R8_fiq

R9 · R9_fiq

R10 · R10_fiq

R11 · R11_fiq

R12 · R12_fiq

R13 · R13_svc – IRQ Mode1: R13_irq · ABT Mode: R13_abt · UND Mode: R13_und · FIQ Mode1: R13_fiq

R14 · R14_svc – IRQ Mode1: R14_irq · ABT Mode: R14_abt · UND Mode: R14_und · FIQ Mode1: R14_fiq

R15 (PC)

CPSR

SPSR_svc – IRQ Mode1: SPSR_irq · ABT Mode: SPSR_abt · UND Mode: SPSR_und · FIQ Mode1: SPSR_fiq

1 Both 32-bit and 26-bit varieties

CPSR and SPSR Registers

As mentioned earlier, the Program Counter and the Program Status Registers have both got their own registers. The Current Process Status Register (CPSR) is detailed below:

0 - 4 – Contents: Processor Mode Flag¹

00000 = User26 Mode

00001 = FIQ26 Mode

00010 = IRQ26 Mode

00011 = SVC26 Mode

10000 = User Mode

10001 = FIQ Mode

10010 = IRQ Mode

10011 = SVC Mode

10111 = ABT Mode

11011 = UND Mode

6 – Contents: FIQ disable flag (set for disable)

7 – Contents: IRQ disable flag (set for disable)

28 – Contents: Conditional Flag - OverFlow (V)

29 – Contents: Conditional Flag - Carry (C)

30 – Contents: Conditional Flag - Zero (Z)

31 – Contents: Conditional Flag - Negative (N)

The VIDC20 chip

The VIDC20 chip controls the computer's video and sound. The data is read from its 64-bit data bus using DMA control and then processed and converted into the necessary analogue signals to drive the video output displays and sound system. The VIDC20 can handle many more display and sound formats than the original VIDC chip found in the Archimedes Hardware. It can also read data from VRAM if installed in the machine, otherwise it reads from DRAM.

Video

Data from the video buffer is converted and processed. They are as follows:

- Data is serialised by the VIDC20 chip into either 1, 2, 4, 8, 16 or 32 bits per pixel
- Passed through a colour look-up palette (256 registers, each is 28-bits wide; 8 Red, 8 Green, 8 Blue and 4 bits for external data)
- The output is then converted by three 8-bit digital to analogue converters (DACs). One for red, green and blue colour
- Output is then used to drive the display output device with a maximum of 16 million possible colours.

The VIDC20 chip can handle any pixel rate up to 110 MHz. The clock selected from one of three source, and can then be further divided by a factor between 1 and 8 (inclusive).

In addition it also contains a phase comparator which allows for a single clock to generate all the required frequencies for any display mode.

Sound

Up to eight separate channels of mono sounds is provided by the VIDC20 chip. The chip can work with 1, 2, 4 or 8 stereo channels using time division multiplexing to synthesize left and right outputs. Each channel can be assigned a stereo position (between left to right).

Data from the buffer is converted and processed. The following steps taken:

- Data from the buffer is read at a programmable rate
- Data is then passed to an 8-bit/16-bit DAC
- DAC uses the stereo image registers to convert the digital samples to a stereo analogue sample
- It is then sent to the computer's internal amplifier

Cursor

The cursor data controls a pointer that is 32 pixels wide, and an unlimited number of pixels high. The pixels can be transparent or can be any of three of the 28-bit wide palette. The cursor can be any shape within these limits.

The IOMD chip

The IOMD chip manages the memory, DMA and I/O within the hardware, and is a replacement to the original IOC and MEMC chips found in the Archimedes Hardware. The features of the IOMD are as follows:

- Memory controller for VRAM and DRAM
- DMA controller (for I/O, sound, cursor and video data)
- Direct interface to ARM processors
- 16-bit steered bus (for on-board peripherals)
- PC keyboard interface
- Mouse interface (Acorn Quadrature style)

[RISC OS Open source page](#)

RTSupport

List of modules

The RTSupport module provides facilities supporting real-time code, at a priority just below that of the interrupt dispatch system, and above that of any SWIs being handled by the foreground application. Multiple priority levels are supported, so routines with tighter timing constraints can pre-empt other real-time routines. Code can be designed either using a callback model (where the system makes a function call to your entry point) or using a thread model (where the rest of the system gets CPU time while your thread sleeps).

In practice

Code typically executes in system (SYS) mode and has its own SYS and SVC stacks, and it may call SWIs (but only re-entrant SWIs) without preventing higher-priority routines from executing. There is no automatic time-slicing of routines; it is assumed that the application takes responsibility for ensuring that the system is not overloaded (although there is a relief valve to cope with overloaded systems, which ensures that the foreground process gets a little CPU time at least a few times per second).

Possible future enhancements:

- Per-routine option of maintaining a floating point context for the routine (traps would need to be disabled while real-time routines execute)
- Per-routine data storage (for example to support C++ exception handling)
- Reusing spare stack frames in the IRQ stack when nearing overflow (useful for timeslicing schemes)
- The facility to recache the priority lookup file after module initialisation

Details

- SWI Calls

[RISC OS Open source page](#)

SATADriver

List of modules

The SATADriver Module provides low-level access to serial ATA devices.

Details

- Service Calls?
- SWI Calls
- *Commands

[RISC OS Open source page](#)

Screen Mode Types

The VDU drivers support several different types of screen mode. Mostly these different mode types are supported in order to provide compatibility with the BBC Micro range of computers.

Standard modes

These are the most common type of screen mode, capable of displaying both text and graphics. The VDU text font is 8×8 pixels.

When using a Mode Selector Block, the kernel will calculate the values of any unspecified mode variables as if a standard mode was selected. This means that care needs to be taken when attempting to select/describe other mode types – see each type below for details.

Gap modes

These are identified by the “gap mode” mode flag being set (bit 2).

Although the VDU font size is unchanged, the character cell which the font is rendered in is now 25% taller, resulting in gaps between the text rows.

To select a gap mode via a mode selector block, you must specify the “gap mode” mode flag, and a suitable value for ScrBRow.

Double-height modes

These are identified by the “Double height VDU chars” mode flag being set (bit 5).

In these modes, 2x upscaling will be used for the VDU font / character cells. By default this will result in characters that are 8×16 pixels in size.

To select a double-height mode via a mode selector block, you must specify the “double height VDU chars” mode flag, and a suitable value for ScrBRow. Currently the kernel only supports 1bpp double-height modes – using other colour depths may result in malfunction.

Hi-res mono modes

These are identified by the “Hi-res mono” mode flag being set (bit 4).

This was a special mode type created for Archimedes machines. The OS would treat the mode as a 1bpp mode (with the only restriction being that the palette is restricted to black and white), but the hardware would be programmed as if it was a 4bpp mode, with the pixel clock and horizontal timing parameters all divided by four. For the mode to be displayed correctly, suitable hardware would have to be fitted to the VED0-VED3 pins on the VIDC chip (and the analogue RGB output ignored).

Although modern RISC OS versions still contain the definition for mode 23, attempting to select it (or any other “hi-res mono” mode) is unlikely to yield useful results.

Teletext

These are identified by the “Teletext” mode flag being set (bit 1).

Under RISC OS, teletext modes provide a software emulation of the text-only teletext mode that was supported by the BBC Micro. The VDU driver will disallow graphics operations (the “non-graphics” mode flag will be set), but in reality it's still a bitmap graphics mode.

In order to recreate the teletext font, teletext modes use either an 8×10 (“gap mode” mode flag set) or 16×20 font (“gap mode” and “double vertical” mode flags set).

To select a teletext mode via a mode selector block, you must specify both the “teletext” and “gap mode” mode flags. However, note that this is only supported on OS versions where the T, TX and TY Mode String attributes are supported (RISC OS Select and RISC OS 5.23+); attempting to select a teletext mode via a mode selector block on other OS versions may fail with an error or cause a bad mode to be entered.

Unlike with other non-standard mode types, specifying the correct ScrRCol and ScrBRow values is optional – the kernel will automatically clamp the values to the allowed range.

BBC gap modes

These are identified by the “BBC gap mode” mode flag being set (bit 3).

On the BBC Micro, gap modes had no video memory allocated to the gap rows. This resulted in the gaps always appearing black, and the hardware text cursor appearing white for any areas where it overlaps the gap. The lack of video memory in the gaps also meant that the MOS VDU driver disabled graphics operations.

Under RISC OS, BBC gap modes are both “gap modes” and “non-graphics”. The gap effect is created by restricting the number of palette entries that are available for use by the VDU font: for modes 3 and 6 (the built-in BBC gap modes), Log2BPP will be 1 (suggesting a 2bpp mode), but NColour will be 1 (suggesting only two colours available). The VDU font will be restricted to colours 0 and 1, while colours 2 and 3 are used for the gap (and gap cursor).

To select a BBC gap mode via a mode selector block, you must specify the “BBC gap mode”, “gap mode”, and “non-graphics” mode flags. Suitable values for ScrBRow and NColour must also be specified. Currently the kernel only supports gap modes where Log2BPP=1 and NColour=1 – using other values may result in malfunction.

Double-pixel modes

These are identified by the value of the Log2BPC mode variable being one higher than Log2BPP.

In double-pixel modes, each logical screen pixel (as addressed by the VDU) is actually two physical screen pixels, in a 2×1 formation. This has the effect of making the VDU text font appear to the user as if it's twice as wide as normal.

Double-pixel modes are still graphics modes, with the VDU graphics routines correctly translating logical pixels to pairs of physical pixels. However other software which writes to screen memory directly may not be aware of the special nature of double-pixel modes, resulting in “half-pixels” being written, or graphics appearing in the wrong position / aspect ratio.

The double-pixel nature of the mode is also reflected in the fact that XWindLimit will be half its usual value – e.g. if the VIDC list specifies a horizontal display with of 640 pixels, XWindLimit will be set to 319. Typically adjusted eigen values will also be used: a mode which is physically 640×480 would typically use eigen values of 1:1. But the double-pixel version of that mode (with 320×480 logical pixels) would use eigen values of 2:1.

To select a double-pixel mode via a mode selector block:

- Specify Log2BPC as being Log2BPP+1
- Specify XWindLimit of (XRes/2)-1
- Specify a suitable ScrRCol value (default ScrRCol is (XRes/8)-1)
- Potentially specify XEig as being higher than normal

Implementing special mode types as numbered modes

For numbered modes, the correct values of all mode variables must be provided either by your Mode Workspace List or by the base mode.

Describing special modes using mode strings

Apart from teletext modes, there is no direct way of describing special modes using mode strings (due to the inability to directly specify the mode flags or Log2BPC).

You can include a mode number at the start of the mode string, however it's unclear whether any OS versions will correctly copy over the attributes of the base mode.

Relationship to Service_EnumerateScreenModes

The Service_EnumerateScreenModes Block only returns basic information about each mode: Its width and height, pixel format, and framerate. This information should be treated as describing the physical properties of the mode, i.e. as if a standard mode type were to be used.

If Service_EnumerateScreenModes listed a 2bpp 640×480 mode as being available, you would translate that definition to a mode selector block as follows:

- X resolution (XRes) = 640
- Y resolution (YRes) = 480
- Log2BPP = 1
- Frame rate = value from Service_EnumerateScreenModes, or -1

With no extra mode variables specified, this will select the standard version of the mode (i.e. 2bpp 640×480). To select other mode types, you must add the correct mode variable values onto the end of the block, as listed below:

Mode type / Mode Flags / ScrRCol / ScrBRow / NColour / Log2BPP / XWindLimit

Standard · &0 – ScrRCol: 79 · ScrBRow: 59 · NColour: 3 · Log2BPP: 1 · XWindLimit: 639

Gap mode · &4 – ScrRCol: 79 · ScrBRow: 47 · NColour: 3 · Log2BPP: 1 · XWindLimit: 639

Double-height · &20 – ScrRCol: 79 · ScrBRow: 29 · NColour: 3 · Log2BPP: 1 · XWindLimit: 639

BBC gap mode · &D – ScrRCol: 79 · ScrBRow: 47 · NColour: 1 · Log2BPP: 1 · XWindLimit: 639

Double-pixel · &0 – ScrRCol: 39 · ScrBRow: 59 · NColour: 3 · Log2BPP: 2 · XWindLimit: 319

Relationship to Service_ModeExtension

When a Service_ModeExtension implementation receives a mode selector block and searches its internal mode database for a suitable VIDC list, it must do so by only paying attention to the physical properties of the mode. I.e. it should determine the dimensions of the mode by looking at the XRes and YRes values, not by looking for any XWindLimit or YWindLimit mode variables (as these are used to indicate the logical dimensions).

To determine the pixel format, the code must look at the Log2BPP value from the block header, and any NColour and mode flags value provided in the mode variable pairs, being careful to ignore irrelevant mode flags (e.g. double height, non-graphic). See Valid Mode Variable Combinations for details.

Relationship to Service_ModeTranslation

For the best user experience, Service_ModeTranslation implementations should attempt to find a replacement mode which matches as many of the attributes of the original mode as possible, including the mode type. However it may be necessary to consider the intent of the choice, e.g. if mode 2 is requested (double-pixel mode with 160 logical pixels per row), it would be sensible to return a non-double pixel mode which has 160 physical pixels per row, if the display hardware can support it and it's likely to result in the desired aspect ratio.

[RISC OS Open source page](#)

Screen Modes

Mode numbers are categorised as below:

Mode · Col x Row · Resolution · Colours · Memory · Mode Type · 0 · 1 · 2 · 3 · 4 · 5 · 8

```

0 · 80 × 32 · 640 × 256 · 2 · 20K · . . * . . . * . . . .
1 · 40 × 32 · 320 × 256 · 4 · 20K · . . * . . . * . . . .
2 · 20 × 32 · 160 × 256 · 16 · 40K · Double-pixel · * . . . * . . . .
3 · 80 × 25 · Text only · 2 · 40K · BBC gap mode · * . . . * . . . .
4 · 40 × 32 · 320 × 256 · 2 · 20K · Double-pixel · * . . . * . . . .
5 · 20 × 32 · 160 × 256 · 4 · 20K · Double-pixel · * . . . * . . . .
6 · 40 × 25 · Text only · 2 · 20K · BBC gap mode · * . . . * . . . .
7 · 40 × 25 · Teletext · 16 · Varies1 · Teletext · * . . . * . . . .
8 · 80 × 32 · 640 × 256 · 4 · 40K · . . * . . . * . . . .
9 · 40 × 32 · 320 × 256 · 16 · 40K · . . * . . . * . . . .
10 · 20 × 32 · 160 × 256 · 256 · 80K · Double-pixel · * . . . * . . . .
11 · 80 × 25 · 640 × 250 · 4 · 40K · Gap mode · * . . . * . . . .
12 · 80 × 32 · 640 × 256 · 16 · 80K · . . * . . . * . . . .
13 · 40 × 32 · 320 × 256 · 256 · 80K · . . * . . . * . . . .

```

[illegible]

Mode / Use

0 - 63 – Use: Screen Modes provided natively by RISC OS

64 - 95 – Use: Third party applications (allocated centrally)

96 - 127 – Use: User defined modes

Notes

1 Mode 7

The BBC Micro series of computers contained a dedicated teletext display chip which would generate the pixel data for a mode 7 screen. This meant that mode 7 only required enough memory necessary to store the character map (1KB of RAM).

Under RISC OS the teletext chip was replaced with a software emulation which performs on-demand conversion of the character map to a standard pixel-oriented framebuffer. The amount of memory required for this framebuffer will vary depending on the OS version and the video hardware capabilities; typically somewhere between 80KB and 640KB.

Other languages documentation

Français

[RISC OS Open source page](#)

ScreenBlanker

List of modules

The ScreenBlanker module provides access to control screen blanking.

Details

- SWI Calls
- Service Calls
- *Commands

[RISC OS Open source page](#)

Scroll_Request

Reason Code 10 · WindowManager

This code is returned from Wimp_Poll and indicates that a user is requesting to scroll a window. It contains the current scroll offsets and the direction that the window is being asked to scroll.

On return

Block

- +0 Window handle
- +4 Visible area minimum x
- +8 Visible area minimum y
- +12 Visible area maximum x
- +16 Visible area maximum y
- +20 Scroll x offset
- +24 Scroll y offset
- +28 Handle to open window behind (-1 top of stack, -2 for bottom of stack)
- +32 Scroll x direction
- +36 Scroll y direction
- +40 Destination icon handle (or -1 for window work area); only present for extended scroll requests

Notes

This event code is used for both “Normal” scroll requests and “Extended” scroll requests. Applications can determine which type of scroll request they’ve received by examining the Scroll x direction & Scroll y direction fields. If the bottom two bits of both fields are zero, it is an extended scroll request. Otherwise, it is a normal scroll request.

“Normal” scroll requests are only sent to windows if one of the Scroll Requests bits are set in the Window Flag of the window being asked to scroll.

“Extended” scroll requests can either be directed towards the window work area or an icon. Window work areas will only receive the event if they advertise support for extended scroll requests via bit 1 of the Extra Window Flags. Icons will only receive the event if they advertise support for via the ‘YS’ validation string flag.

To actually implement the window scroll, applications must call `Wimp_OpenWindow` after the new scroll offsets have been calculated.

The scroll offsets are in OS units, so an applications wishing to scroll in pixels must carry out the necessary scaling to calculate the correct values.

[RISC OS Open source page](#)

SDIODriver

List of modules

The **SDIODriver** module provides the software stack for SDIO devices. It supports a wide range of memory card types and speeds, as well as providing hooks for drivers for SDIO devices like cameras, wi-fi and Bluetooth.

Details

- Service Calls
- SWI Calls
- Technical Details
- *Commands

[RISC OS Open source page](#)

SDIODriver Technical Details

SDIODriver

SD host controllers may be attached via a PCI bus as was anticipated by the SD standard. In this case, the **SDIODriver** would scan the PCI bus for devices of the appropriate class. It should be noted however that at the present time there are no RISC OS platforms with such an architecture, so this is considered a future enhancement.

Low-level interface

Host controllers with alternate attachments are discovered via the HAL Device API. This potentially includes hot-pluggable host controllers, as is true of all HAL Devices. The HAL Device looks like this:

0 – Meaning: Type:

15 – 8 · 5 · Controller of peripheral buses

7 – 0 · 1 · For SDIO controller

2 – Meaning: ID:

15 – 0 · 1 · Generic SDHCI controller

4 – Meaning: Location:

31 – 28 · 5 · (interconnect) for OMAP3

27 – 24 · 1 · (L4 interconnect) for OMAP3

23 – 16 · 0 · 0 for OMAP3

15 – 8 · 0 · 0 for OMAP3

7 – 0 · 0 · 0 for OMAP3

8 – Meaning: Version: Only major version 0 has currently been defined. Within major version 0, minor version 1 indicates the presence of the `TriggerCommand` function pointer (which may still be NULL)

12 – Meaning: Description:

16 – Meaning: Address:

20 – Meaning: Reserved1:

32 – Meaning: Activate:

36 – Meaning: Deactivate:

40 – Meaning: Reset:

44 – Meaning: Sleep:

48 – Meaning: DeviceNumber:

52 – Meaning: TestIRQ:

56 – Meaning: ClearIRQ:

60 – Meaning: Reserved2:

64 – Meaning: Flags:

31 – 3 · 0 · Reserved, must be zero

2 · 1 · the controller doesn't set the error interrupt bit in the normal interrupt status register

1 · 1 · The controller doesn't strip the byte containing the CRC and end bit from R2 responses before placing in the response registers

0 · 0 · All register reads and writes must use 32-bit accesses

68 · 31 – 0 – Meaning: Number of slots supported by this host controller – or technically, the number of standard register sets it provides, which corresponds to the number of CMD lines it drives. Each slot is normally used to drive a single MMC or SD device, but it is possible that multiple devices can be driven by a single slot – for MMC buses this is achieved by the use of an identifying field, the RCA, in the command argument; for SD buses, a Shared Bus configuration can be implemented in hardware, with a separate CLK line for multiple devices that share the same CMD line

72 · 31 – 0 – Meaning: Pointer to array of structures giving further information about each slot

void – Function Pointer: WriteRegister(device *controller, uint32_t slot, void *address, uint32_t value, uint32_t width) · Meaning: function pointer to a custom routine to write a register in the SDHCI controller. If flags bit 0 is set, then writes have already been amalgamated into 32-bit words and the address will be word aligned. Otherwise the fourth parameter represents the value to be written, zero-extended to a 32-bit value, and the fifth contains the width of the register in bytes. This entry is intended for controllers that have special requirements (such as the BCM2835 where the rate of register pokes must be strictly limited). In other cases, a null pointer can be given, and the register will be written using a simple data write operation

uint64_t – Function Pointer: GetCapabilities(device *controller, uint32_t slot) · Meaning: function pointer to a custom routine to read the controller's capabilities register. This is intended for controllers that do not implement this standard SDHCI register; if the register is implemented, you can use a null pointer instead. The value returned from this function should follow the bit layout as defined in the SDHCI standard

uint32_t – Function Pointer: GetVddCapabilities(device *controller, uint32_t slot) · Meaning: function pointer to a custom routine to read which supply voltages are available. This is needed in cases where an external level-shifter is in use, which the controller itself cannot know about. If this is a null pointer, the available voltages are determined from the capabilities register instead. Bits in the return value are defined as follows: bit 0 set => 3.3 V supported bit 1 set => 3.0 V supported bit 2 set => 1.8 V supported other bits reserved, should be 0 If you report more than one bit set, then you must also support 0 V (this is because cards must be powered down in order to change their operating voltage)

void – Function Pointer: SetVdd(device *controller, uint32_t slot, uint32_t voltage) · Meaning: function pointer to a custom routine to set the supply voltage. This is needed if blocks other than the SDHCI controller are involved in power supply or level shifting; if this is not the case then you can use a null pointer and the SDHCI power control register will be used instead. The voltage is specified in mV, and may have value 0 to indicate that the bus should be powered down

void – Function Pointer: SetBusMode(device *controller, uint32_t slot, bool od_not_pp) · Meaning: function pointer to switch the CMD line of the bus between open-drain and push-pull modes. With the MMC bus (unlike the SD bus), multiple cards may share both a CLK and CMD line, and may all simultaneously drive the shared CMD line in response to "bcr" type commands. These commands are used during device-identification mode, so during this mode, the CMD line must be set to open-drain. Push-pull mode is used in data-transfer mode for its speed advantage. A controller that only supports SD cards may use a null pointer for this entry

void – Function Pointer: PostPowerOn(device *controller, uint32_t slot) · Meaning: function pointer which is called after power is supplied to the card. The controller in the OMAP3 has a special initialisation procedure that must be performed at this point – other cases will probably use a null pointer

void – Function Pointer: SetBusWidth(device *controller, uint32_t slot, uint32_t lines) · Meaning: function pointer to a custom routine to set the number of DAT lines used by the controller (1, 4 or 8); may be a null pointer if the controller conforms to the SDHCI specification

uint32_t – Function Pointer: GetMaxCurrent(device *controller, uint32_t slot) · Meaning: function pointer to find the maximum current that can be supplied to the card at the presently selected voltage and bus width. This is needed if there are external power supplies or voltage shifters, or if the maximum current capabilities register is not implemented or reports value 0. You may use a null pointer otherwise. The value returned is in units of mA

uint32_t – Function Pointer: SetSDCLK(device *controller, uint32_t slot, uint32_t freq_khz) · Meaning: function pointer to a custom routine set the SDCLK frequency (in units of 1 kHz) for all SDCLK lines driven by this slot; may be a null pointer if the controller conforms to the SDHCI specification. A controller may set any frequency below the one requested; the function returns the actual frequency used, in kHz,

rounded down

uint32_t – Function Pointer: GetTMCLK(device *controller, uint32_t slot) · Meaning: function pointer to a custom routine to read the timeout clock frequency (TMCLK) in units of 1 kHz; may be a null pointer if the controller conforms to the SDHCI specification and provides a non-zero value in the Timeout Clock Frequency field of the Capabilities register

void – Function Pointer: SetActivity(device *controller, uint32_t slot, uint32_t level) · Meaning: function pointer to turn on or off the disc activity indicator LED for the specified slot on this controller (level 0 = off, level 1 = read access, level 2 = write access); may be a null pointer if the standard SDHCI mechanism is to be used instead

bool – Function Pointer: GetCardDetect(device *controller, uint32_t slot) · Meaning: function pointer that returns the state of the hardware card-detect line for the card in this slot. When multiple MMC devices share the same CMD line, there is no way to associate a card's bus address (RCA) with an independent card detect line, so detection is impossible and this function should return true. The SD Shared Bus configuration is only used in cases where the device is embedded and non-removable, so likewise this function should return true in such cases. Otherwise, this may be a null pointer if the standard SDHCI mechanism is to be used instead

bool – Function Pointer: GetWriteProtect(device * controller, uint32_t slot) · Meaning: function pointer that returns whether the hardware write-protect switch is activated for this slot. Similar concerns regarding devices sharing a CMD line apply so in such cases you should always return false. Otherwise, this may be a null pointer if the standard SDHCI mechanism is to be used instead

void – Function Pointer: TriggerCommand(device *controller, uint32_t slot, uint16_t transfer_mode, uint16_t command) · Meaning: function pointer that performs the equivalent of writes to the Transfer Mode and Command registers on a pure SDHCI controller, and thereby causes a command to be issued on the bus

Further information on HAL Device Descriptor fields can be found [here](#).

SlotInfo Structure

The structures pointed at by slotinfo have the following layout:

0 – Meaning: Flags:

31 – 8 · Reserved (must be zero)

7 · If set => all devices on this slot are integrated onto the board and so are not physically removable, but at least one of them provides a memory function (this flag is used by SDFS to assign a "hard disc" drive number)

6 · If set => the devices on this slot are removable but the card detect line is non-functional, so testing for device presence must be performed by a speculative command

5 – 3 · Number of SDCLK lines controlled by this slot, minus 1

2 – 1 · Data bus width tracked out to device(s):

0 => 1 bit

1 => 4 bits

2 => 8 bits (MMC or embedded SD only)

0 · If set => all devices on this slot are integrated onto the board and so are not physically removable (this flag is used to hide integrated SDIO devices from SDFS)

4 – Meaning: stdregs:

31 – 0 · Logical base addresses of standard register set

Hardware Specific Information

The OMAP3530 has 3 host controllers with one slot on each, but since the beagleboard only tracks out one of them to an SD connector, only this one will be reported by the HAL. For boards like the IGEPv2 which use 2 of the controllers, the HAL will instead report two devices. A driver for a beagleboard expansion card would announce the additional controller(s) once the expansion card had been detected - this is necessary since the HAL can't be expected to have knowledge of how to detect every possible expansion card and how they route their SD buses.

(Actually, to simplify initial development, this device will be implemented in a module, so that complete ROM rebuilds are not required to test each incremental change, but when the code becomes stable it will be moved into the HAL proper.)

High-level interface

First, a note on controller and card addressing. There are a number of fields common to multiple parts of the API:

- Bus number: this is the index of the SD host controller device, in the order of discovery.
- Slot number: this is the index of the SDCLK line for a given host controller. Except for a SD Shared Bus configuration, this is the same as the index of the CMD line, and maps directly on to the register set index as described by the HAL. On an MMC bus, multiple cards may also share the same slot number, but they are identified by a differing RCA (passed as an argument to various commands) instead.

- Unit number: this is the logical unit index within a card. A value of 0 refers to memory operation, values of 1-7 refer to a function index, and other values are reserved.

The SDIODriver is responsible for probing cards, to identify which are memory cards, which are IO cards and which are combo cards, and for IO cards, discovering the set of functions available and the Function Interface Code for each. However, it is not appropriate for it to create further HAL devices to describe functions because to do so would mean hard-coding knowledge of the meaning of each Function Interface Code into the SDIODriver, and additional codes could be allocated at any time; it is better that this knowledge is encapsulated within the driver module for that specific function.

[RISC OS Open source page](#)

SerialDeviceDriver

List of modules

The SerialDeviceDriver module creates a DeviceFS device for the serial port which is present on the 82C710-compatible I/O combo chip used in IOMD and Tungsten machines. It also contains a driver for the 6551 serial port used on early Archimedes machines, although that driver is not enabled in the RISC OS 5 build of the module.

Introduction

The DeviceFS device which is created is compliant with the SerialDeviceSupport protocol, which allows the legacy OS_SerialOp SWI to be used to interact with the serial port.

Note that on Tungsten hardware the I/O combo chip actually provides two serial ports. The first port is made available via SerialDeviceDriver, while the second port is made available via Serial / DualSerial.

[RISC OS Open source page](#)

SerialV

Vectors

All OS_SerialOp calls are implemented via calls to this vector.

Before calling

- Parameters as per OS_SerialOp

On return

- Result as per OS_SerialOp

Notes

SerialV is used to indirect all calls to OS_SerialOp. The default action is to call the OS_SerialOp routine.

[RISC OS Open source page](#)

SeriousErrorV

Vectors

This vector is called during serious error reporting and recovery, such as an Abort. It should not be claimed, but called to notify relevant sections of the OS that a serious error has occurred and/or recovery steps have been taken.

Before calling

R2 Reason code

- Other registers dependent on reason code

R0 Parameter provided to SeriousErrorV

R1 Pointer to null-terminated string

On return

- All registers preserved

R0-R3, R12 Corrupt

In practice

Note that handlers of SeriousErrorV have tighter restrictions placed on them than with other vectors. Specifically, reason code 0 will be called from ABT mode, with IRQ disabled. At the time the call is made the stack pointers for the other privileged modes will be in an indeterminate state (e.g. the abort could be due to a SVC stack overflow), so implementations must be careful to avoid using the stacks for any other modes. For example, this prevents the use of SWI calls, and it prevents the direct use of CMHG veneers in C modules. Similarly, if the abort was due to a faulty interrupt handler, re-enabling IRQs may cause the handler to crash again, so SeriousErrorV handlers must take care to not re-enable interrupts.

For C modules which are only interested in the reason codes that are called in SVC mode, it's recommended to use a pre-veneer in front of the CMHG veneer which will filter out all the codes which you are not interested in. Currently reason code 0 is the only reason code which is not called in SVC mode, but more reason codes might be introduced in future, so having your pre-veneer filter out all unknown codes is the safest approach.

Reason codes

0 – Collect. Entered in ABT mode, IRQ disabled, FIQ undefined

– : R0 = Pointer to register dump

– : R1 = Pointer to (untranslated) error block

1 – Recover. Entered in SVC mode, IRQ disabled

– : R0 = Pointer to translated error block

2 – Report. Entered in SVC mode, IRQ enabled

– : R0 = Pointer to translated error block

3 – Custom Report. Entered in SVC mode, IRQ enabled

– : R0 = Pointer to translated error block

– : R1 = Flags

– : R3 = Pointer to callback function to receive the report

– : R4 = Callback R0

Serious error processing

When a serious error occurs, the kernel undertakes the following procedure:

- A register dump is created, in a temporary location on the ABT or UND stack.
- An appropriate untranslated error block is selected (the error number will be correct, but the message will be a meaningless message file token)

- SeriousErrorV reason code 0 will be called. This allows crash reporting software such as the Debugger module to collect the key machine state at the time of the error (register dump, error code, privileged mode stack dumps, code surrounding the PC, etc).
- The CPU will switch to SVC mode and the SVC stack pointer will be reset to its default value
- The register dump will be copied from the stack to the location specified by the environment handler. This occurs after the SVC stack has been reset so that the kernel can call OS_Memory 24 to check that the target address is valid. If the address is invalid, it will be silently reset to the kernel default address.
- The error block will be translated via MessageTrans
- The stack pointers for the other privileged processor modes will be reset to their default values
- Any other relevant internal state (e.g. IRQsema) will also be reset
- FIQs will be re-enabled
- SeriousErrorV reason code 1 will be called. This reason code is primarily to allow low-level system components to reset themselves, e.g. RTSupport needs to be notified of the SVC stack reset and the IRQsema reset. The system will still be in an indeterminate state, so code must be careful about which SWIs (if any) it calls.
- IRQs will be re-enabled
- SeriousErrorV reason code 2 will be called. This is primarily used to allow crash reporting software to output its report, e.g. by writing it to disc. Other machine state may also be collected at this time (e.g. anything which could not be collected during the "collect" call)
- Finally, OS_GenerateError is called in order to notify the current application of the error

Note that the exact ordering of items is subject to change in future kernel versions, but the overall flow as seen by SeriousErrorV will remain the same.

Custom report generation

Software can request that a custom report is generated by calling SeriousErrorV reason codes 0 and 3. For example, when the ZeroPain module emulates a zero page access, it also invokes SeriousErrorV in order to collect information for inclusion in the log file so that the zero page access can be reported to the author of the relevant software. The process for using these reason codes is as follows:

- Create a temporary register dump and error block somewhere
- Enter ABT mode with IRQs disabled
- Invoke SeriousErrorV reason code 0
- Enter SVC mode with IRQs enabled
- Invoke SeriousErrorV reason code 3

The requirement for reason code 0 to be called in ABT mode means that there is no legal way for software outside the kernel to invoke that reason code. Therefore, at this time, the use of custom reports is restricted to low-level OS components which have enough knowledge of the host system to be able to invoke the vector directly.

The flags for reason code 3 are as follows:

Bit / Meaning

0 – Meaning: 1 Produce annotated text dump

0 Produce raw hex dump

1+ – Meaning: Reserved

The callback function pointer provided in R3 must have the following signature:

The callback function will be called multiple times as the different areas of the report are output.

[RISC OS Open source page](#)

ShellCLI

List of modules

The ShellCLI module provides a single *Command that invokes a command shell from within the Desktop environment.

In practice

Two SWIs are provided by the module for internal use only, and must not be used.

Details

- SWI Calls
- *Commands

[RISC OS Open source page](#)

Software information: Maestro

Software information

Maestro is a basic music player. It can create and edit scores of up to four staves and eight voices. Playback is via the 8-bit internal sound system, or to a MIDI device.

Installation Instructions

Maestro is normally installed in \$.Apps in the standard HardDisc4 image. Help on usage can be found in the !Help file or the RISC OS User Guide.

Issues

Maestro has numerous quirks and limitations. There is at least one redraw bug that causes visual corruption when editing multi-part scores, and playback can be erratic.

If you are experiencing problems with this software, first check to see if a ticket has been raised on the Bug tracker page. You may also want to visit the Bugs forum.

If you have a general query related to the software, you can enquire on the Community Support forum.

History of the component

To view the release history for versions available on the ROOL web site, visit the GitLab Commits page (releases are “tagged” with the version number).

Obtaining the software

This software is usually supplied as part of the standard HardDisc4 build.

The latest version of the software can be obtained from the Packages server.

To obtain the source code, visit the GitLab source page.

[RISC OS Open source page](#)

Software Vector Numbers

Vectors

- 0 · &00 – Vector: UserV · Description of Vector: User (reserved)
- 1 · &01 – Vector: ErrorV · Description of Vector: Error (OS_GenerateError)
- 2 · &02 – Vector: IrqV · Description of Vector: Unknown interrupt
- 3 · &03 – Vector: WrchV · Description of Vector: Write character (OS_WriteC)
- 4 · &04 – Vector: ReadCV · Description of Vector: Read character (OS_ReadC)
- 5 · &05 – Vector: CLIV · Description of Vector: Command line interpreter (OS_CLI)
- 6 · &06 – Vector: ByteV · Description of Vector: OS_Byte indirection (OS_Byte)
- 7 · &07 – Vector: WordV · Description of Vector: OS_Word indirection (OS_Word)
- 8 · &08 – Vector: FileV · Description of Vector: File read/write (OS_File)
- 9 · &09 – Vector: ArgsV · Description of Vector: File arguments read/write (OS_Args)
- 10 · &0A – Vector: BGetV · Description of Vector: File byte read (OS_BGet)
- 11 · &0B – Vector: BPutV · Description of Vector: File byte put (OS_BPut)
- 12 · &0C – Vector: GBPBV · Description of Vector: File byte block get/put (OS_GBPB)
- 13 · &0D – Vector: FindV · Description of Vector: File open (OS_Find)
- 14 · &0E – Vector: ReadLineV · Description of Vector: Read a line of text (OS_ReadLine, OS_ReadLine32)
- 15 · &0F – Vector: FSControlV · Description of Vector: Filing system control (OS_FSControl)
- 16 · &10 – Vector: EventV · Description of Vector: Event vector (OS_GenerateEvent)
- 17 · &11 – Vector: Reserved · Description of Vector: Reserved
- 18 · &12 – Vector: Reserved · Description of Vector: Reserved
- 19 · &13 – Vector: KeyV · Description of Vector: Key vector
- 20 · &14 – Vector: InsV · Description of Vector: Buffer insert (OS_Byte 138)
- 21 · &15 – Vector: RemV · Description of Vector: Buffer remove (OS_Byte 145)
- 22 · &16 – Vector: CnpV · Description of Vector: Count/Purge Buffer (OS_Byte 21 & OS_Byte 152)
- 23 · &17 – Vector: UKVDU23V · Description of Vector: Unknown VDU23 (OS_WriteC)
- 24 · &18 – Vector: UKSWIV · Description of Vector: Unknown SWI
- 25 · &19 – Vector: UKPLOTV · Description of Vector: Unknown VDU25 (OS_Plot)
- 26 · &1A – Vector: MouseV · Description of Vector: Mouse (OS_Mouse)
- 27 · &1B – Vector: VDUXV · Description of Vector: VDU extension vector (OS_WriteC)
- 28 · &1C – Vector: TickerV · Description of Vector: 100Hz pacemaker
- 29 · &1D – Vector: UpCallV · Description of Vector: Warning vector
- 30 · &1E – Vector: ChangeEnvironmentV · Description of Vector: Environment change (OS_ChangeEnvironment)
- 31 · &1F – Vector: SpriteV · Description of Vector: OS_SpriteOp indirection
- 32 · &20 – Vector: DrawV · Description of Vector: Draw SWI (Draw Module SWI Calls)
- 33 · &21 – Vector: EconetV · Description of Vector: Econet activity (Econet SWI Calls)
- 34 · &22 – Vector: ColourV · Description of Vector: ColourTrans SWI (ColourTrans SWI Calls)
- 35 · &23 – Vector: PaletteV · Description of Vector: Read/Write palette vector
- 36 · &24 – Vector: SerialV · Description of Vector: Serial operation (OS_SerialOp)
- 37 · &25 – Vector: FontV · Description of Vector: Font manager
- 38 · &26 – Vector: PointerV · Description of Vector: Mouse drivers
- 39 · &27 – Vector: TimeShareV · Description of Vector: SkyNet
- 40 · &28 – Vector: LowPriorityEventV · Description of Vector: For future expansion
- 41 · &29 – Vector: FastTickerV · Description of Vector: Like TickerV, but faster (RISCOS Ltd)
- 42 · &2A – Vector: GraphicsV · Description of Vector: Graphics hardware abstraction
- 43 · &2B – Vector: UnthreadV · Description of Vector: High-priority callbacks
- 44 · &2C – Vector: VideoV · Description of Vector: Graphics abstraction (RISCOS Ltd)
- 45 · &2D – Vector: SeriousErrorV · Description of Vector: Handling of “serious errors” and exceptions
- 62 · &3E – Vector: NVRAMV · Description of Vector: NVRAM hardware abstraction (RISCOS Ltd)
- 63 · &3F – Vector: RTCV · Description of Vector: RTC hardware abstraction (RISCOS Ltd)

[RISC OS Open source page](#)

Software Vectors

Vectors

The kernel is responsible for the overall control of the Operating System. Extra functionality is provided by modules. Each module is responsible for a handling a specific task, such as font handing, filer windows and sound scheduling. The end result is a modular Operating System that can be easily developed, updated and extended.

In practice

Unlike modules, the kernel cannot be altered in any way. However, RISC OS does provide a method of allowing certain key routines used by the kernel to be changed or replaced. This is achieved by use of software vectors, or vectors for short.

Vector basics

When an ordinary SWI instruction is issued, RISC OS must determine which routine is requested. It does this by looking up the address of the SWI routine, and then branching to it, thereby running the routine. However, if the SWI instruction issued has a corresponding software vector (known as a vectored SWI), the required address is returned from the vector. Normally this would return the address of the standard SWI routine, although this can be altered.

By changing the address of a vectored SWI, this is called claiming a vector, To claim a vector, you use `OS_Claim`.

As detailed above, vectors store the address of the vectored SWI routine. This address is used by the kernel to determine which routine should be called. However, instead of just replacing an address, it is possible for multiple addresses to be added to a vector. In summary:

- Multiple routines (addresses) can be added to a vector
- Vectors always points to the next address to be called
- No routine that has claimed the vector needs know about any other routine that has claimed the same vector
- Vectors can be claimed and released by any piece of software running, and in any order
- Routines are called in the reverse order to which they were claimed

If your routine completely replaces the standard routine, then it is deemed to interpret the call, otherwise it is known to pass on the call.

A vector can be claimed by any application resulting in multiple addresses being added to a vector. This means your routine can:

- Replace the original SWI routine, passing control directly back to the caller
- Call the first address pointed to by the vector (which may be your routine) before calling the next address (which may be the standard routine) which then passes control back to the caller
- Call the first address pointed to by the vector (which may be the standard routine), before calling the next address (which may be your routine) and then pass control back to the caller

Claiming software vectors

To claim a vector, you use `OS_Claim`. This SWI actually removes any identical earlier instances of the routine from the list of addresses. If this is not desired, then it is possible to simply add an address to the list by using `OS_AddToVector`.

Any routine that uses a vector must use the same registers as the default routine for both entry and exit conditions.

If your routine passes the call on, you can deliberately alter some of the registers values to change the effect of the call, however, you must arrange for control to return to your routine again to restore to those that the original routine would have returned. It should then return control back to the calling program.

To remove a routine from the list of those that claim a vector, use `OS_Release`.

When not to intercept a vector

There are some vectors that must not be intercepted, that is, to completely replace the standard routine. This is because the standard routine (also known as the default owner), might perform an important action that must be executed. A good example of this is the `ErrorV?` vector, which by default calls the error handler. If this vector was to be intercepted, then the error handler would never be called, and thus never dealt with.

Multitasking

RISC OS is a co-operative multitasking Operating System, that is, where each application is written to pass control back to the Window Manager after it has done what it needed to do. When multiple tasks are running concurrently, the application memory of the active task is mapped into memory at `&8000`. When the next application is to assume control, the current task is mapped out of memory, and the new application is mapped into the same space (at `&8000`).

Because of this process, any application that claims a vector, must release the vector each time it is mapped out of application space, and then reclaim it again when mapped back in. This is achieved by two SWIs:

- `OS_DelinkApplication` - Release vectors when an application is about to be mapped out
- `OS_RelinkApplication` - Reclaim vectors when an application is about to be mapped back in

Error Handling

Like normal SWIs, routines using most of the vectors can return errors by setting the `V` flag, and making `R0` point to an error block.

However, when generating an error inside a vector routine, the routine must intercept the call instead of passing it on. This is to prevent other routines in the vector chain from malfunctioning; for example if `R0` was being used as an argument for the vector call then the other claimants won't be able to cope with the fact that it's now an error pointer.

Note that some vectors (such as those involving IRQ calls) should never be made to return errors, as there is nowhere sensible for the error to be sent to.

Processor modes

The processor mode used by the routine depends on the vector:

- Routines via `IrqV` are always executed in IRQ mode
- Routines via `EventV`, `InsV`, `RemV`, `CnpV` and `TickerV` are generally executed in IRQ mode. However, they may be executed in SVC mode if called using `OS_CallAVector`, and in certain other unspecified circumstances
- Routines via `GraphicsV` are generally called in SVC mode, except for a few which the kernel will call from IRQ mode. These cases are noted in the details for the individual calls.
- All other routines are executed in SVC mode (the same mode when the SWI instruction is executed)

If you call a SWI from a routine that is in SVC mode you will corrupt the return address held in `R14`. Therefore your routine should use the full, descending stack address by `R13` to save `R14` first.

If your routine will be entered in IRQ mode, then further restrictions apply. These are detailed in full within the Interrupts and handling them?

Returning from a vectored routine

When returning from a vectored routine, the method differs depending on whether the vector is intercepted or whether you are passing on the call. The methods are shown below:

Passing on the call / Return by copying `R14` into the Program counter

Intercepting the call – Return by copying `R14` into the Program counter: Pull an exit address from the stack and jump to it

List of Software Vectors

There are many different software vectors available. The complete list is available [here](#).

[RISC OS Open source page](#)

Sound_ControllerInfo Property Numbers

SoundDMA

/

0 – : Null-terminated controller name string for display to user

1 – : Pointer to mixer HAL device (for internal use only – mixer devices are managed by SoundControl)

2 – : List of supported sample rates (1 word per entry), in ascending order. Units are in 1/1024Hz, as per Sound_SampleRate 2.

[RISC OS Open source page](#)

Sound_ReadSysInfo 1

SoundDMA

The purpose of this call is to read the features of the version of SoundDMA installed on the system.

Before calling

R0 1 (reason code)

On return

R0 0

R1 Flags:

Bit 0: Generates service calls Sound rate changing, Sound starting, Sound stopping

Bit 1: SWIs Sound_SelectDefaultController, Sound_EnumerateControllers, Sound_ControllerInfo supported

Bits 2-31: Reserved

Notes

Many versions of SoundDMA do not return errors when unknown SWIs are called. Therefore, to detect if this SWI is supported, check if R0 has been set to 0 on exit (Valid reason codes start at 1).

[RISC OS Open source page](#)

Sound_ReadSysInfo 2

SoundDMA

The purpose of this call is to read the ID string of the current audio controller device.

Before calling

R0 2 (reason code)

R1 Pointer to buffer for result, or 0 to check required length

R2 Length of buffer

On return

R0 0, or error (e.g. buffer overflow)

R1 Buffer updated with null-terminated string (if sufficient space available)

R2 Length of data (including terminator)

Notes

Many versions of SoundDMA do not return errors when unknown SWIs are called. Therefore, to detect if this SWI is supported, check if R0 has been set to 0 on exit (Valid reason codes start at 1).

If the supplied buffer is not large enough, or the pointer was 0, a buffer overflow error will be generated and no data will be copied to the buffer.

[RISC OS Open source page](#)

Sound_ReadSysInfo Reason Codes

SoundDMA

/

1 – : Return SoundDMA features

2 – : Get current audio controller device ID

[RISC OS Open source page](#)

SoundControl

List of modules

The SoundControl module manages the settings of any audio mixers which are present in the system.

In practice

Most mixers operate by taking one or more input audio signals, mixing them together to produce a combined audio signal (with separate gain and mute settings for each input signal), and then distributing that signal to several outputs – again with separate gain and mute settings for each output.

SoundControl groups the input/output signals for each mixer into different categories. An index parameter is also used in order to cope with situations where a mixer has multiple signals which fall into the same category.

Details

- SoundControl Categories
- SWI Calls
- *Commands

[RISC OS Open source page](#)

SoundControl Categories

SoundControl

Current versions of SoundControl define the following category numbers. Positive numbers are used to identify inputs, while negative numbers identify outputs.

Input Categories /

0 – : System audio – the sound produced by the OS

1 – : Microphone

2 – : Line in

3 – : Aux in, e.g. CD audio

Output Categories /

-1 – : Integrated speaker

-2 – : Headphones

-3 – : Line out

-4 – : Aux out

[RISC OS Open source page](#)

SoundCtrl_ExamineMixer Block

SoundControl

Although the block length shown here is 16 bytes, not all mixers return the same amount of data. The block size value returned by SoundCtrl_ExamineMixer in R3 should always be used to determine how much data is available. Currently defined block sizes are 4 and 16 bytes. Be aware that larger sizes may be introduced in the future – your code should only read as many bytes as it understands.

0-1 – : Flags:

Bit 0 set => Gain setting is fixed

Bit 1 set => Mono

Bit 2 set => Channel should be muted by default

Bits 3-15: Reserved

2-3 – : Category

4-7 – : Minimum gain

8-11 – : Maximum gain

12-15 – : Minimum step size/granularity of gain changes

Notes

All gain values are specified in units of 1/16dB, e.g. -16 corresponds to -1dB. Because not all mixers allow such a fine level of control over their gain, the word at offset 12 is used to indicate the minimum step size that is supported by the channel.

[RISC OS Open source page](#)

SpriteExtend SWI Calls

SpriteExtend

The SpriteExtend module is required to be loaded in order for many OS_SpriteOp calls to function; see OS_SpriteOp for details.

OS_SpriteOp SWI Calls

Note that in practical terms the SpriteExtend module will almost always be loaded, as it has been included in the OS ROMs since RISC OS 2.00. However additional OS_SpriteOp reason codes and features have been added since then, and so the appropriate version number or feature checks should be performed where necessary.

JPEG SWI Calls

The SpriteExtend module also provides JPEG rendering facilities, via the following SWIs:

- JPEG_Info
- JPEG_FileInfo

- JPEG_PlotScaled
- JPEG_PlotFileScaled
- JPEG_PlotTransformed
- JPEG_PlotFileTransformed
- JPEG_PDriverIntercept

[RISC OS Open source page](#)

SpriteUtils

List of modules

The SpriteUtils module provides a several sprite-related *Commands, mainly focused around manipulating the contents of the system sprite area.

Details

- *Commands

[RISC OS Open source page](#)

SpriteV

Vectors

All OS_SpriteOp calls are implemented via calls to this vector.

Before calling

- Parameters as per OS_SpriteOp

On return

- Result as per OS_SpriteOp

Notes

The kernel is the default SpriteV claimant, and implements many of the OS_SpriteOp calls itself. Several other important calls are implemented by SpriteExtend.

[RISC OS Open source page](#)

Squash

List of modules

The Squash module provides compression and decompression facilities to the Operating System. The module interface is designed to be restartable, so its facilities can occur from a variety of address locations. Operations involving I/O can easily be constructed.

In practice

The module is used by the Squash application to generate Squash files (&FCA). It currently uses a 12-bit LZW algorithm.

Details

- SWI Calls
 - Squash file format?
- [RISC OS Open source page](#)

Squash_Compress Flags and Status

Squash

Squash_Compress Flags

Bits / Meaning if set

- 0 – Meaning if set: Continuing existing operation
- 1 – Meaning if set: More input remains after this
- 2 – Meaning if set: 0 (reserved for future use)
- 3 – Meaning if set: Return workspace needed, and max output size (all other bits must be 0 for this flag to work)
- 4 - 31 – Meaning if set: 0 (reserved for future use)

Squash_Compress Statuses

Bits / Meaning if set

- 0 – Meaning if set: Operation complete
- 1 – Meaning if set: Input exhausted (R3 = 0)
- 2 – Meaning if set: Output space exhausted (R5 < 12)

[RISC OS Open source page](#)

Squash_Decompress Flags and Status

Squash

Squash_Decompress Flags

Bits / Meaning if set

- 0 – Meaning if set: Continuing existing operation
- 1 – Meaning if set: More input remains after this
- 2 – Meaning if set: Assume all output will fit in buffer (if bits 0, 1 clear a faster algorithm will be used)
- 3 – Meaning if set: Return workspace needed and max output size (all other bits must be zero)
- 4 - 31 – Meaning if set: 0 (reserved for future use)

Squash_Decompress Statuses

Bits / Meaning if set

- 0 – Meaning if set: Operation complete
- 1 – Meaning if set: Input exhausted (R3 > 12)
- 2 – Meaning if set: Output space exhausted (R5 = 0)

[RISC OS Open source page](#)

SWI Error Block

Introduction To SWIs

SWI error blocks are the key mechanism by which errors are returned from SWIs. Whenever a SWI call exits with the V flag set, R0 will contain a pointer to an error block, with the following structure:

Offset / Contents

+0 – Contents: Error number

+4 – Contents: Error String (zero terminated)

In practice

The error number is broken down as follows:

0-7 – Meaning when set: Basic error number

8-23 – Meaning when set: Error owner or Error Generator

24-29 – Meaning when set: Reserved. Must be 0

30 – Meaning when set: Defined to be 0

31 – Meaning when set: Error is serious (hardware). Owner is then;

0 Machine failure (data abort, undefined instruction, etc.)

1 Co-Processor failure

2 Floating Point (FPA)

3 Econet

6 Floating Point (VFP)

7 Python OS failure (used by errors in the host implementation within RISC OS Pyromaniac)

Notes

X SWI's are allowed to generate (rather than return) serious errors.

Starting with RISC OS 5.23, whenever a SWI exits with the V flag set, some sanity checks will be performed on R0 to try and make sure it's a valid error block pointer. This includes sanity-checking the pointer value, as well as the error number (e.g. making sure no reserved bits are set). These checks are designed to help catch programming mistakes before they are able to cause problems elsewhere – e.g. on a system running with high processor vectors, a null error pointer will result in a crash. Without help from the kernel, the SWI which generated the bad error pointer may be hard to track down.

If a bad error pointer is detected then it will be swapped for a pointer to a different error block, with the error message describing which SWI is at fault.

For compatibility with existing programs, `XOS_GenerateError` is exempt from SWI error pointer validity checks (it is essentially a no-op as far as error generation is concerned, and many programs take advantage of this and abuse the SWI by using it for other purposes, e.g. as a method of triggering callbacks or converting null-terminated strings to BASIC form). Similarly, `XMessageTrans_ErrorLookup` is exempt from error number validity checks, as some code operates by passing in a dummy error number and then replacing it with a valid one in the returned error block (although often this isn't necessary and the correct error number could have just as easily be placed in the input block). All other SWIs, whether called in X or non-X forms, will have the error pointer check performed.

[RISC OS Open source page](#)

System Control Handlers

Controlling sources: Hdr:EnvNumbers managed by the allocations manager

- 0 · Memory limit** – OS_ChangeEnvironment R1 value: Memory limit · R2 value: Unused · R3 value: Unused
- 1 · Undefined instruction** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Unused · R3 value: Unused
- 2 · Prefetch abort** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Unused · R3 value: Unused
- 3 · Data abort** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Unused · R3 value: Unused
- 4 · Address exception** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Unused · R3 value: Unused
- 5 · Other exceptions (reserved)** – OS_ChangeEnvironment R1 value: Unused · R2 value: Unused · R3 value: Unused
- 6 · Error** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Handler R0 · R3 value: Error buffer
- 7 · CallBack** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Handler R12 · R3 value: Register dump buffer
- 8 · BreakPoint?** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Handler R12 · R3 value: Register dump buffer
- 9 · Escape** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Handler R12 · R3 value: Unused
- 10 · Event?** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Handler R12 · R3 value: Unused
- 11 · Exit** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Handler R12 · R3 value: Unused
- 12 · Unused SWI?** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Handler R12 · R3 value: Unused
- 13 · Exception registers** – OS_ChangeEnvironment R1 value: Register dump buffer · R2 value: Unused · R3 value: Unused
- 14 · Application space** – OS_ChangeEnvironment R1 value: Memory limit · R2 value: Unused · R3 value: Unused
- 15 · Currently active object** – OS_ChangeEnvironment R1 value: CAO pointer · R2 value: Unused · R3 value: Unused
- 16 · UpCall** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Handler R12 · R3 value: Unused
- 32 · Branch Through 0 Handler?** – OS_ChangeEnvironment R1 value: Handler code · R2 value: Unused · R3 value: Unused
- 33 · Command Line Area?** – OS_ChangeEnvironment R1 value: Unused · R2 value: Size of command line area · R3 value: Command line area
- 34 · Exception Dump Region** – OS_ChangeEnvironment R1 value: Unused · R2 value: Size of exception dump region · R3 value: Exception dump region

[RISC OS Open source page](#)

System Device Numbers

Controlling sources: Hdr:DevNos managed by the allocations manager

Note: The following table is only valid for IOMD-based pre-HAL machines. For machines with a HAL (even a HAL running on IOMD), programmers should make no assumption about device numbers following any fixed scheme. Instead, the appropriate OS calls or hardware-specific documentation should be used to determine the correct device numbers.

- 0** – Meaning: Printer interrupt from 82C710/1
- 1** – Meaning: Unused (was Low battery warning on IOC)
- 2** – Meaning: Floppy disc indexer
- 3** – Meaning: VSync Pulse
- 4** – Meaning: Power on reset
- 5** – Meaning: Timer 0
- 6** – Meaning: Timer 1
- 7** – Meaning: FIQ downgrade - reserved for the current owner of FIQ
- 8** – Meaning: Expansion card FIQ downgrade; this should normally be masked off
- 9** – Meaning: IDE hard disc interrupt (was Sound system buffer change on IOC)
- 10** – Meaning: Serial port interrupt from 82C710/1 also mapped to FIQ device 4
- 11** – Meaning: Network interrupt
- 12** – Meaning: Floppy disc interrupt
- 13** – Meaning: Expansion card interrupt

- 14 – Meaning: Keyboard serial transmit register empty
- 15 – Meaning: Keyboard serial receive register full
- 16 – Meaning: DMA channel 0
- 17 – Meaning: DMA channel 1
- 18 – Meaning: DMA channel 2
- 19 – Meaning: DMA channel 3
- 20 – Meaning: DMA sound channel 0
- 21 – Meaning: DMA sound channel 1
- 22 – Meaning: ARM 7500 mouse port receive register full
- 23 – Meaning: ARM 7500 mouse port transmit register empty
- 24 – Meaning: ARM 7500 joystick A-to-D completion
- 25 – Meaning: ARM 7500 event 1
- 26 – Meaning: ARM 7500 event 2

Notes

PCI device numbers are allocated dynamically - see PCI_ReadInfo (bit 11).

[RISC OS Open source page](#)

System Variables

System Variables, as its name suggests, are Global variables that store various settings pertaining to various aspects of the current Operating System environment.

In practice

The table below details the most common uses of System Variables.

Application variables – Description: It is common for Applications to specify several system variables

Alias variables – Description: How to change or add command associated with a System Variable

FileSwitch variables – Description: Relates to current Filing System in use

File type variables – Description: Specify how specific file type should be handled by the OS

Filing Systems – Description: Specify how absent Filing Systems should be handled

Filer – Description: Specify how to adjust the behaviour when dropping a file on the filer icon

CLI – Description: Settings specific to the Command Line Interpreter

Configuration – Description: Various OS related settings are stored as System Variables

System – Description: Specify pathnames that will be searched for files during read or execute operations

Obey files – Description: Specify the directory from which an Obey file is being run

Time and date – Description: Used to evaluate time, date and years

Return codes – Description: Provides access to the last returned value given by OS_Exit

!System and !Scrap – Description: Provides full pathnames to the respective directories

Desktop – Description: Various settings and values specific to the desktop environment

TaskWindow – Description: Specify pathname to application that was used to start up task windows

Devices – Description: Various settings specific to devices attached to the system

Printing – Description: Specify various settings specific to the Printing device

Application Variables

When developing an application, a number of different System Variables will be necessary. Some are frequently used by developers, while others are for more specialised purposes. The following is a list of different application System Variables.

In the examples listed below, you should replace App with your own application name, adhering to the Naming conventions guidelines. Where you need a system variable and can't find a relevant one below, you should use your own, naming it App\$.

Example / Description

App\$Dir – Description: Provides the full pathname of the directory that holds the Application App. This is typically set in the applications !Run file. e.g. Set App\$Dir <Obey\$Dir>

App\$Path – Description: Provides the full pathname of the directory that holds the Application App. An App\$Path variable differs from App\$Dir variable in two important respects: The pathname includes a trailing '.' The variable may hold a set of pathnames, separated by commas Note: It is more common to use an App\$Dir variable rather than an App\$Path variable. As an example, an App\$Path variable might be set in the application's !Run file by the following: Set App\$Path <Obey\$Dir>.,%.App. (if the application held further resources in the subdirectory App of the library

App\$Path_Message – Description: Provides an alternative error message to be used if the path App: cannot be found. The message is then used instead of the default one provided by the Operating System

App\$Options – Description: Provides start-up options of the application App An option that can be either on or off should consist of a single character, followed by the character '+' or '-' (eg M+ or S-) Other options should consist of a single character, followed by a number (e.g. P4 or F54) Options should be separated by spaces; so a complete string might be F54 M+ P4 S- This variable is typically used to save the state of an application to a desktop boot file, upon receipt of a desktop save message. A typical line output to the boot file might be: Set App\$Options F54 M+ P4 S- You should only save those options that differ from the default, and hence not output a line at all if the application is in its default state. You should however be prepared to read options that set the default values, in case users explicitly add such options

App\$PrintFile – Description: Provides the name of the file or system device to which the application App prints. This is usually a printer, and would be set in your application's !Run file by: Set App\$PrintFile printer

App\$Resources – Description: Provides the full pathname of the directory that holds application App resources. This would usually be set in your application's !Run file: Set App\$Resources App:Resources Note the use of App: to make use of App\$Path

App\$Running – Description: Provides a value indicating if application App is running. It should have the value 'Yes' if true. This might be used in the application's !Run file as follows: If "App\$Running" <> "" then Error App is already running Set App\$Running Yes When the application stops running, you should use *Unset to delete the variable

Alias Variables

Example / Description

Alias\$Command – Description: Provides the ability to define a new command named Command. For example: Set Alias\$Mode echo |<22>|<%0> By using the name of an existing command, you can change how it works.

FileSwitch Variables

Example / Description

FileSwitch\$... – Description: Contains the name of the current filing system, and FileSwitch\$TemporaryFilingSystem contains the name of the temporary filing system. FileSwitch\$SpecialField contains the last special field to have been evaluated as a path was processed. Please see FileSwitch section for more details

FileSwitch\$FilingSystem\$... – Description: Most filing systems provide system variables used to store their currently selected directory, previously selected directory, library directory, and user root directory. For a filing system fs, these are respectively: FileSwitch\$fs\$CSD FileSwitch\$fs\$PSD FileSwitch\$fs\$Lib FileSwitch\$fs\$URD

File type Variables

Example / Description

File\$Type_XXX – Description: Provides the textual name for a given file type of the hexadecimal value XXX. It is typically set in the !Boot file of an application that uses (read and/or write) the file type. For example: Set File\$Type_XXX TypeName The reason the !Boot file is used rather than the !Run file is so that the file type can be converted to text from the moment its 'parent' application is first seen, rather than only from when it is run

Alias\$@LoadType_XXX, Alias\$@PrintType_XXX, Alias\$@RunType_XXX – Description: Sets the commands that are used to load, print and run a file with a file type of the hexadecimal value XXX. They are typically set in the !Boot file of an application that uses (read and/or write) the file type. For example: Set Alias\$@PrintType_XXX /<Obey\$Dir> -Print Set Alias\$@RunType_XXX /<Obey\$Dir> Note: The above lines both have a trailing space (unable to be shown). The reason the !Boot file is used rather than the !Run file is so that files of the given type can be loaded, printed and run from the moment their 'parent' application is first seen, rather than only from when it is run. More information is available in the FileSwitch Use Of File Types page

Filing System Variables

Example / Description

FilingSystem\$Path_Message – Description: Provides an alternative error message to be used if the given FilingSystem cannot be found. The error message is then used instead of the default message provided by the Operating System

Filer Variables

Example / Description

FSTitle\$DefaultDir – Description: Defines an alternative directory to be used, instead of the root directory, when files are dropped on the iconbar filer icon to start a copy/move operation. An example to change the directory of the RAM filing system to is: Set RAMFiler\$DefaultDir Public This sets a variable so that the RAMFiler saves the file to the Public folder on the RAM filing system instead of the root directory.

CLI Variables

Example / Description

CLI\$Prompt – Description: Provides the prompt to be used at the command line interpreter. By default this is the '*' character. An example to display the time as the prompt is by: SetMacro CLI\$Prompt <Sys\$Time> * This sets a macro so that the system time variable is evaluated each time the command prompt is displayed

Configuration Variables

Example / Description

Copy\$Options, Count\$Options, Wipe\$Options – Description: Sets the behaviour of the *Copy, *Count and *Wipe commands

System Path Variables

Example / Description

File\$Path and Run\$Path – Description: Provides pathnames where files are searched for during, respectively, read and execute operations. They are both path variables, which means that, in common with other path variables, they consist of a comma separated list of pathnames, each which has a trailing '.' character If you wish to add a pathname to one of these variables, you must ensure that you append it once, and once only. For example, to add the 'bin' subdirectory of an application to Run\$Path, you could use the following lines in the application's !Boot file: If "<App\$Path>" = "" then Set Run\$Path <Run\$Path>,<Obey\$Dir>.bin. Set App\$Path <Obey\$Dir>. More information on File\$Path and Run\$Path is available [here](#)

Obey Variables

Example / Description

Obey\$Dir – Description: Provides the pathname of the directory from which an Obey file is being run, and may be used by commands within that Obey file. The value is set by the OS automatically. Note that it is not set to the full parent name, only the part of the string passed to the command as the pathname. So if you change the current directory or filing system during the obey file, then it would not be valid any more. Ideally, you should invoke Obey files (and applications, which are started by an Obey file named !Run) by using their full pathname, and preceding that by either a forward slash / or the word Run , for example: / adfs::MikeWinnie\$.Odds'nSods.MyConfig Run adfs::MikeWinnie\$.Odds'nSods.MyConfig This ensures that Obey\$Dir is set to the full pathname of the Obey file.

Time and Date Variables

Example / Description

Sys\$Time, Sys\$Date, Sys\$Year – Description: These are code variables that are evaluated at the time of their use to give, respectively, the current system time, date and year. An example of how they can be used is shown above under CLI Variable

Sys\$DateFormat – Description: Sets the format in which the date is presented by the OS_ConvertStandardDateAndTime SWI call. The format used by this variable is available [here](#)

Return Code Variables

Example / Description

Sys\$ReturnCode – Description: Provides the last return value given by the OS_Exit

Sys\$RCLimit – Description: Sets the maximum return value that will not generate an error

!System and other !Boot related Variables

Example / Description

System\$Dir and System\$Path – Description: Provides the full pathname of the System application. They have the same value, save that System\$Path has a trailing '.', whereas System\$Dir does not. You must not change their values Note: There are two versions of this pathname for reasons of backward compatability

Wimp\$Scrap – Description: Provides the full pathname of the Wimp scrap file used by the file transfer protocol. You must not use this variable for any other purpose, nor change its value

Wimp\$ScrapDir – Description: Provides the full pathname of a scrap directory within the Scrap application, which you may use to store temporary files. You must not use this variable for any other purpose, nor change its value

Desktop Variables

Example / Description

Desktop\$File – Description: Provides the desktop boot file that was used to start the desktop

Wimp\$State – Description: Provides the current state of the Wimp. If the desktop is running, it has the value 'desktop'; otherwise it has the value 'commands'

TaskWindow Variables

Example / Description

TaskWindow\$Server – Description: Provides the pathname of the application used to start up task windows

Devices Variables

Example / Description

DeviceFS\$Device\$Options – Description: Provides the default options for a DeviceFS device

Printer Variables

Example / Description

PrinterType\$n – Description: Provides the path used to print to printer type n. For example: *Show PrinterType\$0 PrinterType\$0 : null:

Type of System Variables

There are several types of system variable

String · Contains any characters you like; these are returned when the string is read – How to Set: *Set · How to Remove: *Unset

Integer · are four-byte signed integers – How to Set: *SetEval · How to Remove: *Unset

Macros · Macros are strings that are passed through OS_GSTrans when the string is read. This means that if the macro contains references to variables or other OS_GSReadable items, the appropriate translation takes place whenever the variable is accessed – How to Set: *SetMacro · How to Remove: *Unset

Example: To set the command line prompt (CLI\$Prompt) to the current time, followed by a space, you would use:

```
*SetMacro CLI$prompt <Sys$Time><&20>
```

Machine Code · A routine is called whenever the variable is to be read, and another when it is set. This allows great flexibility in the way in which such variables behave – How to Set: OS_SetVarVal · How to Remove: OS_SetVarVal

Example: you could make a variable directly control a CMOS RAM location using this technique

Sys\$Time is a good example of a code variable

Notes

- All the different System Variable types can be: set with OS_SetVarVal read with OS_ReadVarVal unset with OS_SetVarVal
- Any non-code variables can be: removed using the *Unset command
- List all System Variables: displayed using the *Show command

Naming of System Variables

You must not use wholly numeric names. e.g. 123 as this causes difficulties when the OS_GSRead and OS_GSTrans operations are used to lookup the value of the variable. In particular, they will always take <123> to mean the ASCII code 123, and will not attempt to look up the name as a variable.

Names may contain any non-space, non-control character.

When a variable is created, the case of the letters is preserved, however, when returning the values, the case is not important.

You can use the characters '#' and '*' – just like looking up filenames.

[RISC OS Open source page](#)

SystemDevices

List of modules

The SystemDevices module provides a number of different system devices, which in some ways behave like files. You can use them anywhere you would normally use a file name as a source or destination. This is achieved by using a method called redirection.

Details

- Technical Details

[RISC OS Open source page](#)

SystemDevices Technical Details

SystemDevices

System Devices can be useful with commands such as *Copy, and the redirection operators (> and <):

List of System Devices

kbd: • Y – Output: Y · Provide by Module Name: SystemDevices · Description: The keyboard · Notes: Reading a line at a time using OS_ReadLine

rawkbd: • Y – Output: Y · Provide by Module Name: SystemDevices · Description: The keyboard · Notes: Reading a character at a time using OS_ReadC

null: • Y – Output: Y · Provide by Module Name: SystemDevices · Description: The 'null' device · Notes: Gives no input/output

printer: • N – Output: Y · Provide by Module Name: SystemDevices · Description: The printer · Notes: Currently configured printer

vdv: • N – Output: Y · Provide by Module Name: SystemDevices · Description: The screen · Notes: Using GSRead format passed to OS_WriteC

rawvdv: • N – Output: Y · Provide by Module Name: SystemDevices · Description: The screen · Notes: Via the VDU drivers and OS_WriteC

netprint: • N – Output: Y · Provide by Module Name: NetPrint · Description: The printer · Notes: Currently configured printer

device: • Y – Output: Y · Provide by Module Name: DeviceFS · Description: Device filing system · Notes: -

parallel: • N – Output: Y · Provide by Module Name: DeviceFS · Description: Parallel port · Notes: -

serial: • N – Output: Y · Provide by Module Name: DeviceFS · Description: Serial port · Notes: -

pipe: • Y – Output: Y · Provide by Module Name: PipeFS · Description: The pipe filing system

resource: • Y – Output: Y · Provide by Module Name: ResourceFS · Description: The resource filing system

Redirection

Example / Description

***Copy myfile printer:** – Description: Send myfile to the printer

***Cat { > printer: }** – Description: List the files in the current directory to the printer

Suppressing Output

By using the System Device named null: it is possible to suppress unwanted output from a command script or program:

Example / Description

***myprogram { > null: }** – Description: Run myprogram with no output

Input Devices

You can only open one file for input on kbd: at once as it has buffered input; normal line editing facilities are available. If you try to open kbd: a second time whilst the first file is open, you will get returned a handle of 0, or an error if the appropriate bit is set in the open mode passed to FileSwitch. Ctrl-D in the input line will yield EOF when it is read from the buffer.

You can open rawkbd: as many times as you like, even if a file is open on kbd:. It uses XOS_ReadC (without echoing to the screen) to read characters. No EOF condition exists on rawkbd:; the program reading it must detect an input value/pattern and stop on that.

No files exist on any of these devices. If you call OS_File 5 on the devices it will always return object type 0, so you cannot use them for input to programs that need to load an entire file at once for processing.

netprint Device

The netprint: system device, as well as using it in place of file names, you can also be used with certain commands that normally use the name of a filing system.

printer Device

This device allows various special fields to specify different types of printers. They are:

printer#null: – Printer Type: Null Printer

printer#sink:

printer#parallel: – Printer Type: Parallel Printer

printer#centronics:

printer#serial: – Printer Type: Serial Printer

printer#rs423:

printer#user: – Printer Type: Printer Type 3

printer# n : – Printer Type: Printer Type n (where n is in the range 0 - 255)

It is possible to open multiple files on printer:, provided they are on different devices and using different buffers.

Other Output Devices

You can open as many files as you wish on the 'other output devices', which are:

- null:
- vdu:
- rawvdu:

H% = OPENOUT "rawvdu:"

SYS"OS_Byte",199,H%,0

type here...

*Spool

When you type everything is sent to the vdu, which outputs it and then uses XOS_BPut to send it to the spool file handle.

This in turn sends it (through another mechanism, OS_PrintChar) to the screen again! The *Spool at the end clears up.

In addition to byte-oriented operations, you are allowed to perform file save operations on the output devices.

The difference between vdu: and rawvdu: is that the former is filtered using the configured DumpFormat, whereas the latter sends its characters straight to the VDU drivers.

[RISC OS Open source page](#)

TaskManager

List of modules

The Task Manager module provides several functions for task management. Currently it allows one to find a task with a given handle, to enumerate all currently active tasks, to initiate a desktop shutdown, and to start a new desktop task.

Details

- SWI Calls
- *Commands

[RISC OS Open source page](#)

TaskWindow

List of modules

The TaskWindow module allows programs that do not call Wimp_Poll to be pre-emptively scheduled in the RISC OS desktop.

In practice

The following terminology is used:

- 'Child' refers to the task created by the *TaskWindow command
- 'Parent' refers to the task being used to display the Child's output

Any screen output produced by the Child is intercepted and sent in Wimp messages to the Parent.

Details

- Messages
- SWI Calls
- *Commands

[RISC OS Open source page](#)

TaskWindow_Ego

Code &808C2 · TaskWindow

This message is sent to your task after a *StartTask xxx -txt &xxxxxxx command. It allows the Parent to identify which Child is announcing its task-id.

Message Block

+4 · Task handle of child

+20 · Parent's txt handle (as passed to *TaskWindow or *ShellCLI_Task)

Notes

This is the first and last time the txt-handle will be used. After this call, the child's task-id will be used instead.

[RISC OS Open source page](#)

TaskWindow_Input

Code &808C0 · TaskWindow

This message is used to give input data (key-presses) to the child task.

Message Block

+4 · Child task id

+20 · Size

+24 · Data...

Notes

Larger amounts of data can be sent with the Data Transfer Protocol. Just send a Message_DataSave message, and wait for the Message_RAMFetch message.

[RISC OS Open source page](#)

TaskWindow_Morio

Code &808C3 · TaskWindow

This message is sent to Parent when Child exits.

Message Block

No message data

[RISC OS Open source page](#)

TaskWindow_Morite

Code &808C4 · TaskWindow

This message is sent by Parent to kill Child.

Message Block

No message data

[RISC OS Open source page](#)

TaskWindow_NewTask

Code &808C5 · TaskWindow

This message is broadcast by the TaskWindow module when a *TaskWindow command has been executed without proper -task and -txt flags.

Message Block

No data passed

Notes

If receiving task can support taskwindows, it should:

- Acknowledge this message via User_Message_Acknowledge;
- Issue a Wimp_StartTask, where R1+20 contains the *Command to be run.

[RISC OS Open source page](#)

TaskWindow_Output

Code &808C1 · TaskWindow

This message is sent when a child has produced output data.

Message Block

+4 · Child task id

+20 · Size (max 232 bytes)

+24 · Data ...

Notes

The process for using the data is:

- Acknowledge this message by User_Message_Acknowledge;
- Display the data sent.

[RISC OS Open source page](#)

TaskWindow_Resume

Code &808C7 · TaskWindow

This message is sent by Parent to resume a previously suspended Child.

Message Block

No message data

[RISC OS Open source page](#)

TaskWindow_Suspend

Code &808C6 · TaskWindow

This message is sent by Parent to suspend Child.

Message Block

No message data

[RISC OS Open source page](#)

Territory Numbers

Controlling sources: Hdr:Countries managed by the allocations manager

It appears that territory and country numbers share the same sequence.

1 – Territory: UK

2 – Territory: Master

3 – Territory: Compact

4 – Territory: Italy

5 – Territory: Spain

6 – Territory: France
7 – Territory: Germany
8 – Territory: Portugal
9 – Territory: Esperanto
10 – Territory: Greece
11 – Territory: Sweden
12 – Territory: Finland
13 – Territory: Unused
14 – Territory: Denmark
15 – Territory: Norway
16 – Territory: Iceland
17 – Territory: Canada1
18 – Territory: Canada2
19 – Territory: Canada
20 – Territory: Turkey
21 – Territory: Arabic
22 – Territory: Ireland
23 – Territory: Hong Kong
24 – Territory: Russia
25 – Territory: Russia2
26 – Territory: Israel
27 – Territory: Mexico
28 – Territory: LatinAm
29 – Territory: Australia
30 – Territory: Austria
31 – Territory: Belgium
32 – Territory: Japan
33 – Territory: MiddleEast
34 – Territory: Netherlands
35 – Territory: Switzerland
36 – Territory: Wales
48 – Territory: USA
49 – Territory: Wales2
50 – Territory: China
51 – Territory: Brazil
52 – Territory: SAfrica
53 – Territory: Korea
54 – Territory: Taiwan
70 – Territory: DvorakUK
71 – Territory: DvorakUSA
80 – Territory: ISO1
128 – Territory: Faroe
129 – Territory: Albania
130 – Territory: Safrica (old)
131 – Territory: Bengal
132 – Territory: Bulgaria
133 – Territory: ByeloRuss
134 – Territory: Czech
135 – Territory: Devenag
136 – Territory: Farsi
137 – Territory: Gujarati
138 – Territory: Estonia

139 – Territory: Gaelic
140 – Territory: GreeceAnct
141 – Territory: Greenland
142 – Territory: Hungary
143 – Territory: Lapp
144 – Territory: Latvia
145 – Territory: Lithuania
146 – Territory: Macedonia
147 – Territory: Malta
148 – Territory: Poland
149 – Territory: Punjab
150 – Territory: Romania
151 – Territory: SerboCroat
152 – Territory: Slovak
153 – Territory: Slovene
154 – Territory: Tamil
155 – Territory: Ukraine
156 – Territory: Swiss1 (Swiss-French)
157 – Territory: Swiss2 (Swiss-German)
158 – Territory: Swiss3 (Swiss-Italian)
159 – Territory: Swiss4 (Swiss-Romansch)
[RISC OS Open source page](#)

TerritoryManager

List of modules

The TerritoryManager module provides the functionality for applications to access 'territory modules'. Each territory module provides the services and information necessary for both RISC OS and the application to be easily adapted for use in different regions of the world (i.e. territories).

Details

- Service Calls
- SWI Calls
- *Commands

[RISC OS Open source page](#)

The Acorn Terminal Interface Protocol

The Acorn Terminal Interface Protocol (Acorn TIP) is used to communicate between a terminal emulator and a protocol module. It is possible to integrate your own terminal emulators and protocol modules with those provided by the TCP/IP Protocol Suite.

Protocol Modules

The purpose of a Protocol Module is to convert one of the many different protocols computers use for input and output to the Acorn TIP.

- Data passing between a terminal emulator and a protocol module uses the Acorn TIP, and passes over a logical link

- Data passing between a protocol module and a remote machine or process uses whatever protocol the module is designed to support, and passes over a connection.

How to develop a Protocol Module

It is strongly advised that before developing a Protocol Module, you should first familiarise yourself with how a RISC OS relocatable module works. Your protocol module must conform to the standards specified.

Service Calls

Your Protocol Module must support all service calls detailed.

SWI Calls

Your Protocol Module must support various SWIs, and they vary based on the functionality it will provide. These must be at the defined offsets from your module's SWI base number, which is allocated by RISC OS Open. To support many of these SWIs you will need to send suitable commands over the physical connection to the remote host.

- 0 · Protocol_OpenLogicalLink?** – Supports sending of data: · Supports sending of data & file transfer: · Supports receipt of data: · Supports receipt of data & file transfer: · Optional: -
- 1 · Protocol_CloseLogicalLink?** – Supports sending of data: · Supports sending of data & file transfer: · Supports receipt of data: · Supports receipt of data & file transfer: · Optional: -
- 2 · Protocol_GetProtocolMenu?** – Supports sending of data: · Supports sending of data & file transfer: · Supports receipt of data: · Supports receipt of data & file transfer: · Optional: -
- 3 · Protocol_OpenConnection?** – Supports sending of data: · Supports sending of data & file transfer: · Supports receipt of data: · Supports receipt of data & file transfer: · Optional: -
- 4 · Protocol_CloseConnection?** – Supports sending of data: · Supports sending of data & file transfer: · Supports receipt of data: · Supports receipt of data & file transfer: · Optional: -
- 5 · Protocol_TransmitData?** – Supports sending of data: · Supports sending of data & file transfer: · Supports receipt of data: - · Supports receipt of data & file transfer: - · Optional: -
- 6 · Protocol_DataRequest?** – Supports sending of data: - · Supports sending of data & file transfer: - · Supports receipt of data: · Supports receipt of data & file transfer: · Optional: -
- 7 · Protocol_MenuItemSelected?** – Supports sending of data: · Supports sending of data & file transfer: · Supports receipt of data: · Supports receipt of data & file transfer: · Optional: -
- 8 · Protocol_UnknownEvent?** – Supports sending of data: · Supports sending of data & file transfer: · Supports receipt of data: · Supports receipt of data & file transfer: · Optional: -
- 9 · Protocol_GetLinkState?** – Supports sending of data: · Supports sending of data & file transfer: · Supports receipt of data: · Supports receipt of data & file transfer: · Optional: -
- 10 · Protocol_Break?** – Supports sending of data: · Supports sending of data & file transfer: · Supports receipt of data: · Supports receipt of data & file transfer: · Optional: -
- 11 · Protocol_SendFile?** – Supports sending of data: - · Supports sending of data & file transfer: · Supports receipt of data: - · Supports receipt of data & file transfer: - · Optional: -
- 12 · Protocol_SendFileData?** – Supports sending of data: - · Supports sending of data & file transfer: · Supports receipt of data: - · Supports receipt of data & file transfer: - · Optional: -
- 13 · Protocol_AbortTransfer?** – Supports sending of data: - · Supports sending of data & file transfer: · Supports receipt of data: - · Supports receipt of data & file transfer: · Optional: -
- 14 · Protocol_GetFileInfo?** – Supports sending of data: - · Supports sending of data & file transfer: - · Supports receipt of data: - · Supports receipt of data & file transfer: · Optional: -
- 15 · Protocol_GetFileData?** – Supports sending of data: - · Supports sending of data & file transfer: - · Supports receipt of data: - · Supports receipt of data & file transfer: · Optional: -
- 17 · Protocol_GetFile?** – Supports sending of data: - · Supports sending of data & file transfer: - · Supports receipt of data: - · Supports receipt of data & file transfer: · Optional: -
- 18 · Protocol_DirOp?** – Supports sending of data: - · Supports sending of data & file transfer: - · Supports receipt of data: - · Optional: -

Data Structures

Your protocol module must keep two different types of data structure constantly updated, as terminal emulators may directly access these any time they need to. These are:

- A single protocol information block which contains the following information:
- A poll word for each logical link that shows the status of that link by the state of various bit flags:

Protocol Information Block

Offset / Contents

- 0** – Contents: Pointer to protocol name string
- 4** – Contents: Pointer to protocol version string
- 8** – Contents: Pointer to protocol copyright string
- 12** – Contents: Maximum number of connections allowed by module
- 16** – Contents: Current number of open connections

The three strings are all null-terminated, and have a maximum length of 30 characters. More information is available on Protocol_OpenLogicalLink? (Offset 0).

Poll Word

Bit / Meaning when set

- 0** – Meaning when set: Data is pending
- 1** – Meaning when set: File is pending
- 2** – Meaning when set: Paused operation is to continue

For more information, please see Protocol_OpenConnection?.

Multiple Links and Connections

All protocol modules must (if physically possible) support multiple logical links, and multiple connections.

Developing a Terminal Emulator

There are various functions that your Terminal Emulator will need to support. This section details which SWI calls you need to use for many such functions, and outlines how to use them.

Finding available and compatible Protocols

To find what protocols are available and compatible with the needs of your emulator, you must:

- Repeatedly issue Service_FindProtocols until it is not claimed
- Then issue Service_FindProtocolsEnd

Choosing a protocol and opening a link

For your user to choose a protocol, you'll probably want to give them a menu of the ones you found to be available. Once they've made the choice, you can then issue Service_ProtocolNameToNumber to find the base SWI number of their chosen protocol module. You can then use this base number to call Protocol_OpenLogicalLink?, since its offset from the base number you just found is zero.

You can also use the facilities outlined in the section below entitled Protocol Modules and the Wimp to provide menus so that your user can set up the way the protocol and connection will work.

Opening a Connections

- Call Protocol_OpenConnection?
- Call Protocol_GetLinkState? to determine if the connection eventually is successful or not. (Sometime the protocol module won't immediately be able to open the connection)

Closing a Connection and a Link

- Call Protocol_CloseConnection?, or
- To close a logical link, call Protocol_CloseLogicalLink? - this also closes any associated connections

Examining the Poll Word

When a connection is established/opened, you set the address of a poll word. The protocol module sets bits in this word when it needs attention. It's vital that your emulator regularly examines this word so that the protocol module gets adequate service.

Note: It is good practice to check the poll word each time a null event from `Wimp_Poll` is received.

Sending Data

Sending data is achieved by Calling `Protocol_TransmitData?`

Receiving Data

When the protocol module receives data over a connection, it will notify your emulator by setting a bit in the poll word. To get the data forwarded to your emulator, call `Protocol_DataRequest?`.

Sending Files

- Call `Protocol_SendFile?` to give details of the file to the protocol module
- Scan the poll word bits to identify when the protocol module is ready to receive the file data
- Send the file in one more more data packets by repeatedly calling `Protocol_SendFileData?`
- Call `Protocol_GetFileData?` to mark the end of the file transfer

Receiving Files

When the protocol module receives a file over a connection, it will notify your emulator by setting a bit in the poll word.

- To get the file forwarded to your emulator, call `Protocol_GetFileInfo?`
- Scan the poll word to identify when the protocol module shows it is ready to forward the file
- Call `Protocol_GetFileData?` until all the data packets making up the file have been received

Note: To explicitly get a file, call `Protocol_GetFile?`

Aborting File Operations

- To abort any file operation, call `Protocol_AbortTransfer?`

Directory Operations

There are no SWIs specified in the Acorn TIP to send, receive or get entire directories in one call. Instead, a single SWI call, `Protocol_DirOp?` is provided. This call allows you to create a directory, move into a directory, and move one level up a directory tree.

Combining this SWI with the ones outlined above allows you to move around a remote file system, creating directories, and sending and getting files at will (subject, to access rights).

Protocol Modules and the Wimp

Several calls are provided which help interaction between the Wimp and protocol menus. These are necessary because the 'pick and mix' nature of protocol modules and terminal emulators means you'll have to combine menus from each; and because protocol modules are not foreground tasks, and so don't receive notice of menu selections and Wimp events.

Retrieving a Protocol's Menu Tree

- Call `Protocol_GetProtocolMenu?` and you combine it with your emulator's menu tree
- If a user clicks on the protocol module's part of the menu tree, pass it on using `Protocol_MenulitemSelected?`
- To pass on a Wimp event to a protocol module, call `Protocol_UnknownEvent?`. You should do this for every event your emulator can't deal with, as the protocol module may be able to.

Generating a Break

- Generate a break over the connection by calling Protocol_Break?

[RISC OS Open source page](#)

TickerV

Vectors

This vector is called every centisecond, and must never be intercepted.

Before calling

--

Notes

If your routine takes > 100µs you may re-enable IRQ, as long as you can cope with/prevent re-entrancy.

[RISC OS Open source page](#)

TLB_InvalidateAll ARMop

OS

The TLB or TLBs are to be globally invalidated.

Before calling

--

On return

R0 Corrupt

[RISC OS Open source page](#)

TLB_InvalidateEntry ARMop

OS

The TLB or TLBs are to be invalidated for the entry at the given logical address.

Before calling

R0 Logical address of entry to invalidate (page aligned)

On return

R0 Corrupt

Notes

The address must be page aligned (4k).

[RISC OS Open source page](#)

UCS Font Manager

FontManager

This call works only by looking at encoding files - it cannot guarantee that a given character is actually defined in a font file, but it can say which characters definitely aren't, by returning with R2 set to -1.

In practice

The UCS Font Manager is a development of the RISC OS Font Manager whose primary purpose is to allow access to more than 256 characters in a font by using UCS character codes.

Unlike the earlier Bitstream-based prototypes, the UCS Font Manager is fully backwards compatible with older versions of the Font Manager and supports almost all existing API calls. The standard RISC OS font format is used.

For backwards compatibility, font encodings continue to be specified in terms of PostScript glyph names. This is slightly space inefficient, but allows a pleasing elegance of design.

2.1 Internal changes

The major internal difference between the UCS Font Manager (version 3.41+) and earlier versions is that bitmaps are held in the cache in file order.

Since RISC OS 3 the Font Manager has been capable of using fonts with glyphs in an arbitrary order and remapping - this is used in the standard Corpus, Homerton and Trinity ROM fonts to make them available in any of Latin 1-4.

The remapping from external (e.g. Latin1) character codes to internal (eg /Base0) glyph codes was performed as outlines were turned into bitmaps, and as metrics were scaled. Hence slave fonts in the cache contained data in, say, Latin1 order, while master fonts stored data in /Base0 order.

This made sense because external codes always contained 256 characters, allowing several arrays to be fixed size, irrespective of the size of the font file. Unfortunately, the new external code UCS contains 2,147,483,648 characters, scuppering that idea. Hence the Font Manager now stores everything in file order, including bitmaps, metrics and kerning information.

This now leads to some inefficiencies in the font cache. Using the two fonts "FHomerton.Medium\ELatin1" and "FHomerton.Medium\ELatin2" will result in two master fonts and two slave fonts, each pair containing the same outlines and same bitmaps, differing only in the stored encoding tables. It may be worth looking at this at some stage.

Another minor change is that metrics are no longer scaled and stored in slave fonts. Only the unscaled master metrics are stored, and scaled on demand.

2.2 Default encodings

As of Font Manager 3.40, the behaviour of the Font Manager when presented with a font identifier with no encoding has changed subtly, but significantly.

Previously, asking for "Homerton.Medium" would cause the Font Manager to call Territory_AlphabetIdentifier to find the default alphabet for the system territory. Hence, on a UK machine you would get "FHomerton.Medium\ELatin1". Asking for "Selwyn" would not apply any encoding.

As of Font Manager 3.40, the behaviour is now to check the current alphabet using OS_Byte 71, and to get the name via Service_International. This means that the command *Alphabet now affects both the system font and future outline font claims.

If a font is a "symbol" font - i.e. contains an "IntMetrics" file rather than an "IntMetric" file, then the default encoding is "Glyph", rather than the system encoding, unless the system encoding is "UTF8", in which case this is treated as the default encoding for all fonts.

2.3 Encodings

To support the UCS, a number of changes have been made to the RISC OS font encoding file scheme. Firstly, the %RISCOS_BasedOn directive is now ignored. Up to now, asking for a font with encoding Latin1 would always try to find

IntMetric0 and Outlines0 files, because the file Encodings.Latin1 contained the directive "%RISCOS_BasedOn 0".

Now the font manager will find the first file fitting the template "IntMet*", and use it and its associated Outlines file. (This means that it is no longer possible to have more than one set of metrics/outlines per font.) It will then deduce the base encoding of that file from its suffix. If the file is, say, IntMetri32, it will use the encoding file "Font:Encodings./Base32". If the file is just IntMetrics, it will use the file "Encoding" in the font directory. If that isn't present, it will use the encoding file "Font:Encodings./Default".

This change means that to claim a font with Latin1 encoding no longer requires that the font file be in /Base0 order.

Next, the encoding generation has been relaxed:

- Target encoding files (eg Latin1) no longer need to have exactly 256 entries.
- If a glyph specified in a target file is not present in a base file, then that glyph is treated as if it were /.notdef, rather than an error being generated.

This permits us to claim MaruGothic.Medium (the Funai 3 Japanese font) with encoding \ELatin1, even though it doesn't contain all the Latin1 glyphs. Possibly more importantly, it allows us to claim any font with encoding \EUTF8. No font contains all UCS/UTF-8 glyphs :)

The final change to the encoding scheme is an "extension" to the encoding file format that prevents us having to have a 2 billion line long UTF8 encoding file.

The font manager now supports a format compatible with the Adobe Glyph List, which, unlike the straight list of glyph names, allows a "sparse" encoding.

For example, the following encoding file

```
0005;fred;SYMBOL FOR FRED
```

```
0002;Jim;SYMBOL FOR JIM
```

```
0007;Harry;SYMBOL FOR HARRY
```

```
# Comment
```

```
22;000C;Tom;SYMBOL FOR TOM
```

is equivalent to the old style encoding file

```
/.notdef /.notdef /Jim /.notdef
```

```
/.notdef /fred /.notdef /Harry
```

```
/.notdef /.notdef /.notdef /.notdef
```

```
/Tom
```

The line format is [XX;]XXXX;NNN;CCCC – the first field is ignored if less than 4 digits long. The CCCC field is ignored.

This format allows the UTF8 encoding file to be a direct concatenation of the main Adobe Glyph List and ZapfDingbats Glyph List.

2.4 UTF8 Encoding

In many respects, the encoding "UTF8" is just like any other encoding. All encodings may now contain up to 2³¹ characters, and the format used for the UTF8 file can be used with any encoding. However, a number of special cases are invoked by the encoding being named "UTF8".

2.4.1 UTF8 encoding file

The UTF8 file in Funai 3 was based on version 1.1 of the Adobe Glyph List. This has now been updated to version 1.2, and all font encoding files have been updated in line.

2.4.2 /uniXXXX identifiers

If the target encoding file being processed is called "UTF8", then any identifiers in the base encoding file fitting the template /uniXXXX (where XXXX is a 4 digit upper-case hexadecimal number) will be mapped in at character code XXXX. This covers the majority of UCS characters, which don't have PostScript glyph names.

Also, as of version 3.43, the form /uniXXXXYYYY is accepted, when XXXX is a high-surrogate (D800-DBFF) and YYYY is a low-surrogate (DC00-DFFF), to represent UCS characters in the range U+00010000 to U+0010FFFF, as specified in version 1.1 of Adobe's "Unicode and Glyph Names"

There is currently no way to specify codes above U+00110000.

The prioritisation of double-mappings described in “Unicode and Glyph Names” is operative. For example, if a font’s encoding contains “/mu”, the glyph will be given UCS codes 00B5 and 03BC. If a font’s encoding contains both “/mu” and “/uni03BC”, then /mu will be mapped at 00B5, and /uni03BC at 03BC.

However, for non-UTF8 encodings, names must currently match exactly. So if a Greek encoding file contains “/mu”, then it will only match a glyph called “/mu”, not one called “/uni03BC”, and vice-versa.

2.4.3 UTF-8/UTF-16 encoding

If the target encoding for a font handle is “UTF8”, then UTF-8 and UTF-16 processing will be invoked for 8 and 16-bit forms of Font_Paint, etc. respectively. (See below).

2.5 Glyph Encoding

A new encoding “Glyph” is available. This is a special case which requests a font with no encoding applied. Thus the font “\FHomerton.Medium\EGlyph” would contain all 300+ Homerton.Medium glyphs, in the order they are in the font file.

2.6 8/16/32 bit string forms

The SWIs Font_Paint and Font_ScanString now accept strings in 8, 16 or 32-bit forms. (All other SWIs only accept 8-bit strings). This is invoked by bits in R2:

bit 12 set / R1 points to a 16-bit string (bit 13 must be clear)

bit 13 set – R1 points to a 16-bit string (bit 13 must be clear): R1 points to a 32-bit string (bit 12 must be clear)

A 16-bit string must be half-word aligned, and consists of a number of 16-bit halfwords. A 32-bit string must be word aligned, and consists of a number of 32-bit words.

For example, the string “Hello” might be represented as follows:

8-bit / 48 65 6C 6C 6F 00

16-bit – 48 65 6C 6C 6F 00: 0048 0065 006C 006C 006F 0000

32-bit – 48 65 6C 6C 6F 00: 00000048 00000065 0000006C 0000006C 0000006F 00000000

This allows access to characters outside the first 256. Control codes work as before, but have to be introduced by a character of the appropriate width. The parameters of control sequences are handled differently in 16 and 32-bit modes:

16-bit parameter forms:

17: – dx/dy comprised of 2 16-bit words: 1 16-bit word

18: – dx/dy comprised of 2 16-bit words: 3 16-bit words

19: – dx/dy comprised of 2 16-bit words: 2 x (RR00, BBGG word pairs) + 1 16-bit word

21: – dx/dy comprised of 2 16-bit words: sequence of 16-bit words, terminated by a word < &0020

25: – dx/dy comprised of 2 16-bit words: 2 16-bit words

26: – dx/dy comprised of 2 16-bit words: 1 16-bit word

27: – dx/dy comprised of 2 16-bit words: 4 32-bit words (word-aligned)

28: – dx/dy comprised of 2 16-bit words: 6 32-bit words (word-aligned)

32-bit parameter forms:

17: – 1 32-bit words: 1 32-bit word

18: – 1 32-bit words: 3 32-bit words

19: – 1 32-bit words: 3 32-bit words

21: – 1 32-bit words: sequence of 32-bit words, terminated by a word < &00000020

25: – 1 32-bit words: 2 32-bit words

26: – 1 32-bit words: 1 32-bit word

27: – 1 32-bit words: 4 32-bit words

28: – 1 32-bit words: 6 32-bit words

Regardless of the setting of the 16/32-bit bits, string lengths continue to be in bytes (such as R7 on entry to Font_Paint).

2.7 UTF-8/UTF-16 string forms

If a font's encoding is "UTF8", then rather than taking "straight" 8 and 16 bit strings, strings are decoded according to UTF-8 or UTF-16. For example, the byte sequence in UTF-8 represents character &A3. The halfword sequence in UTF-16 represents character &10000. 32-bit strings are treated normally (hence the string is treated as UCS-4).

UTF-8 allows access to all 31 bits of the UCS in 8-bit mode. UTF-16 allows access to between 20 and 21 bits of the UCS in 16-bit mode.

Again, string lengths continue to be in bytes. Illegal or incomplete multibyte/multiword sequences are treated as character &FFFD. This includes "long" UTF-8 forms such as .

2.8 File format extensions

The basic file formats remain unaltered. However, various bugs/limitations in the Font Manager and FontEd's handling of large files have been fixed.

- 16-bit composite characters are now supported (but see note below about dependency bytes).
- FontEd now handles an arbitrary number of dependency bytes. However the Font Manager only handles 4 dependency bytes (32 bits). Thus composite character inclusions must be within the first 32 chunks (1024 characters).
- Bitmap files with > 256 characters are now supported. Font_MakeBitmap now produces version 8 files. Bitmap files are expected to be in the same order and directory as the Outlines file (so latin1 etc subdirectories are no longer used.)

Some limitations (as of 3.43) still remain; for example scaffold data is still limited to 64K.

3.1 SaveFontCache/LoadFontCache

*LoadFontCache now checks the base address of a saved font cache before loading it and reports a sensible error if not correct. Note that the font cache format for the UCS Font Manager is totally incompatible with previous versions.

3.2 Font_ReadFontMetrics

As of version 3.42, Font_ReadFontMetrics call will no longer return kerning information into the buffer in R5. Further, the bbox, xwidth and ywidth information is only returned for the first 256 characters of the font.

This call is of doubtful use for fonts with up to 2 billion characters, so you should instead use Font_CharBBox, or Font_ScanString, instead of using R1 to R3 in this call.

To obtain kerning information, you can now pass a buffer in R6 - this will return kerning information in a different (but similar) format. Note that unlike the other data returned by this call, the kerning data is unscaled (in 1/1000ths em, rather than millipoints), and character codes are internal codes. You can use Font_EnumerateCharacters to map external codes to internal codes.

The kerning information is indexed by a hash table. The hash function used is:

(first letter) EOR (second letter ROR 4)

where the rotate happens in 8 bits.

Size / Description

256x4 – Description: hash table giving offset from table start of first kern pair for each possible value (0-255) of hash function

4 – Description: offset of end of all kern pairs from table start

4 – Description: flag word: bit 1 set => no x offsets bit 2 set => no y offsets bit 31 set => 'short' kern pairs

? – Description: kern pair data

Each kern pair consists of the internal code of the first letter of the kern pair, followed by the internal code of the second letter of the kern pair, followed by the x offset in 1/1000 em, followed by the y offset in 1/1000 em.

If bit 31 of the flag word is clear, then each kern pair is held in 2 words, each field being 16 bits:

word 0 / bits 0-15 / first character code

word 0 – bits 0-15: bits 15-31 · first character code: second character code

word 1 – bits 0-15: bits 0-15 · first character code: x offset

word 1 – bits 0-15: bits 16-31 · first character code: y offset

If bit 31 of the flag word is set, then each kern pair is held in 1 word. This can only happen if all kern pairs apply to the first 256 characters, and kerning is only in one direction:

bits 0-7 / first character code

bits 8-15 – first character code: second character code

bits 16-31 – first character code: x or y offset

3.3 Font_EnumerateCharacters (SWI &400A9)

This call can be used to determine which characters are present in a font, and which glyphs in the underlying font file characters map to.

On entry /

R0 – : Font handle (0 for current)

R1 – : Character code (0 to start enumeration)

On exit /

R0 – : Corrupted

R1 – : Next character (-1 if this character is the last)

R2 – : Internal character code of this character (-1 if unmapped)

Interrupts

Interrupt status is undefined Fast interrupts are enabled

Processor Mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

In practice

For example, for the font “\FHomerton.Medium\EUTF8”, a call sequence might be:

On entry / On exit / /

R1 = 0 · R1 = &20 (space) R2 = -1

..

R1 = &10F (d-caron) · R1 = &112 (E macron) · R2 = &151

R1 = &112 (E-macron) · R1 = &113 (e macron) · R2 = &185

R1 = &113 (e-macron) · R1 = &116 (E dot) · R2 = &195

..

R1 = &FB02 (fl ligature) · R1 = -1 · R2 = &FF

So, in this example, we see that the fl ligature character (Unicode FB02), is character &FF within the Homerton.Medium font file.

This call may be of use in conjunction with Font_ReadFontMetrics's kerning data, and for font manipulation by MakePSFont. Also, it would allow a character map program to scan through the UCS space to find defined characters.

Related SWIs

None

Related vectors

None

3.4 *Configure FontMax

As of version 3.42, the size of FontMax is set in units of 64K, rather than 4K. The same byte of CMOS is used, but it is now possible to configure FontMax up to a maximum of 16,320K. See *Configure FontMax.

4. Future enhancements

There are still a number of outstanding features that would be desirable:

- Glyph substitution, eg "A" + "combining acute" = "Aacute", or "f" + "i" = "fi ligature".
- Allowing more than 64K of scaffolding data.
- Glyph aliases (eg a glyph might match both "Dstroke" and "Eth").
- Compression.
- Dependencies outside the first 1024 characters.
- More scaffold lines per character (for CJK fonts)
- Displaying a symbol for a missing character.
- PostScript printing (the current PostScript printer driver can't handle the UTF8 encoding).

5. References

- Adobe, "Unicode and Glyph Names" version 1.1.
- <http://partners.adobe.com/supportservice/devrelations/typeforum/unicodegn.html> (Copy lodged in ROMFonts' Doc directory.)
- ISO/IEC 10646-1:1993, "Universal Multiple-Octet Coded Character Set (UCS)"
- The Unicode Standard, Version 2.1

[RISC OS Open source page](#)

UKPLOTV

Vectors

This vector is called when an VDU 25 plot command or OS_Plot SWI is issued with an unknown parameter.

Before calling

R0 Plot command code

On return

R0 Preserved

Notes

Whenever an unrecognised byte is found following 25 in the VDU stream, or OS_Plot is called with an unrecognised R0, then this vector is called. R0 contains this unknown plot command code - the current and 3 previous positions of the graphics cursor can be read with OS_ReadVduVariables from variables &8C - &93, allowing custom plot routines to be implemented. The default owner of this vector does nothing - unknown plot options are ignored.

[RISC OS Open source page](#)

UKSWIV

Vectors

This vector is called when an SWI with an unknown number is issued.

Before calling

R0-R8 Register parameters for SWI

R11 SWI number

Notes

When an SWI is called, the following steps are taken:

- Checks if the SWI number is < 512 – if so it calls the appropriate kernel SWI
- Checks whether any relocatable module defines that SWI via its SWI table
- If no module claims the SWI it is passed to this vector, where it may be claimed
- Any SWIs still unrecognised is passed to the unknown SWI handler, set up by OS_ChangeEnvironment.

[RISC OS Open source page](#)

UKVDU23V

Vectors

This vector is called when a VDU 23 command is issued with an unknown parameter.

Before calling

R0 VDU 23 option

R1 Pointer to VDU 23 queue

On return

R0 Preserved

R1 Preserved

Notes

Whenever an unrecognised byte (18-25,28-31) is found following 23 in the VDU stream this vector is called. R0 contains this unknown byte – it is also the first of the 9 bytes in the VDU queue, pointed at by R1.

The default owner of this vector does nothing – unknown VDU 23 options are ignored.

[RISC OS Open source page](#)

Undefined Instruction Handler

OS_ChangeEnvironment

The undefined instruction handler is entered in the following state:

In practice

- The processor will be in UND32 mode, with IRQs disabled
- FIQ state will match that at the time of the exception
- R14 & SPSR_und will contain the exception address and saved PSR, corresponding to when the exception was raised by the CPU. R14 will be pointing to the instruction after the instruction that caused the exception.
- The other core registers will be as per when the exception was raised by the CPU. However if your code examines the registers it must take care to use the correct register bank (e.g. if the exception was from user mode, it should look at R14_usr, not R14_und).
- R13 will be a valid stack pointer
- CP15 registers relating to exception processing will be in the correct state corresponding to that exception

If the handler is capable of handling the exception (e.g. by emulating the instruction) then it can resume execution by performing a suitable exception return operation.

Notes

On 26bit OS versions, the CPU will be in SVC26 mode, with R14 containing the combined LR+PSR.

The kernel's default handler will save a copy of the ARM integer registers (corresponding to the CPU mode the exception occurred in) to the Exception Registers Block, reset the stack pointers for all the privileged CPU modes, and then call OS_GenerateError to report an Undefined Instruction error.

[RISC OS Open source page](#)

UpCall Handler

OS_ChangeEnvironment

Called by the kernel to warn an applications of particular situations.

Before calling

R0 Reason code

R12 Workspace pointer

All other registers dependent on reason code

SVC mode, IRQ+FIQ state undefined

In practice

If the UpCall handler is called and R0 = UpCall_NewApplication (256), all handlers should be restored to their previous values and you should then return to the caller preserving all registers, letting the new application start.

[RISC OS Open source page](#)

UpCallV

Vectors

This vector is called when an Upcall is generated by the OS.

Before calling

R0 UpCall reason code

Notes

To quote the OS header files the 'can you deal with this before I go bang' vector.

[RISC OS Open source page](#)

URL_Fetcher

List of modules

The URL (Universal Resource Locator) module is a general purpose module for fetching data from various Internet services. The purpose of the module is to provide a uniform entry point into a set of "fetcher" protocols (e.g. FTP, HTTP, Gopher, NNTP, etc.), without the need for a client application to understand how that protocol works. This is done using a number of generalised URL SWIs. The fetcher protocols modules with which the URL module communicates, are called only by the URL module itself. The entry points into the protocol modules have similar names to the entry points into the URL module, but these are NOT the same, despite similarities.

In practice

Each client fetch occurs with in the context of a 'session'. Each session is identified by a different session identifier. Client session identifiers are issued by the URL module upon request and remain valid until the client informs the URL module to discard the session. Subsequently, session identifiers may be re-issued by the URL module for new sessions. Only a single object fetch can be performed in any one given session. Sessions cannot be re-used by clients, even if a prior object fetch in that session has completed.

The typical client usage of the system is:

- Obtain a session identifier (SWI URL_Register)
- Start fetching an object (SWI URL_GetURL)
- Repeatedly, whilst multi-tasking if in the desktop environment: - Read blocks of data (SWI URL_ReadData) - Process that data
- Discard session (SWI URL_Deregister)

If an application decides it requires a premature termination (e.g. the user asked the application to quit whilst an object was being downloaded), then the application calls SWI URL_Stop immediately and then discards the session with SWI URL_Deregister. Typical clients, such as web browsers, will, most likely, have several sessions active concurrently.

The URL module uses its own session identifiers that are passed in many of the SWI interfaces to the protocol modules which are not those known to the client application - the URL module maintains its own private sessions into the protocol modules. Service calls are also provided to ease interaction between the URL module and the fetchers, mainly to inform other modules of the arrival or departure of a particular module.

Each protocol module accepts data and returns results as per the HTTP protocol. Thus any extra client data associated with a request (passed in R4 to SWI URL_GetURL) will take the format of a (possibly empty) set of HTTP headers, an empty line and then the data; and each response will start with an HTTP/1.1 Response-Line of the format: "HTTP/1.1 200 OK" followed

by various headers identifying the content-type of the retrieved data, followed by an empty line, followed by the data itself.

Details

- SWI Calls
- Service Calls
- *Commands

Further information on how the URL_Fetcher module communicates with the protocol modules is found in the URL Fetcher API Specification. A start has been made in transferring this information to wiki pages. See URL_Fetcher Protocol module interface.

[RISC OS Open source page](#)

URL_Fetcher Protocol module interface

URL_Fetcher

The protocol module SWI interface is only called by the URL module. URL module clients should never call the ReadData/Status/GetData/Stop SWIs directly. The protocol modules are required to supply a SWI interface. There are currently 4 SWIs that need to be supported which run from SWI_base to SWI_base+3. New SWIs common to all protocol modules will only be added at the low end of the SWI range. Protocol modules must generate standard SWI not known error (error number &1E6) if they receive a call which they do not understand, so that the URL module can determine that they do not support the SWI. Note that there is no general requirement to use SWIs from offset 0 into a SWI chunk, although it makes sense to do this. Protocol modules which support multiple protocols should ensure that they do not place their internal "SWI bases" less than 16 SWIs apart to allow space to future expansion. e.g. AcornHTTP registers http: as &83F80 and https: as &83F90.

Protocol module interface

Protocol-specific SWIs should be added at the top end of the SWI chunk (ie start at SWI_base+63 and work down) – the AcornHTTP module uses that range to provide clients with access to its HTTP cookie management code, for example.

Note

The session identifiers used by the URL module to talk to the protocol modules are not the same identifiers used by clients to talk to the URL module. They are not interchangeable.

SWI interface

- Protocol_GetData
- Protocol_Status
- Protocol_ReadData
- Protocol_Stop

[RISC OS Open source page](#)

USBDriver

List of modules

The USBDriver module coordinates communication to peripherals connected via the Universal Serial Bus.

In practice

Its SWIs are intended for use by host controller drivers (such as OHCI Driver) and are of little use to applications or USB peripheral drivers which should use the DeviceFS interface instead.

USBDriver includes basic USB peripheral drivers for hubs, keyboards, and mice.

Details

- Service Calls
- SWI Calls
- *Commands

[RISC OS Open source page](#)

User_Drag_Box

Reason Code 7 · WindowManager

This code is returned from Wimp_Poll and indicates that all the mouse buttons are released to finish a user drag that was initiated by Wimp_DragBox.

On return

Block

+0 Dragbox minimum x

+4 Dragbox minimum y

+8 Dragbox maximum x

+12 Dragbox maximum y

Notes

This call only provides the final position of the dragbox. To find where the dragbox ended, Wimp_GetPointerInfo must be called.

[RISC OS Open source page](#)

User_Message

Reason Code 17 · WindowManager

This code is returned from Wimp_Poll and indicates that another task has broadcast a message to any of the following:

On return

Block

+0 Length of block
+4 Task handle of sender
+8 My ref (senders reference number for this message)
+12 0
+16 Message Code
+20 ... data...

In practice

- this specific task
- a specific window belonging to this task
- to all tasks

Notes

The message code determines the meaning of the message sent.

No acknowledgement of receiving this message is required by the message sender.

The number of message codes is very large, and not every code needs to be supported, however, every task must take action when receiving Message_Quit. This code is broadcast when a task is asked to quit via the Task Display window (via Task Manager).

A task that does not understand a message code should simply ignore it. Tasks should not respond to messages for no reason.

[RISC OS Open source page](#)

User_Message_Acknowledge

Reason Code 19 · WindowManager

This code is returned from Wimp_Poll and indicates that the message sent from this task was not acknowledged by any recipients.

On return

Block

+0 Length of block
+4 Task handle of sender
+8 My ref (senders reference number for this message)
+12 Your ref (senders reference number for a previous message)
+16 Message Code
+20 ... data...

Notes

A task that does not understand a message code should simply ignore it. Tasks should not respond to messages for no reason. When no tasks respond to a User_Message_Recorded message, the Wimp will send this message back to the task that originally sent the message.

The message code determines the meaning of the message sent.

[RISC OS Open source page](#)

User_Message_Recorded

Reason Code 18 · WindowManager

This code is returned from `Wimp_Poll` and indicates that another task has broadcast a message to any of the following:

On return

Block

+0 Length of block

+4 Task handle of sender

+8 My ref (senders reference number for this message)

+12 Your ref (senders reference number for a previous message, if this a reply, otherwise 0)

+16 Message Code

+20 ... data...

In practice

- this specific task
- a specific window belonging to this task
- to all tasks

Notes

The message code determines the meaning of the message sent.

Unlike `User_Messages`, tasks that send this type of message can accept an acknowledgement message that the message was received.

A task can acknowledge the message by calling `Wimp_SendMessage` with the event code of 19 (`User_Message_Acknowledge`). `Your_ref` should be set to the original code sent (`my_ref`). This will prevent the sender from receiving its original message back from the Wimp with event code 19.

Another method of acknowledging a message is to send a reply message using event code `User_Messages` or `User_Message_Acknowledge` again with `your_ref` parameter set to the original message's `my_ref`.

Both types of acknowledgement must take place before the next call to `Wimp_Poll`.

A task that does not understand a message code should simply ignore it. Tasks should not respond to messages for no reason.

[RISC OS Open source page](#)

Using PCI Devices

PCI

In RISC OS, PCI devices can be identified in one of two ways. Users generally need only be aware of the PCI system when expansion cards are added to the machine, and will use the PCI slot number, in commands like `*Unplug` or `*PCIDevices`. Slot numbers are fixed for a given system.

PCI Device Identification

Device drivers use a PCI handle – each function of each PCI device in the system is given a unique, permanent non-zero handle when it is discovered. Handles remain constant as long as the device remains present in the system, but can change when the system is rebooted, or if a card is removed and reinserted (if the system supports hot-plugging). Handles do not

change if the PCI Manager is reloaded. Handles are assigned from 1 upwards, so can be searched linearly, much like Expansion Cards.

PCI Interrupts

Since PCI interrupt lines are shared by multiple devices, the interrupt handling mechanism is a little different from that used by traditional expansion cards. To find the device vector number that will be called for a particular card, use `PCI_ReadInfo` with bit 11 set in the `R0` bitmask. To install a device driver, use `OS_ClaimDeviceVector` and pass in this vector number with bit 31 set to indicate that the call may be passed on to other vector claimants. You must also ensure that interrupts for this vector are enabled by calling the HAL routine `HAL_IRQEnable` (HAL entry number 1) with this vector number as the parameter. Note however that when you remove a device driver, you should not disable interrupts for this vector.

When your device driver routine is called, check whether your card is interrupting. If it is, perform the necessary interrupt processing and claim the vector by pulling the return address from the stack. If it is not, pass on the call by returning to the address in `R14`. If you are writing a device driver in C, use the CMHG 'vector-handlers' veneer and return 0 to claim the call or 1 to pass it on.

When the interrupt processing code needs to clear the interrupt on the PCI card, there are two things that must be done before interrupts are re-enabled or you return from the handler. Firstly, if clearing the interrupt involves writing to a register on the PCI card, you must ensure that the write has been flushed out to the card. This can be achieved by simply performing a read operation from a safe area or register on the card. Secondly, you should then call the HAL routine `HAL_IRQClear` (HAL entry number 3) with the vector number as the parameter. Note that this is not actually required for the lyonix hardware, but should be included for compatibility with possible future hardware platforms.

If no handler claims a vector for a particular interrupt, the kernel will disable that interrupt line. For correct operation of the system it is vital that vector claiming and interrupt clearing are performed correctly.

[RISC OS Open source page](#)

VDU 17

VDU Codes

Defines text foreground and background colours. Identical in use to BASIC statement `COLOUR`.

Syntax

`VDU 17,col`

Parameters

`col` · Colour

Notes

Colour reference is the logical colour, which may differ to the actual colour.

[RISC OS Open source page](#)

VDU 18

VDU Codes

Defines graphics foreground and background colours. Identical in use to BASIC statement GCOL.

Syntax

VDU 18,plotmode,col

Parameters

plotmode / 0|1|2|3|4

col – 0|1|2|3|4: Colour

Notes

Plotmode specifies how the colour is to be placed on the screen, as tabulated below.

Colour reference is the logical colour, which may differ to the actual colour.

Plotmode / Action

- 0 – Action: Plot the colour specified
- 1 – Action: OR the specified colour with that already there
- 2 – Action: AND the specified colour with that already there
- 3 – Action: EOR (Exclusive-OR) the specified colour with that already there
- 4 – Action: Invert the colour already there

[RISC OS Open source page](#)

VDU 19

VDU Codes

Selects the actual colour that is to be displayed for each logical colour.

Syntax

VDU 19,logical colour,mode,red,green,blue

Parameters

logical colour / Colour to set

mode – Colour to set: how to set the colour

red, green, blue – Colour to set: physical colour information

In practice

VDU 19 defines the colour palette relationship. It causes a specified logical colour for either the screen, border or pointer to be represented by a given physical colour.

The action depends on the value of 'mode' as follows:

mode = 0 - 15 · logical colour = physical colour specified by mode parameter. red, green and blue are ignored, and should be zero

mode = 16 · both flash palettes for logical colour = red units red, green units green, blue units blue

mode = 17 · first flash palette for logical colour = red units red, green units green, blue units blue

mode = 18 · second flash palette for logical colour = red units red, green units green, blue units blue

mode = 20-23 · text fg, bg, graphics fg, bg colour change [Cerilica]

mode = 24 · border colour = red units red, green units green, blue units blue; logical colour is not used, and should be zero

mode = 25 · logical colour (1 - 3) of pointer = red units red, green units green, blue units blue

mode = 31 · Teletext Level 3.5 palette, colour 0-31 [Cerilica]

If you add 128 to the 'mode' value, you also set the 'supremacy' bit of the appropriate palette entry. This is used when the computers' video is mixed with an external video source, to provide a superimposed image.

Notes

Use of semi-colon punctuation in BASIC sends number as two-byte pair: lo followed by hi.

[RISC OS Open source page](#)

VDU 23

VDU Codes

Multi-purpose command:

VDU 23,0 – Function: Interlace, cursor appearance and extensions

VDU 23,1 – Function: Cursor appearance

VDU 23,2-5 – Function: ECF pattern definition

VDU 23,6 – Function: Dashed line pattern

VDU 23,7 – Function: Text window scrolling

VDU 23,8 – Function: Text area clearing

VDU 23,9-10 – Function: Flashing colour timing

VDU 23,11 – Function: Default ECF patterns

VDU 23,12-15 – Function: Simplified ECF pattern definition

VDU 23,16 – Function: Cursor movement

VDU 23,17 – Function: Miscellaneous

VDU 23,18 – Function: Teletext emulation control

VDU 23,23 – Function: Direct UDG output [Cerilica]

VDU 23,25 – Function: FontManager colours & thresholds

VDU 23,26 – Function: FontManager font selection

VDU 23,27 – Function: System sprite operations

VDU 23,28 – Function: Extended coordinate text window [Cerilica]

VDU 23-255 – Function: Define User Defined Graphic

In practice

Other actions are passed to the UKVDU23V vector.

Syntax

VDU 23,action,...8 bytes...

Parameters

Refer to specific uses linked above.

Notes

Some functions are provided by third party modules. The following are further subclassed: VDU 23,0, VDU 23,1, VDU 23,17, VDU 23,18, VDU 23,27.

[RISC OS Open source page](#)

VDU 23,16

VDU Codes

Controls the movement of the cursor after displaying a character.

Syntax

VDU 23,16,eor,and,eor2,and2,..4 zeros..

Parameters

- – Modifies the CursorFlags, which have the following effects:: b0 = Scroll protection ("column 81" mode)
- – Modifies the CursorFlags, which have the following effects:: b1 = Positive X is left (else right)
- – Modifies the CursorFlags, which have the following effects:: b2 = Positive Y is up (else down)
- – Modifies the CursorFlags, which have the following effects:: b3 = X is vertical (else horizontal)
- – Modifies the CursorFlags, which have the following effects:: b4 = Y wraps (else scrolls)
- – Modifies the CursorFlags, which have the following effects:: b5 = Cursor doesn't move (else X+=1)
- – Modifies the CursorFlags, which have the following effects:: b6 = VDU 5 cursor doesn't increment Y when X gets too large
- – Modifies the CursorFlags, which have the following effects:: b7 = reserved

eor2, and2 – Modifies the CursorFlags, which have the following effects:: 3rd Party per-direction flags [Cerilica]

Notes

The current state can be read using OS_Word 10,11 (or by the official API of OS_Byte 160 via 3rd party support).

[RISC OS Open source page](#)

VDU 24

VDU Codes

Enables definition of graphics window, i.e. area of screen inside which graphics can be drawn with the DRAW and PLOT statements. Graphics (including sprites) are cropped to appear only within the graphics window.

Syntax

VDU 24,lx;by;rx;ty;

Parameters

lx / Left X (inclusive)

by – Left X (inclusive): Bottom Y (inclusive)

rx – Left X (inclusive): Right X (inclusive)

ty – Left X (inclusive): Top Y (inclusive)

Notes

Four coordinates given as two bytes each, relative to the graphics origin. Use of semi-colon punctuation sends number as two-byte pair: lo followed by hi.

If the graphics window has a negative width or height, it will not crop the graphics, instead the whole screen will be accessible to plotting operations. Care should be taken if this is not the desired result.

[RISC OS Open source page](#)

VDU 25

VDU Codes

Identical in use to BASIC statement PLOT.

Syntax

VDU 25,k,x;y;

Parameters

k / Plotmode

x – Plotmode: x-coordinate

y – Plotmode: y-coordinate

Notes

Use of semi-colon punctuation sends number as two-byte pair: lo followed by hi.

Plotmode specifies what kind of point, line or triangle is to be drawn, as tabulated below.

0 – Action: Move relative to last point

1 – Action: Draw line relative in the current graphics foreground colour

2 – Action: Draw line relative in the logical inverse colour

3 – Action: Draw line relative in the current graphics background colour

4 – Action: Move to absolute position

5 – Action: Draw line absolute in the current graphics foreground colour

6 – Action: Draw line absolute in the logical inverse colour

7 – Action: Draw line absolute in the current graphics background colour

8 - 15 – Action: As 0 - 7 but with the last point in the line omitted

16 - 23 – Action: As 0 - 7 but with a dotted line

24 - 31 – Action: As 0 - 7 but with a dotted line and without the last point on the line

32 - 63 – Action: Reserved

64 - 71 – Action: As 0 - 7 but only a single point is plotted

72 - 79 – Action: As 0 - 7 but to draw a horizontal line to the left and right of the point until a colour other than the current background colour is reached

80 - 87 – Action: As 0 - 7 but plot and fill a triangle

[RISC OS Open source page](#)

VDU 28

VDU Codes

Enables definition of text window, i.e. area of screen to which future text is written (rather than being written anywhere on the screen).

Syntax

VDU 28,lx;by;rx;ty;

Parameters

lx / Left X

by – Left X: Bottom Y

rx – Left X: Right X

ty – Left X: Top Y

Notes

Four coordinates are given as character positions, with the origin being 0,0 at the top left of the screen. Maximum values depend on the current screen mode.

[RISC OS Open source page](#)

VDU 29

VDU Codes

Defines graphics origin.

Syntax

VDU 29,x;y;

Parameters

x / x-coordinate

y – x-coordinate: y-coordinate

Notes

Use of semi-colon punctuation sends number as two-byte pair: lo followed by hi.

[RISC OS Open source page](#)

VDU 31

VDU Codes

Moves text cursor to new character position.

Syntax

VDU 31,x,y

Parameters

x / x-coordinate

y – x-coordinate: y-coordinate

Notes

Coordinates are given as character positions, with the origin being 0,0 at the top left of the screen. Maximum values depend on the current screen mode.

[RISC OS Open source page](#)

VDU Codes

A table below lists the VDU codes and their meanings. Some VDU codes require parameters sending as extra bytes; e.g. VDU 22 (select screen mode) requires the screen mode passing as a parameter (one extra byte). The number of extra bytes required is also listed.

· # Hex

0 · &00 · @ · 0 · Does nothing (flushes multicode sequences)

1 · &01 · A · 1 · Sends next character to printer only

2 · &02 · B · 0 · Enables printer

3 · &03 · C · 0 · Disables printer

4 · &04 · D · 0 · Writes text at text cursor

5 · &05 · E · 0 · Writes text at graphics cursor

6 · &06 · F · 0 · Enables VDU driver

7 · &07 · G · 0 · Generates bell sound

8 · &08 · H · 0 · Moves cursor back one character

9 · &09 · I · 0 · Moves cursor forward one character

10 · &0A · J · 0 · Moves cursor down one line

11 · &0B · K · 0 · Moves cursor up one line

12 · &0C · L · 0 · Clears text window/form feed

13 · &0D · M · 0 · Carriage return (moves cursor to start of subsequent line)

14 · &0E · N · 0 · Turns on paged mode

15 · &0F · O · 0 · Turns off paged mode

16 · &10 · P · 0 · Clears graphics window

17 · &11 · Q · 1 · Defines text colour

18 · &12 · R · 2 · Defines graphics colour

19 · &13 · S · 5 · Defines logical colour

20 · &14 · T · 0 · Restores default logical colours

21 · &15 · 0 · Disables VDU drivers

22 · &16 · V · 1 · Selects screen mode

VDU 23 · &17 · W · 9 · Multi-purpose command:

23,0 · Sets the interlace and controls cursor appearance

23,1 · Controls text cursor appearance

23,2-5 · Defines ECF pattern and colours

23,6 · Sets dot-dash line style

23,7 · Scrolls text window or screen

23,8 · Clears a block of the text window

23,9 · Sets first flash time

23,10 · Sets second flash time

23,11 · Sets default patterns

23,12-15 · Defines simple ECF patterns and colours

23,16 · Controls cursor movement after printing

23,17,0-3 · Sets the tint for a colour

23,17,4 · Chooses ECF patterns

23,17,5 · Exchanges text foreground and background colours

23,17,6 · Sets ECF origin

23,17,7 · Sets character size/spacing

23,18-24 · Reserved for future expansion

23,25-26 · Private Font Manager calls

23,27 · Private Sprite Manager calls

23,28-31 · Reserved for use by application programs

23,32-255 · Redefines printable characters
 24 · &18 · X · 8 · Defines graphics window
 25 · &19 · Y · 5 · PLOT command
 26 · &1A · Z · 0 · Restores default windows
 27 · &1B · [· 0 · Does nothing (cancels multicode sequences)
 28 · &1C · \ · 4 · Defines text window
 29 · &1D ·] · 4 · Defines graphics origin
 30 · &1E · ^ · 0 · Homes text cursor
 31 · &1F · _ · 2 · Moves text cursor
 127 · &7F · 0 · Delete

Notes

"Multicode sequence" support such as UTF-8 and Shift-JIS is provided by 3rd party modules.

[RISC OS Open source page](#)

VDU Variables

Unlike the Mode Variables, many of the VDU variables represent changeable properties of the video system.

0-127 · &0-&7F – Meaning: See Mode Variable Numbers
128 · &80 – Name: GWLCol · Meaning: Graphics Window – Lefthand Column · Co-ordinates 4: (ic)
129 · &81 – Name: GWBRow · Meaning: Graphics Window – Bottom Row · Co-ordinates 4: (ic)
130 · &82 – Name: GWRCol · Meaning: Graphics Window – Righthand Column · Co-ordinates 4: (ic)
131 · &83 – Name: GWTRow · Meaning: Graphics Window – Top Row · Co-ordinates 4: (ic)
132 · &84 – Name: TWLCol · Meaning: Text Window – Lefthand Column
133 · &85 – Name: TWBRow · Meaning: Text Window – Bottom Row
134 · &86 – Name: TWRCol · Meaning: Text Window – Righthand Column
135 · &87 – Name: TWTRow · Meaning: Text Window – Top Row
136 · &88 – Name: OrgX · Meaning: X coord of graphics Origin · Co-ordinates 4: (ec)
137 · &89 – Name: OrgY · Meaning: Y coord of graphics Origin · Co-ordinates 4: (ec)
138 · &8A – Name: GCsX · Meaning: Graphics Cursor X coord · Co-ordinates 4: (ec)
139 · &8B – Name: GCsY · Meaning: Graphics Cursor Y coord · Co-ordinates 4: (ec)
140 · &8C – Name: OlderCsX · Meaning: Oldest gr. Cursor X coord · Co-ordinates 4: (ic)
141 · &8D – Name: OlderCsY · Meaning: Oldest gr. Cursor Y coord · Co-ordinates 4: (ic)
142 · &8E – Name: OldCsX · Meaning: Previous gr. Cursor X coord · Co-ordinates 4: (ic)
143 · &8F – Name: OldCsY · Meaning: Previous gr. Cursor Y coord · Co-ordinates 4: (ic)
144 · &90 – Name: GCsIX · Meaning: Graphics Cursor X coord · Co-ordinates 4: (ic)
145 · &91 – Name: GCsIY · Meaning: Graphics Cursor Y coord · Co-ordinates 4: (ic)
146 · &92 – Name: NewPtX · Meaning: New point X coord · Co-ordinates 4: (ic)
147 · &93 – Name: NewPtY · Meaning: New point Y coord · Co-ordinates 4: (ic)
148 · &94 – Name: ScreenStart · Meaning: As used by VDU drivers
149 · &95 – Name: DisplayStart · Meaning: As used by display hardware
150 · &96 – Name: TotalScreenSize · Meaning: Memory allocated to screen
151 · &97 – Name: GPLFMD · Meaning: GCOL action for foreground col
152 · &98 – Name: GPLBMD · Meaning: GCOL action for background col
153 · &99 – Name: GFCOL · Meaning: Graphics foreground col
154 · &9A – Name: GBCOL · Meaning: Graphics background col
155 · &9B – Name: TForeCol · Meaning: Text foreground col
156 · &9C – Name: TBackCol · Meaning: Text background col

- 157 · &9D – Name: GFTint · Meaning: Graphics foreground tint
- 158 · &9E – Name: GBTint · Meaning: Graphics background tint
- 159 · &9F – Name: TFTint · Meaning: Text foreground tint
- 160 · &A0 – Name: TBTint · Meaning: Text background tint
- 161 · &A1 – Name: MaxMode · Meaning: Highest built-in numbered mode known to kernel
- 162 · &A2 – Name: GCharSizeX · Meaning: X size of VDU5 chars (pixels)
- 163 · &A3 – Name: GCharSizeY · Meaning: Y size of VDU5 chars (pixels)
- 164 · &A4 – Name: GCharSpaceX · Meaning: X spacing of VDU5 chars (pixels)
- 165 · &A5 – Name: GCharSpaceY · Meaning: Y spacing of VDU5 chars (pixels)
- 166 · &A6 – Name: HLineAddr · Meaning: Address of horizontal line-draw routine
- 167 · &A7 – Name: TCharSizeX · Meaning: X size of VDU4 chars (pixels)
- 168 · &A8 – Name: TCharSizeY · Meaning: Y size of VDU4 chars (pixels)
- 169 · &A9 – Name: TCharSpaceX · Meaning: X spacing of VDU4 chars (pixels)
- 170 · &AA – Name: TCharSpaceY · Meaning: Y spacing of VDU4 chars (pixels)
- 171 · &AB – Name: GcolOraEorAddr · Meaning: Addr of colour blocks for current GCOLs
- 172 · &AC – Name: VIDCClockSpeed · Meaning: VIDC clock speed in kHz³
- 174 · &AE – Meaning: Left border size¹
- 175 · &AF – Meaning: Bottom border size¹
- 176 · &B0 – Meaning: Right border size¹
- 177 · &B1 – Meaning: Top border size¹
- 192 · &C0 – Meaning: Current GraphicsV driver number²
- 256 · &100 – Name: WindowWidth · Meaning: Width of text window in chars
- 257 · &101 – Name: WindowHeight · Meaning: Height of text window in chars

Notes

1 Available from RISC OS 5.07 onwards

2 Available from RISC OS 5.21 onwards. Also available in RISC OS Select, for the current VideoV driver number.

3 Introduced in RISC OS 3, removed in RISC OS 5

4 (ic) indicates values which are stored in internal coordinates: 0,0 is the top-left pixel of the screen, with positive X pointing right and positive Y pointing down.

(ec) indicates values which are stored in external coordinates, i.e. OS units. Positive X points right, and positive Y points up. Most external coordinates are relative to the graphics origin, except the graphics origin itself (variables 136 and 137), which is a coordinate relative to the bottom-left pixel of the screen. The XEig and YEig mode variables can be used to apply the scale conversion between OS units and pixels.

[RISC OS Open source page](#)

VDUXV

Vectors

This vector is called when VDU output has been redirected by OS_WriteC destination flag (setting bit 5).

Before calling

R0 Byte send to VDU

On return

R0 Preserved

C flag set if byte is to be sent to printer

Notes

If bit 5 of the output stream flag (see OS_Byte 3) has been set then characters which would have been sent to the display are routed to this vector instead. Characters sent to other output streams (e.g. *Spool file, printer) are unaffected.

If the C flag is set on return, then the character (unless it is the printer ignore character (see OS_Byte 6)) is sent to the VDU printer stream (if bits 2 and 6 are clear in the output stream flag). This is independent of the setting of bit 3 in the output stream flag.

The PRM is not clear as to whether this can send the byte twice if bit 3 is set (printer enabled). Confused? :-)

[RISC OS Open source page](#)

VFP Support

List of modules

The VFP Support module manages the VFP/NEON coprocessor hardware. Most of the SWIs provided by the module focus around the creation and use of VFP contexts. Each program which wishes to use VFP/NEON must create at least one context. The primary use of the context is as a buffer to save the hardware registers and state when another program becomes active, for example when a Wimp task calls Wimp_Poll and another task is paged in. If a program attempts to execute a VFP/NEON instruction without having a context active then the instruction will abort.

Language support

Note that depending on the programming language and support libraries your program uses, context creation and management may or may not be handled for you:

- For pure assembler programs and BASIC programs utilising assembler, you must create and manage contexts manually
- For C or C++ programs created using versions of GCC which have full VFP/NEON support, context creation and management is handled by the C runtime
- For C/C++ programs created by compilers which don't provide full VFP/NEON support (e.g. current versions of the ROOL C/C++ tools), context creation and management must be performed manually

For other programming languages, consult the relevant documentation supplied with the compiler/interpreter.

Usage notes

The VFP Support module allows the use of VFP/NEON contexts from user-mode applications, module code running in supervisor mode, and interrupt handlers.

The Wimp is fully aware of VFP contexts and will switch them automatically when switching tasks; when a task is resumed the context it was previously using will be reactivated. However for other situations (single-tasking programs started from the command line, SWI calls, interrupt handlers, etc.) no automatic context switching occurs. For these situations it is your responsibility (or the responsibility of e.g. the C runtime library) to ensure the previously active context is restored on exit from your code.

Note: Wimp versions prior to 5.60 contained a bug that prevented VFP contexts from being handled correctly if certain types of Wimp_Poll post-filters were used. Therefore it's recommended that Wimp-based VFP applications ensure Wimp 5.60 or later (RISC OS 5.24). If supporting older Wimps is necessary, the VFP context must be deactivated and reactivated manually around Wimp_Poll calls.

Details

- SWI Calls

[RISC OS Open source page](#)

VideoOverlay

List of modules

The VideoOverlay module acts as a manager for any graphics/video overlays that are supported by the system, e.g. for directly displaying YUV video frames. Its SWIs act as a high-level interface which applications can use to create and manipulate overlays.

In practice

Internally, it uses the GraphicsV API to manage the overlays (e.g. reason codes 20-28). The GraphicsV overlay API is a low-level interface that should not be used directly by programs, as any overlay operations which are performed via GraphicsV may conflict with operations that VideoOverlay (or a replacement manager module) may be performing.

The current version of the module is functional, but incomplete. Once completed, its full set of duties are expected to include:

- Rendering (or coordinating the rendering) of alpha/TCK sections to allow Z-order overlays to be visible
- Associating overlays with Wimp windows/tasks Allows Z-order to be automatically updated Ensures overlay is destroyed if task exits or window is destroyed
- Notifying programs of when their overlays have been lost
- The public API doesn't use GraphicsV overlay IDs directly; instead a different ID system is used, to prevent a program which has lost its overlay due to a mode change from accidentally interacting with an overlay that belongs to a different program
- Dealing with some of the complexities of the "Basic" overlay type
- Software emulation of overlays
- Allowing screen output to be redirected to an overlay

With the overlay ID system that VideoOverlay uses, zero is a suitable value for programs to use as an invalid overlay ID.

Since generic hardware overlays aren't a critical system component, the VideoOverlay module is typically disc-based rather than ROM-based. Thus, programs which expect to use it should attempt to RMLoad it from !System, in the usual manner.

Details

- SWI Calls

[RISC OS Open source page](#)

Wimp_TextOp 0

Reason Code 0 · WindowManager

With a reason code of 0, this call sets the text colours that will be used when plotting a font using this call with reason code 2.

Before calling

R0 0

R1 Foreground colour

R2 Background colour

On return

R0 Corrupt

R1 Preserved

R2 Preserved

Notes

The format used when setting colours is &BBGGRR00.

[RISC OS Open source page](#)

Wimp_TextOp 1

Reason Code 1 · WindowManager

With a reason code of 1, this call returns the width of a specified string using the current desktop font.

Before calling

R0 1

+ b31 to return coverage [4.3+, not 5]

R1 Pointer to ctrl-terminated string

R2 Number of Characters to scan, or

0 for whole string

On return

R0 String width in OS units

R1 Preserved

R2 Preserved

Notes

This is often used prior to plotting a string to an icon, in case the icon needs to be resized.

The value returned is in OS units.

In RO4/6 b31 requests coverage: the maximum of advance width and the bounding box.

[RISC OS Open source page](#)

Wimp_TextOp 2

Reason Code 2 · WindowManager

With a reason code of 2, this call plots text on screen using the current desktop font.

Before calling

R0 2 + bits (See below)

Bits

28 Vertically centre at the Y co-ordinate

29 Co-ords are relative to the window being redrawn

30 Vertical adjust, so baseline of characters is at same Y co-ordinate as system font

31 Right justify (as per R4, R5)

R1 Pointer to null-terminated string

R2 -1 ·

R3 -1

R4 X co-ordinate

R5 Y co-ordinate

On return

R0 String width in OS units

R1 Preserved

R2 Preserved

Notes

Bits 30-28 are used to specify the justification of the text.

This call should be used from within a redraw loop.

Because an outline font may be used, the background colour of the font must be specified, so that the anti-aliasing colours can be set-up.

BUG: The PRMs incorrectly describe the string as "ctrl-terminated". This is not true when using System Font, so the string must always be null-terminated.

[RISC OS Open source page](#)

Wimp_TextOp 3

Reason Code 3 · WindowManager

With a reason code of 3, this call calculates how many characters of a string can be displayed with a specific width specified in OS units.

Before calling

R0 3

R1 Pointer to ctrl-terminated string

R2 Width in OS units

R3 Split character

On return

R0 Pointer to split point

R1 Preserved

R2 Preserved

Notes

This is very useful when applications implement their own font display routines, so that fonts can be wrapped correctly on screen.

[RISC OS Open source page](#)

Wimp_TextOp 4

Reason Code 4 · WindowManager

With a reason code of 4, this call fills the buffer with the string dependant on the OS width specified.

Before calling

R0 4

R1 Pointer to ctrl-terminated string

R2 Pointer to output buffer

R3 buffer size, or

0 to find required size

R4 Maximum width in OS units

On return

R0 Required buffer size

R1 Preserved

R2 Preserved

R3 Preserved

Notes

The buffer is filled with the string, and dependant on the width, may be truncated but with a '...' suffix appended.

Depending on the alphabet and Wimp font, either an ellipsis character or three full-stops will be used.

BUG: RO4/6 fail to allow for the width of WimpSymbol characters, causing incorrect truncation.

[RISC OS Open source page](#)

Window

List of modules

The Window module provides the Window class and many built-in gadget types.

Details

- Events?
- Gadgets?
- Methods?
- Service Calls?
- SWI Calls
- Technical Details?

[RISC OS Open source page](#)

WindowManager

List of modules

The Window Manager module provides two very important components of the OS:

In practice

- A cooperative multitasking execution environment to allow multiple user-facing programs to coexist
- A windowing system to allow the user to interact with the running programs

The Window Manager module and the environment it provides are commonly referred to as the Wimp. The user interface that the Wimp provides is also known as the RISC OS Desktop. Programs that run under the control of the Wimp can also be referred to as Applications or Tasks. Within the Window Manager section, the words Program, Application and Task should be treated as synonyms.

In order to ensure a consistent user experience, the RISC OS Style Guide should be consulted for information on how Wimp tasks should look and feel, and how they should interact with the user and each other.

Details

- Overview
- Technical Details
- Service Calls
- SWI Calls
- Messages
- *Commands

[RISC OS Open source page](#)

WindowManager Application Resource Files

WindowManager

RISC OS stores applications in a manner different from that of most other operating systems. Instead of having application files and resource files spread across multiple folders, all application related files are stored within the application's own 'folder'. This is called the 'Application Directory'. All information relating to how an application functions is stored within the application directory.

In practice

Applications are readily identified by their '!' suffix; often pronounced as a 'Pling'. e.g !Edit, !Draw, !Boot.

Applications are loaded by double-clicking the application's icon, but double-clicking with the shift key depressed opens the application directory, thereby showing all the files that make up the application.

Applications are made up of several special files; the following table describes them.

!Boot – File Type: Obey (&FEB) · Description: This file is automatically executed when the application is first seen by the Filer. It is typically used to set up the sprite to be used for the application's icon and to name special file-types.

!Run – File Type: Obey (&FEB) or TaskObey (&FD7) · Description: This file is executed when the application directory's icon is double-clicked. Typically, it will check the existence of the resources needed by the application before starting it up.

!RunImage – File Type: Varies · Description: This is a traditional name of the application's main code.

!Sprites – File Type: Sprite (&FF9) · Description: This file is used to store sprites used by the application.

!Choices – File Type: Varies · Description: This file used be used to store application's settings. Nowadays a separate file in !Boot.Choices is used, so that choices are retained between upgrades.

!Help – File Type: Varies · Description: This file is `filer_run` when the user selects the Help option in the Filer Submenu for the application directory.

!Messages – File Type: Text (&FFF) · Description: This file is used to store the text used for windows, dialog boxes and menus

!Templates – File Type: Template (&FEC) · Description: This file is used to store window definitions

The names `!Boot`, `!Run`, `!Sprites` and `!Help` are fixed. The other names are merely traditional, and can be whatever the user chooses.

!Boot File

`!Boot` files are automatically executed by the Wimp when the application is initially seen by the Filer.

An example of a `!Boot` file for a typical application called `!Foo` is shown below.

```
Set Foo$Dir <Obey$Dir>
IconSprites <Foo$Dir>.!Sprites
Set Alias$RunType_f00 Run <Foo$Dir>.!Run %*0
Set File$Type_f00 FOO
...
```

This example code sets a system variable, in line 1, holding the application directory's pathname and loads the sprite file into the system spritearea in line 2. In line 3 it defines the run-action for a file of type `&f00`: that its pathname is passed as a command-line argument to the application. In line 4 the textual designation for filetype `&f00` is defined.

The setting of the run-action for `FOO` files, in line 3, will have the effect that when a file of type `FOO` is double-clicked, if no running task responds to the Wimp's message that a file of this type has been double-clicked, then the file will be passed to a new invocation of the application.

Annotations `fn1`. The Wimp will add a suffix to the sprite file depending on the screen mode in use at the time.

!Run File

The `!Run` file is executed when the user double-clicks the application's icon, or when a command to run the application is executed. The `!Run` file should ensure that the resources which the application needs are available before issuing the command to start it running.

An example of the `!Run` file for our mythical `Foo` application is shown below.

```
| Has the application been booted yet?
If "<Foo$Dir>" = "" then run <Obey$Dir>.!Boot
| Now we can use Foo$Dir safely.
| If no commandline arguments display help.
if "%*0" = "" then filer_run <Foo$Dir>.!Help
| Required versions of modules to load application
RMEnsure FPEmulator 4.03 RMLoad System:Modules.FPEmulator
RMEnsure FPEmulator 4.03 Error You need FPEmulator 4.03 or Later
| Memory Management
WimpSlot -min 256K -max 512K
| Run the application code
Run <Foo$Dir>.!RunImage %*0
```

The `!Boot` file copied the value of `Obey$Dir` into `Foo$Dir`. Other `Obey` files could change the setting of `Obey$Dir` so it is safer to make a copy and use that instead. This new reference to the application's path can then be safely used within the application itself.

In the `!Run` file the application checks to see if it has been booted, and boots itself if it has not. Then it checks to see if any arguments have been passed to it. In this case it displays its Help service if none have.

The application then checks that any required modules are present and loaded by using `*RMEnsure`. The first use of `RMEnsure` checks to see if the right version of the module is loaded, and loads it if not. The second issues an error message if it is still not loaded. Some applications which need to ensure the presence of many modules use a dedicated `Obey` file `!Modules` to deal with this, which is called from the `!Run` file.

The Application then tries to assign a dedicated amount of memory for the application using `*WimpSlot`. More memory can always be set aside if required by the application if required.

Finally, it then tries to execute the `!!RunImage` file. Only one `%` sign is required this time, and there must be no CR/LF after this line, else the file will remain open until the application is quit.

Notes:

- `*WimpSlot` must be called after loading modules. This is because loading modules requires memory too, so should be loaded first.
- If shared system resources are required, then the application should check the `System$Path` is defined and produce an error if it is not. e.g.

```
*If "<System$Path>" = "" Then Error 0 System resources is not found
```

!RunImage File

This file is conventionally used to store the main code of the application. The `!Run` file calls this file after it has checked that the system can load the application successfully.

The date stamp on this file is also used as the date stamp of the application, if the Filer is displaying in 'Full Info' mode.

!Sprites File

This file is used to store several important sprites used by the application. In actual use, the Wimp may use a different file depending on the screen mode in use. See Icon Sets for more details.

The Wimp can assign a different sprite to any type of file type. Many common file types have already pre-assigned sprites. If an application handles a file type that is not already handled natively by RISC OS, then it should specify a sprite that represents that particular file type.

The following sprites should be available within the relevant `!Sprites` file.

Name / Description

!foo – Description: To represent the application itself

sm!foo – Description: To represent the application when filer is set to display small icons

file_f00 – Description: To represent the filetype within Filer windows

small_f00 – Description: To represent the filetype within Filer windows when the Filer is set to display small icons

!Choices File

Applications that allow for the Applications settings to be changed by the user used to store the settings in the `!Choices` file. The data format used to store the settings can range from a simple text file to a proprietary data format. Because this makes it easy for the settings to be overwritten when the application is updated, nowadays settings are kept separately in `!Boot.Choices.Foo`.

!Help File

The `!Help` file is typically text file (filetype `&FFF`), or an HTML file (filetype `&FAF`) that is used to convey the purpose of the application and any other important information. In principle any displayable document type could be used. Alternatively it is an Obey file that checks which document types are available and issues appropriate commands for displaying the information.

If the `!Help` file is present it will appear as a sub-menu option, if the user clicks the Menu button over the applications icon within the filer window.

!Messages File

This is a file that is often used to store text to be used by an application. i.e. windows, dialog boxes and menus. This enables a developer to quickly customize the application for different languages by supplying different message files for each language supported.

Resource files, such as message files should be loaded from removable storage at the same time as the application. This ensures that the application does not have to request access later when the path may no longer be accessible. e.g. application was loaded from a CD, that is no longer present.

Template Files

Template files are used to store window definitions that the application can load and display. For a full description please see Template Files.

RISC OS has many guidelines on the appropriate use of resource files. For a complete detailed guide on resource files, please see the RISC OS Style Guide.

[RISC OS Open source page](#)

WindowManager Applications And Documents

WindowManager

In addition to loading, editing and saving documents, applications must also know how to load and quit themselves. This page details how they should load and quit and successfully integrate document handling.

In practice

The term 'Document' is used in this part to denote files that an application can handle; e.g. text file for a text editor

Loading an Application

There are two standard ways for an application to be loaded by a user.

/ Methods

1 – Methods: Double-click on the application's icon within a filer window. Upon loading, the application icon should be displayed on the icon bar, and no application windows should be displayed.

2 – Methods: Double-click an appropriate document icon using either the Select or Adjust mouse buttons. Upon loading, the application icon should appear on the icon bar, and a window should appear with the file contents being displayed.

Icons that appear on the icon bar should have a width and height of 68 OS units square. More information on this topic is available in the Icon Bar page.

Application icons should always respond to the menu mouse button by opening a menu.

Quitting an Application

Applications must have a 'Quit' option available on the icon bar menu. To adhere to RISC OS guidelines, it should also be the last menu item.

Applications should also respond to the Message_PreQuit event code, as this gives the application an opportunity to 'respond' to the application quitting or even the computer shutting down.

Loading Documents

There are two standard ways for a document to be loaded by an application.

/ Methods of Loading a File

1 – Methods of Loading a File: Double-click an appropriate document icon using either the Select or Adjust mouse buttons. Upon loading, the application icon should appear on the Icon bar, and a window should open to display the document.

2 – Methods of Loading a File: Drag a document from a filer window onto the application's icon that is displayed on the Icon bar. This assumes that the application has been already been loaded.

When a document icon has been double-clicked, the Wimp issues a Message_DataOpen event code. Applications wishing to load the file should respond with Message_DataLoadAck.

Saving Documents

RISC OS manages the saving of documents in a different way from that of many other Operating Systems. Instead of the more traditional 'Save As' Dialog box where the user has to choose where to save the document, RISC OS uses a 'Save As' window consisting of an icon representing the document and a writeable text box.

Three methods of saving a file are available to the user.

/ Methods of Saving a File

- 1 – Methods of Saving a File: Entering appropriate file name, then dragging the document icon to any filer window.
- 2 – Methods of Saving a File: Clicking the OK button if the file has already been saved.
- 3 – Methods of Saving a File: Using hot keys or function keys to display the 'Save As' window.

Note: All file related actions should be available as a sub-menu within the 'File' menu option.

For more details on how to open and close menus, please see Wimp_CreateMenu.

Closing Documents

The most common method of a user requesting to close a document is by clicking on the window's close icon with the Select mouse button. Upon any close icon being clicked, the Wimp issues a Close_Window_Request event code.

Applications should respond to this event code within their Wimp_Poll routine.

If the document has been changed since it was last saved then it should open a window asking if the document should be saved or discarded. An application should cater for the following scenarios.

/ Response / Action

- 1 – Response: Yes · Action: Open 'Save As' window, and if the document is saved then close the documents window.
- 2 – Response: No · Action: Close the documents window.
- 3 – Response: Cancelled. · Action: Do nothing.

Depending on the mouse button used when clicking on the close icon, two different outcomes are possible.

Mouse Button Differences

Select / As above.

Adjust – As above.: As above, but should open the document filer window if not already on the screen after closing the documents window.

To determine which mouse button was used when clicking the close icon, the application should call Wimp_GetPointerInfo directly after it responds to the Close_Window_Request event code.

To obtain the document's path name, applications should remove the leafname from the document's file name. An application can open a filer window using Message_FilerOpenDir.

Creating Documents

Documents created by being loaded, or by an application itself, should be displayed to the user in a consistent manner. The guidelines are:

Guideline / Description

Initial window – Description: Should be both vertically and horizontally centred on the screen.

Subsequent window – Description: Should be horizontally centred on the screen, but 48 OS units under the previous window, unless the bottom of the window would overlap the icon bar.

Overlapped Icon bar – Description: Windows that would overlap the icon bar, should be displayed at the same horizontal and vertical co-ordinates as the initial window. All subsequent windows would then be displayed as above.

The size of window when opened should follow the guidelines below:

Attribute / Maximum Size

Width – Maximum Size: 700 OS units

Height – Maximum Size: 500 OS units

Editing Documents

Applications should know whether a document has been changed since it was last saved. This is often referred to as the update flag.

A document that has been edited since it was loaded should have an asterisk appended to the end of the windows Title text. You do not need to update the entire window to update a window title bar. See Wimp_ForceRedraw for details.

Inserting Documents

The Wimp also allows for a document to be inserted into an existing document by dragging a document's icon from within a filer window into an already open document window.

It is the applications responsibility to handle this task. The Wimp issues a `Wimp_SendMessage` that the application should respond to within its `Wimp_Poll` routine.

RISC OS has many guidelines on Application and documents. For a complete detailed guide on dialog boxes, please see the RISC OS Style Guide.

[RISC OS Open source page](#)

WindowManager Changing The Pointer

WindowManager

The Wimp allows for applications to change the mouse pointer via `Wimp_SpriteOp` where `R0=&24(36)`. The Wimp achieves this by transforming (and scaling if necessary) a sprite.

In practice

Sprites used for a pointer should have a suffix of 'ptr_'. e.g. the standard arrow shape is named `ptr_default`.

Applications should only change the mouse pointer when it is within the work area of one of its windows.

Note: Each sprite should have its own palette.

Notification

The Wimp provides two event codes for applications to know when the pointer enters and leaves a window. This allows applications to change the pointer between entering and leaving a window.

1 / `Pointer_Entering_Window`

2 – `Pointer_Entering_Window`: `Pointer_Leaving_Window`

Tasks should also respond to the `Message_ModeChange` event code because the Wimp resets the pointer to its default shape upon a screen-mode change. In other words, if a pointer is over an application window when a screen-mode change occurs, it the responsibility of the application to ensure that the correct mouse pointer is used.

[RISC OS Open source page](#)

WindowManager Colour

WindowManager

Among other aspects of its window environment the Wimp permits configuration of colour. An application has control over the colour of text, of window outlines and of many others parts. The Wimp provides different methods for managing the colour of applications:

Type / Description

Basic – Description: Uses one of 16 possible colours

Advanced – Description: Uses 24-bit colour

In practice

The Wimp knows which method to use by checking the Extra Window Flags within a windows definition.

Note: The advanced approach to colour handling requires at least version x.xx of the Window Manager.

4-bit Colour

The Wimp uses 16 pre-set colours which are mapped to the available colours in the current screen mode, regardless of whether there are more or fewer available colours.

The available colours are:

- 0 – Colour: White
- 1 – Colour: Grey(1) · Default Usage: Work area background, icon bar background, scroll bar inner
- 2 – Colour: Grey(2) · Default Usage: Title background
- 3 – Colour: Grey(3) · Default Usage: Scroll bar outer
- 4 – Colour: Grey(4) · Default Usage: Desktop background
- 5 – Colour: Grey(5)
- 6 – Colour: Grey(6)
- 7 – Colour: Black · Default Usage: Title text, window text
- 8 – Colour: Dark Blue
- 9 – Colour: Yellow
- 10 – Colour: Green
- 11 – Colour: Red
- 12 – Colour: Cream · Default Usage: Title background – Input focus
- 13 – Colour: Army Green
- 14 – Colour: Orange
- 15 – Colour: Light Blue

Translation

The following tables details how the Wimp maps the 16 pre-set Wimp colours within non-16 colour screen modes.

2 Colour Mode

16 Colour # / Mapping

- 0 – Mapping: White
- 1-6 – Mapping: Patterned
- 7 – Mapping: Black
- 8-15 – Mapping: White or Black (closest Match)

4 Colour Mode

16 Colour # / Mapping

- 0 – Mapping: White
- 1 – Mapping: Light grey
- 2 – Mapping: Dark grey
- 3 – Mapping: Black
- 8-15 – Mapping: White or Black (closest Match)

Greater than 16 Colour Mode

Translation to closest possible colour is undertaken. However, an exact match may not always be possible.

Note: Manual translation of colours is possible if required. An application should use the ColourTrans module to achieve this

Translation Exceptions

The Wimp translates colours when the current screen mode does not match the required colour depth. The following table shows when translations occur and any exceptions.

When / Exceptions

Windows – Exceptions: Translation of colours unless explicitly told otherwise. This is set within the windows windows flags using bit 10

Sprites – Exceptions: Colour translations are provided by the OS_SpriteOp reason code &34 (52)

Icons – Exceptions: Patterned colours in icons cannot be displayed within 2-colour modes

Caret – Exceptions: Translations of colours unless told otherwise

24-bit Colour

The Wimp can also handle Colours using 24-bit colour.

The table below details which parts of a window can be set with 24-bit colour.

/ Window Element

1 – Window Element: Title text

2 – Window Element: Title background

3 – Window Element: Work area text

4 – Window Element: Work area background

5 – Window Element: Scroll bar inner

6 – Window Element: Scroll bar outer

The colours are in the format BBGGRR.

Translation

Where 24-bit colour has been specified for a window, but a particular window element has not got it's colour set, then the Wimp will use the applicable default 16 colour equivalent.

Manual translation of colours is possible if required. An application should use the ColourTrans module to achieve this.

Notification

The Wimp will issue an event code when the palette changes because of a change of screen mode. This allows applications to know about the change so they can update their content.

The Wimp issues Message_ModeChange which applications should respond to so that any mode specific data can be changed.

For a full detailed description on colour, please see the RISC OS Style Guide.

[RISC OS Open source page](#)

WindowManager Dialog Boxes

WindowManager

Dialog boxes are pre-defined windows for informing, and getting input from, the user.

In practice

The Wimp does not provide a way of changing pre-defined dialog boxes. However, it is possible to create windows that mimic the look and feel of dialog boxes to achieve the same effect.

There are three ways of using a dialog box.

/ Type / Description

1 – Type: Notification · Description: To display information to the user

2 – Type: Static · Description: To allow the user to enter information or make a choice

3 – Type: Menu Entry · Description: As a window that is opened via a menu system option

Opening a Dialog Box

There are two methods of opening a dialog box:

1 · Use the Wimp_OpenWindow to open a dialog box like a standard window

2 · Use Wimp_CreateMenu if the dialog box is to be opened via the menu system. This ensures that the dialog box is opened as if it was a menu window. i.e a non-persistent window that will close if the mouse pointer is clicked outside of the dialog box

Note: If a dialog box is available via the menu system and has a keyboard shortcut assigned, then the dialog should be opened as a menu (option 2 above) as this will ensure that it will be opened in the same manner as it would via the menu system.

Icons

Any type of icon can be assigned within a user-defined dialog box, as it is just a standard window.

RISC OS has many guidelines on the appropriate use of dialog boxes. For a complete detailed guide on dialog boxes, please see the RISC OS Style Guide.

[RISC OS Open source page](#)

WindowManager Dragging

WindowManager

RISC OS uses dragging in a consistent way throughout the Desktop.

In practice

There are two main types of drag operations:

/ Operation / Description

1 – Operation: System · Description: For dragging a window's position, size or scroll offsets. This is mostly used with windows that have no title bar, or those with no Adjust Size icon.

2 – Operation: User · Description: For moving or re-sizing objects within a window.

Basic Dragging Methodology

The Wimp and the application must work together to implement dragging. A brief summary of how the Wimp and the application work together are shown below.

- 1 · User starts a drag using the mouse button (Select or Adjust mouse buttons only)
- 2 · Wimp issues the Mouse_Click event code
- 3 · Application responds to mouse click event code and initiates Wimp_DragBox to start the drag process
- 4 · User ends the drag
- 5 · Wimp issues a User_Drag_Box to signal the end of a drag
- 6 · Application handles the drag, ensuring the all related windows are updated correctly

Mouse Clicks

Window work-areas and icons can easily be configured to respond to dragging. This is achieved by the Window Button Types within the window definition. The Button Flag tells the Wimp which (if any) button clicks and drags it should respond to.

Although a window or icon can be set to respond to drags, it does not exclude other mouse clicks. The result of this, is that the Wimp will still issue a Mouse_Click event code specifying a single mouse click.

Likewise, upon the end of drag the Wimp may issue a Mouse_Click event code specifying a Double Click. The Button flag specifies this.

Miscellaneous Information

Applications must ensure that windows are updated correctly during a drag. An object could be re-sized or moved within a window, or it could also be moved between windows themselves. Updating of a window should use Wimp_UpdateWindow.

Applications that require a drag to be confined to a window must handle the situation when the mouse pointer nears the edge of the window. Then it scrolls the window in the chosen direction or remain unmoved.

If a scroll has been initiated and the mouse button has been released afterwards, it leaves the possibility for further mouse clicks during a drag, for opening a menu etc.. . Applications should be written to take into account this possibility.

To keep track of the drag, the application should stop following the drag on a `Pointer_Leaving_Window` event code, and start again on a `Pointer_Entering_Window`.

[RISC OS Open source page](#)

WindowManager Errors

WindowManager

Errors can be very difficult to detect and fix and may cause both the user and developer great frustration if left unfixed.

In practice

If an application unfortunately fails to deal with an error, it should, as a bare minimum report the error to the user in a friendly manner, explaining what has happened.

Common Errors

/ Common Error

- 1 – Common Error: Always check for errors when calling a Wimp SWI.
- 2 – Common Error: The redraw loops can be a source of infinite error loops. If a redraw fails, then the Wimp will again ask for the area to be redrawn, which cause the same error as before and so on.
- 3 – Common Error: Missing Fonts can cause to infinite looping if error checking is not implemented.
- 4 – Common Error: `Wimp_CreateWindow` and `Wimp_CreateIcon` should always be error proof, to ensure that no data are lost if the calls fail.
- 5 – Common Error: Check cases concerning running out of space.

Error Messages

The table below provides a quick overview of some of the most common error messages that your application may have to deal with.

- &280** – Error Message: Wimp unable to claim work area. The RMA area is full.
- &281** – Error Message: Invalid Wimp operation in this context. Some operations are only allowed after a call to `Wimp_Initialise`.
- &284** – Error Message: Window definition won't fit. No room in RMA for window.
- &286** – Error Message: `Wimp_GetRectangle` called incorrectly.
- &287** – Error Message: Input focus window not found.
- &288** – Error Message: Illegal window handle.
- &289** – Error Message: Bad work area extent. Visible window is set to display a non-existent part of the work area.
- &29F** – Error Message: Bad parameter passed to Wimp in R1.

[RISC OS Open source page](#)

WindowManager General Principles

WindowManager

One of the strengths of RISC OS is the consistency of the user interface offered by applications and the operating system. A description of the general principles of writing compliant applications can be found in the RISC OS Style Guide.

In practice

The Style Guide covers ease of use, consistency, hardware and software configurations, file-handling, naming fonts and application- and user-support.

Compatibility

Although the authors of the Programmer's Reference Manuals (PRMs) have taken great care to ensure that the information provided is clear and correct, it is important that certain rules be followed by a developer. Below are some key guidelines that help to ensure fewer development-problems with previous, current and future versions of RISC OS.

Reserved fields · must be set to 0, i.e. reserved words must be 0, and all reserved bits unset

Wimp_Poll · Unknown Wimp_Poll event codes must be ignored by an application. Errors must not be generated

WIMP Version · Applications should check the Wimp version number to identify the functionality available, and either adapt themselves accordingly or report an error if no work arounds are possible using OS_GenerateError

Window Handles · Be careful when handling unknown window handles. It is quite possible that an unrecognised window handle is owned by another application

Wimp Tasks · Wimp tasks that are Modules must obey certain rules

Key Presses · Any tasks that are set to receive Key_Pressed events must pass on all unrecognised keys to Wimp_ProcessKey. Failure to do so will result in the 'hot key' facility not working

Responsiveness

The RISC OS Operating System allows fast responsive applications to be written. For a complete detailed description on how to optimise the responsiveness of your application(s) please view the Screen Handling section of the RISC OS Style Guide.

Colour

RISC OS allows a wide range of screen resolutions and colour depths. It is considered good practise to ensure that an application is not tied to any specific resolution or colour depth.

Terminology

Applications should adhere to the RISC OS Style Guide in user-interactions. This ensures a consistent user-experience across all applications.

[RISC OS Open source page](#)

WindowManager Icons

WindowManager

An icon is a rectangular area of a window's workspace. It can be used to display either text, or a sprite, or both text and sprite, automatically. It needs the help of the application if it is to display something else to the user; for example, a graph displaying share prices.

In practice

Icons can also be writable. That is, the Wimp will deal with user input, and also handle certain editing functions such as Delete and left and right cursor movements.

Icon Definition

Icons are defined by 32 bytes of data and 12 bytes of data named Icon Data. The icon's dimensions must also be provided in OS units. A full detailed description of an icon can be found in the Icon Block page. A quick overview can be found below.

Icons have handles that are unique within the window they reside in. Thus, an icon can be uniquely defined by it's window/icon handles.

Note: User icon handles begin from zero, whereas system area icons (close, title bar, etc) have negative icon numbers.

Dimensions

Two sets of x and y co-ordinates define the size of the icon within a window.

Icon Flags

Thirty two bytes of data are used to specify numerous settings for the icon, such as Border type, horizontal and vertical alignment, colour, font data, how the button responds to button clicks and whether the icon needs help from the application to be redrawn correctly.

The full detailed description is available in the Icon Flags page.

Icon Data

The Icon Data are divided into three three words (4 bytes each). Icons can be configured to use either Direct or Indirect data for the following attributes of an icon:

- 1 / The text to be displayed in the icon
- 2 – The text to be displayed in the icon: The name of the sprite to be displayed within the icon
- 3 – The text to be displayed in the icon: Both the text to be displayed and the name of the sprite to be used

If the data is set as Direct, a 12 character limit is imposed.

If the data is set as Indirect, then there is no limit as to how many character can be used to define the above attributes.

Icon Creating and Deleting

There are three ways that an icon can be created. They are:

- 1 / Created at the same time as the window.
- 2 – Created at the same time as the window.: Created at run-time.
- 3 – Created at the same time as the window.: Created virtually at run-time using Wimp_PlotIcon during a redraw loop. The advantage of using a virtual icon is that they do not occupy permanent storage.

Wimp_CreateIcon and Wimp_DeleteIcon are used to create and delete an icon. Two important side effects of using these calls are:

- 1 / The window is not redrawn.
- 2 – The window is not redrawn.: The icon handles may change.

Another method of deleting an icon is to use Wimp_SetIconState. Bit 23 of the window flags when using this call must be set, to ensure that the icon is deleted and not simply redrawn on top of itself.

Icon Selection

Icons can be defined to be part of an Exclusive Selection Group(ESG) group. That is, only one icon per ESG can be selected at a given time. The default is for an icon not to be part of any ESG. Note: These are often referred to as Radio Buttons.

When an icon has been selected using the Wimp, the icon colours are inverted, and any other icons in the ESG will be deselected.

Mouse Button Differences

Select · When an icon has been clicked with the Select mouse button it is selected regardless of its previous state.

Adjust · When an icon is clicked with the Adjust mouse button the state of the icon is toggled between its previous and current state.

However, if an icon has its adjust bit set in the window flag, then using the Adjust mouse button to toggle the icon's state will not have any affect on the other icons within the same ESG.

An icon can be disabled so it cannot be selected. In this instance, the Wimp shades the icon to denote that it cannot be selected.

Icon Sprites

Sprites used in an icon can come from the following sources:

- 1 / The system-sprite pool
- 2 – The system-sprite pool: The Wimp sprite pool

3 – The system-sprite pool: A user defined memory allocation

However, the use of system-sprites is not recommended as certain operations cannot be performed on them; for example, scaling and colour translation.

Wimp sprites are useful because one copy can be used by many applications.

User-sprites are used when sprites unique to the application are required.

The Wimp-sprite area is divided in two parts: a permanent part held in ROM and a transient expandable part held in the RMA.

The Wimp_SpriteOp allows automatic access to Wimp Sprites by name. This is a read-only access call, and the only operation allowed on Wimp sprites that can change them is the MergeSpriteFile (reason code 11) or the equivalent *IconSprites command.

For a detailed description on defining the appearance and size of sprites, see the Sprites and Icons? section in the RISC OS Style Guide.

[RISC OS Open source page](#)

WindowManager Keyboard Input And Text Handling

WindowManager

An application using the Wimp should always perform all of its input handling using the Wimp_Poll routine, rather than the older OS_ReadC or OS_Byte calls directly.

Input Focus

Only one window within the Wimp can have the input focus at any given time. In other words, only one window will accept user input via the keyboard. The term used for this is input focus.

The window with input focus should be highlighted so that it is easy for the user to see which window has the focus. This is usually achieved by a highlighted Title Bar, and a caret to show the current input focus position in the window's work area or icon, whichever has the focus.

A window can gain the input focus by the following methods.

/ Description

1 – Description: If the mouse pointer is clicked using Select or Adjust over a writeable icon. The caret is positioned and sized automatically by the Wimp.

2 – Description: By calling the Wimp_SetCaretPosition. The application has control of the caret settings. Positioning of the caret must be calculated using window position, scroll offsets and pointer position.

Notification

The Wimp notifies applications when windows gain and lose focus using the Gain_Caret and Lose_Caret respectively. This allows for an application to respond to the change in caret position. It also is very important if an application is drawing its own caret and not relying on the Wimp's caret.

Key Presses

The Wimp handles the majority of key presses if a writeable icon has the focus. The left and right cursor keys work as expected and the text, caret position and index are all updated. Other key presses, and all keys pressed while the input focus is not in a writeable icon must be managed by the application.

Notification

The Wimp notifies applications about key presses through the Key_Pressed event code via Wimp_Poll.

The Wimp provides information about the keys pressed as well as the standard Wimp caret.

It is the responsibility of the application to decide what actions to perform based on the keys pressed. i.e. pressing on the down cursor key, usually means to moves focus to the next item within the window.

Function Keys and Hot Keys

Function keys and 'hot' keys must be handled by the application itself. They are reported via the standard notification (see above), but are passed as special codes with bit 8 set.

Applications should pass the value of unknown or unwanted function and hot keys back to the Wimp using `Wimp_ProcessKey` so that other applications (that have bit 12 of its Window Flags) can deal with the key press if they so desire. They are called in the order they appear in the window stack, top most first.

An application should return via `Wimp_Poll` once it has carried out any actions.

Notes:

- If a function key is passed back to the Wimp and the input focus is a writeable icon, then the Wimp will expand the function key definition and insert as much as possible of the string into the icon
- Function key F12 and any key-press combination is reserved by RISC OS, so applications must not respond to this key press
- If an applications purpose is to grab hot keys, bit 6 of Window Flags should be set so that the window can be displayed off screen.

Special Characters

Special key combinations can be created by the use of the Alt key. The use of these combinations is common within international keyboards. To ensure that these combinations are permitted, applications should allow for the use of the top-bit-set characters.

Escape Key

As the Wimp continually loops round `Wimp_Poll`, applications can easily detect if the escape has been pressed by checking for the escape button character code 27 (&1B). However, if an application is to perform an action that will take a considerable amount of time then another option is available.

Handling Escape Keys Manually

If an application is going to perform an action that will take a considerable amount of time before it calls `Wimp_Poll`, then it may be wise to enable its own Escape key trapping.

Enable Escape Key / *FX 299,0

Disable Escape Key – *FX 299,0: *FX 229,1 then OS_Byte 124

Note: It is important that an application disable Escape key trapping before it calls `Wimp_Poll`.

It is also a good idea to display the mouse pointer as an hourglass during actions that will take a considerable amount of time, as the user is less likely to interrupt if progress is visible. You should not, however, attempt to change the pointer while the hourglass is still showing.

Upon an application closing down via `Wimp_CloseDown`, the Wimp restores the Escape key to its previous state, along with all other settings changed by the application. i.e. cursor keys, function keys etc...

RISC OS has many guidelines on the appropriate use of Keys presses and Character handling. For a complete detailed guide, please see the RISC OS Style Guide.

[RISC OS Open source page](#)

WindowManager Layout Of Windows

WindowManager

Windows consist of two separate areas:

Coordinate System

system-area / Contains the title bar, scroll bar indicators, close icon etc.

work-area – Contains the title bar, scroll bar indicators, close icon etc.: Contains all the application-specific data to be displayed to the user

The system-area does not normally get updated by the application, except, perhaps, the Title Bar.

Work-area

The work-area is where the application will display information to the user; you can think of it as a document where the words and pictures may be displayed.

The following table shows all the important co-ordinate-systems for dealing with a work-area.

work-area dimensions · Specify the minimum and maximum x and y co-ordinates of the entire work-area. This also referred to as the 'extent' of the window

work-area visibility · Specify the width and height of the work-area visible to the user

work-area scroll offsets · Specify the co-ordinate offsets of the horizontal and vertical scroll bars. This determines which part of the work-area is to be displayed at the top left of the visible work-area

screen position co-ordinates · Specify where the window should be displayed on the screen using x and y co-ordinates

Note: All co-ordinates are provided in OS graphics units.

Work-area Example 1

The following snippet of code specifies the size of the work-area. It does not, however, specify the size of the work-area that is to be shown.

In this example it would produce a work-area 2000 units high and 1000 units wide. The origin of x and y co-ordinates are from the top left, hence the height is a negative number.

```
work_area_x_min = 0
work_area_y_min = -2000
work_area_x_max = 1000
work_area_y_max = 0
```

Work area Example 2

The following snippet of code specifies the size of the work-area and the size of the work-area to be shown.

In this example it would produce a work-area 2000 units high and 1000 units wide, but displaying only a 200 high by 100 wide part of it.

```
work_area_x_min = 0
work_area_y_min = -2000
work_area_x_max = 1000
work_area_y_max = 0
visible_area_x_min = 0
visible_area_y_min = 0
visible_area_x_max = 100
visible_area_y_max = 200
```

Work area Example 3

The following snippet of code specifies the size of the work-area, the size of the work-area to be shown and which part of the work-area is to be shown.

In this example it would produce a work-area 2000 units high and 1000 units wide, but displaying only a 200 high by 100 wide from the x and y offsets of 20 and 100 respectively.

```
work_area_x_min = 0
work_area_y_min = -2000
work_area_x_max = 1000
work_area_y_max = 0
visible_area_x_min = 0
visible_area_y_min = 0
visible_area_x_max = 100
visible_area_y_max = 200
scroll_offset_x = 20
scroll_offset_y = -100
```

Work area Example 4

The following snippet of code specifies the size of the work-area, the size of the work-area to be shown, which part of the work-area is to be shown and where on the screen this work-area should be displayed.

In this example it would produce a work-area 2000 units high and 1000 units wide, display only 200 high by 100 wide from the x and y offsets of 20 and 100 respectively and display it on the screen at (300, 600). Please note: that the screen co-ordinates originate from the bottom left of the screen, as opposed to the top left for windows.

```
work_area_x_min = 0
work_area_y_min = -2000
work_area_x_max = 1000
work_area_y_max = 0
visible_area_x_min = 0
visible_area_y_min = 0
visible_area_x_max = 100
visible_area_y_max = 200
scroll_offset_x = 20
scroll_offset_y = -100
screen_x = 300
screen_y = 600
```

Co-ordinate Calculations

Although at first glance, the calculations may seem daunting, they do get easier the more you use them. A rule of thumb is that work-area co-ordinate origins are the top left of a window, where as the screen co-ordinates are from the bottom left of the screen.

A number of ways of retrieving windows co-ordinates exist.

- They may be made available via the Wimp_Poll routine
- They can be retrieved by calling one of numerous Wimp SWIs such as Wimp_GetWindowState

Note: The co-ordinates above are only a small subset of larger set of data required to create a window as per SWIs such as Wimp_CreateWindow.

Window Stacks

Windows are able to overlap other windows, and in order to keep track of the order of overlapping windows, the wimp also maintains 'depth'. This is called the window stack. The window at the top of the stack is deemed to be the forefront window on the screen.

A window can change its window stack position by user intervention such as clicking on the Title Bar or the Back icon within the system area of a window.

A window can determine its own window stack value by specifying which window that it must appear behind. Another option is simply stating whether it should be 'top' or 'bottom' in the stack.

Window Flags

Many attributes of a window are defined by setting a series of flags in the window block; can it be moved, does it have a Title Bar, does it ignore mouse clicks etc? There are many other settings that can be changed in addition to these to provide a comprehensive set of window configurations.

In addition to the window definition, any initial icons belonging to a window are also defined.

For a complete view of a window's Flags and all associated data, see `Wimp_CreateWindow`.

[RISC OS Open source page](#)

WindowManager Memory Management

WindowManager

The Wimp is responsible for managing memory across all system areas. e.g. screen memory, application workspace, system sprites, font cache etc...

In practice

Although the system has direct control over all aspects of memory, RISC OS allows the user limited control of some memory allocations. This is achieved via Task Display which is accessible by clicking on the Task Manager icon on the icon bar.

Task Display

Task Display is an application provided by the Operating System that gives a graphical representation of the system memory resources using bars to display the amount of memory set aside for each part of the system.

The user can re-size the length of a bar to adjust the allocated memory. Reductions in memory of one part of the system can mean an increase in another should the user require it. For example, reducing the memory allocated to the RAM disc could mean more memory for allocation to system sprites. This allows the user to decide how to make best use of the system's memory.

Two bars, named 'Free' and 'Next' can be altered by the user. They provide the size of free available memory available, and the maximum amount of memory that can be used by the next application to be loaded. This is only used if the application does not explicitly specify how much memory is required by using the `*WimpSlot` command.

Memory Mapping

RISC OS uses co-operative, not the more widespread pre-emptive method of multi-tasking. When an application is given control via a return from `Wimp_Poll`, the Wimp must map the correct pages of physical memory to the application's workspace, which starts at logical memory address &8000 (32,768).

Note: The smallest unit of mapped memory is called a 'Page' and is usually either 8K or 32K in size.

This memory mapping is carried out independently by the Wimp without any action on the part of the application.

Memory Allocation

Applications can (and usually should) specify the amount of memory they need to run without problem. This is achieved by `Wimp_SlotSize`. This has the same effect as manually adjusting the size of the 'Next' bar within Task Display.

If they require more memory, applications can use the Wimp's free pool of memory. Only applications executing in SVC (supervisor) mode can make use of this memory; it is protected against user-mode access. In addition, while the memory is claimed for use, the Wimp cannot dynamically alter the memory size of other parts of the system. So to reduce any potential problems, applications that request the memory should not claim it for extended periods of time. i.e. across multiple `Wimp_Polls`.

To claim use of this memory, an application should call `Wimp_ClaimFreeMemory`.

It is also possible for a user dynamically to adjust an application's memory allocation by dragging the application's bar within Task Display. However, this is only possible if the application agrees to this when it is first loaded. When an application is loaded, the Wimp issues `Message_SetSlot` and if the application will allow its memory to be dynamically allocated by the user, should respond. The Wimp will also issue the same message if a user tries to alter the application bar that has already specified it will allow the memory allocated to be changed.

Memory Tips

Applications with complex or heavy memory requirements should use `Wimp_SlotSize` at run time. This allows applications to both take and give back memory to the Wimp.

BASIC programs may use the `END=&xxxx` to call `Wimp_SlotSize` instead.

Applications written in a high level language such as C/C++ should call `Wimp_SlotSize` directly or use an appropriate call to specify memory requirements.

Applications should never re-configure a machine or kill modules to claim enough memory to load. Such actions are strongly discouraged.

[RISC OS Open source page](#)

WindowManager Menus

WindowManager

The Wimp provides a simple yet powerful menu system . Menus may be nested and the user can choose to keep menus open.

Opening a Menu

RISC OS reserves the middle mouse button for opening menus. Wherefore this button is called the Menu button. The Wimp issues the appropriate `Mouse_Click` event code upon registering a click of the menu button.

An application will usually respond to the event code and open a menu by calling `Wimp_CreateMenu`.

The Wimp passes a pointer to a data structure that describes a list of menu items, and each menu item may have further pointers to its submenus (if any).

Menus can also be made to open a window rather than simply another menu. The window can have icons, sprites and user defined data as required.

Creating a Menu

As the Wimp notifies the application when a menu is to be opened, the menu tree can be built dynamically at runtime. This allows for context-driven menus easily to be created. The window and icon handles passed by the Wimp give the necessary information for creating a menu relevant to the user.

Mouse Clicks

Clicks of the menu button over a window are always reported by the Wimp regardless of the Window Button Types attribute.

When a user makes a selection within a menu (using any mouse button) the Wimp generates another event `Menu_Selection`. The application should then respond to this by decoding the selected menu item(s) and performing the required actions.

The Wimp provides the `Wimp_DecodeMenu` call to aid in calculating which menu item has been selected in which menu.

Mouse Button Differences

Select · The Select button selects the menu item under the mouse pointer and closes the menu.

Adjust · The Adjust button selects the menu item but should keep the menu open to allow the user to make more selections. Some assistance from the application is required, however. See `Wimp_CreateMenu` for more details

Writeable Icons

Menu items can also be made to handle text input. This allows for very easy input from the user while a menu is open.

RISC OS has many guidelines on the appropriate use of menus. For a complete detailed guide on menus please see the RISC OS Style Guide.

[RISC OS Open source page](#)

WindowManager Mode Independence

WindowManager

Wimp applications should work at all screen resolutions and colour depths that the Wimp allows and should not set a particular screen mode upon loading.

In practice

To obtain details about the current screen mode, use `OS_ReadVduVariables` and use the returned values within the application.

Notification

The Wimp issues `Message_ModeChange` to which applications should respond so that any mode specific data can be changed.

Usage

The Wimp provides access to the current screen mode and also useful colour translations and scaling operations.

Notes:

- Within 256 colour screen modes, some EOR plotting tricks do not work quite as expected.
- Within 2 colour modes, the Wimp uses ECF patterns for Wimp colours 1 to 6 and it should be noted that some EOR plotting tricks do not work on these, and use of the `Wimp_TransferBlock` can cause alignment problems with the patterns.

RISC OS has many guidelines on the mode independence. For a complete detailed guide, please see the RISC OS Style Guide.

[RISC OS Open source page](#)

WindowManager Mouse

WindowManager

The Select button should be used as the user's primary method for 'selecting', choosing or dragging things.

Select and Adjust

The Adjust button should be used to perform variations on the function of the select button. It should not simply duplicate the action of the Select button.

Another common use of the mouse buttons is in conjunction with specific key-presses, such as the Shift or Alt keys.

Although this may be useful in some instances, the user should not be expected to try every variation to use an application.

The application-designer should try to minimize the intellectual and manual effort required by its user-interface.

Menu Button

A dedicated mouse-button is used for opening menus; it is often called the Menu button. For consistency all applications should use the Menu button for opening menus.

Double Clicks

The notification of double-clicks to an application is provided by a single event code, rather than by multiple event codes.

[RISC OS Open source page](#)

WindowManager Overview

WindowManager

As in most modern Operating Systems with window environments, windows are used to contain information and display it to the user. The display and updating of windows is shared between the application and the Window Manager.

In practice

The Window Manager and the application communicate by way of a message-passing scheme. The Window Manager sends messages when something needs updating. Applications must be able to respond to these messages so that they can update the necessary windows.

Depending on the complexity of the application, much of the work required to keep windows' contents up to date can be delegated to the Wimp.

Icons

Icons constitute another important feature of the Window Manager. An Icon is a rectangular area within a window whose contents can be one of the following: text, a sprite¹, text & a sprite or User-drawn graphic.

The table below details which type of icon can be updated automatically.

Icon Type / Responsible for Update

Text – Responsible for Update: Wimp (Automatic)

Sprite – Responsible for Update: Wimp (Automatic)

Text & Sprite – Responsible for Update: Wimp (Automatic)

User-Drawn Graphic – Responsible for Update: Application

Menus

Menus are an integral part of the whole RISC OS Window environment. Unlike many other Operating Systems, Menus under RISC OS are pop-up rather than pull-down. Menus are opened by pressing the dedicated menu-button on the mouse.

Menus are context-sensitive: they depend upon the position of the mouse when the menu is opened.

Menus may be nested. That is, a menu entry can lead to another menu, recursively to any desired depth. A useful feature is that each menu (or leaf as it is also known) is a window in its own right and thus can be moved while it is open.

Multi-tasking Method

RISC OS uses co-operative multi-tasking. By calling the Wimp's polling routine a task hands over control to the Wimp, which may pass on control to other tasks. Each application regains control on return from the call to the Wimp's polling routine.

To ensure a smooth running Window environment, each application must not consume too much time between Polling, otherwise other tasks will be prevented from running.

However, it is possible to enforce pre-emptive² multi-tasking for non-Wimp applications by running them within a task window.

Memory Management

Each application, when it has control, is given the illusion that it alone has access to the application workspace. For this to occur, some memory management must be accomplished in advance of switching between Applications.

Application workspace starts at &8000.

The amount of workspace that an application has is set before it starts up.

Communication

As described above, each application must be able to respond to the messages from the Window Manager to ensure it can update its windows and their contents correctly (if it has any).

In addition to this, applications can communicate with each other. Again, this is provided by a powerful message-passing scheme.

1 A Sprite is the name given to the native RISC OS pixel graphics file. It can be thought of as the RISC OS equivalent of a BitMap File(BMP) on Windows.

2 Pre-emption is a method of multi-tasking whereby the the OS takes control from applications. An application will in general not know in advance when this will happen.

[RISC OS Open source page](#)

WindowManager Polling

WindowManager

Within each Multi-tasking application there must be a polling loop. It is the polling loop's duty to respond to events. An event can be generated by either the user or the Operating System itself.

In practice

An application calls its own polling routine to discover if there are any events pending for it. The event is returned in register R0, and other data, depending on the event, are returned in the message block whose address is returned in R1.

The application can utilize the polling routine to execute some code 'in the background'. For example, a clock application may update the clock display if no more pressing events are pending.

A pseudocode version of Polling routine may look like:

```
SYS"Wimp_Initialise"
finished = FALSE
DIM blk 255
REPEAT
SYS"Wimp_Poll",0,blk TO event
CASE event OF
WHEN 0:... Null_Reason_Code
WHEN 1:... Redraw_Window_Request
... etc.
ENDCASE
UNTIL finished
SYS"Wimp_CloseDown" End Application
```

Masking Events

It is also possible to specify that your application is not interested in certain events. This stops the Window Manager from passing control needlessly to your application when these events occur. This method is called Masking.

Masking improves the efficiency of the system, and ensures a more responsive experience for the user.

Note: Some events cannot be masked, as it is vital that an application respond to them. An example of this is the `Open_Window_Request`.

Null Events

The Null Event is returned by the wimp (unless masked) if no events have occurred since the last poll.

If your application does not need to respond to null events (i.e. responding to no events) then you should mask it out to improve efficiency.

If you do need to respond to null events you should use `Wimp_PollIdle` rather than `Wimp_Poll`, unless the user is directly involved (e.g. when dragging an object) and responsiveness is important.

[RISC OS Open source page](#)

WindowManager Redrawing Windows

WindowManager

It is the shared responsibility of the Wimp and the application to ensure that windows on the screen are updated. The Wimp does not, after all, know what the contents of a window should be.

In practice

The Wimp is responsible for informing applications when a window needs to be redrawn and which part of a window is to be redrawn. It is the responsibility of an application actually to redraw the contents of a window.

When a window needs to be redrawn, it is quite possible that only part of the window needs to be updated, and hence the Wimp will inform the application which rectangular part(s) of a window needs to be redrawn. The Wimp splits these parts into non-overlapping rectangles so that the application can either;

- efficiently update only the rectangular parts of a window
- simply, but inefficiently, update the whole window in one go

An application receives notification that one of its windows needs redrawing by receiving a `Redraw_Window_Request` event from the `Wimp_Poll`.

Upon notification, the application should enter a loop in which it checks with the Wimp which rectangular parts of a window need updating. The application itself must then take responsibility for updating each rectangular part of the window.

The snippet of pseudocode below shows how an application would deal with this.

```
REM block is the Wimp_Poll block
SYS"Wimp_RedrawWindow",,block TO flag
WHILE flag :REM Get each rectangular are needing updating
Redraw contents of the appropriate rectangular part of a window
SYS"Wimp_GetRectangle",,block TO flag
ENDWHILE
Return to polling loop
```

The block returns values that are of use for updating a window. Full details are available in the `Wimp_GetRectangle` page.

The areas specified by the `Wimp_RedrawWindow` call are automatically cleared to the window's background colour by the Wimp. The task must determine how to update the window's contents,taking into account the visible work-area and scroll offsets.

Important

The following operations should not be used within the redraw loop

- `Wimp_BlockCopy` should only be used outside of the redraw loop to move an area of the work-area;
- EOR operations should only be used when updating a dragged object within a window, and not the window's contents themselves.

By ensuring that these operations are excluded from the window redraw loop, the same code can be reused to draw the window's contents to a printer.

A full description on improving redraw speed is available in the Redrawing Speed? section in the RISC OS Style Guide.

[RISC OS Open source page](#)

WindowManager Relocatable Modules

WindowManager

A relocatable module is a piece of software which, when loaded, acts as an extension to the operating system or as a replacement for an existing module. They run in an area of memory known as the Relocatable Module Area (RMA). They have a particular format, and RISC OS has facilities for organizing them. They can be used to provide either *commands or SWIs or both for use by software.

In practice

Relocatable modules can receive and respond to the service calls generated by the Wimp. They must also be able to terminate themselves if required, for example during a *RMTidy operation.

Note: Modules that do not handle service calls correctly can cause system wide failure, and such bugs that can be very difficult to track down.

[RISC OS Open source page](#)

WindowManager RISC OS System Icons

WindowManager

In RISC OS 2 the Wimp provided support for numerous screen resolutions and colour depths. In RISC OS 3 it became possible to use alternative sprites for system icons.

In practice

The system icons (title bar, close icon, scroll bars etc...) can easily be customized.

Please note: This document assumes RISC OS 3 or above.

Icon Sets

Software may provide several different sets of icons, each tailored to a particular screen resolution and colour depth. The Wimp will then pick the most appropriate set to use when loading the icons (e.g. via *IconSprites or *ToolSprites).

Filename suffixes are used to identify the different icon set variants. The table below summarises the different filenames (assuming a base name of 'ISprites'), the dimensions for a standard icon sprite, and the different sprite modes that are acceptable.

16 Cols · 256 Cols · 32K/16M Cols

ISprites · 90 × 45 · 34 × 17 · 12 · 15 · RISC OS 3.5

ISprites22 · 90 × 90 · 34 × 34 · 20 or 27 · 21 or 28 · RISC OS 3.5

ISprites23 · (obsolete 90 × 90 DPI mono mode 23)

ISprites11 · 180 × 180 · 68 × 68 · RISC OS 3.5 · RISC OS 3.5 · RISC OS 3.5

ISpritesA · 90 × 45 – Standard Dimensions: 34 × 17 · Sprite mode/format: RISC OS Select alpha mask

ISpritesA2 · 90 × 90 – Standard Dimensions: 34 × 34 · Sprite mode/format: RISC OS Select alpha mask

ISpritesA1 · 180 × 180 – Standard Dimensions: 68 × 68 · Sprite mode/format: RISC OS Select alpha mask

Each set of system icons should be based on the same 'look and feel', and be designed for optimised appearance in screen modes with the above characteristics. See also colour support for information about which sprite formats are acceptable for

different Wimp versions.

The minimum requirement is for the base !Sprites file to be supplied, and the Wimp will fall back to this icon set if no other can be found.

For software which loads sprites manually, Wimp_Extend (sprite suffix), Wimp_ReadSysInfo 29, and Wimp_ReadSysInfo 2 may be used to read information about which icon set the Wimp would prefer to use for the current screen mode.

Note that alpha-masked sprites are only supported for use as icon sprites; tool sprites do not support their use.

Naming Convention

Each system icon has a special name.

For sake of simplicity, the icons will be grouped by where they are shown within a window. Thus:

1 / Top Bar consisting of the Title Bar, Close Icon, Back Icon etc...

2 – Top Bar consisting of the Title Bar, Close Icon, Back Icon etc...: Vertical Scroll Bar

3 – Top Bar consisting of the Title Bar, Close Icon, Back Icon etc...: Horizontal Scroll Bar

Top Bar Icons

Description / Un-clicked / Clicked / Plot Order

Back icon · bicon – Clicked: pbicon · Plot Order: 1st

Close icon · cicon – Clicked: pcicon · Plot Order: 2nd

Title bar (left end cap) · tbarlcap – Clicked: ptbarlcap · Plot Order: 3rd

Title bar1 (middle bottom) · tbarmidb – Clicked: ptbarmidb · Plot Order: 4th

Title bar1 (middle top) · tbarmidt – Clicked: ptbarmidt · Plot Order: 5th

Title bar (right end cap) · tbarrcap – Clicked: ptbarrcap · Plot Order: 6th

Iconize icon · iicon – Clicked: piicon · Plot Order: 7th

Notes:

- All these icons have the same height with the exception of Title Bar middle sprites.
- Title bars on window menus are drawn using the same sprites as those for windows.

Vertical Scroll Bar

Scroll Up Arrow · uicon – Clicked: puicon · Plot Order: 1st

Scroll Down Arrow · dicon – Clicked: pdicon · Plot Order: 2nd

Page Down Area (end cap) · vwellbcap – Clicked: - · Plot Order: 3rd

Page Down Area (section) · vwellb – Clicked: - · Plot Order: 4th

Slider (bottom end cap) · vbarb – Clicked: pvbarb · Plot Order: 5th

Slider (middle section) · vbarmid – Clicked: pvbarmid · Plot Order: 6th

Slider (top end cap) · vbart – Clicked: pvbart · Plot Order: 7th

Page Up Area (section) · vwellt – Clicked: - · Plot Order: 8rd

Page Up Area (end cap) · vwelltcap – Clicked: - · Plot Order: 9nd

Toggle Size icon2 · ticon, ticon1 – Clicked: pticon, pticon1 · Plot Order: 10th

Adjust Size icon3 · sicon, bicon – Clicked: psicon · Plot Order: 11th

Note: All these icons have the same width.

Horizontal Scroll Bar

Scroll Left Arrow · licon – Clicked: plicon · Plot Order: 1st

Scroll Right Arrow · ricon – Clicked: pricon · Plot Order: 2nd

Page Left Area (end cap) · hwelllcap – Clicked: - · Plot Order: 3rd

Page Left Area (section) · hwelll – Clicked: - · Plot Order: 4th

Slider (left end cap) · hbarl – Clicked: phbarl · Plot Order: 5th

Slider (middle section) · hbarmid – Clicked: phbarmid · Plot Order: 6th

Slider (right end cap) · hbarr – Clicked: phbarr · Plot Order: 7th

Page Right Area (section) · hwelrr – Clicked: - · Plot Order: 8rd

Page Right Area (end cap) · hwelrncap – Clicked: - · Plot Order: 9nd

Note: All these icons have the same height.

Annotations

1 These Title Bar icons must have the same width, but can have different heights.

2 The Toggle Size icon must have the same height as the icons in the Top bar and Horizontal scroll bar (see above and below). Two different icons may be used depending on the state of the window (maximum size or previous size). The width of these icons must also be the same as that of the vertical scroll bar.

3 The blicon sprite is used when the Adjust Size icon is not present but both horizontal and vertical scrollbars are shown.

Icon Borders

So that icons do not overlap and so produce unwanted effects, it is necessary for some rows and columns to be transparent.

The table below details which icons must cater for this.

Back icon – Location: Top Bar · Transparency: right edge 1 pixel

Close icon – Location: Top Bar · Transparency: right edge 1 pixel

Title bar (left end cap) – Location: Top Bar · Transparency: solid

Title bar (right end cap) – Location: Top Bar · Transparency: solid

Toggle Size icon – Location: Vertical Scroll Bar · Transparency: left edge 1 pixel

Adjust Size icon2 – Location: Vertical Scroll Bar · Transparency: top edge 1 pixel

Scroll Up Arrow – Location: Vertical Scroll Bar · Transparency: top edge 1 pixel

Scroll Down Arrow – Location: Vertical Scroll Bar · Transparency: top edge 1 pixel

Page Up Area (end cap) – Location: Vertical Scroll Bar · Transparency: solid

Page Down Area (end cap) – Location: Vertical Scroll Bar · Transparency: solid

Scroll Left Arrow – Location: Vertical Scroll Bar · Transparency: left edge = black vertical

Scroll Left Arrow – Location: Vertical Scroll Bar · Transparency: right edge 1 pixel

Page Left Area (end cap) – Location: Horizontal Scroll Bar · Transparency: solid

Page Right Area (end cap) – Location: Horizontal Scroll Bar · Transparency: solid

Common Icons

Icon sprite sets can be further subdivided into 7 categories. They are:

/ Category / Most Commonly used Icons

1 – Category: Filer Icons · Most Commonly used Icons: directory, directory, directoryo, small_dir, small_diro, small_app, file_xxx, small_xxx

2 – Category: Icon Bar Icons · Most Commonly used Icons: switcher, !configure, display, floppydisc, harddisc, ramfs, romapps, small_fs, network

3 – Category: Dialog Box Icons · Most Commonly used Icons: yes, no, dontcare, radiooff, radioon, optoff, opton, tick, up, down, left, right, tick, ,

4 – Category: Pointer Icons · Most Commonly used Icons: ptr_default, ptr_double, ptr_menu, ptr_write, ptr_hand, ptr_direct, ptr_cross, ptr_confirm

5 – Category: Pinboard Icons · Most Commonly used Icons: ic_edit, ic_filer, ic_draw, ic_paint, ic_?

6 – Category: Application Icons · Most Commonly used Icons: !Draw, !Edit, !Paint, etc...

7 – Category: Miscellaneous Icons · Most Commonly used Icons: error

Icon Design

Each icon set (!Sprites22 etc...) should have a similar style.

Text is not recommended in system icons.

Icons should not attempt to be photographic-quality pictures. They should be clear, iconic representations of their referends.

The active point of a pointer-icon, should be a transparent pixel, if the mask is present.

Speed Optimisation

System icons get drawn fastest when the iconset appropriate to the screen mode is used. That is, the number of bits per pixel should be the same as the screen mode, and when the palette matches the default Wimp palette. If the standard

Wimp palette is in use then it's customary to not give the sprite any palette at all - the Wimp will automatically use the standard palette when plotting it, and it will help to reduce both RAM and disc space, especially when dealing with large numbers of icons.

The Wimp caches the memory addresses of all the icons in use so that it does not have to search by sprite name. This cache is automatically updated as required, making it maintenance-free and fully transparent to other software.

Searching for the most common icons is faster when they are placed at the beginning of the !Sprites file.

Colour Support

For RISC OS 2, sprites should all use the standard Wimp palette, as the Wimp will not perform any colour translation when plotting the sprites.

From RISC OS 3 onwards, the Wimp uses ColourTrans when preparing sprites for plotting, allowing the sprites to make use of arbitrary palettes. When RISC OS 3.5 was introduced it became possible to use true-colour sprites for Wimp icons. However note that in both these cases the same sprite suffixes are used as for RISC OS 2 - and so when designing software for older systems, authors must be careful to only supply sprites that will work with the oldest supported OS version, or to implement their own logic to select between different icon sets.

With RISC OS 5.21 (Wimp 5.34) support was introduced for alpha-masked Wimp icons. These have their own filename suffixes, and the Wimp will only attempt to use them if it detects that the rest of the OS understands the required sprite formats, making it safe to supply the sprites with software which also needs to be compatible with older systems. Note that when enabling this feature, the Wimp only checks if RISC OS Select alpha-channel sprites are supported - support for RISC OS 5 alpha-masked sprites is not checked nor guaranteed. This is to allow the same sprites to be used with compatible versions of RISC OS Select (although current versions of the Wimp supplied with RISC OS Select will not look for and load alpha-masked sprites automatically).

[RISC OS Open source page](#)

WindowManager Shared Resources

WindowManager

Applications should store all their resources within their own application directory. However, if an application is using data that other applications may want to access, the preferred approach is to create a shared resource directory which contains the data. The application's !Boot file should set up a system-variable holding the shared-resource directory's pathname.

In practice

An example of a shared resource directory is !System, which is used as a location for resources used by many applications.

Note

Applications should not add their own resources to !System, they should be added to their own shared resource directories. Shared-resource directories' names are centrally maintained by RISC OS Open Ltd, and must be requested using the !Allocate application.

To download the !Allocate application and for more information on registering and allocating names for use with RISC OS, please click [here](#).

[RISC OS Open source page](#)

WindowManager System Font

WindowManager

The Wimp allows any Outline font to be displayed. However, it is good practice to use the Desktop font rather than a font specified by an application.

In practice

The system font can also be used within the Wimp. This practice is now deprecated.

The system font was a standard 8 by 8 pixel character set whose use was necessary before the introduction of Outline Fonts. These can be tailored to specific requirement. E.g. Anti-Aliasing, Bold, Kerning etc...

The Wimp defines the system font to be 16 OS units wide by 32 OS units high regardless of screen resolution in use. This may result in scaling to ensure that its physical size is constant.

Note: An Outline font has been created specifically to mimic the look of the system font.

[RISC OS Open source page](#)

WindowManager Template Files

WindowManager

Besides `Wimp_CreateWindow`, the Wimp also provides a much simpler and easier approach.

In practice

Template files are used to store the definitions of windows and any font or icon data associated with each. By using a template editor application, a user can easily specify windows' attributes such as size, position, title text and icons.

Although template editors make creating windows easier, they cannot handle windows which are child windows of other windows. Each window (parent and child) is deemed separate and needs to be defined individually. This is because window nesting information is not held within template files.

Usage

To use a template file within an application, the Wimp provides several calls.

SWI / Description

Wimp_OpenTemplate – Description: Open a template file

Wimp_LoadTemplate – Description: Load a particular window within the template file

Wimp_CloseTemplate – Description: Close the template file

Wimp_OpenWindow – Description: Used to display a window

Operating System Templates

Many of the system-templates used by RISC OS are stored in the Resource Filing System. The pathname of these files are: `Resource:$.Resources`. It is possible to load these templates into a template editor application and use them as a foundation to build application-specific template files.

It is also possible to over-ride the use of the system-template files by setting `App$Path` where `App` is the name of the application. These variables contain comma-separated list of prefixes, in which the Wimp will search for the template file to open.

Issues

Template files use references to indirect data and to fonts. They must both be handled with care.

Indirect Data

When a template file is created and then saved, all references to the indirect data must point to a location within the file itself. Before loading a template file, the application must set aside memory to be used for window definitions and any indirect data. The pointers to indirect data are then changed to the new locations in memory.

Note: If there is not enough memory to load a template, Wimp_LoadTemplate will return an error.

Fonts

Template-files reference fonts by name and size. For applications to know which font handles these are mapped to, the application must provide an array of data where each reference to a font used is given a specific font handle. The Wimp provides several calls for managing fonts.

SWI / Description

Font_FindFont – Description: Generate a font handle based on a fonts name and size (specified in the template file)

Font_LoseFont – Description: Remove the reference to a specific font handle

Font_ReadDefn – Description: Obtain new font information using the font handle

The array of data used for referencing fonts is 256 bytes in size, and must be used when calling Wimp_LoadTemplate. Each element within the 256 byte array should be initialized to zero in advance of the first call. Font handles are received each time the Wimp calls Font_FindFont. The font handles themselves are indices into the 256 bytes array.

Element *i* is increased each time font handle *i* is returned.

The end result is that when Wimp_LoadTemplate returns, the array contains a count of how many times each font handle was allocated. On closing a window or indeed quitting an application, it should scan the array and call Font_LoseFont the given number of times for non-zero entries.

Notes:

- Applications can obtain the actual font handles by examining the returning window block from Wimp_LoadTemplate
- Each window can use a different 256 byte data array, which may make handling of the fonts easier, however, it does increase the memory requirements slightly
- Windows within a template file known not to have any references to specific fonts, can safely be loaded with no 256 byte data array. The Wimp will, however, report an error if any font references are found
- If references to fonts are used, applications must also rescan its fonts upon receiving Message_ModeChange? by the Wimp. By Calling Font_ReadDefn for each relevant font handle, changing the correct x and y resolution and calling Font_FindFont again. The new font handle can be assigned to the applicable window by using Wimp_SetIconState

[RISC OS Open source page](#)

WindowManager The Icon Bar

WindowManager

The iconbar is a window which is located at the bottom of the screen. Its purpose is three-fold.

- 1 · To provide access to filing systems and device icons (left side of iconbar)
- 2 · To provide access to loaded applications (right side of iconbar)
- 3 · To provide access to the Task Manager and Display Manager (right most side of iconbar)

In practice

It provides access by displaying an icon that represents the application, filing system or device icon.

If there are too many icons on the iconbar, then the Wimp will automatically scroll the iconbar whenever the mouse pointer is positioned close to either end of the bar.

An icon placed on the iconbar can also have text underneath it. It is up to the application to decide the purpose of the text.

Registering an Icon

When an application, filing system or device wishes to be registered on the iconbar, the `Wimp_CreateIcon` is used (with the appropriate window handle). It is the window handle that determines where on the iconbar the icon should be placed.

In adherence to the RISC OS Style Guide the icon used should be the sprite used within the Filer for the application itself. See for example; the !Edit application.

Mouse Clicks

When the mouse pointer is clicked on an icon, the Wimp returns the `Mouse_Click` event (with a window handle of -2) to the application which created the icon originally.

Also, calling `Wimp_GetPointerInfo` returns a window handle of -2 when the pointer is partially or completely over the iconbar.

Icon Dimensions

When the Wimp is called to place an icon on the bar (registering an icon), it uses the icon's x co-ordinates (see `Wimp_CreateIcon`) only to determine its width. The Wimp then decides how best to position the icon.

However, the Wimp does use both the icon's y co-ordinates to determine the icon's vertical position. Consequently applications must be aware of the 'standard' dimensions of the iconbar, in order to position their icons correctly.

Icons that appear on the iconbar should have a width and height of 68 OS units square.

For a full detailed description on how to position Icons can be found in the Positioning icons on the iconbar? section within the RISC OS Style Guide.

[RISC OS Open source page](#)

WindowManager Time

WindowManager

RISC OS actually has two clocks;

In practice

- the hardware clock
- the software clock.

Both clocks are timed independently and can diverge by a few seconds per day. The clocks are only re-synchronised when the computer hardware is reset.

Usage

Tasks that require the use of time and date settings should always use the time provided by the software clock. The software clock measures time in centi-seconds.

Note: Times are represented using 32-bits, and it should be remembered that the number used can become negative (when bit 31 is set). So when comparing times, the expression

`(new_time - old_time) > 100`

is better than

`new_time > old_time + 100`

[RISC OS Open source page](#)

WindowManager Updating Windows

WindowManager

When the Wimp has notified an application that a window needs redrawing, the process of updating it must begin. An application cannot simply update the appropriate area of the screen as other windows may be overlapping it.

In practice

There are two approaches to how an application can update a window:

1 / Permanent redraw to a window

2 – Permanent redraw to a window: Temporary redraw to a window

Both options have their own benefits within an application. Each option will be discussed in further detail below.

Permanent Redraw

This is useful when a window needs to be updated to reflect a change that will occur to a window. e.g. when a paragraph of text has been reformatted in a word processor.

This is achieved by calling `Wimp_ForceRedraw` which will clear the update area automatically ready for the new contents to be redrawn by the application.

Once called, the application returns to the central loop where the `Redraw_Window_Request` originated.

Note: The Wimp will return a `Redraw_Window_Request` from `Wimp_Poll`, which should be responded to.

Temporary Redraw

This is useful when a window needs to be updated to reflect a temporary change that will occur to a window. e.g. when the mouse is dragging an object across a window.

This is achieved by calling `Wimp_UpdateWindow` and performing the appropriate operations. Unlike `Wimp_ForceRedraw`, no clearing of the area takes place. This allows for the application to retain the existing window contents and update the relevant part using EOR; for example, when dragging a line.

Note: The Wimp will return a list of rectangles within the work area that need updating which should be responded to.

[RISC OS Open source page](#)

WindowManager Window Panes

WindowManager

Window panes are similar to standard windows with one major difference: they are attached to other windows. These windows are called Tool Windows. Normally panes are attached to the outer edge of Tool windows. As a Tool window gets dragged by a mouse pointer, so do the panes.

In practice

They are often used to display commonly used icons in an application.

Definition

Window panes are actually defined as a standard window with one difference; within the Window Flags, bit 5 is set to specify a pane window.

This flag is used in two different circumstances:

- 1 · When a pane gets the input focus, the tool window is highlighted.
- 2 · When toggling the tool window, the Wimp must treat panes as transparent.

Usage

Although window panes can increase the productivity of an application, they do require more effort. For a start, the application itself must handle all the management of window panes.

Important aspects to consider when dealing with window panes are:

- 1 · When a Tool window is closed, any attached panes must also be closed by the application.
- 2 · When a Tool window is opened, any attached panes must be opened by the application.

Notes:

- To open a pane window at the correct co-ordinates, an application should first get the co-ordinates of the Tool window and then calculate the appropriate position of the pane.
- Unwanted redraws can be caused if the order in which windows are opened is not optimised.

[RISC OS Open source page](#)

WindowManager Window System Area

WindowManager

The system-area is the part of a window that display icons (controls) that affect the position and display of a window. Each control is detailed below, with differences between the Select and Adjust mouse buttons.

Title Bar

Contains the name of the window, often referred to as the 'window caption' in other Operating Systems. Dragging the Title Bar causes the whole window to be moved. It is an icon. i.e it can have its own font type, alignment and colour settings.

SWIs & Event Information

Open_Window_Request · If RISC OS is configured to display solid window dragging rather than just window outlines, then each mouse movement will generate one of these requests

Redraw_Window_Request · If RISC OS is configured to display solid window dragging rather than just window outlines, then each mouse movement will generate one of these requests after the Open_Window_Request

Mouse Button Differences

Select · The window moves and is also brought to the top of the window stack

Adjust · The window moves, but the window stack depth remains unchanged

Close Icon

Used to generate a request for a window be closed. It is the responsibility of an application to decide how to respond to the event request.

SWI & Event Information

Wimp_CloseWindow · If an application decides to close the window, then this SWI should be called

Close_Window_Request · Wimp issues this event when the close icon has been clicked

Redraw_Window_Request · Wimp issues this event when the window is closed, to all those windows which were obscured by it and are now visible

Mouse Button Differences

Select / The window should close

Adjust – The window should close: The window should close, but also open up the parent window (if applicable)

Back Icon

Used to send the window to the back of the window stack (across all applications). This makes it the least visible window within the Window Manager.

SWI & Event Information

Redraw_Window_Request · Wimp issues the event when the window is sent to the back of the window stack, to all those windows which were obscured by it and are now visible

Mouse Button Differences

Select / The window should go to the back of the window stack

Adjust – The window should go to the back of the window stack: Same as Select

Toggle Size Icon

Used to toggle the window between its maximum size and the previous size used.

SWI & Event Information

Open_Window_Request · Wimp issues the event so that an application can update the work-area region of the resized window

Mouse Button Differences

Select · Toggles the size of the window between the maximum and previous size used. The window stack also toggles between the front and its previous depth setting

Adjust · Toggles the size of the window between the maximum and previous size used. The window stack depth remains unchanged

Adjust Size Icon

Used to resize the window both horizontally and vertically (if permitted).

SWI & Event Information

Open_Window_Request · Wimp can generate a continuous stream of these events while being resized or one single event at the end of the resize. This setting is on a per window basis

Redraw_Window_Request · Wimp may issue these events if other windows 'behind' this window need updating due to the window resizing

Mouse Button Differences

Select · Resizes the window and brings it to the front of the window stack

Adjust · Resizes the window but the window stack depth remains unchanged

Iconize Icon

Used to hide the window and place an icon on either the Pinboard or Icon bar that can be used to re-display the window if clicked.

SWI & Event Information

Message_Iconize · Wimp issues this event when the Iconize icon has been clicked

Mouse Button Differences

Select / Minimizes the window to either the Pinboard or Icon bar

Adjust – Minimizes the window to either the Pinboard or Icon bar: Same as Select

Vertical Scroll Bar

It is used to advance or 'move' the work-area in a vertical manner. Note: All scroll operations leave the window's stack unchanged.

The vertical scroll bar is made up of five discrete elements, each with different methods of advancing the work-area. They are:

Element / Vertical Advancement

Scroll Up Arrow – Vertical Advancement: Offsets the window by 32 OS units

Page Up Area (above the slider) – Vertical Advancement: Adjusts the Scroll offsets by the height of the visible work-area

Slider – Vertical Advancement: Dragging the slider allow for accurate positioning of the work-area

Page Down Area (below the slider) – Vertical Advancement: Adjusts the Scroll offsets by the height of the visible window work-area

Scroll Down Arrow – Vertical Advancement: Offsets the window by 32 OS units

SWI & Event Information

Open_Window_Request · Wimp issues this event after each time the scroll bar has changed position so the work-area contents can be updated. This is the default setting for a window

Scroll_Request · Wimp can generate a continuous stream of these events or a single event if either the Scroll Arrows or Page Area are clicked. The application can then decide how much to scroll. This setting is on a per window basis

Wimp_OpenWindow · This SWI is used to update the contents of the window if the application has control over how much the work-area should be scrolled by. i.e using Scroll_Requests

Scroll Up & Scroll Down Arrow

Select / Moves the work-area by 32 OS units in the appropriate direction

Adjust – Moves the work-area by 32 OS units in the appropriate direction: Moves the work-area by 32 OS units in the reverse direction

Page Up & Page Down Areas

Select · Moves the work-area by the height of the visible work area in appropriate direction

Adjust · Moves the work-area by the height of the visible work-area in reverse direction

Slider

Select / Allows accurate vertical positioning of the work-area

Adjust – Allows accurate vertical positioning of the work-area: Allows accurate vertical and horizontal positioning of the work-area. Note: The horizontal Scroll bar must also be visible for this

Horizontal scroll Bar

It is used to advance or 'move' the work-area in a horizontal manner. Note: All scroll operations leave the window's stack unchanged.

The horizontal scroll bar is made up of five discrete elements, each with different methods of advancing the work-area. They are:

Element / Horizontal Advancement

Scroll Left Arrow – Horizontal Advancement: Offsets the window by 32 OS units

Page Left Area (above the slider) – Horizontal Advancement: Adjusts the Scroll offsets by the width of the visible work area

Slider – Horizontal Advancement: Dragging the slider allow for accurate positioning of the work-area

Page Right Area (below the slider) – Horizontal Advancement: Adjusts the Scroll offsets by the width of the visible window work area

Scroll Right Arrow – Horizontal Advancement: Offsets the window by 32 OS units

SWI & Event Information

Open_Window_Request · Wimp issues this event after each time the scroll bar has changed position so the work area contents can be updated. This is the default setting for a window

Scroll_Request · Wimp can generate a continuous stream of these events or a single event if either the Scroll Arrows or Page Area are clicked. The application can then decide how much to scroll. This setting is on a per window basis

Wimp_OpenWindow · This SWI is used to update the contents of the window if the application has control over how much the work-area should be scrolled by. i.e using Scroll_Requests

Scroll Left & Scroll Right Arrow

Select / Moves the work-area by 32 OS units in the appropriate direction

Adjust – Moves the work-area by 32 OS units in the appropriate direction: Moves the work-area by 32 OS units in the reverse direction

Page Left & Page Right Areas

Select · Moves the work-area by the width of the visible work area in appropriate direction

Adjust · Moves the work-area by the width of the visible work area in reverse direction

Slider

Select / Allows accurate horizontal positioning of the work area

Adjust – Allows accurate horizontal positioning of the work area: Allows accurate horizontal and horizontal positioning of the work area.

Note: The horizontal Scroll bar must also be visible for this

Window Definition

When a window is created, Window flags are used to determine which system-area controls should be displayed. To find out more about this see the Window Flags pages.

[RISC OS Open source page](#)

WordV

Vectors

This vector is called to indirect all calls to OS_Word.

Before calling

R0 Reason code

R1 Pointer to parameter block in memory

Notes

OS_Word is entered with interrupts disabled, so claimants of this vector should focus on rapid execution, else your routine must enable interrupts and be prepared to be re-entered.

The default action is to call the MOS word handler.

[RISC OS Open source page](#)

WrchV

Vectors

This vector is used to indirect all calls to OS_WriteC, and in turn OS_NewLine, OS_PrettyPrint, OS_Write0, OS_WriteI, OS_WriteN and OS_WriteS which call it.

Before calling

R0 "Character" to write to active output streams (not a byte)

On return

R0 Preserved

Notes

In practice these SWIs may not actually call `OS_WriteC` or this vector if there is no reason to do so (i.e. the vector is not claimed, spooling and printer are inactive).

Importantly the interpretation of `R0` is highly context-dependent. If there is a VDU queue (`OS_Byte 218`) then it's a data byte. Otherwise its width is defined by the current Alphabet (`OS_Byte 71`).

Alphabets such as UTF-16 (117) or Unicode (118) imply `R0` is 16 bit or 21 bit respectively, and implementations of UTF-8 (111) and Shift-JIS (122) may support multicode sequences. Use `Service_International 332` to read the character code mask and multicode flags for an Alphabet.

Once irrelevant bits have been masked, then if it's between 0 and 31 it will usually be a control code (though during `OS_Plot 208-215` that's a FontManager control code).

Serious bugs have been caused by assuming that `R0` is a byte on entry (it isn't) or that it can be ANDed with `&FF` (it can't). e.g. All versions of TaskWindow crash or hang up if you type `VDU-12345678`.

[RISC OS Open source page](#)

2026 local supplements

These manuals were found in /Volumes/Stuff/C Sources/PRM. They describe local builds, ports, and proposed interfaces. Check each manual and the installed module version before using an interface; inclusion here does not mean it is present on every RISC OS 5 system.

BlendTable

RISC OS 5 Modernisation Unify – blend translation tables

Version 0.10C | 04 September 2026

BlendTable is a ReWrite 32-bit RISC OS 5 module for the Modernisation Unify project. It restores the Select 0.08 public surface on APCS-32 systems. This manual documents version 0.10C. BlendTable builds cached alpha/blend translation tables for the current screen mode. Icon borders and image renderers use it when compositing against desktop colours.

Calls listed: BlendTable_GenerateTable, BlendTable_UnlockTable

Source: BlendTable-PRM-2026-09-04.pdf · 3 pages

Use the source supplement for its complete register contracts and known gaps.

ClipboardHolder

RISC OS 5 Modernisation Unify – global clipboard holder

Version 0.18C | 04 September 2026

ClipboardHolder is a ReWrite 32-bit RISC OS 5 module for the Modernisation Unify project. It restores the Select 0.17 public surface on APCS-32 systems. This manual documents version 0.18C. ClipboardHolder holds the global clipboard when applications exit, and offers simple Copy/Paste SWIs so clients need not implement the full ClaimEntity / DataRequest / DataSave protocol themselves.

Calls listed: ClipboardHolder_Copy, ClipboardHolder_Paste

Source: ClipboardHolder-PRM-2026-09-04.pdf · 3 pages

Use the source supplement for its complete register contracts and known gaps.

Desktop

RISC OS Titanium build – C port

Version 2.78C | 03 September 2026

The Desktop module starts the desktop: it wakes every dormant module task in turn, boots the ROM applications, starts whichever of them the CMOS configuration asks for, and finally runs whatever the caller asked for on the *Desktop command line itself. It has one command and no SWIs; almost everything it does, it does as the direct effect of that one command.

Source: Desktop-PRM-2026-09-03.pdf · 4 pages

Use the source supplement for its complete register contracts and known gaps.

DragASprite

RISC OS Titanium build – C port

Version 0.22C | 03 September 2026

The DragASprite module drags a sprite around the screen on the caller's behalf, following the pointer (or a fixed box) until the mouse button is released, in place of the plain dashed outline Wimp_DragBox draws on its own.

Calls listed: DragASprite_Start, DragASprite_Stop

Source: DragASprite-PRM-2026-09-03.pdf · 5 pages

Use the source supplement for its complete register contracts and known gaps.

DragAnObject

RISC OS Titanium build – C port

Version 0.11C | 03 September 2026

The DragAnObject module drags an arbitrary shape around the screen - not a sprite already in existence, but whatever a caller-supplied routine draws - by rendering it twice into a throwaway pair of sprites, working out a mask from the difference between them, and handing the result to DragASprite to do the actual dragging.

Calls listed: DragAnObject_Start, DragAnObject_Stop

Source: DragAnObject-PRM-2026-09-03.pdf · 5 pages

Use the source supplement for its complete register contracts and known gaps.

DrawChart

RISC OS 5 Modernisation Unify – chart rendering SWIs

Version 0.09C | 04 September 2026

DrawChart is a ReWrite 32-bit RISC OS 5 module for the Modernisation Unify project. It restores the Select 0.08 public surface on APCS-32 systems. This manual documents version 0.09C. DrawChart renders simple bar and line charts either directly with OS_Plot or as Draw objects for embedding in DrawFiles.

Calls listed: DrawChart_Render, DrawChart_DrawFile, DrawChart_CheckPoint, DrawChart_DrawObject, DrawChart_UnpackObject

Source: DrawChart-PRM-2026-09-04.pdf · 3 pages

Use the source supplement for its complete register contracts and known gaps.

Filer

RISC OS Titanium build – C port

Version 3.05 | 04 September 2026

The Filer is the desktop file manager: directory viewers, the icon-bar device icons that open them, and the commands other tasks use to open, close, boot and run objects without knowing how a viewer is drawn. It is a Wimp module task started by *Desktop through *Desktop_Filer. It exports no SWIs of its own.

Source: Filer-PRM-2026-09-04.pdf · 10 pages

Use the source supplement for its complete register contracts and known gaps.

FilerAct

RISC OS Titanium build – C rewrite of the Filer action window

Version 1.01C | 03 September 2026

Filer_Action provides the background operations the Filer offers on a selection: copy, move, delete, set access, set type, count, copy-move, copy-local, stamp and find. The Filer starts one Filer_Action task per operation, tells it what was selected and what to do, and the task shows the familiar action window ("Copying files", "Deleting files" and so on) with its progress fields, buttons and options menu.

Source: FilerAct-PRM-2026-09-03.pdf · 8 pages

Use the source supplement for its complete register contracts and known gaps.

FilerFind

RISC OS Titanium build – C port

Version 0.41 | 03 September 2026

FilerFind is the application-provided replacement for the Filer's built-in Find. It is a RISC_OSLib C Wimp task that searches a directory tree for names matching a pattern (with optional type, size and date criteria) and shows the hits in a results window.

Source: FilerFind-PRM-2026-09-03.pdf · 3 pages

Use the source supplement for its complete register contracts and known gaps.

FilterManager

RISC OS Titanium build – C port

Version 0.01C | 03 September 2026

FilterManager is the desktop filter broker. The Window Manager accepts only one direct registration per Wimp_RegisterFilter type; FilterManager holds those registrations and fans each call out to any number of task- or module-owned filters registered through its own SWI chunk.

Source: FilterManager-PRM-2026-09-03.pdf · 7 pages

Use the source supplement for its complete register contracts and known gaps.

FontMap

RISC OS 5 Modernisation Unify – font name mapping

Version 0.05C | 04 September 2026

FontMap is a ReWrite 32-bit RISC OS 5 module for the Modernisation Unify project. It restores the Select 0.04 public surface on APCS-32 systems. This manual documents version 0.05C. FontMap translates foreign or Windows font names into RISC OS FontManager names so documents and HTML viewers find a usable face.

Calls listed: FontMap_Translate

Source: FontMap-PRM-2026-09-04.pdf · 3 pages

Use the source supplement for its complete register contracts and known gaps.

Free

RISC OS Titanium build – C port

Version 0.43C | 03 September 2026

The Free module implements the desktop free-space display: one window per filing system showing the size, free space and used space of a mounted disc, drawn as three proportional bars, and kept up to date as files are written.

Calls listed: Free_Register, Free_DeRegister

Source: Free-PRM-2026-09-03.pdf · 6 pages

Use the source supplement for its complete register contracts and known gaps.

IconBorderPlain

RISC OS 5 Modernisation Unify – plain icon borders

Version 0.04C | 04 September 2026

IconBorderPlain is a ReWrite 32-bit RISC OS 5 module for the Modernisation Unify project. It restores the Select 0.03 public surface on APCS-32 systems. This manual documents version 0.04C. IconBorderPlain registers plain rectangular icon-border filters with FilterManager so the Window Manager draws Select-style slab borders.

Source: IconBorderPlain-PRM-2026-09-04.pdf · 3 pages

Use the source supplement for its complete register contracts and known gaps.

IconBorderRound

RISC OS 5 Modernisation Unify – rounded icon borders

Version 0.29C | 04 September 2026

IconBorderRound is a ReWrite 32-bit RISC OS 5 module for the Modernisation Unify project. It restores the Select 0.28 public surface on APCS-32 systems. This manual documents version 0.29C. IconBorderRound registers rounded icon-border filters and provides *WimplIconBorder for Select-era rounded furniture.

Source: IconBorderRound-PRM-2026-09-04.pdf · 3 pages

Use the source supplement for its complete register contracts and known gaps.

LibraryHelp

RISC OS 5 Modernisation Unify – library keyword help

Version 0.14C | 04 September 2026

LibraryHelp is a ReWrite 32-bit RISC OS 5 module for the Modernisation Unify project. It restores the Select 0.13 public surface on APCS-32 systems. This manual documents version 0.14C. LibraryHelp extends *Help so library and alias keywords can be described. It exports no SWIs.

Source: LibraryHelp-PRM-2026-09-04.pdf · 3 pages

Use the source supplement for its complete register contracts and known gaps.

PathUtils

RISC OS 5 Modernisation Unify – path variable helpers

Version 0.11C | 04 September 2026

PathUtils is a ReWrite 32-bit RISC OS 5 module for the Modernisation Unify project. It restores the Select 0.10 public surface on APCS-32 systems. This manual documents version 0.11C. PathUtils maintains comma-separated path variables such as Run\$Path and File\$Type_FFF. It appends, prepends or removes a component while ensuring the component appears only once.

Calls listed: PathUtils_EnumeratePath, PathUtils_JoinPath, PathUtils_RemovePath

Source: PathUtils-PRM-2026-09-04.pdf · 3 pages

Use the source supplement for its complete register contracts and known gaps.

Pinboard

RISC OS Titanium build – C port

Version 1.08C | 03 September 2026

The Pinboard module provides the desktop background: the picture behind everything else, the file and directory icons pinned on top of it, the icons an iconised window collapses to, and the small set of directory icons - "TinyDirs" - that can be added to the icon bar instead of the backdrop. Nothing about it is a filing system or an application in its own right; it is a place other things can be put.

Source: Pinboard-PRM-2026-09-03.pdf · 7 pages

Use the source supplement for its complete register contracts and known gaps.

Pinboard3 Widgets

RISC OS TITANIUM – MODULE AND WIDGETS

Version 3.00 | 03 September 2026

This manual documents Pinboard 3.00 and the desktop widgets it hosts: the clock, weather card, calendar, news card, RISCOSNet search bar, sticky notes, watermark and translucent icon bar. It covers the module's public *Command interface, the service calls it claims, the system variables it reads, and the companion Configure plug-ins !PinSetup and !WidgetStp.

Source: Pinboard3-Widgets-PRM-2026-09-03.pdf · 11 pages

Use the source supplement for its complete register contracts and known gaps.

SharedCLibrary 7.00

Version 7.00 | Revised 6 September 2026

This manual defines the Unified SharedCLibrary 7.00 release: its architecture, exported initialization SWIs, library-chunk contract, APCS-32 client requirements, StubsG companion library, NorcroftUniversal build, Core 7 deployment, error handling and verification evidence.

Calls listed: LibInitAPCS_A, LibInitAPCS_R, LibInitAPCS_32, LibInitModuleAPCS_32

Source: SharedCLibrary-7.00-PRM-2026-09-04.pdf · 22 pages

Use the source supplement for its complete register contracts and known gaps.

SpriteExtend

RISC OS 5 – Lyonix upward (ARMv5TE to ARMv8)

Version 2.00 | 03 September 2026

SpriteExtend is the sprite extension module. It claims SpriteV (&1F) and intercepts the OS_SpriteOp reason codes the kernel does not implement itself – scaled and transformed plotting, tiling, row and column insert/delete, palettes, alpha, and CheckSpriteArea – and it provides the JPEG_* SWIs used to inspect and plot JFIF images.

Source: SpriteExtend-PRM-2026-09-03.pdf · 6 pages

Use the source supplement for its complete register contracts and known gaps.

SysLog

RISC OS 5 Modernisation Unify – system diagnostic log

Version 0.62C | 04 September 2026

SysLog is a ReWrite 32-bit RISC OS 5 module for the Modernisation Unify project. It restores the Select 0.61 public surface on APCS-32 systems. This manual documents version 0.62C. SysLog provides a system-wide diagnostic log with level filtering, indentation and optional session files.

Calls listed: SysLog_LogMessage, SysLog_GetLogLevel, SysLog_FlushLog, SysLog_SetLogLevel

Source: SysLog-PRM-2026-09-04.pdf · 3 pages

Use the source supplement for its complete register contracts and known gaps.

TaskManager

RISC OS Titanium build – C port

Version 1.55C | 03 September 2026

The Switcher module, whose title is TaskManager, is the desktop's Task Manager: the window listing every application and module task, the memory they are using, and the free space and largest free block in the RMA and in each dynamic area, together with the menu that shuts the machine down, restarts it, or drops to the command line. Everything the icon bar's Task Manager icon offers is implemented here.

Calls listed: TaskManager_TaskNameFromHandle, TaskManager_EnumerateTasks, TaskManager_Shutdown, TaskManager_StartTask

Source: TaskManager-PRM-2026-09-03.pdf · 8 pages

Use the source supplement for its complete register contracts and known gaps.

UnifiedToolbox

Version 1.0 | 3 September 2026

The Unified Toolbox brings the RISC OS Select and RISC OS 6 Toolbox additions to RISC OS 5 while retaining the established RISC OS 5 object interfaces. It is intended for applications which use Select-authored resource files, extended Window gadgets, Toolbox timers, deferred submenus or shared redraw callbacks.

Source: UnifiedToolbox-PRM-2026-09-03.pdf · 8 pages

Use the source supplement for its complete register contracts and known gaps.

Zipper

RISC OS 5 Modernisation Unify – ZIP archive access

Version 0.28C | 04 September 2026

Zipper is a ReWrite 32-bit RISC OS 5 module for the Modernisation Unify project. It restores the Select 0.27 public surface on APCS-32 systems. This manual documents version 0.28C. Zipper reads and writes ZIP archives through a handle-based SWI API. This port supports store-method (compression 0) archives without a ZLib dependency.

Calls listed: Zipper_UnZipOpen, Zipper_UnZipClose, Zipper_UnZipInfo, Zipper_UnZipEnumerate, Zipper_UnZipFileInfo, Zipper_UnZipFileOpen, Zipper_UnZipFileClose, Zipper_UnZipFileRead

Source: Zipper-PRM-2026-09-04.pdf · 3 pages

Use the source supplement for its complete register contracts and known gaps.

Alphabetical entry index

Page references point to entries in this PDF. PDF search also finds names, register labels, and notes.

SWI calls

AcornSSL_Close 3
AcornSSL_Connect 3
AcornSSL_Creat 4
AcornSSL_CreateSession 4
AcornSSL_Getpeername 5
AcornSSL_Getsockname 5
AcornSSL_Getsockopt 5
AcornSSL_ioctl 6
AcornSSL_Read 6
AcornSSL_Recv 7
AcornSSL_Send 7
AcornSSL_Setsockopt 8
AcornSSL_Shutdown 8
AcornSSL_Stat 9
AcornSSL_Version 9
AcornSSL_Write 9
ADFS_ATAPIOp 10
ADFS_ControllerType 11
ADFS_DescribeDisc 11
ADFS_DiscOp 12
ADFS_DiscOp64 12
ADFS_Drives 12
ADFS_ECCSAndRetries 13
ADFS_FlpProcessDCB 13
ADFS_FreeSpace 13
ADFS_FreeSpace64 14
ADFS_HDC 14
ADFS_IDEDeviceInfo 15
ADFS_IDEUserOp 16
ADFS_LockIDE 17
ADFS_MiscOp 17
ADFS_PowerControl 18
ADFS_Retries 19
ADFS_SectorDiscOp 19
ADFS_SetIDEController 20
ADFS_VetFormat 20
ARM_IMB 21
ARM_IMBRange 21
ATA_Op 22
ATAPI_GetDrives 23
BASICTrans_Error 23
BASICTrans_HELP 24
BASICTrans_Message 24
BCMSupport_AllocPropertyBuffer 25
BCMSupport_FreePropertyBuffer 25

BCMSupport_MBSync 25
BCMSupport_SendMBMessage 26
BCMSupport_SendPropertyBuffer 27
BCMSupport_SendTempPropertyBuffer 29
BlendTable_GenerateTable 29
BlendTable_UnlockTable 30
Buffer_Create 31
Buffer_Deregister 31
Buffer_GetInfo 32
Buffer_InternalInfo 32
Buffer_LinkDevice 33
Buffer_ModifyFlags 33
Buffer_Register 34
Buffer_Remove 34
Buffer_Threshold 34
Buffer_UnlinkDevice 35
CD_AudioLevel 35
CD_AudioPause 35
CD_AudioStatus 36
CD_ByteCopy 36
CD_CheckDrive 37
CD_CloseDrawer 37
CD_Control 37
CD_ConvertToLBA 38
CD_ConvertToMSF 38
CD_DiscChanged 39
CD_DiscHasChanged 39
CD_DiscUsed 39
CD_DriveReady 40
CD_DriveStatus 40
CD_EjectButton 41
CD_EnquireAddress 41
CD_EnquireDataMode 41
CD_EnquireTrack 42
CD_GetAudioParms 43
CD_GetAudioParms 0 43
CD_GetParameters 44
CD_Identify 44
CD_Inquiry 45
CD_IsDrawerLocked 46
CD_LastError 46
CD_OpenDrawer 47
CD_PlayAudio 47
CD_PlayTrack 48
CD_Prefetch 48
CD_ReadAudio 49
CD_ReadData 49
CD_ReadSubChannel 50
CD_ReadUserData 50
CD_Register 51
CD_Reset 51
CD_SCSIUserOp 52

CD_SeekTo 53
CD_SetAudioParms 53
CD_SetAudioParms 0 54
CD_SetParameters 54
CD_StopDisc 55
CD_Supported 55
CD_Unregister 56
CD_Version 56
CDFS_ConvertDriveToDevice 57
CDFS_DescribeDisc 57
CDFS_GetBufferSize 58
CDFS_GetNumberOfDrives 58
CDFS_GiveFileType 58
CDFS_SetBufferSize 59
CDFS_SetNumberOfDrives 59
CDFS_Truncation 59
CDFS_WherIsFile 60
ColourPicker_CloseDialogue 60
ColourPicker_DeregisterModel 61
ColourPicker_HelpReply 61
ColourPicker_ModelSWI 61
ColourPicker_OpenDialogue 62
ColourPicker_ReadDialogue 62
ColourPicker_RegisterModel 63
ColourPicker_SetColour 63
ColourPicker_UpdateDialogue 64
ColourTrans_ColourNumberToGCOL 64
ColourTrans_ConvertCIEToRGB 65
ColourTrans_ConvertCMYKToRGB 65
ColourTrans_ConvertDeviceColour 66
ColourTrans_ConvertDevicePalette 66
ColourTrans_ConvertHSVToRGB 67
ColourTrans_ConvertRGBToCIE 67
ColourTrans_ConvertRGBToCMYK 68
ColourTrans_ConvertRGBToHSV 68
ColourTrans_GCOLToColourNumber 69
ColourTrans_GenerateTable 69
ColourTrans_InvalidateCache 70
ColourTrans_MiscOp 70
ColourTrans_ReadCalibration 70
ColourTrans_ReadPalette 71
ColourTrans_ReturnColourNumber 71
ColourTrans_ReturnColourNumberForMode 72
ColourTrans_ReturnFontColours 72
ColourTrans_ReturnGCOL 73
ColourTrans_ReturnGCOLForMode 73
ColourTrans_ReturnOppColourNumber 74
ColourTrans_ReturnOppColourNumberForMode 74
ColourTrans_ReturnOppGCOL 75
ColourTrans_ReturnOppGCOLForMode 75
ColourTrans_SelectGCOLTable 76
ColourTrans_SelectTable 76

ColourTrans_SetCalibration 77
ColourTrans_SetColour 77
ColourTrans_SetFontColours 78
ColourTrans_SetGCOL 78
ColourTrans_SetOppGCOL 79
ColourTrans_SetOppTextColour 79
ColourTrans_SetTextColour 80
ColourTrans_WriteCalibrationToFile 80
ColourTrans_WriteLoadingsToFile 80
ColourTrans_WritePalette 81
Debugger_Disassemble 81
Debugger_DisassembleThumb 82
DeviceFS_CallDevice 82
DeviceFS_Deregister 83
DeviceFS_DeregisterObjects 83
DeviceFS_ReceivedCharacter 84
DeviceFS_Register 84
DeviceFS_RegisterObjects 85
DeviceFS_Threshold 85
DeviceFS_TransmitCharacter 86
DHCP_Execute 86
DHCP_GetOption 87
DHCP_GetState 87
DHCP_Inform 88
DHCP_SetOption 88
DHCP_Version 89
DMA_AllocateLogicalChannels 89
DMA_DeregisterChannel 90
DMA_ExamineTransfer 90
DMA_QueueTransfer 91
DMA_RegisterChannel 91
DMA_ResumeTransfer 92
DMA_SuspendTransfer 93
DMA_TerminateTransfer 93
DOSFS_DiscFormat 94
DOSFS_LayoutStructure 94
DragAnObject_Start 95
DragAnObject_Stop 96
DragASprite_Start 97
DragASprite_Stop 97
Draw_Fill 98
Draw_FillClipped 98
Draw_FlattenPath 99
Draw_ProcessPath 99
Draw_Stroke 100
Draw_StrokeClipped 101
Draw_StrokePath 101
Draw_TransformPath 102
DrawFile_BBox 103
DrawFile_DeclareFonts 103
DrawFile_Render 104
Econet_AbandonAndReadReceive 105

Econet_AbandonReceive 105
Econet_AbandonTransmit 106
Econet_AllocatePort 106
Econet_ClaimPort 106
Econet_ConvertStatusToError 107
Econet_ConvertStatusToString 107
Econet_CreateReceive 108
Econet_DeAllocatePort 108
Econet_DoImmediate 109
Econet_DoTransmit 109
Econet_EnumerateReceive 110
Econet_EnumerateTransmit 110
Econet_ExamineReceive 111
Econet_NetworkState 111
Econet_PacketSize 112
Econet_PollTransmit 112
Econet_PrintBanner 113
Econet_ReadLocalStationAndNet 113
Econet_ReadProtection 113
Econet_ReadReceive 114
Econet_ReadStationNumber 114
Econet_ReadTransportName 115
Econet_ReadTransportType 115
Econet_ReleasePort 116
Econet_SetProtection 116
Econet_StartImmediate 116
Econet_StartTransmit 117
Econet_Version 118
Econet_WaitForReception 118
FileCore_Create 119
FileCore_DescribeDisc 120
FileCore_DiscardReadSectorsCache 120
FileCore_DiscFormat 120
FileCore_DiscOp 121
FileCore_DiscOp 0 122
FileCore_DiscOp 1 122
FileCore_DiscOp 15 123
FileCore_DiscOp 2 123
FileCore_DiscOp 3 124
FileCore_DiscOp 4 124
FileCore_DiscOp 5 125
FileCore_DiscOp 6 125
FileCore_DiscOp 7 126
FileCore_DiscOp 8 126
FileCore_DiscOp 9 127
FileCore_DiscOp64 127
FileCore_Drives 128
FileCore_Features 129
FileCore_FloppyStructure 129
FileCore_FreeSpace 130
FileCore_FreeSpace64 130
FileCore_LayoutStructure 131

FileCore_MiscOp 131
FileCore_MiscOp 0 132
FileCore_MiscOp 1 133
FileCore_MiscOp 2 133
FileCore_MiscOp 3 134
FileCore_MiscOp 4 134
FileCore_MiscOp 5 135
FileCore_MiscOp 6 136
FileCore_MiscOp 7 136
FileCore_SectorDiscOp 137
FilerAction_SendSelectedDirectory 137
FilerAction_SendSelectedFile 137
FilerAction_SendStartOperation 138
Filter_DeRegisterCopyFilter 139
Filter_DeRegisterPostFilter 140
Filter_DeRegisterPostIconFilter 140
Filter_DeRegisterPostRectFilter 141
Filter_DeRegisterPreFilter 141
Filter_DeRegisterRectFilter 142
Filter_RegisterCopyFilter 142
Filter_RegisterPostFilter 143
Filter_RegisterPostIconFilter 143
Filter_RegisterPostRectFilter 144
Filter_RegisterPreFilter 144
Filter_RegisterRectFilter 145
Font_ApplyFields 145
Font_CacheAddr 146
Font_Caret 146
Font_CharBBox 147
Font_ConverttoOS 147
Font_Converttopoints 148
Font_CurrentFont 148
Font_CurrentRGB 148
Font_DecomposeMenu 149
Font_EnumerateCharacters 150
Font_FindCaret 150
Font_FindCaretJ 151
Font_FindField 151
Font_FindFont 152
Font_FutureFont 152
Font_FutureRGB 153
Font_ListFonts 153
Font_ListFonts (Basic) 154
Font_ListFonts (Menu) 154
Font_LookupFont 155
Font_LoseFont 156
Font_MakeBitmap 156
Font_Paint 156
Font_ReadColourTable 157
Font_ReadDefn 157
Font_ReadDefn (information syntax) 158
Font_ReadDefn (size syntax) 158

Font_ReadEncodingFilename 159
Font_ReadFontMax 159
Font_ReadFontMetrics 160
Font_ReadFontPrefix 161
Font_ReadInfo 161
Font_ReadScaleFactor 162
Font_ReadThresholds 162
Font_ScanString 162
Font_SetColourTable 163
Font_SetFont 164
Font_SetFontColours 164
Font_SetFontMax 165
Font_SetPalette 165
Font_SetScaleFactor 166
Font_SetThresholds 166
Font_StringBBox 167
Font_StringWidth 167
Font_SwitchOutputToBuffer 168
Font_UnCacheFile 168
FPEmulator_Abort 169
FPEmulator_ActivateContext 170
FPEmulator_ChangeContext 171
FPEmulator_ContextLength 171
FPEmulator_DeactivateContext 172
FPEmulator_ExceptionDump 172
FPEmulator_InitContext 173
FPEmulator_LoadContext 173
FPEmulator_SaveContext 174
FPEmulator_Version 175
Free_DeRegister 175
Free_Register 175
Freeway_Enumerate 176
Freeway_Read 177
Freeway_Register 177
Freeway_Serial 178
Freeway_Status 178
Freeway_Write 179
FSLock_ChangeStatus 180
FSLock_Status 180
FSLock_Version 181
GPIO_Features 181
GPIO_ReadData 181
GPIO_ReadOE 182
GPIO_WriteData 182
GPIO_WriteEvent 182
GPIO_WriteOE 182
Hourglass_Colours 183
Hourglass_LEDs 183
Hourglass_Off 184
Hourglass_On 184
Hourglass_Percentage 185
Hourglass_Smash 185

Hourglass_Start 186
HTTP_AddCookie 186
HTTP_ConsumeCookie 187
HTTP_DeregisterMethod 187
HTTP_EnumerateCookies 188
HTTP_GetData 189
HTTP_RegisterMethod 189
IIC_Control 190
InverseTable_Calculate 191
InverseTable_SpriteTable 191
Joystick_CalibrateBottomLeft 192
Joystick_CalibrateTopRight 192
Joystick_Read 193
Joystick_Read 0 193
Joystick_Read 1 194
JPEG_FileInfo 194
JPEG_Info 195
JPEG_PDriverIntercept 195
JPEG_PlotFileScaled 196
JPEG_PlotFileTransformed 196
JPEG_PlotScaled 197
JPEG_PlotTransformed 198
Message_ColourPickerResetColourRequest 199
MessageTrans_CloseFile 199
MessageTrans_CopyError 199
MessageTrans_Dictionary 200
MessageTrans_EnumerateTokens 200
MessageTrans_ErrorLookup 201
MessageTrans_FileInfo 201
MessageTrans_GSLookup 202
MessageTrans_Lookup 202
MessageTrans_MakeMenus 203
MessageTrans_OpenFile 204
NetFS_ConvertDate 204
NetFS_DoFSOp 205
NetFS_DoFSOpToGivenFS 205
NetFS_EnableCache 206
NetFS_EnumerateFS 206
NetFS_EnumerateFSContexts 207
NetFS_EnumerateFSList 207
NetFS_GetObjectUID 208
NetFS_ReadCurrentContext 208
NetFS_ReadFSName 209
NetFS_ReadFSNumber 209
NetFS_ReadFSTimeouts 210
NetFS_ReadUserId 210
NetFS_SetCurrentContext 211
NetFS_SetFSName 211
NetFS_SetFSNumber 211
NetFS_SetFSTimeouts 212
NetFS_UpdateFSList 212
NetPrint_BindPSName 213

NetPrint_ConvertStatusToString 213
NetPrint_ListServers 214
NetPrint_ListServers 0 214
NetPrint_ListServers 1 215
NetPrint_ListServers 2 216
NetPrint_ReadPSName 216
NetPrint_ReadPSNumber 217
NetPrint_ReadPSTimeouts 217
NetPrint_SetPSName 218
NetPrint_SetPSNumber 218
NetPrint_SetPSTimeouts 218
OS_AbortTrap 219
OS_AddCallBack 221
OS_AddToVector 222
OS_AMBControl 222
OS_Args 223
OS_Args 0 223
OS_Args 1 224
OS_Args 2 224
OS_Args 254 225
OS_Args 255 225
OS_Args 3 225
OS_Args 4 226
OS_Args 5 226
OS_Args 6 227
OS_Args 7 227
OS_Args 8 228
OS_Args 9 228
OS_BGet 229
OS_BinaryToDecimal 229
OS_BPut 229
OS_BreakCtrl 230
OS_BreakPt 230
OS_Byte 231
OS_Byte 0 233
OS_Byte 1 234
OS_Byte 10 234
OS_Byte 106 235
OS_Byte 11 235
OS_Byte 112 236
OS_Byte 113 236
OS_Byte 114 237
OS_Byte 117 237
OS_Byte 118 238
OS_Byte 12 238
OS_Byte 120 239
OS_Byte 121 239
OS_Byte 122 240
OS_Byte 124 240
OS_Byte 125 241
OS_Byte 126 241
OS_Byte 127 242

OS_Byte 128 242
OS_Byte 129 243
OS_Byte 129 (OS) 243
OS_Byte 129 (scan range) 243
OS_Byte 129 (scan single) 244
OS_Byte 129 (time) 244
OS_Byte 13 245
OS_Byte 134 245
OS_Byte 135 246
OS_Byte 138 246
OS_Byte 139 247
OS_Byte 14 247
OS_Byte 143 248
OS_Byte 144 248
OS_Byte 145 249
OS_Byte 15 249
OS_Byte 152 250
OS_Byte 153 250
OS_Byte 156 251
OS_Byte 16 252
OS_Byte 160 252
OS_Byte 161 253
OS_Byte 162 253
OS_Byte 163 254
OS_Byte 163 (reset) 254
OS_Byte 163 (return sprite information) 254
OS_Byte 163 (return status) 255
OS_Byte 163 (set) 255
OS_Byte 165 256
OS_Byte 17 256
OS_Byte 176 256
OS_Byte 177 257
OS_Byte 178 257
OS_Byte 18 258
OS_Byte 181 258
OS_Byte 182 259
OS_Byte 188 259
OS_Byte 189 260
OS_Byte 19 260
OS_Byte 190 261
OS_Byte 191 261
OS_Byte 192 262
OS_Byte 193 262
OS_Byte 194 263
OS_Byte 195 263
OS_Byte 196 264
OS_Byte 197 264
OS_Byte 198 265
OS_Byte 199 265
OS_Byte 2 266
OS_Byte 20 266
OS_Byte 200 267

OS_Byte 201 267
OS_Byte 202 268
OS_Byte 203 268
OS_Byte 204 269
OS_Byte 21 269
OS_Byte 211 270
OS_Byte 212 270
OS_Byte 213 271
OS_Byte 214 271
OS_Byte 216 272
OS_Byte 217 272
OS_Byte 218 273
OS_Byte 219 273
OS_Byte 220 274
OS_Byte 221 - 228 274
OS_Byte 229 275
OS_Byte 230 275
OS_Byte 236 276
OS_Byte 237 276
OS_Byte 238 277
OS_Byte 240 278
OS_Byte 241 278
OS_Byte 242 279
OS_Byte 243 279
OS_Byte 245 280
OS_Byte 246 280
OS_Byte 247 281
OS_Byte 25 281
OS_Byte 250 282
OS_Byte 251 282
OS_Byte 253 283
OS_Byte 254 283
OS_Byte 255 284
OS_Byte 3 284
OS_Byte 4 285
OS_Byte 5 285
OS_Byte 6 286
OS_Byte 7 286
OS_Byte 70 287
OS_Byte 71 287
OS_Byte 8 288
OS_Byte 9 288
OS_CallAfter 289
OS_CallASWI 289
OS_CallASWIR12 290
OS_CallAVector 290
OS_CallBack 291
OS_CallEvery 291
OS_ChangedBox 292
OS_ChangeDynamicArea 292
OS_ChangeEnvironment 293
OS_ChangeRedirection 294

OS_CheckModeValid 294
OS_Claim 295
OS_ClaimDeviceVector 295
OS_ClaimProcessorVector 297
OS_ClaimScreenMemory 298
OS_CLI 298
OS_Confirm 299
OS_Control 299
OS_ConvertBinary1 300
OS_ConvertBinary2 300
OS_ConvertBinary3 301
OS_ConvertBinary4 301
OS_ConvertCardinal1 302
OS_ConvertCardinal2 302
OS_ConvertCardinal3 303
OS_ConvertCardinal4 303
OS_ConvertDateAndTime 304
OS_ConvertFileSize 304
OS_ConvertFixedFileSize 305
OS_ConvertFixedNetStation 305
OS_ConvertHex1 306
OS_ConvertHex2 306
OS_ConvertHex4 307
OS_ConvertHex6 307
OS_ConvertHex8 308
OS_ConvertInteger1 308
OS_ConvertInteger2 309
OS_ConvertInteger3 309
OS_ConvertInteger4 310
OS_ConvertNetStation 310
OS_ConvertSpacedCardinal1 311
OS_ConvertSpacedCardinal2 311
OS_ConvertSpacedCardinal3 312
OS_ConvertSpacedCardinal4 312
OS_ConvertSpacedInteger1 313
OS_ConvertSpacedInteger2 313
OS_ConvertSpacedInteger3 314
OS_ConvertSpacedInteger4 314
OS_ConvertStandardDateAndTime 315
OS_ConvertVariform 315
OS_CRC 316
OS_DelinkApplication 316
OS_DynamicArea 317
OS_DynamicArea 0 318
OS_DynamicArea 1 319
OS_DynamicArea 10 319
OS_DynamicArea 2 320
OS_DynamicArea 20 321
OS_DynamicArea 21 321
OS_DynamicArea 22 322
OS_DynamicArea 23 323
OS_DynamicArea 24 324

OS_DynamicArea 25 325
OS_DynamicArea 26 325
OS_DynamicArea 27 326
OS_DynamicArea 3 326
OS_DynamicArea 4 327
OS_DynamicArea 5 327
OS_DynamicArea 8 328
OS_DynamicArea 9 328
OS_EnterOS 329
OS_EnterUSR26 330
OS_EnterUSR32 330
OS_EvaluateExpression 331
OS_Exit 331
OS_ExitAndDie 332
OS_File 332
OS_File 0 333
OS_File 1 334
OS_File 10 334
OS_File 11 335
OS_File 12 335
OS_File 13 336
OS_File 14 336
OS_File 15 337
OS_File 16 338
OS_File 17 338
OS_File 18 339
OS_File 19 339
OS_File 2 340
OS_File 20 340
OS_File 21 341
OS_File 22 341
OS_File 23 342
OS_File 24 343
OS_File 255 343
OS_File 3 344
OS_File 4 344
OS_File 5 345
OS_File 6 345
OS_File 7 346
OS_File 8 346
OS_File 9 347
OS_Find 347
OS_FindMemMapEntries 348
OS_FSControl 348
OS_FSControl 0 350
OS_FSControl 1 350
OS_FSControl 10 351
OS_FSControl 11 351
OS_FSControl 12 352
OS_FSControl 13 352
OS_FSControl 14 353
OS_FSControl 15 353

OS_FSControl 16 354
OS_FSControl 17 354
OS_FSControl 18 355
OS_FSControl 19 355
OS_FSControl 2 355
OS_FSControl 20 356
OS_FSControl 21 357
OS_FSControl 22 357
OS_FSControl 23 358
OS_FSControl 24 358
OS_FSControl 25 359
OS_FSControl 26 359
OS_FSControl 27 360
OS_FSControl 28 360
OS_FSControl 30 361
OS_FSControl 31 361
OS_FSControl 32 362
OS_FSControl 33 362
OS_FSControl 35 363
OS_FSControl 36 363
OS_FSControl 37 364
OS_FSControl 38 364
OS_FSControl 39 365
OS_FSControl 4 365
OS_FSControl 40 366
OS_FSControl 41 366
OS_FSControl 42 367
OS_FSControl 43 367
OS_FSControl 44 367
OS_FSControl 45 368
OS_FSControl 46 368
OS_FSControl 47 369
OS_FSControl 48 369
OS_FSControl 49 369
OS_FSControl 5 370
OS_FSControl 50 370
OS_FSControl 51 371
OS_FSControl 52 371
OS_FSControl 53 372
OS_FSControl 54 372
OS_FSControl 55 373
OS_FSControl 56 373
OS_FSControl 57 374
OS_FSControl 58 374
OS_FSControl 6 375
OS_FSControl 7 375
OS_FSControl 8 376
OS_FSControl 9 376
OS_GBPB 377
OS_GBPB 1 377
OS_GBPB 10 378
OS_GBPB 11 379

OS_GBPB 12 380
OS_GBPB 2 381
OS_GBPB 3 381
OS_GBPB 4 382
OS_GBPB 5 383
OS_GBPB 6 383
OS_GBPB 7 384
OS_GBPB 8 385
OS_GBPB 9 385
OS_GenerateError 386
OS_GenerateEvent 387
OS_GetEnv 387
OS_GSInit 388
OS_GSRead 388
OS_GSTrans 389
OS_Hardware 389
OS_Hardware 0 390
OS_Hardware 1 390
OS_Hardware 2 391
OS_Hardware 3 391
OS_Hardware 4 392
OS_Hardware 5 392
OS_Heap 393
OS_Heap 0 393
OS_Heap 1 394
OS_Heap 2 394
OS_Heap 3 394
OS_Heap 4 395
OS_Heap 5 395
OS_Heap 6 396
OS_Heap 7 396
OS_Heap 8 397
OS_HeapSort 398
OS_HeapSort32 398
OS_IICOp 399
OS_InstallKeyHandler 399
OS_IntOff 400
OS_IntOn 400
OS_LeaveOS 400
OS_Memory 401
OS_Memory 0 402
OS_Memory 10 402
OS_Memory 12 403
OS_Memory 13 404
OS_Memory 14 404
OS_Memory 15 405
OS_Memory 16 405
OS_Memory 17 406
OS_Memory 18 407
OS_Memory 19 408
OS_Memory 20 410
OS_Memory 21 411

OS_Memory 22 411
OS_Memory 23 412
OS_Memory 24 413
OS_Memory 25 414
OS_Memory 6 415
OS_Memory 64 415
OS_Memory 65 416
OS_Memory 7 416
OS_Memory 8 417
OS_Memory 9 417
OS_MMUControl 418
OS_MMUControl 0 419
OS_MMUControl 1 419
OS_MMUControl 2 420
OS_Module 421
OS_Module 0 422
OS_Module 1 422
OS_Module 10 423
OS_Module 11 423
OS_Module 12 424
OS_Module 13 424
OS_Module 14 425
OS_Module 15 425
OS_Module 16 425
OS_Module 17 426
OS_Module 18 426
OS_Module 19 427
OS_Module 2 427
OS_Module 20 428
OS_Module 21 428
OS_Module 22 429
OS_Module 23 429
OS_Module 24 430
OS_Module 3 430
OS_Module 4 431
OS_Module 5 431
OS_Module 6 432
OS_Module 7 432
OS_Module 8 433
OS_Module 9 433
OS_Mouse 433
OS_NewLine 434
OS_NVMemory 434
OS_NVMemory 0 435
OS_NVMemory 1 435
OS_NVMemory 2 436
OS_NVMemory 3 436
OS_NVMemory 4 437
OS_NVMemory 6 437
OS_NVMemory 7 438
OS_PlatformFeatures 438
OS_PlatformFeatures 0 439

OS_PlatformFeatures 32 440
OS_PlatformFeatures 33 440
OS_PlatformFeatures 34 441
OS_PlatformFeatures 35 442
OS_Plot 442
OS_Pointer 444
OS_Pointer 0 445
OS_Pointer 1 445
OS_Pointer 2 446
OS_PrettyPrint 446
OS_PrintChar 447
OS_ReadArgs 447
OS_ReadC 448
OS_ReadDefaultHandler 448
OS_ReadDynamicArea 449
OS_ReadEscapeState 449
OS_ReadLine 450
OS_ReadLine32 451
OS_ReadMemMapEntries 451
OS_ReadMemMapInfo 452
OS_ReadModeVariable 452
OS_ReadMonotonicTime 453
OS_ReadPalette 453
OS_ReadPoint 454
OS_ReadRAMFsLimits 454
OS_ReadSysInfo 455
OS_ReadSysInfo 0 455
OS_ReadSysInfo 1 456
OS_ReadSysInfo 10 456
OS_ReadSysInfo 11 457
OS_ReadSysInfo 12 457
OS_ReadSysInfo 13 458
OS_ReadSysInfo 14 458
OS_ReadSysInfo 15 459
OS_ReadSysInfo 2 459
OS_ReadSysInfo 3 460
OS_ReadSysInfo 4 461
OS_ReadSysInfo 5 461
OS_ReadSysInfo 6 462
OS_ReadSysInfo 7 462
OS_ReadSysInfo 8 463
OS_ReadSysInfo 9 463
OS_ReadUnsigned 464
OS_ReadVarVal 465
OS_ReadVduVariables 466
OS_Release 466
OS_ReleaseDeviceVector 467
OS_RelinkApplication 467
OS_RemoveCallBack 468
OS_RemoveCursors 468
OS_RemoveTickerEvent 468
OS_Reset 469

OS_RestoreCursors 469
OS_ResyncTime 470
OS_ScreenMode 470
OS_ScreenMode 0 471
OS_ScreenMode 1 471
OS_ScreenMode 10 472
OS_ScreenMode 11 472
OS_ScreenMode 13 473
OS_ScreenMode 14 473
OS_ScreenMode 15 474
OS_ScreenMode 2 475
OS_ScreenMode 3 475
OS_ScreenMode 4 476
OS_ScreenMode 5 476
OS_ScreenMode 6 477
OS_ScreenMode 64 477
OS_ScreenMode 65 478
OS_ScreenMode 66 478
OS_ScreenMode 67 479
OS_ScreenMode 68 479
OS_ScreenMode 7 480
OS_ScreenMode 8 481
OS_ScreenMode 9 481
OS_SerialOp 481
OS_SerialOp 0 482
OS_SerialOp 1 482
OS_SerialOp 10 483
OS_SerialOp 2 483
OS_SerialOp 3 484
OS_SerialOp 4 484
OS_SerialOp 5 485
OS_SerialOp 6 485
OS_SerialOp 8 485
OS_SerialOp 9 486
OS_ServiceCall 486
OS_SetCallBack 487
OS_SetColour 487
OS_SetECFOrigin 488
OS_SetEnv 488
OS_SetMemMapEntries 489
OS_SetVarVal 489
OS_SpriteOp 490
OS_SpriteOp 10 491
OS_SpriteOp 11 492
OS_SpriteOp 12 492
OS_SpriteOp 13 493
OS_SpriteOp 14 493
OS_SpriteOp 15 494
OS_SpriteOp 16 494
OS_SpriteOp 17 495
OS_SpriteOp 2 496
OS_SpriteOp 23 496

OS_SpriteOp 24 497
OS_SpriteOp 25 497
OS_SpriteOp 26 498
OS_SpriteOp 27 498
OS_SpriteOp 28 499
OS_SpriteOp 29 499
OS_SpriteOp 3 500
OS_SpriteOp 30 500
OS_SpriteOp 31 501
OS_SpriteOp 32 501
OS_SpriteOp 33 502
OS_SpriteOp 34 502
OS_SpriteOp 35 503
OS_SpriteOp 36 503
OS_SpriteOp 37 504
OS_SpriteOp 38 505
OS_SpriteOp 39 507
OS_SpriteOp 40 507
OS_SpriteOp 41 508
OS_SpriteOp 42 508
OS_SpriteOp 43 509
OS_SpriteOp 44 510
OS_SpriteOp 45 510
OS_SpriteOp 46 511
OS_SpriteOp 47 511
OS_SpriteOp 48 512
OS_SpriteOp 49 512
OS_SpriteOp 50 513
OS_SpriteOp 51 513
OS_SpriteOp 52 514
OS_SpriteOp 53 515
OS_SpriteOp 54 515
OS_SpriteOp 55 516
OS_SpriteOp 56 517
OS_SpriteOp 57 518
OS_SpriteOp 58 519
OS_SpriteOp 60 519
OS_SpriteOp 61 520
OS_SpriteOp 62 521
OS_SpriteOp 65 522
OS_SpriteOp 8 522
OS_SpriteOp 9 523
OS_SubstituteArgs 523
OS_SubstituteArgs32 524
OS_SWINumberFromString 525
OS_SWINumberToString 525
OS_SynchroniseCodeAreas 526
OS_UnusedSWI 527
OS_UpCall 527
OS_UpCall 1 528
OS_UpCall 10 529
OS_UpCall 11 530

OS_UpCall 12 530
OS_UpCall 15 531
OS_UpCall 16 532
OS_UpCall 17 532
OS_UpCall 19 533
OS_UpCall 2 533
OS_UpCall 24 534
OS_UpCall 25 535
OS_UpCall 256 535
OS_UpCall 257 536
OS_UpCall 3 536
OS_UpCall 4 539
OS_UpCall 6 540
OS_UpCall 7 541
OS_UpCall 8 541
OS_UpCall 9 542
OS_UpdateMEMC 542
OS_ValidateAddress 543
OS_Word 544
OS_Word 0 545
OS_Word 1 545
OS_Word 10 546
OS_Word 11 546
OS_Word 12 547
OS_Word 13 547
OS_Word 14 548
OS_Word 15 548
OS_Word 2 548
OS_Word 21 549
OS_Word 22 549
OS_Word 3 550
OS_Word 4 550
OS_Word 7 551
OS_Word 9 551
OS_Write0 552
OS_WriteC 552
OS_WriteEnv 553
OS_Writel 553
OS_WriteN 553
OS_WriteS 554
Parallel_HardwareAddress 554
Parallel_Op 555
Parallel_Op 0 555
Parallel_Op 1 556
Parallel_Op 2 556
PCI_ConfigurationRead 557
PCI_ConfigurationWrite 557
PCI_EnumerateFunctions 558
PCI_FindByClass 558
PCI_FindByID 559
PCI_FindByLocation 559
PCI_HardwareAddress 560

PCI_IORead 560
PCI_IOWrite 561
PCI_LogicalAddress 561
PCI_MemoryRead 562
PCI_MemoryWrite 562
PCI_RAMAlloc 563
PCI_RAMFree 563
PCI_ReadHeader 564
PCI_ReadID 564
PCI_ReadInfo 564
PCI_ReturnNumber 565
PCI_SpecialCycle 565
Portable_CommandBMU 566
Portable_Control 566
Portable_EnumerateBMU 567
Portable_Idle 567
Portable_ReadBMUVariable 568
Portable_ReadBMUVariables 568
Portable_ReadFeatures 569
Portable_ReadSensor 570
Portable_SleepTime 570
Portable_Speed 571
Portable_Speed2 571
Portable_Speed2 0 572
Portable_Speed2 1 572
Portable_Speed2 2 573
Portable_Speed2 3 573
Portable_Speed2 4 574
Portable_Speed2 5 574
Portable_Speed2 6 575
Portable_Speed2 7 575
Portable_Speed2 8 576
Portable_Speed2 9 576
Portable_Status 577
Portable_Stop 577
Portable_WriteBMUVariable 578
RamFS_DescribeDisc 578
RamFS_DiscOp 579
RamFS_DiscOp64 579
RamFS_Drives 580
RamFS_FreeSpace 580
ResourceFS_DeregisterFiles 580
ResourceFS_RegisterFiles 581
RT_ChangePriority 581
RT_Deregister 582
RT_ReadInfo 582
RT_Register 583
RT_TimedYield 586
RT_Yield 586
RTC_Adjust 587
RTC_Features 588
RTC_Read 588

RTC_Write 589
ScreenBlanker_Control 589
ScreenBlanker_Control 2 590
ScreenBlanker_Control 3 590
ScreenBlanker_Control 4 591
ScreenBlanker_Control 5 591
ScreenBlanker_Control 6 591
ScreenBlanker_Control 7 592
ScreenModes_EnumerateAudioFormats 592
ScreenModes_Features 593
ScreenModes_ReadInfo 594
ScreenModes_ReadInfo 0 594
ScreenModes_ReadInfo 1 595
ScreenModes_ReadInfo 2 595
SDIO_ClaimDeviceVector 596
SDIO_Control 596
SDIO_Control 0 597
SDIO_Control 1 598
SDIO_Control 2 598
SDIO_Control 3 599
SDIO_Control 4 599
SDIO_Control 5 600
SDIO_Control 6 600
SDIO_ControllerFeatures 601
SDIO_Enumerate 601
SDIO_Enumerate 0 602
SDIO_Enumerate 1 602
SDIO_Initialise 603
SDIO_Initialise 0 604
SDIO_Initialise 1 604
SDIO_Op 605
SDIO_ReadRegister 606
SDIO_ReleaseDeviceVector 607
SharedCLibrary_LibInitAPCS_32 607
SharedCLibrary_LibInitAPCS_A 609
SharedCLibrary_LibInitAPCS_R 610
SharedSound_HandlerInfo 611
SharedSound_HandlerVolume 611
SharedSound_InstallDriver 612
SharedSound_InstallHandler 612
SharedSound_RemoveHandler 615
SharedSound_SampleRate 615
Shell_Create 615
Shell_Destroy 616
Socket_Accept 616
Socket_Accept_1 616
Socket_Bind 617
Socket_Close 617
Socket_Connect 618
Socket_Creat 618
Socket_Getpeername 619
Socket_Getpeername_1 619

Socket_Getsockname 620
Socket_Getsockname_1 620
Socket_Getsockopt 621
Socket_Gettsize 621
Socket_lcntl 622
Socket_Listen 622
Socket_Read 623
Socket_Readv 623
Socket_Recv 624
Socket_Recvfrom 624
Socket_Recvfrom_1 625
Socket_Recvmsg 625
Socket_Recvmsg_1 626
Socket_Select 626
Socket_Send 627
Socket_Sendmsg 627
Socket_Sendmsg_1 628
Socket_Sendto 628
Socket_Setsockopt 629
Socket_Shutdown 629
Socket_Stat 630
Socket_Sysctl 630
Socket_Version 631
Socket_Write 631
Socket_Writev 632
Sound_AttachNamedVoice 632
Sound_AttachVoice 633
Sound_Configure 633
Sound_Control 634
Sound_ControllerInfo 634
Sound_ControlPacked 635
Sound_Enable 635
Sound_EnumerateControllers 636
Sound_InstallVoice 636
Sound_InstallVoice 0 637
Sound_InstallVoice 0,0 637
Sound_InstallVoice 1 637
Sound_InstallVoice 2 638
Sound_InstallVoice 3 638
Sound_InstallVoice V 639
Sound_LinearHandler 639
Sound_LinearHandler 0 640
Sound_LinearHandler 1 640
Sound_LogScale 641
Sound_Mode 641
Sound_Mode 0 642
Sound_Mode 1 642
Sound_Mode 3 643
Sound_Pitch 643
Sound_QBeat 644
Sound_QFree 644
Sound_QInit 645

Sound_QInterface 645
Sound_QRemove 645
Sound_QSchedule 645
Sound_QSDispatch 646
Sound_QTempo 646
Sound_ReadControlBlock 646
Sound_ReadSysInfo 647
Sound_RemoveVoice 647
Sound_SampleRate 648
Sound_SampleRate 0 648
Sound_SampleRate 1 649
Sound_SampleRate 2 649
Sound_SampleRate 3 650
Sound_SelectDefaultController 650
Sound_SoundLog 651
Sound_Speaker 651
Sound_Stereo 652
Sound_Tuning 652
Sound_Volume 653
Sound_WriteControlBlock 654
SoundCtrl_ExamineMixer 654
SoundCtrl_GetMix 655
SoundCtrl_SetMix 655
Squash_Compress 656
Squash-Decompress 656
Super_Sample45 657
Super_Sample90 657
TaskManager_EnumerateTasks 658
TaskManager_Shutdown 658
TaskManager_StartTask 659
TaskManager_TaskNameFromHandle 659
TaskWindow_TaskInfo 660
Territory_Alphabet 660
Territory_AlphabetIdentifier 660
Territory_AlphabetNumberToName 661
Territory_CharacterPropertyTable 661
Territory_Collate 662
Territory_ControlTable 662
Territory_ConvertDateAndTime 663
Territory_ConvertOrdinalsToTime 663
Territory_ConvertStandardDate 664
Territory_ConvertStandardDateAndTime 664
Territory_ConvertStandardTime 665
Territory_ConvertTextToString 665
Territory_ConvertTimeFormats 666
Territory_ConvertTimeStringToOrdinals 667
Territory_ConvertTimeToOrdinals 667
Territory_ConvertTimeToUTCOrdinals 668
Territory_DaylightRules 668
Territory_DaylightRules 0 669
Territory_DaylightRules 1 669
Territory_DaylightSaving 670

Territory_DaylightSaving 0 670
Territory_DaylightSaving 1 671
Territory_DaylightSaving 2 671
Territory_DaylightSaving 3 672
Territory_DaylightSaving 4 672
Territory_DaylightSaving 5 673
Territory_Deregister 673
Territory_Exists 674
Territory_IME 674
Territory_LowerCaseTable 674
Territory_NameToNumber 675
Territory_Number 675
Territory_NumberToName 675
Territory_PlainTable 676
Territory_ReadCalendarInformation 676
Territory_ReadCurrentTimeZone 677
Territory_ReadSymbols 677
Territory_ReadTimeZones 679
Territory_Register 679
Territory_RepresentationTable 680
Territory_Select 680
Territory_SelectAlphabet 681
Territory_SelectKeyboardHandler 681
Territory_SetTime 681
Territory_TransformString 682
Territory_UpperCaseTable 682
Territory_ValueTable 683
Territory_WriteDirection 683
URL_Deregister 684
URL_EnumerateProxies 685
URL_EnumerateSchemes 686
URL_GetURL 687
URL_ParseURL 688
URL_ProtocolDeregister 691
URL_ProtocolRegister 692
URL_ReadData 693
URL_Register 694
URL_SetProxy 694
URL_Status 696
URL_Stop 697
USBDriver_DeregisterBus 697
USBDriver_InsertTransfer 698
USBDriver_RegisterBus 698
USBDriver_ScheduleSoftInterrupt 699
USBDriver_TransferComplete 699
USBDriver_Version 699
VFPSupport_ActiveContext 700
VFPSupport_ChangeContext 700
VFPSupport_CheckContext 701
VFPSupport_CreateContext 701
VFPSupport_DestroyContext 702
VFPSupport_ElementaryFunctions 703

VFPSupport_ExamineContext 704
VFPSupport_ExceptionDump 705
VFPSupport_FastAPI 706
VFPSupport_Features 706
VFPSupport_Features 0 707
VFPSupport_Features 1 707
VFPSupport_Features 2 708
VFPSupport_Features 3 709
VFPSupport_Version 709
VideoOverlay_Create 710
VideoOverlay_Destroy 711
VideoOverlay_DiscardBuffer 711
VideoOverlay_DisplayBuffer 711
VideoOverlay_MapBuffer 712
VideoOverlay_RedrawWindow 712
VideoOverlay_SetPosition 713
VideoOverlay_SetScale 713
VideoOverlay_SetWindow 714
VideoOverlay_UnmapBuffer 714
VideoOverlay_Vet 714
Wimp_AddMessages 715
Wimp_AutoScroll 716
Wimp_BaseOfSprites 716
Wimp_BlockCopy 717
Wimp_ClaimFreeMemory 717
Wimp_CloseDown 718
Wimp_CloseTemplate 718
Wimp_CloseWindow 719
Wimp_CommandWindow 719
Wimp_Createlcon 720
Wimp_Createlcon (icon bar syntax) 720
Wimp_Createlcon (window syntax) 721
Wimp_CreateMenu 721
Wimp_CreateSubMenu 722
Wimp_CreateWindow 722
Wimp_DecodeMenu 723
Wimp_Deletelcon 723
Wimp_DeleteWindow 724
Wimp_DragBox 724
Wimp_Extend 724
Wimp_Extend (get validation string) 725
Wimp_Extend (read slot size) 726
Wimp_Extend (sprite suffix) 726
Wimp_Extend (sprite suffix) (RISC OS Select) 727
Wimp_Extend (stack information) 727
Wimp_Extend (window information) 728
Wimp_ForceRedraw 728
Wimp_ForceRedraw (extended syntax) 729
Wimp_ForceRedraw (mark invalid area syntax) 729
Wimp_GetCaretPosition 730
Wimp_GetlconState 730
Wimp_GetMenuState 731

Wimp_GetPointerInfo 731
Wimp_GetRectangle 732
Wimp_GetWindowInfo 732
Wimp_GetWindowOutline 733
Wimp_GetWindowState 733
Wimp_GetWindowState (extended syntax) 734
Wimp_GetWindowState (original syntax) 734
Wimp_Initialise 735
Wimp_LoadTemplate 736
Wimp_LoadTemplate (load syntax) 736
Wimp_LoadTemplate (request size syntax) 737
Wimp_OpenTemplate 738
Wimp_OpenWindow 738
Wimp_OpenWindow (extended syntax) 738
Wimp_OpenWindow (original syntax) 739
Wimp_PlotIcon 740
Wimp_Poll 741
Wimp_PollIdle 741
Wimp_ProcessKey 742
Wimp_ReadPalette 742
Wimp_ReadPixTrans 743
Wimp_ReadSysInfo 743
Wimp_RedrawWindow 744
Wimp_RegisterFilter 744
Wimp_RemoveMessages 744
Wimp_ReportError 745
Wimp_ResizeIcon 745
Wimp_SendMessage 746
Wimp_SetCaretPosition 747
Wimp_SetColour 747
Wimp_SetColourMapping 748
Wimp_SetExtent 748
Wimp_SetFontColours 749
Wimp_SetIconState 749
Wimp_SetMode 750
Wimp_SetPalette 750
Wimp_SetPointerShape 751
Wimp_SetWatchdogState 751
Wimp_SlotSize 752
Wimp_SpriteOp 752
Wimp_StartTask 753
Wimp_TextColour 753
Wimp_TextOp 754
Wimp_TransferBlock 754
Wimp_UpdateWindow 755
Wimp_WhichIcon 755

Service calls

Service_ADFSPodule 757
Service_ADFSPoduleIDE 757
Service_ADFSPoduleIDEDying 758
Service_BufferStarting 758

Service_CalibrationChanged 759
Service_ClaimFIQ 759
Service_ClaimFIQInBackground 760
Service_CloseFile 760
Service_ColourPickerLoaded 761
Service_DCIDriverStatus 761
Service_DCIFrameTypeFree 762
Service_DCIProtocolStatus 762
Service_DesktopWelcome 763
Service_DeviceDead 763
Service_DeviceFSCloseRequest 763
Service_DeviceFSDying 764
Service_DeviceFSStarting 764
Service_DiscDismounted 765
Service_DisplayChanged 765
Service_DisplayFormatHelp 766
Service_DisplayStatus 767
Service_DrawObjectDeclareFonts 768
Service_DrawObjectRender 768
Service_DynamicAreaCreate 769
Service_DynamicAreaRemove 769
Service_DynamicAreaRenummer 769
Service_EconetDying 770
Service_EnumerateFormats 770
Service_EnumerateFormats Block 771
Service_EnumerateNetworkDrivers 771
Service_EnumerateScreenModes 772
Service_EnumerateScreenModes Block 772
Service_Error 773
Service_FilerDying 774
Service_FilterManagerDying 774
Service_FilterManagerInstalled 774
Service_FindNetworkDriver 775
Service_FindProtocols 775
Service_FindProtocolsEnd 776
Service_FontsChanged 777
Service_FreewayStarting 777
Service_FreewayTerminating 777
Service_FSRedeclare 778
Service_Hardware 778
Service_Hardware 0 778
Service_Hardware 1 779
Service_Help 779
Service_IdentifyDisc 780
Service_IdentifyFormat 781
Service_IIC 781
Service_IIC 0 782
Service_International 782
Service_International 0 783
Service_International 1 783
Service_International 2 784
Service_International 3 784

Service_International 332 785
Service_International 4 785
Service_International 5 786
Service_International 6 786
Service_International 7 787
Service_International 8 787
Service_International 9 788
Service_InternetStatus 789
Service_InvalidateCache 789
Service_KeyHandler 790
Service_LookupFileType 790
Service_Memory 791
Service_MemoryMoved 791
Service_MessageFileClosed 792
Service_MIDI 792
Service_MIDI 0 793
Service_MIDI 1 793
Service_ModeChange 794
Service_ModeChanging 794
Service_ModeExtension 795
Service_ModeFileChanged 796
Service_ModeTranslation 796
Service_ModulePostFinal 797
Service_ModulePostInit 797
Service_ModulePreInit 798
Service_MonitorLeadTranslation 799
Service_MouseTrap 799
Service_NetFS 800
Service_NetFSDying 800
Service_NetworkDriverStatus 800
Service_NewApplication 801
Service_PagesSafe 801
Service_PagesSafe64 802
Service_PagesUnsafe 802
Service_PagesUnsafe64 803
Service_PCI 804
Service_PCI 0 804
Service_PCI 1 805
Service_PDriverChanged 805
Service_PDriverGetMessages 806
Service_PDriverGetMessages Block 806
Service_PDriverStarting 806
Service_PDumperDying 807
Service_PDumperStarting 807
Service_Portable 807
Service_Portable 0 808
Service_Portable 1 808
Service_Portable 3 809
Service_Portable 4 809
Service_Portable 5 810
Service_Portable Ports 810
Service_Portable Reason Codes 810

Service_PostInit 811
Service_PostReset 811
Service_PreModeChange 811
Service_PreReset 812
Service_Print 812
Service_ProtocolDying 813
Service_ProtocolNameToNumber 813
Service_ReAllocatePorts 813
Service_ReleaseFIQ 814
Service_ResourceFSDying 814
Service_ResourceFSStarted 815
Service_ResourceFSStarting 815
Service_RTCSynchronised 816
Service_ScreenBlanked 816
Service_ScreenRestored 817
Service_SCSIAttached 817
Service_SCSIDetached 818
Service_SCSIDying 818
Service_SCSIStarting 819
Service_SDIODescribeFIC 819
Service_SDIOSlotAcquired 819
Service_SDIOSlotAttached 820
Service_SDIOSlotDetached 820
Service_SDIOSlotReleased 821
Service_SDIOWUnitAttached 822
Service_SDIOWUnitDetached 822
Service_ShutDown 823
Service_ShutdownComplete 823
Service_Sound 823
Service_Sound 10 824
Service_Sound 11 824
Service_Sound 8 825
Service_Sound 9 825
Service_Sound Reason Codes 826
Service_StartedFiler 826
Service_StartedWimp 827
Service_StartFiler 827
Service_StartUpFS 828
Service_StartWimp 828
Service_SwitchingOutputToSprite 829
Service_SyntaxError 829
Service_TaskManagerAcknowledgements 830
Service_TerritoryManagerLoaded 831
Service_TerritoryStarted 831
Service_TimeZoneChanged 831
Service_UKByte 832
Service_UKCommand 832
Service_UKCompression 833
Service_UKConfig 833
Service_UKStatus 834
Service_UKWord 834
Service_URLModule 835

Service_URLProtocolModule 835
Service_URLProtocolModule 0 836
Service_URLProtocolModule 1 836
Service_USB 837
Service_USB 0 837
Service_USB 1 838
Service_USB 3 838
Service_USB 4 839
Service_ValidateAddress 839
Service_WimpCloseDown 840
Service_WimpPalette 840
Service_WimpRegisterFilters 841
Service_WimpReportError 841
Service_WimpSaveDesktop 841
Service_WimpSpritesMoved 842

***Commands**

*Access 843
*AddApp 843
*AddFS 844
*AddMap 844
*AddTinyDir 844
*AddToRMA 845
*ADFS 845
*Alias 845
*Alphabet 846
*Alphabets 846
*Append 846
*AppPath 847
*AppSize 847
*AppSlot 848
*ARP 848
*Audio 849
*Back 849
*BackDrop 849
*Backup 850
*BASIC 850
*BASIC64 851
*BASICFPA 851
*BASICVFP 852
*BlankTime 852
*BreakClr 853
*BreakList 853
*BreakSet 853
*Build 854
*Bye 854
*Cache 855
*Canonical 855
*Cat 855
*CDDevices 856
*CDFS 856
*CDir 856

- *CDSpeed 857
- *ChangeDynamicArea 857
- *ChannelVoice 857
- *CheckMap 858
- *Close 858
- *ColourTrans 858
- *ColourTransLoadings 859
- *ColourTransMap 859
- *ColourTransMapSize 860
- *Compact 860
- *Configure 860
- *Configure ADFSBuffers 863
- *Configure ADFSDirCache 863
- *Configure AutoDST 863
- *Configure Baud 864
- *Configure Boot 864
- *Configure BootNet 865
- *Configure BST 865
- *Configure Cache 865
- *Configure Caps 866
- *Configure CDROMBuffers 866
- *Configure CDROMDrives 866
- *Configure Country 867
- *Configure Data 867
- *Configure Delay 868
- *Configure Dir 868
- *Configure Drive 868
- *Configure DST 869
- *Configure DumpFormat 869
- *Configure FileSystem 869
- *Configure Floppies 870
- *Configure FontMax 870
- *Configure FontMax1 870
- *Configure FontMax2 871
- *Configure FontMax3 871
- *Configure FontMax4 872
- *Configure FontMax5 873
- *Configure FontSize 873
- *Configure FreewayAutoAddress 874
- *Configure FS 874
- *Configure GMT 874
- *Configure HardDiscs 875
- *Configure IDEDiscs 875
- *Configure Ignore 876
- *Configure Language 876
- *Configure Lib 876
- *Configure LMNameServer 877
- *Configure LMTransport 877
- *Configure Loud 878
- *Configure Mode 878
- *Configure MonitorType 878
- *Configure MouseStep 879

- *Configure MouseType 879
- *Configure NoBoot 880
- *Configure NoCaps 880
- *Configure NoDir 880
- *Configure NoDST 881
- *Configure NoNum 881
- *Configure NoScroll 881
- *Configure Num 882
- *Configure NVidia 882
- *Configure Print 883
- *Configure PrinterBufferSize 883
- *Configure PS 884
- *Configure Quiet 884
- *Configure RamFSSize 884
- *Configure Repeat 885
- *Configure RMASize 885
- *Configure ScreenSize 885
- *Configure Scroll 886
- *Configure SCSIFSDrive 886
- *Configure ShareBoot 886
- *Configure ShCaps 887
- *Configure SoundDefault 887
- *Configure SoundSystem 888
- *Configure SpriteSize 888
- *Configure Step 888
- *Configure Sync 889
- *Configure SystemSize 889
- *Configure Territory 889
- *Configure TimeZone 890
- *Configure Truncate 890
- *Configure TV 891
- *Configure WimpAutoMenuDelay 891
- *Configure WimpDoubleClickDelay 891
- *Configure WimpDoubleClickMove 892
- *Configure WimpDragDelay 892
- *Configure WimpDragMove 892
- *Configure WimpFlags 893
- *Configure WimpFont 893
- *Configure WimpMenuDragDelay 894
- *Configure WimpMode 894
- *Continue 894
- *Copy 895
- *CopyBoot 895
- *Count 896
- *Countries 896
- *Country 897
- *Create 897
- *Debug 897
- *Defect 898
- *Delete 898
- *DeskFS 899
- *Desktop 899

- *Desktop_ 899
- *DeviceInfo 900
- *DHCPExecute 900
- *DHCPInfo 901
- *DimTime 901
- *Dir 901
- *Dismount 902
- *DMAChannels 902
- *Do 902
- *DOSMap 903
- *Drive 903
- *Dump 903
- *Echo 904
- *Eject 904
- *EnumDir 904
- *Error 905
- *Eval 905
- *Ex 906
- *Exec 906
- *FileInfo 906
- *Filer_Action 907
- *Filer_Boot 907
- *Filer_CloseDir 907
- *Filer_DClickHold 908
- *Filer_Layout 908
- *Filer_OpenDir 909
- *Filer_Options 909
- *Filer_Run 910
- *Filer_Truncation 910
- *Filer_Visibility 911
- *Filters 911
- *FontCat 911
- *FontInstall 912
- *FontLibrary 912
- *FontList 912
- *FontMerge 913
- *FontRemove 913
- *Format 913
- *Free 914
- *FreePool 914
- *FreezeTime 914
- *FS 915
- *FSLock_ChangePassword 915
- *FSLock_Lock 916
- *FSLock_Status 916
- *FSLock_Unlock 916
- *FwAddNet 917
- *FwShow 917
- *FX 917
- *Go 918
- *GOS 918
- *GPIODevices 919

- *Help 919
- *HOff 920
- *HOn 920
- *I am 920
- *IconSprites 921
- *IF 921
- *IfConfig 921
- *IfThere 922
- *Ignore 923
- *IICDevices 923
- *InetGateway 923
- *InetInfo 924
- *Info 924
- *InitStore 925
- *Install_CheckSystem 925
- *Install_DeMerge 925
- *Install_Log 926
- *Install_LogDir 926
- *Install_Merge 927
- *Install_NoLogDir 928
- *Install_Remove 929
- *Install_Update 929
- *Key 930
- *Keyboard 930
- *LanMan 930
- *LCat 931
- *LEx 931
- *Lib 931
- *List 932
- *ListFS 932
- *ListPS 932
- *Load 933
- *LoadCMOS 933
- *LoadFontCache 933
- *LoadModeFile 934
- *Lock 934
- *Logon 935
- *Map 935
- *Memory 935
- *MemoryA 936
- *MemoryI 936
- *MimeMap 937
- *MixVolume 937
- *ModelList 938
- *Modules 938
- *Mount 939
- *NameDisc 939
- *NameDisk 939
- *Net 940
- *NetMap 940
- *NetProbe 940
- *NetStat 941

- *Networks 941
- *NoDir 941
- *NoLib 942
- *NoURD 942
- *Obey 942
- *Opt 1 943
- *Opt 4 943
- *Pass 944
- *Path 944
- *PathMacro 944
- *PCIDevices 945
- *PCIInfo 945
- *Pin 946
- *Pinboard 946
- *PinboardOptions 946
- *Ping 947
- *PipeCopy 948
- *Play 948
- *PlayList 948
- *PlayMSF 949
- *PoduleLoad 949
- *Podules 949
- *PoduleSave 950
- *Pointer 950
- *PowerOff 950
- *PrepPath 951
- *Print 951
- *PS 951
- *QSound 952
- *Quit 952
- *RAM 953
- *ReadMimeMap 953
- *Remove 953
- *RemoveTinyDir 954
- *RemPath 954
- *Rename 954
- *Render 955
- *Repeat 955
- *ResolverConfig 956
- *ResourceFS 956
- *RMClear 956
- *RMEnsure 957
- *RMFaster 957
- *RMInsert 957
- *RMKill 958
- *RMLoad 958
- *RMReInit 959
- *RMRun 959
- *RMTidy 959
- *ROMModules 960
- *Route 960
- *RouteDTraceOff 961

- *RouteDTraceOn 961
- *Run 961
- *SafeLogon 962
- *SATADevices 962
- *Save 962
- *SaveCMOS 963
- *SaveFontCache 963
- *SaveModeFile 964
- *SChoose 964
- *SCopy 965
- *ScreenLoad 965
- *ScreenSave 965
- *SCSI 966
- *SCSIDevices 966
- *SCSIsoftUSB_PopUpDelay 966
- *SDelete 967
- *SDFS 967
- *SDIODEvices 967
- *SDIOSlots 968
- *SDisc 968
- *Set 969
- *SetEval 969
- *SetMacro 969
- *SetPS 970
- *SetStation 970
- *SetType 971
- *SFlipX 971
- *SFlipY 971
- *SGet 972
- *Shadow 972
- *Share 972
- *ShareFS 973
- *ShareFSIcon 973
- *ShareFSLogoff 973
- *ShareFSLogon 974
- *ShareFSWindow 974
- *Shares 974
- *ShellCLI 975
- *ShellCLI_Task 975
- *ShellCLI_TaskQuit 975
- *Show 976
- *ShowFPRegs 976
- *ShowFree 977
- *ShowRegs 977
- *ShowVFPPRegs 978
- *ShrinkRMA 978
- *Shut 978
- *ShutDown 979
- *SInfo 979
- *SList 980
- *SLoad 980
- *SMerge 980

- *SNew 981
- *Sound 981
- *SoundGain 982
- *Speaker 982
- *Spool 982
- *SpoolOn 983
- *SRename 983
- *SSave 984
- *Stamp 984
- *StartDesktopTask 984
- *Status 985
- *Stereo 985
- *Stop 985
- *Supported 986
- *TaskWindow 986
- *Tempo 987
- *Territories 987
- *Territory 987
- *Time 988
- *ToolSprites 988
- *Tuning 988
- *TV 989
- *Type 989
- *Unalias 990
- *Unlock 990
- *Unplug 990
- *Unset 991
- *UnShare 991
- *Up 991
- *URD 992
- *URLProtoShow 992
- *USB Buses 992
- *USBConflInfo 992
- *USBDevices 993
- *USBDevInfo 993
- *USBQuirk 993
- *USBReset 993
- *USBSetConfig 993
- *USBSetInterface 994
- *VCGenCmd 994
- *Verify 994
- *Voices 995
- *Volume 995
- *Where 995
- *WhichDisc 996
- *WimpKillSprite 996
- *WimpMode 996
- *WimpPalette 997
- *WimpScroll 997
- *WimpSlot 998
- *WimpTask 998
- *WimpVisualFlags 999

*WimpWriteDir 1000

*Wipe 1000

*X 1001

Messages

Message_AlarmGoneOff 1002

Message_AlarmSet 1002

Message_AppControl 1003

Message_ClaimEntity 1003

Message_ClaimEntity Flags 1003

Message_ColourPickerCloseDialogueRequest 1003

Message_ColourPickerColourChanged 1004

Message_ColourPickerColourChoice 1004

Message_ColourPickerOpenParentRequest 1004

Message_DataLoad 1005

Message_DataLoadAck 1005

Message_DataOpen 1006

Message_DataRequest 1006

Message_DataSave 1007

Message_DataSaveAck 1007

Message_DataSaved 1008

Message_DeviceClaim 1008

Message_DeviceInUse 1009

Message_DragClaim 1009

Message_DragClaim Flags 1009

Message_Dragging 1010

Message_Dragging Flags 1010

Message_FilerAction 1010

Message_FilerAddSelection 1011

Message_FilerCloseDir 1011

Message_FilerControlAction 1011

Message_FilerOpenDir 1012

Message_FilerOpenDirAt 1012

Message_FilerOpenDirAt ViewMode 1012

Message_FilerSelection 1013

Message_FilerSelection ViewMode 1013

Message_FilerSelectionDirectory 1014

Message_FontChanged 1014

Message_FSLockStateChangeConfirm 1014

Message_FSLockStateChanged 1014

Message_FSLockStateChangeRequest 1015

Message_HelpEnable 1015

Message_HelpReply 1015

Message_HelpRequest 1016

Message_Iconize 1016

Message_IconizeAt 1016

Message_IconizeAt Flags 1017

Message_MenusDeleted 1017

Message_MenuWarning 1017

Message_ModeChange 1018

Message_Notify 1018

Message_OpenConfigWindow 1018

Message_PaletteChange 1019
Message_Plugin_Abort 1019
Message_Plugin_Action 1019
Message_Plugin_Busy 1020
Message_Plugin_Close 1021
Message_Plugin_Closed 1021
Message_Plugin_Focus 1022
Message_Plugin_Notify 1022
Message_Plugin_Open 1022
Message_Plugin_Opening 1023
Message_Plugin_Reshape 1024
Message_Plugin_Reshape_Request 1024
Message_Plugin_Status 1025
Message_Plugin_StreamAsFile 1025
Message_Plugin_StreamDestroy 1025
Message_Plugin_StreamNew 1026
Message_Plugin_StreamWrite 1027
Message_Plugin_StreamWritten 1027
Message_Plugin_Unlock 1028
Message_Plugin_URLAccess 1028
Message_PluginQuit 1029
Message_PluginQuitAbort 1029
Message_PluginQuitContinue 1029
Message_PreQuit 1030
Message_Quit 1030
Message_RAMFetch 1031
Message_RAMTransmit 1031
Message_ReleaseEntity 1032
Message_ReleaseEntity Flags 1032
Message_SaveDesktop 1032
Message_SetSlot 1033
Message_Shutdown 1033
Message_SlotSize 1033
Message_TaskCloseDown 1034
Message_TaskInitialise 1034
Message_TaskNamels 1034
Message_TaskNameRq 1035
Message_TaskStarted 1035
Message_WindowClosed 1035
Message_WindowInfo 1036

Reference topics

!Alarm 1037
!Chars 1037
!Draw 1037
!Edit 1038
!Help 1038
!Paint 1038
About RISC OS 1039
About the Programmer's Reference Manuals 1041
AcornHTTP 1041
AcornHTTP SWI Calls 1042

AcornSSL 1043
Address Exception Handler 1043
Alphabet Numbers 1044
Application Space Handler 1045
Archimedes Hardware 1045
ARM Hardware Overview 1051
ARMv7 compatibility primer 1052
AUNMsgs 1057
BASIC 1057
BASICTrans 1057
Battery Management Unit status flags 1058
Battery Management Unit variables 1058
BBCEconet 1059
BCMSupport 1060
BootNet 1061
Buffer Numbers 1061
Buffers 1061
ByteV 1062
Cache_CleanAll ARMop 1063
Cache_CleanInvalidateAll ARMop 1063
Cache_CleanInvalidateRange ARMop 1064
Cache_InvalidateAll ARMop 1064
Cache_RangeThreshold ARMop 1065
CallBack Handler 1065
CDFS Control Block 1066
CDFS Technical Details 1066
CDFSDriver 1067
CDFSDriver Technical Details 1068
Cerilica_StyledText 1070
Character Codes 1070
Character Entering Buffer Event 1071
Character Sets 1072
Close_Window_Request 1073
CnpV 1074
Colour Mapping Descriptor 1074
ColourPicker 1075
ColourTrans 1075
ColourTrans_ReadPalette Flags 1076
ColourTrans_SelectGCOLTable Flags 1076
ColourTrans_SetGCOL Flags 1076
ColourTrans_SetTextColour Flags 1076
ColourTrans_WriteCalibrationToFile Flags 1077
ColourTrans_WritePalette Flags 1077
ColourV 1077
Configure 1078
Currently Active Object Handler 1080
Data Abort Handler 1081
Date Format String Tokens 1081
Debugger 1082
Debugger Exception Dumps 1082
DeskFS 1084
DeviceFS 1084

DeviceFS Technical Details 1084
DHCP 1086
DisplayManager 1086
DMB_Read ARMop 1087
DMB_ReadWrite ARMop 1087
DMB_Write ARMop 1088
DOSFS 1088
DragASprite 1089
Draw 1089
Draw Dash Block 1090
Draw Fill Style 1090
Draw Join_Cap Block 1091
Draw Module Technical Details 1091
Draw Path Block 1094
Draw_ProcessPath Fill Style 1095
Draw_ProcessPath Special Values 1095
Draw_StrokeClipped Clip Block 1096
DSB_Read ARMop 1096
DSB_ReadWrite ARMop 1096
DSB_Write ARMop 1097
DualSerial 1098
Dynamic Area Numbers 1098
Econet Technical Details 1099
Econet Vector Reason Codes 1100
EconetV 1101
Error Handler 1101
Escape Event 1102
Escape Handler 1103
Event 0 1103
Event 1 1104
Event 10 1104
Event 11 1104
Event 12 1105
Event 14 1105
Event 15 1105
Event 16 1106
Event 17 1106
Event 19 1107
Event 21 1107
Event 22 1107
Event 28 1108
Event 3 1108
Event 4 1108
Event 5 1109
Event 7 1109
Event 8 1109
Event 9 1110
Event Numbers 1110
Events 1111
Events Introduction 1112
EventV 1113
Exception Dump Region 1113

Exception Registers Block 1114
Exit Handler 1114
File Descriptor Blocks 1115
File Formats 1115
File formats: DrawFile 1115
File formats: MusicFile 1120
File formats: Relocatable Module 1124
File formats: Squash file 1131
File formats: Utility 1132
File Server Protocol Interface 1135
File Types 1135
FileCore FS Descriptor Block 1160
FileCore Timeline 1160
FileCore_DiscOp64 Address 1161
FileCore_Features Mask 1161
FileCore_FloppyStructure Flags 1162
FileCore_MiscOp 1 Result Flags 1162
Filer 1162
Filer_Action 1163
FileSwitch 1163
FileSwitch Key Features 1164
FileSwitch Paths To Search For Files 1166
FileSwitch Use Of File Types 1167
Filing System Numbers 1168
FilterManager 1169
Flags For File Operations 1170
Floating Point Contexts 1171
Floppy Disc Formats 1171
Font_DecodeMenu Flags 1172
Font_DecodeMenu Result 1172
Font_MakeBitmap Flags 1173
Font_Paint Block 1173
Font_Paint Flags 1173
Font_Paint Special Characters 1174
Font_Paint Special Characters Parameters 1174
Font_ScanString Block 1175
Font_ScanString Flags 1175
Font_SwitchOutputToBuffer Flags 1176
FontManager 1176
Format Of Sprite 1177
Format Of Sprite Area 1178
Free 1179
Freeway 1179
Freeway Technical Details 1179
FSLock 1180
FSLock Technical Details 1181
Gain_Caret 1181
GPIO 1182
GPIO_ReadEvent 1182
GraphicsV 1183
GraphicsV 1 1185
GraphicsV 10 1185

GraphicsV 11 1186
GraphicsV 12 1186
GraphicsV 13 1187
GraphicsV 13 0 1188
GraphicsV 13 1 1188
GraphicsV 13 2 1189
GraphicsV 14 1190
GraphicsV 15 1190
GraphicsV 16 1191
GraphicsV 17 1191
GraphicsV 18 1192
GraphicsV 19 1193
GraphicsV 2 1195
GraphicsV 20 1195
GraphicsV 21 1197
GraphicsV 22 1197
GraphicsV 23 1198
GraphicsV 24 1198
GraphicsV 25 1199
GraphicsV 26 1199
GraphicsV 27 1200
GraphicsV 28 1201
GraphicsV 3 1201
GraphicsV 4 1202
GraphicsV 5 1202
GraphicsV 6 1203
GraphicsV 7 1204
GraphicsV 8 1205
GraphicsV 9 1206
HAL Hardware 1207
HAL Video API 1208
HAL_VideoBufferAlignment 1210
HAL_VideoFeatures 1210
HAL_VideoFlybackDevice 1211
HAL_VideoFramestoreAddress 1211
HAL_VideoIICOp 1211
HAL_VideoOutputFormat 1212
HAL_VideoPixelFormatList 1212
HAL_VideoPixelFormats 1213
HAL_VideoReadPaletteEntry 1214
HAL_VideoRender 1214
HAL_VideoSetBlank 1215
HAL_VideoSetDAG 1215
HAL_VideoSetInterlace 1216
HAL_VideoSetMode 1216
HAL_VideoSetPowerSave 1217
HAL_VideoStartupMode 1217
HAL_VideoUpdatePointer 1218
HAL_VideoVetMode 1219
HAL_VideoWritePaletteEntries 1219
HAL_VideoWritePaletteEntry 1220
Hardware Vectors 1220

Hourglass 1224
IMB_Full ARMop 1225
IMB_List ARMop 1225
IMB_Range ARMop 1226
Installer 1226
InsV 1227
Internal Communications 1228
International 1228
InternationalKeyboard 1229
Internet 1229
Interrupts 1229
Introduction To *Commands 1231
Introduction To Filing Systems 1233
Introduction To SWIs 1235
InverseTable 1238
IrqV 1239
Joystick 1239
Key_Pressed 1239
Keyboard ID 1240
Keyboard Scan Codes 1240
KeyV 1243
KeyV 0 1243
KeyV 1 1244
KeyV 16 1244
KeyV 2 1245
KeyV 3 1245
KeyV 4 1245
List of HAL APIs 1246
Lose_Caret 1246
Low-Level Internal Key Numbers 1247
Memory Limit Handler 1249
Menu_Selection 1250
Message Codes 1250
MessageTrans 1254
MMU_Changing ARMop 1255
MMU_ChangingEntries ARMop 1255
MMU_ChangingEntry ARMop 1256
MMU_ChangingUncached ARMop 1256
MMU_ChangingUncachedEntries ARMop 1257
MMU_ChangingUncachedEntry ARMop 1257
Monitor Types 1257
Mouse_Click 1258
NetFiler 1258
NetPrint 1259
NetStatus 1259
Null_Reason 1260
OMAP3 Hardware 1260
OMAPVideo 1262
Open_Window_Request 1267
OS Codes 1267
OS_Byte 128 Reason Codes 1268
OS_Byte 160 VDU Variable Numbers 1268

OS_Byte 221 - 228 Key Codes 1269
OS_Byte 242 Baud Rates 1270
OS_Byte CMOS Settings 1270
OS_Byte Introduction 1273
OS_ChangedBox Block 1275
OS_ClaimDeviceVector Flag 1275
OS_ConvertVariform Types 1275
OS_Find Path Bits 1276
OS_FSControl 10 Options 1276
OS_FSControl 52 Object Types 1276
OS_GBPB 10 Buffer 1277
OS_GBPB 11 Buffer 1277
OS_GBPB 12 Buffer 1278
OS_Memory 0 Flags 1278
OS_Memory 10 Flags 1279
OS_Memory 12 Flags 1279
OS_Memory 13 Flags 1280
OS_Memory 14 Flags 1280
OS_Memory 16 Flags 1280
OS_Memory 19 Input Function 1281
OS_Memory 19 Output Function 1282
OS_Memory 8 Flags 1283
OS_Memory 9 Controller Numbers 1283
OS_Memory Page Blocks 1284
OS_Memory Page Blocks (64 bit) 1284
OS_MMUControl 1 Flags 1284
OS_PlatformFeatures 0 Flags 1285
OS_PlatformFeatures 34 Flags 1286
OS_ReadArgs Examples Page 1288
OS_ReadArgs Input String 1291
OS_ReadArgs Output Buffer 1292
OS_ReadArgs Syntax String 1292
OS_ReadLine Flags 1293
OS_ReadSysInfo 6 Items 1293
OS_ReadSysInfo 8 Flags 1295
OS_ReadSysInfo 8 Platform Class 1295
OS_ReadUnsigned Flags 1296
OS_ReadVarVal Variable Types 1296
OS_SetColour Flags 1296
OS_SynchroniseCodeAreas Flags 1297
OS_Word 0 Buffer 1297
OS_Word 13 Buffer 1298
OS_Word 22 Type 1298
OS_Word 9 Input Block 1298
OS_Word 9 Output Block 1299
OS_Word Introduction 1299
PaletteV 1300
PaletteV 1 1300
PaletteV 11 1301
PaletteV 14 1301
PaletteV 15 1302
PaletteV 16 1302

PaletteV 2 1302
PaletteV 3 1303
PaletteV 4 1303
PaletteV 5 1304
PaletteV 6 1304
PaletteV 7 1304
PaletteV 8 1305
PaletteV 9 1305
PCI 1305
PCI_HardwareAddress Address Index 1306
PCI_HardwareAddress Input Flags 1306
PCI_HardwareAddress Output Flags 1306
PCI_LogicalAddress Flags 1307
PCI_MemoryRead Flags 1307
PCI_ReadInfo Request 1307
Percussion 1308
Pinboard 1308
Pointer Device Numbers 1308
Pointer_Entering_Window 1309
Pointer_Leaving_Window 1309
PointerV 1310
PointerV 0 1310
PointerV 1 1311
PointerV 2 1311
PointerV 3 1312
PointerV 4 1312
PointerV 9 1313
PollWord_NonZero 1314
Portable 1314
Portable Feature Matrix 1316
Portable_Control Mask 1319
Prefetch Abort Handler 1320
Printer Driver Numbers 1320
Processor Vector Numbers 1321
Program Environment 1321
RAMFSFiler 1322
ReadLineV 1322
Reason Codes 1323
Redraw_Window_Request 1324
RemV 1324
Resolver 1325
ResourceFiler 1325
ResourceFS 1325
RiscPC Hardware 1326
RTSupport 1330
SATADriver 1330
Screen Mode Types 1331
Screen Modes 1333
ScreenBlanker 1335
Scroll_Request 1335
SDIODriver 1336
SDIODriver Technical Details 1336

SerialDeviceDriver 1339
SerialV 1339
SeriousErrorV 1340
ShellCLI 1342
Software information: Maestro 1342
Software Vector Numbers 1343
Software Vectors 1344
Sound_ControllerInfo Property Numbers 1346
Sound_ReadSysInfo 1 1346
Sound_ReadSysInfo 2 1346
Sound_ReadSysInfo Reason Codes 1347
SoundControl 1347
SoundControl Categories 1347
SoundCtrl_ExamineMixer Block 1348
SpriteExtend SWI Calls 1348
SpriteUtils 1349
SpriteV 1349
Squash 1349
Squash_Compress Flags and Status 1350
Squash-Decompress Flags and Status 1350
SWI Error Block 1351
System Control Handlers 1352
System Device Numbers 1352
System Variables 1353
SystemDevices 1357
SystemDevices Technical Details 1357
TaskManager 1358
TaskWindow 1359
TaskWindow_Ego 1359
TaskWindow_Input 1360
TaskWindow_Morio 1360
TaskWindow_Morite 1360
TaskWindow_NewTask 1360
TaskWindow_Output 1361
TaskWindow_Resume 1361
TaskWindow_Suspend 1361
Territory Numbers 1361
TerritoryManager 1363
The Acorn Terminal Interface Protocol 1363
TickerV 1367
TLB_InvalidateAll ARMop 1367
TLB_InvalidateEntry ARMop 1367
UCS Font Manager 1368
UKPLOTV 1373
UKSWIV 1374
UKVDU23V 1374
Undefined Instruction Handler 1375
UpCall Handler 1375
UpCallV 1376
URL_Fetcher 1376
URL_Fetcher Protocol module interface 1377
USBDriver 1378

User_Drag_Box 1378
User_Message 1378
User_Message_Acknowledge 1379
User_Message_Recorded 1380
Using PCI Devices 1380
VDU 17 1381
VDU 18 1382
VDU 19 1382
VDU 23 1383
VDU 23,16 1384
VDU 24 1384
VDU 25 1385
VDU 28 1385
VDU 29 1386
VDU 31 1386
VDU Codes 1387
VDU Variables 1388
VDUXV 1389
VFPSupport 1390
VideoOverlay 1391
Wimp_TextOp 0 1391
Wimp_TextOp 1 1392
Wimp_TextOp 2 1392
Wimp_TextOp 3 1393
Wimp_TextOp 4 1394
Window 1394
WindowManager 1395
WindowManager Application Resource Files 1395
WindowManager Applications And Documents 1398
WindowManager Changing The Pointer 1400
WindowManager Colour 1400
WindowManager Dialog Boxes 1402
WindowManager Dragging 1403
WindowManager Errors 1404
WindowManager General Principles 1404
WindowManager Icons 1405
WindowManager Keyboard Input And Text Handling 1407
WindowManager Layout Of Windows 1409
WindowManager Memory Management 1411
WindowManager Menus 1412
WindowManager Mode Independence 1413
WindowManager Mouse 1413
WindowManager Overview 1414
WindowManager Polling 1415
WindowManager Redrawing Windows 1416
WindowManager Relocatable Modules 1417
WindowManager RISC OS System Icons 1417
WindowManager Shared Resources 1420
WindowManager System Font 1421
WindowManager Template Files 1421
WindowManager The Icon Bar 1422
WindowManager Time 1423

WindowManager Updating Windows 1424
WindowManager Window Panes 1424
WindowManager Window System Area 1425
WordV 1428
WrchV 1428