Contents

Introduction

Functional specs

3rd Party

RISC OS 5

RISC OS Select

Kernel

I/O

Desktop

Wimp

Hardware

Time

Networking

Graphics

Programmer

Indexes

These documents are not necessarily complete.

Consult individual documents and authors for details of their completeness.

About these documents

Introduction

The documentation in this collection is a staging area for documentation that does not yet have any home. It is intended to hold documents which are partially complete, or which are useful but yet ready for acceptance into general documentation.

Collection areas

There are different areas in this collection, which are intended to arbitrarily divide the content into their source or intended use. The structure here is not intended to be indicative of the final structure of the documentation.

Functional specs ('acorn')

This area is intended to contain functional specifications from the Acorn era which have not yet made it to PRM documentation. Functional specifications contain references to design and implementation decisions and discussion of problems with the system which are not appropriate for the reference manuals. However they are important steps along to way to creating documentation,

3rd Party documentation ('3rdparty')

This area contains 3rd party documentation from a variety of sources to provide a staging point before these documents are made more widely available.

Document information

Maintainer(s):	Gerph <gerph@gerph.org>			
History:	**Revision**	**Date**	**Author**	**Changes**
	1	29 Aug 2021	Gerph	**Initial version**
				● Created to hold cut and paste, and demonstrate indexing.
	2	07 Oct 2021	Gerph	**Added 3rd party**
Disclaimer:	© Gerph, 2021.			

Cut-and-Paste

1. Overview

1.1 This Document

This document supersedes the cut-and-paste and drag-and-drop protocol application notes [1] and [2], and specifies the cut-and-paste / drag-and-drop abilities to be added to the Wimp, plus the Clipboard module that supports the Wimp (and whose facilities will also be of use to future applications intended to support the protocols).

Key terms are defined and some familiar terms are also redefined more precisely in the glossary (§13), and the reader is recommended to read this first.

Where information is fundamentally new in this specification - non-obvious consequences of and clarifications and extensions to the existing protocol, the relevant section is underlined to bring attention to the fact.

The document may be slightly weighted towards text-handling applications, but this is because the user interface is generally more complicated in such cases. It is expected that the reader be able to extrapolate meanings to apply to any possible fundamental type of data.

1.2 Cut-and-Paste

Global-clipboard-based or cut-and-paste data transfer involves data being removed or copied from any document in the desktop to a notional "clipboard", then pasted from the clipboard into the same document, or any other document in the desktop, whether managed by the same application or not. On the way, data translation is performed in such a way as to minimise the information loss about the data.

This interface involves three operations, cut, copy and paste, which may be performed in any order. Any one data transfer will require at least two of these operations, in addition to the choice of the original selection and the destination point; the process is somewhat clumsy (and unintuitive, since the clipboard is hidden from view), so this is the least preferred technique of the two described in this document. However, it must still be provided, as it does allow some operations which cannot be achieved in any other way, and can perform other operations faster than by drag-and-drop.

1.3 Drag-and-Drop

Drag-and-drop is similar to cut-and-paste, but with the cut/copy performed by pressing a mouse button, and the paste by releasing the button at the destination. Full drag-and-drop compliance combines the features of conventional, simple drag-and-drop, as commonly used in Save dialogue boxes, with the data translation abilities of the global clipboard, and overcomes the abstract nature of the global clipboard by the displaying throughout the drag a bounding box, dithered sprite/object or representation of the insertion point.

The user interface is simpler to use - consisting typically of two mouse drags (one to select and one to move or copy), with a requirement for at most one keypress (the Shift key swaps the meaning of copying and moving). However, the programming interface is more complicated, because continuously negotiated transferable data types, destination positioning and rendering of objects and pointers are required in addition to everything needed for cut-and-paste. This complexity is possibly responsible for the low implementation rate of the protocol to date.

1.4 General

The Clipboard module is primarily to enable cut-and-paste and drag-and-drop to be reliably implemented for writable icons, which are handled automatically by the Wimp. (The fundamental problem is that the established cut-and-paste and drag-and-drop systems are Wimp-message-based, and the Wimp itself is poorly equipped to send and receive messages.)

However, the module is secondarily to handle the complex protocols on behalf of any application that chooses to do so - and in the process, producing a more uniform user interface. This is to be the preferred method of implementing the protocols in future, although in special cases, the tasks it performs can be split into four separate areas, any combination of which can be taken advantage of by the same application:

- clipboard management - supporting cut and copy operations
- clipboard procurement - supporting paste operations
- sending drag-and-drop data
- receiving drag-and-drop data

2. Outstanding Issues

2.1 Bounding Box Discrepancies

It is possible that data may have different "real life" bounding boxes in different data types - for example, a DrawFile sprite object may be transformed and/or scaled, and thus have a different bounding box to the underlying sprite. Thus if a transformed sprite object were dragged from Draw to a sprite editor window, the bounding box would not represent the final position of the sprite.

3. Technical Background

3.1 This Document

Some important pre-existing technical terms are rigorously defined in the context of cut-and-paste / drag-and-drop in the Glossary in §13, along with some new terms introduced in this document.

Where technical background information is relatively straightforward (no more than one or two short paragraphs), it is included alongside the appropriate part of §4 or §5, for ease of reference.

3.2 Previous Documents

Simple drag-and-drop operations, such as those employed by Save dialogue boxes, do not employ any inter-task negotiation during the drag, and use the plain DataSave/DataLoad protocol during the drop, as described in the Programmer's Reference Manual [3].

The Style Guide [4] first indicates selection models, then describes an overview of cut-and-paste and drag-and-drop behaviour, before referring the reader to the relevant Support Group Application Notes [1] and [2].

3.3 Previous Applications

Old-style selection model / copy-and-move implementations are still in existence, especially so in the case of text editors, (e.g. Zap, StrongEd, SrcEdit) which have often followed Edit's example. Such schemes typically involve only three actions to perform an operation (select region, position caret, move/copy keypress) rather than the four (select region, cut/copy keypress, position caret, paste keypress), but have the disadvantage that they can present both a caret and a selection to the user at the same time, which is potentially confusing. Old behaviour will remain deprecated, but new cut-and-paste / drag-and-drop applications must be able to interact with tasks that use it.

Other applications may support just cut-and-paste, or just drag-and-drop. Drag-and-drop will be the favoured technique in future, due to the simpler actions required (select region, drag region), but cut-and-paste must also be supported as well, to cater for cases when drag-and-drop cannot be used (e.g. when copying on to a menu tree).

Some aspects (e.g. pointer shape, window) of applications already written to the application note guidelines (e.g. DataPower, EasiWriter/TechWriter) are already inconsistent, due to the lack of detail in the application notes. Applications will in future be encouraged to follow the more detailed specification herein.

4.User Interface

4.1 Selection

4.1.1 Protocol

4.1.1.1 Rendering

Selection is rendered either by a recolouring an object (with its photographic negative or otherwise) or by drawing a bounding box, typically in red, and optionally with one or more "handles" for resizing and/or rotating operations as appropriate:
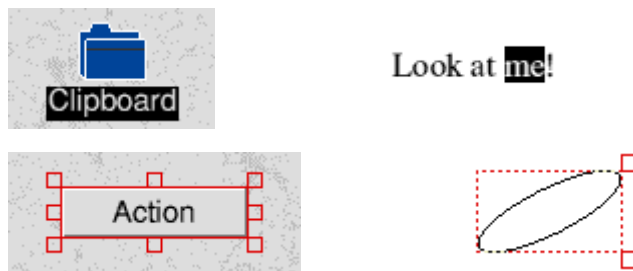
It should be emphasised that a caret must never appear within the same window as a selection at any time, not even during the selecting drag. Placing a new caret or selection removes any caret or selection previously active in the same window. (The definition of a window for these purposes is a top-level window and its panes and all their child windows.)

However, if during (or immediately after) the selection process, the selection would have to be drawn with zero width (i.e. for text selections, when the two ends snap to the same character boundary), a caret must be displayed instead. It is helpful to consider the caret as a zero-width selection; only one selection may be present within one window at a time, so the exclusivity of the caret and selection is an extension of this concept.

When either a caret or a selection is placed in a document, the window must gain the input focus. This will happen automatically for a Wimp-drawn user caret, but in the case of application-drawn carets and selections, the application must position the Wimp caret in the window, but marked as invisible, in order to achieve the same effect. Simply changing the window border colour (as is possible in Wimps since RISC OS 4) is not acceptable.

When a caret or selection is placed in a different drag-and-drop window, the old selection must be redrawn as a shaded selection, not left as is or removed entirely. The caret must be removed entirely (or optionally redrawn as a shadow caret); if the application uses a Wimp-drawn user caret, the caret will be removed for it

automatically. The LoseCaret and GainCaret events must not be used to determine when this is necessary, as the Wimp may "borrow" the caret temporarily while a menu is open. A Wimp message exists to indicate when removal or redraw of the caret and selection is necessary, and must be used in preference to the events (see §5.1.1).

Note that a selection may also be non-shaded but not have the input focus if an application not adhering to the cut-and-paste / drag-and-drop protocols had grabbed the Wimp caret (and therefore the input focus). Similarly, non-Wimp-drawn carets may be deprived of the input focus under similar circumstances while neither being removed, nor being replaced with a shadow caret.

4.1.1.2 Mouse Events

Non-contiguous selections (just about everything except text and DTP documents) will continue to be handled as described in the Style Guide [4], with the proviso that clicking Select over an already-selected object must not deselect anything, as a Select click event always precedes the Select drag event which initiates a drag-and-drop operation. Appropriate action must also be taken to un-shade shaded selections when necessary.

On the other hand, a more detailed behaviour for contiguous selections must be adhered to in future. In summary:

- Select click outside the selection (or when there is no selection, or at one end of a selection): position the caret at the pointer position, and flag the next Select drag as creating a selection.
- Select click on a selection: if the selection was shaded, un-shade it. Make sure the window has the input focus. Flag the next Select drag as being a drag-and-drop drag.
- Select click in a "dead" region of a window (e.g. in a page border): un-shade any selection or replace any shadow caret with the caret, if either exists. Make sure the window has the input focus. Optionally, flag the next Select drag as causing an window scroll operation (as Impression, TechWriter etc. do as present), but certainly not as starting a selection or drag-and-drop operation.
- Select drag-start event: remove the caret, and set the selection from the old caret position to the current pointer position. Alternatively, start the drag-and-drop operation (see §4.4.1). The exact meaning is determined by the flag that was set at the Select click stage.
- During Select drag: if creating a selection, adjust the most recently touched end of the selection to the pointer position at regular intervals; autoscroll the window if necessary, using the SWI Wimp_AutoScroll introduced in the RISC OS 4 Wimp. For what to do during a drag-and-drop drag, see

§4.4.1.

- Select double-click: select a word (as defined in the Style Guide), irrespective of whether the click is on an existing selection or not.
- Select click-drag (button pressed, then released, then pressed again within the double-click limits, then held or moved according to the drag limits): equivalent to a normal selection-delimiting drag, except that the selection limits are rounded to word boundaries (excluding whitespace at either end).
- Adjust click when there is no caret or selection: position the caret at the pointer position, unless there was a shadow caret, in which case, position the caret where the shadow caret was. Set a flag to indicate that there was no caret or selection before the Adjust click (the shadow caret doesn't count); do not rely on the fact that there is no selection displayed when the drag event is generated to flag this, as a zero-width selection may have been displayed as a caret instead.
- Adjust click when there is a caret: remove the caret, and set the selection from the old caret position to the current pointer position.
- Adjust click when there is a selection: grow or shrink the selection so that the nearest end of the selection moves to the pointer position. (Remember to un-shade the selection and/or gain the input focus if necessary.)
- Adjust click in a "dead" region of a window: the same as Select in these circumstances.
- Adjust drag-start event: unless there was no caret or selection before the preceding Adjust click, adjust the most recently touched end of the selection to the pointer position.
- During Adjust drag: unless there was no caret or selection before the preceding Adjust click, adjust the most recently touched end of the selection to the pointer position at regular intervals; autoscroll the window if necessary, using SWI Wimp_AutoScroll.

Other operations (including Adjust double-clicks, higher-multiple-clicks and combinations with shifting keys) are left to the application to respond to as it sees fit - they might select a line of text, or select something in another layer, or whatever. Typically, higher-multiple clicks of Select will select progressively larger blocks of text. For single-line items such as writable icons, three clicks means select the entire line. Once the maximum number of clicks is reached, the next click is interpreted as for a single click, so for writable icons, a quadruple-click sets the position of the caret.

In order to make selections over a larger area than can be displayed in a window, during selecting drags of contiguous selections and

select box drags of non-contiguous selections, autoscrolling can be implemented. However, since there is rarely a meaning to making a selection spanning several windows, there is only one meaning to moving the pointer off the window, and so there must be no need for a pause over the autoscrolling zone to precede commencement of scrolling, as this would merely slow down the user's actions.

4.1.1.3 Keypresses

There are some special keypresses relating to cut-and-paste that affect the selection. Obviously, these only apply to selections that have the input focus (and therefore never apply to a shaded selection). The keypresses are:

- Ctrl-Z: clear the selection (i.e. undraw the highlights), and place the caret (if appropriate) at the right-hand end of the old selection (or the left-hand end in a right-to-left language).
- Ctrl-V or Insert: delete (not cut) the selected data, and place the caret (if appropriate) at the end of the newly inserted text.
- Ctrl-X, Backspace or Delete: cut the selected data and place the caret (if appropriate) where the selection was.
- Ctrl-K: delete (not cut) the selected data and place the caret (if appropriate) where the selection was.

Then there are a number of special behaviours for textual regions:

- Left-arrow/up-arrow: clear the selection, and process the keypress as though the caret had been at the left of the selection.
- Right-arrow/down-arrow: clear the selection, and process the keypress as though the caret had been at the right of the selection.
- Any other repositioning keypresses (Home, Tab etc.) behave along similar lines, as appropriate to the application.
- Any other keypresses that would normally insert one or more characters: perform a cut operation, then position the caret where the selection was, and process the keypress as normal.

Any other keypresses must not affect the selection.

During drags (both those that set a selection and those that copy or move one), no keypresses that would normally affect the selection must be acted upon.

4.1.1.4 Scope

When a caret or selection is placed in the same window where one

already exists, the old one is removed (not just re-rendered as a shaded selection). In order for this to be consistent with the use of input focus colouring of windows, all carets and selections must be unique within a group of windows characterised thus: a top-level (non-nested) window, all its panes, and all windows nested within the window or one of its panes. If a task does its own selection handling but the window or one of its panes also uses writable icons, the task will need to monitor caret/selection updates to the writable icons in order to deselect its own selections.

Carets and selections must not be preserved when a window is closed, deleted or iconized (check for Open_Window_Requests with handle-to-open-behind of -3 to detect iconization). The Wimp takes care of everything for Wimp-drawn carets, and automatically removes the input focus in any case. If a selection can be made in a dialogue box opened from a menu, then the task must act as though the window were being closed when receiving Message_MenusDeleted, as tasks are not sent the usual Close_Window_Request for such windows.

When the window is being closed or deleted, application-drawn carets and selections must be marked as absent, but when it the window is being iconized, carets and selections drawn by the application must be flagged as a shadow caret (if supported) or a shaded selection, respectively, ready for the next redraw request.

4.1.2 Clipboard Module

The Clipboard is not involved in the selection process.

4.1.3 Writable Icons

Up to one writable icon selection may exist in each window, and the selection will only be un-shaded if the window has the input focus.

4.1.3.1 Rendering

Carets within writable icons will be Wimp colour 11 (red), irrespective of the background colour of the icon. This will be achieved by using (Wimp colour 11 EOR background colour), calculated in GCOL space, as the colour to EOR on to the icon.

Selections and shaded selections will be drawn by switching the foreground and background colours, then fading them if appropriate. A gap of 4 OS units will be left before the top and bottom borders (if any) of the icon. Therefore, a typical writable icon will look like this in its three states:
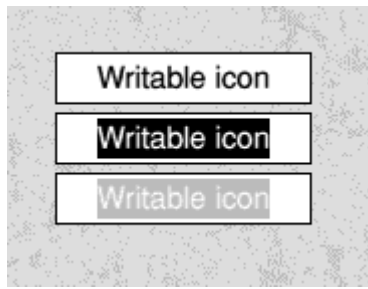
Using this method is better than EORing a block of colour, especially in the shaded selection case, where the anti-aliasing of the text is destroyed by an EOR operation. It also means that non-standard writable icons are catered for sensibly as well with no extra effort:

In the past, during writable icon redraws caused by scrolling of the icon (caused, for example, by repositioning of the caret), there has been a certain amount of flicker, both of the text, and of the caret itself, especially for large writable icons. This will be exaggerated substantially if the same technique is used to draw selections; to reduce flicker in both cases, a new algorithm will be written to deal with icon updates following caret / selection / ghost caret changes, utilising block copies wherever possible.

4.1.3.2 Scrolling

Icons where the text is less wide than the icon are relatively simple; the text has a fixed position, irrespective of caret and selection position. But it is likely that where the text is wider than the icon, occasions will arise where the user needs access to areas of the text string that are normally hidden, in order to set one or both ends of a selection. The matter is similar to the requirement for icon scrolling to position the ghost caret during a drag-and-drop selection (see §4.4).

So, while the user is delimiting a selection, or when a ghost caret is displayed in the icon, an autoscrolling scheme will be followed, directly analogous to that used for windows in Wimp_AutoScroll.

Note in particular:

● The speed of scrolling is proportional to the distance the

pointer has moved beyond the inside edge of the autoscrolling "pause" zone. This is because this scheme allows fine user control of both acceleration and deceleration.

- When delimiting a selection, autoscrolling will start as soon as the pointer enters the "pause" zone - i.e. a pause time of zero is used. Conversely, to start autoscrolling during a drag-and-drop operation, the pointer must be held over the pause zone for the configured pause time. This matches the equivalent behaviour for autoscrolling of windows.

- While document windows are generally of a comparable size, hence the similar pause zone widths, the size of writable icons can vary dramatically from icon to icon - compare, for example, a writable icon that is part of a numeric field, with the URL at the top of a web browser window. Scrolling speeds that would suit a small icon would be painfully slow for a very large one, and usable speeds for a large icon would scroll a small icon far too quickly. Therefore, the scrolling speed of a writable icon when the pointer is at one end will be proportional to its width. However, it is also desirable that the scrolling speed ramp up at the same rate, irrespective of icon size; these two constraints imply that a fixed proportion of the width of any icon needs to be allocated as the autoscroll pause zone - we will use 1/4 of the width at each end, as illustrated to scale below:
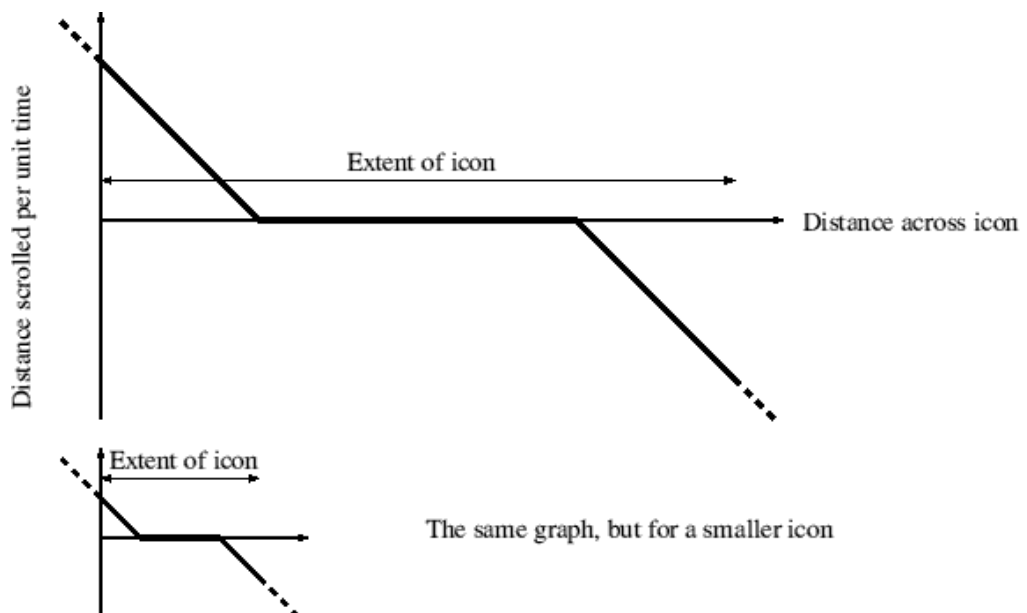
Below, the autoscrolling zone is cross-hatched; the autoscrolling pause zone is the intersection of the autoscrolling zone with the icon bounding box:
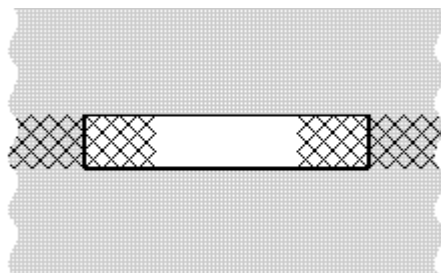
4.1.3.3 Mouse Events

Mouse events in writable icons will follow the general behaviour, as specified in §4.1.1.2, but with a couple of slight changes. The definition of a word will match that used for Shift-arrow navigation, i.e. treating both spaces and the '.' character as word delimiters.

On the second click of a double click with Adjust, the selection established by the first Adjust click will be extended outwards at both ends to include complete words.

Clicks with either Select or Adjust will not affect the text origin, unless they are setting the caret position (which will still be centred as far as possible, for consistency with old Wimps). Neither double-click operation will affect the text origin either, unless a scroll was caused by the first click of the two.

During a drag, while the pointer is over the central zone between the autoscrolling zones, no scrolling occurs. The autoscrolling zones act just like those of windows. After each scroll step (not before), the selection end is determined by the closest character boundary to the pointer.

When a drag starts, any movement of the text which was performed at the time of the click event is undone. This is necessary because otherwise we have introduced a relative movement between the text and the pointer which was not intended by the user, and the alternative (moving the pointer) is less in the style of the RISC OS user interface. Consider, for example, if the user clicks Select at the right hand end of an icon where there is a lot of text further to the right which is clipped out of view: if the user starts dragging to the left, but as a result of the initial click, the text had jumped quickly to the left of the pointer and so the user is now creating a selection to its right; worse still, if the pointer is still over the autoscrolling zone, the initial character may start scrolling off the left of the icon, leaving a large selection in the opposite direction to that intended by the user!

4.1.3.4 Keypresses

These will in general be handled as in §4.1.1.3. Note in particular:

- Only keypresses as specified in the validation string would normally insert characters, so any others (except Ctrl-X and its synonyms, of course) will not cut any selected text.
- Whenever a keypress (including Ctrl-X and synonyms) causes the caret to be repositioned, a traditional, centred caret will be used.
- When a paste is performed, and so an entirely new selection is set, the selection will be centred within the icon (unless it is wider than the icon, in which it will be right-aligned).

4.1.3.5 Wimp Selections and Menus

When the pointer moves over a writable menu item, or when a dialogue box containing writable icons is opened from a menu, the Wimp automatically places the caret in the menu item, or the first writable icon, respectively. The Wimp remembers the position of the caret before it does this, and then returns the caret to its old location afterwards.

This behaviour will be extended to check for Wimp selections that have the input focus before the caret is placed. If the same selection still exists afterwards (i.e. a selection has not been made within the menu structure in the meantime), then the input focus will be returned to it.

Note that selections cannot be made in writable menu items, as any clicks are considered as choices from the menu tree before being considered as requests to set the caret position, let alone setting a selection. Also note that drags cannot be made to an icon in a menu structure, as the click that starts the drag will close the menu structure before the drag begins!

Cut and paste will work as specified for writable icons in dialogue boxes in menu trees, and pasting (but obviously not cut or copy) will work for writable menu items.

4.1.3.6 Password icons

Cutting, copying and dragging from a password icon, or pasting or dragging into one, is not permitted for security reasons. To give the user some feedback, the Wimp issues a system beep if the user attempts to do so. Selecting text in a password icon is permitted, although the only action that can be performed on it is deletion.

4.1.3.7 Application-altered Indirected Data

On occasions, the text of a writable icon is altered by code other than the Wimp's writable icon handling code (and as a prerequisite, the text data has to be indirected). A common example of this is the

writable numeric range, where adjuster arrows may be used to alter the value inside the accompanying writable icon.

Altering the data does not, in itself, cause any screen updates to be done; applications have to force a redraw of the icon for the new value to be displayed. During the redraw, the caret is redrawn, but only using the last work-area-origin-relative co-ordinates calculated the last time the caret was positioned. If the new data requires a different text origin, the caret will then appear incorrectly positioned. To cater for this, nearly if not all applications set the position of the caret again, as well as forcing a redraw of the icon.

A similar situation could arise with selections (and even ghost carets) - but since no existing applications know about selections, they will not be able to cater for the "feature" in the API. For example, suppose the value 99 was selected in a centred numeric range, then the up-arrow was pressed; the result would be as below:

To work around this, separate checksums will be kept for the text of the icon currently containing the selection and the ghost caret. Each time an icon is redrawn, a new checksum is calculated, and if the checksum has changed, the selection or ghost caret will be removed. This is because the change to the text has probably invalidated the selected text anyway.

A variant on this approach will be used to fix the equivalent long-standing bug in the case of carets. One potential fix which we have to reject is to simply remove the caret when the text changes, because in many cases the application already has its own workaround whereby it reapplies the caret, so with each change of the text, the window's title bar would flicker due to losing and re-gaining the input focus. Instead, the Wimp will recalculate the caret position, assuming the same index into the string is required - unless the icon's numeric flag (icon flag bit 20) is set, in which the caret will be kept at the same index from the end of the string, to preserve the decimal place being edited. This way, future applications need not include the workaround at all.

4.1.3.8 Scope

In addition to the rules in §4.1.1.4., any combination of caret, ghost caret and selection must be removed when an icon is deleted. Also, when a menu is closed, any selection in a dialogue box linked to the menu must be removed (the caret already is removed in these cases).

It will not be possible for any caret, selection or ghost caret to be placed in a writable icon that is shaded. If any caret, selection or ghost caret is present in a writable icon when it becomes shaded, they will be removed.

The Wimp selection and both carets will be removed when the Wimp font is changed, but this will be the responsibility of the task that is changing the font - namely !Configure (or more precisely, a configure plug-in).

If an icon is resized using Wimp_ResizeIcon, any of the caret, ghost caret and selection which are present in the icon will be marked absent (although no redrawing will occur immediately, because Wimp_ResizeIcon expects to be followed by a separate redraw operation anyway).

4.1.3.9 Draggable-Writable (Type 14) Icons

Type 14 (draggable) writable icons are much rarer than standard, type 15 writable icons, and in the past, have only differed in that drag events are reported to the task. Some applications (such as Fresco) have taken advantage of these icons to implement a simpler form of drag-and-drop; such behaviour would be broken if the steps described above were employed for type 14 icons. Type 14 icons could also be a useful special case, where sub-units of the information in the icon have no meaning on their own, and where only the entire text can logically be dragged-and-dropped.

Therefore, all of the rest of §4.x.3 and §5.x.3 (with the exception of developments specific to carets, such as the bugfix in §4.1.3.7) will only apply to type 15 writable icons.

4.2. Cut and Copy

4.2.1. Protocol

"Cut" and "Copy" menu options, if provided, must be placed as described in the Style Guide; the options must be shaded if there is currently no selection in the window.

A cut operation must be performed when the task receives a Ctrl-X, Backspace or Delete keypress (i.e. Wimp key codes &008, &018 and &07F) or when "Cut" is chosen from the menu. When a keypress suitable for inserting data is received, or when data is dragged-and-dropped on to the selection's window, or pasted when a selection is active, the selection must also be cut prior to performing the raw operation.

A copy operation must be performed when "Copy" is chosen from the menu, or when the task receives a Ctrl-C keypress (Wimp key code

&003) - but not when Wimp key code &18B is received, because although it resulted from a press of the "Copy" key on the Archimedes, A30x0, A4000 and A5000, on all other machines it will be generated by the "End" key.

Both cut and copy will cause a copy of the selected data to be placed on the clipboard overwriting any data already there. (See §13 for a definition of clipboard in this context.) No attempt must be made to render the clipboard; it is a hidden, abstract entity. The data on the clipboard is of indeterminate data type; a data type to use for the transfer is negotiated between the clipboard owner and the pasting task at paste-time, and may involve either or both tasks performing data translation.

The only difference between cut and copy is that the selected data must be removed from the main document in the case of a cut operation. The selection remains unchanged in the case of a copy operation (i.e. it is not deselected).

If the data cut or copied to the clipboard is of type text, the newlines (if any) must be represented by ASCII &0A.

4.2.2. Clipboard Module

The use, or not, of the Clipboard module to handle cut and copy operations will not affect the cut or copy user interfaces, even though this entails some complication of the programming interface (see §5.2.2).

4.2.3. Writable Icons

Keypresses will be honoured as described in §4.2.1 - although individual icons don't and shouldn't have menus, so the description of performing cuts and copies via a menu is inappropriate. The data held in the writable icon will always be plain text, and the exported data can only be the same, so management of the clipboard can and will be delegated to the Clipboard (and as a consequence of this, a Message_ClaimEntity 4 will be issued by the Clipboard every time data is cut or copied to the clipboard from a writable icon, including those in menu structures).

4.3. Paste

4.3.1. Protocol

A"Paste" menu option, if provided, must be placed as described in the Style Guide; the option must be shaded if there is no data on the clipboard suitable for pasting into the document, even though this may entail a slight delay before opening of the submenu while the

application interrogates the current owner of the clipboard.

The keypresses Ctrl-V and Insert (Wimp key codes &016 and &1CD) are both equivalent to choosing "Paste" from the menu.

If there is a selection present in the window before the paste operation, it must be deleted before the paste takes place; swapping the clipboard contents and the selection would prevent the same data being pasted multiple times. The new data is inserted at the caret, or where the old selection was positioned, and the pasted data is automatically selected, so that the user can immediately cut it again, should it be so desired.

If the data pasted from the clipboard is of type text, any instances of ASCII &0A, &0D, or both codes adjacently in either order must be treated equally, as a single newline.

4.3.2. Clipboard Module

The use, or not, of the Clipboard module to handle paste operations will not affect the paste user interface.

4.3.3. Writable Icons

Keypresses will be honoured as described in §4.3.1 - although individual icons don't and shouldn't have menus, so the description of performing pastes via a menu is inappropriate. Handling the protocol for obtaining the pasted data will be delegated to the Clipboard.

Pasted data must be available in text (data type &FFF) form, or else the keypress will be ignored. Only text up to the first instance of ASCII &00, &0A or &0D, or the length of the spare space in the data buffer (plus the length of any selection), will be considered; if this string contains other control characters, or characters forbidden by the validation string, the operation will be faulted with a beep. In this case, no characters are inserted and any pre-existing selection is neither deselected nor deleted.

4.4. Drag

### 4.4.1. Protocol

4.4.1.1. General

When the user starts a drag-and-drop drag (which will always be with the Select button, at least for text selections), the selection is not deselected. When the drag ends, the new data is selected, which means that, unless the drag was a move operation, or the destination is in the same window as the source, the old selection will

subsequently be redrawn as a shaded selection.

If, during any drag operation, the Escape key is pressed, the drag must be aborted. Any other keypresses during a drag must be ignored (except of course for the status of the Shift key at the beginning of the drag, which is responsible for exchanging the meanings of copy and move operations).

4.4.1.2. Pointers

During a drag, the pointer shape is changed to the new standard alternative pointer shape ptr_drop; this must be used instead of the alternatives employed by DataPower, TechWriter and others. The new pointer shape will be added to the Wimp's RAM sprite pool by the Clipboard module, so that it is always available for tasks to use.

ptr_default   ptr_drop   TechWriter   DataPower
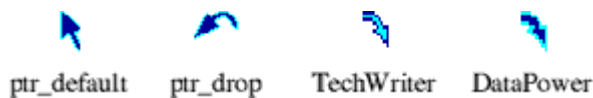
The new pointer retains the styling of the default pointer, plus the handed-ness of it, while implying a lifting operation consistent with the drop shadows added by the DragASprite and DragAnObject modules, and yet is not completely dissimilar to the existing third party pointers.

The ptr_drop pointer must remain in use throughout the drag operation, with the sole exception of during an autoscroll, when the Wimp's autoscroll pointers are used in preference - see §4.4.1.5.

4.4.1.3. Dragboxes

Linked to the pointer position, there will be at all times during the drag either a representation of the (potential) drop position, or of the original data, but not both. Which is used depends on both the sending and (potentially-) receiving tasks, and on the type of data being dragged: if the receiving task understands at least one of the the data types, it will draw the drop position; if not, the sending task is responsible for the representation of the original data.

The representation of the original data, when required, can take the form of either a rotating-dash Wimp box of the same size as the original selection, or of a DragASprite or DragAnObject rendering. As ever, if the CMOS indicates as such, a dragbox must be used instead of a DragASprite or DragAnObject drag. If a selection consisting of multiple, non-contiguous objects is to be represented without using a dragbox, the Wimp sprite "package" must be used, to match the Filer's behaviour.

Whether a dragbox is to be used or not, the representation is (of course) updated automatically by the Wimp to follow the pointer. The box, sprite or object must always keep the same position relative to the pointer's active point as the original bounding box did at the click that started the drag - except that in the special case of the "package" sprite, the sprite must always be centred over the pointer's active point.

4.4.1.4. Ghost Carets

The representation of the drop position - known as the ghost caret - has two typical forms. When the pointer is over a primarily textual region, and the task understands at least one of the available data types, the ghost caret can be displayed as a grey I-beam, "snapped" to the nearest character boundary. When the pointer is over a layout-based region, a grey bounding box, scaled according to the zoom setting of the window, can be displayed, snapped to any grid, guidelines, frame boundaries etc. as appropriate. The two are not necessarily mutually exclusive; a DTP package might, for example, want to display an I-beam when underneath a text drag, but a scaled bounding box when underneath a sprite drag. If neither form is appropriate, and the application knows of no other appropriate rendering either, the sending task must be informed (or be allowed to continue) to display the dragbox, sprite or whatever.

While the task that technically owns the drag continues to be the sending task, the receiving task is responsible for drawing any ghost caret. Therefore the ghost caret position is only updated at each pass through the underlying message protocol, approximately four times a second. In order to prevent the sending task's dragbox or sprite from coexisting with the ghost caret, and thus cluttering the target area to an unnecessary extent, provision is made in the protocol for the receiving task to request that the drag be replaced with a "drag point" (type 7) drag for as long as the receiving task is displaying a ghost caret.

In textual documents, if during the drag, the pointer is positioned over the original selection, the ghost caret must not be displayed - the dragbox must be displayed instead. This is because dragging text

inside itself has no meaning.

4.4.1.5. Scrolling

During a drag, when the pointer is over a window that can scroll, autoscrolling must be turned on using the SWI Wimp_AutoScroll. For more details of the effect of this SWI, see [5], but note that unless the pointer is held still near the edge of the window for a period, no scrolling will occur. Since determination of the pause zone is dependent upon positioning of panes etc., the activation and deactivation of Wimp_AutoScroll is the responsibility of the receiving task.

4.4.2. Clipboard Module

The use, or not, of the Clipboard module to handle drag operations will not affect the drag user interface.

4.4.3. Writable Icons

Drags from writable icons will use the ptr_drop pointer, and a rotating-dash dragbox matched in size to the selection. If the pointer has not moved since the click, the drag will initially look like this (with the dashes rotating):

Drags to writable icons (including drags within the same icon) will use an I-beam ghost caret. The ghost-caret drawing facility of the Wimp will also be made available to applications, in order to ensure that all I-beam ghost carets are drawn to the same colour and design. The colour used will be the ColourTransed version of palette entry &80808000 (50.2% grey) in order to attain maximum contrast when EORed over every possible colour.

If the pointer is dragged over the autoscrolling zone (as defined in §4.1.3.2), the icon will be scrolled in order to let the ghost caret be positioned in an out-of-sight part of the icon. This will happen even if the selection fills the icon (meaning that the ghost caret cannot be positioned anywhere in the icon) because there would be no visual clues as to why the autoscrolling was not occurring in this case.

Window autoscrolling will not be initiated while the pointer is being dragged over a writable icon. However, if autoscrolling of the icon's window is already in progress and the pointer moves on to a writable

icon, the ghost caret will not be placed in the writable icon.

4.5. Drop

4.5.1. Protocol

4.5.1.1. Sending

When a drag-and-drop drag ends, the sending task attempts to transfer data to the task under the pointer, or if a drag-and-drop dialogue was in effect, to the receiving task (which can only be different from the task under the pointer if the receiving task is autoscrolling one of its windows). The data type actually transferred is negotiated between the sending and receiving tasks at the time of the drop; it may entail either or both tasks performing data translation.

The decision whether to delete the original data when the drag ends (i.e. whether to copy or move the selection) is based upon the state of the Shift key when the drag began, and upon whether the pointer position at the end of the drag is in the same window as at the start, or not. Drags within a window move the data unless Shift is held down; drags between windows copy the data unless Shift is held down. Shift reverses the meaning of the drag, so within a region, the selection is copied, and between regions, the selection is moved.

The destination task can also insist that the operation be a move irrespective of the above; this is to allow for trashcan applications. Drags to non-drag-and-drop applications (including the Filer) are treated the same as drags to a different window.

In some circumstances, such as dropping data onto a directory viewer, the filename used in the data transfer protocols will become visible to the user. For these to be meaningful to the user, these filenames should follow the convention of concatenating the source of the data with the textual filetype, for example "IconText" or "PaintSprite".

When generating data of type text that includes newline characters, you must use ASCII &0A to terminate lines.

4.5.1.2. Receiving

To the destination task, a drop will appear the same as a non-drag-and-drop DataSave (inter-application data transfer) operation, except that the Wimp message is subtly marked (by virtue of having a non-zero your_ref field) as having resulted from previous messaging (i.e. the drag-and-drop dialogue). Assuming the task doesn't reject the data as being unsuitable, this is sufficient for the task to know what

to do with the data:

If a drag-and-drop drop,

- If a ghost caret was being displayed, the insertion point is set to the last known ghost caret position.
- Otherwise, the insertion point is set to the position specified in the message (i.e. the pointer position), snapped if necessary.

If a non-drag-and-drop drop,

- If a caret (shadow or not) or selection (shaded or not) is being displayed, the insertion point is set there.
- Otherwise, the insertion point is set to the position specified in the message, snapped if necessary.

If the insertion point thus determined lies on a selection (shaded or not), the said selection must be cut to the clipboard. (This is the only circumstance in which the clipboard is affected by a drag-and-drop operation.) The new data is inserted, and is then selected itself.

If the insertion point lies on the source selection, no actions must be taken. The selection remains selected.

Received text data must be correctly handled whether newlines (if any) are indicated using ASCII &0A, &0D, or both characters in either order.

4.5.2. Clipboard Module

The use, or not, of the Clipboard module to handle drop operations will not affect the drop user interface.

4.5.3. Writable Icons

The requirements for acceptance of dropped data are the same as for pasted data (see §4.3.3).

Text dragged from a writable icon is not terminated in any way - the "file" length determines the amount of text. The leafname used for icon-sourced text will be "IconText"; because the data transfer message handling will be delegated to the Clipboard, there will be no opportunity to change the leafname so as not to overwrite an existing file of the same name.

5. Programming Interface and Data Interchange

These two sections have been combined because any programming interfaces being specified are intimately connected to data interchange, and it makes no sense to discuss the programming interfaces before the data interchange they relate to.

5.1. Selection

5.1.1. Protocol

Handling mouse and key events relating to and rendering of selections is the responsibility of the task. The task may use Wimp_SetCaretPosition to delegate drawing of the caret, but non-I-beam carets and selections need to be drawn during window update and redraw code. To give a window the input focus without displaying the Wimp caret (for example, when setting a selection), Wimp_SetCaretPosition must be called with R4 bit 25 set.

Whenever a cut-and-paste / drag-and-drop task gains either the caret or the selection, it must broadcast the following event 17 Wimp message with both flag bits 0 and 1 set:

Message_ClaimEntity (&0000F)

This message is broadcast by a task claiming the cut-and-paste / drag-and-drop caret, selection or clipboard

Message

Offset	Contents
R1+12	your ref: 0
R1+20	Flags:

Bit(s)	Meaning
0	Caret or selection being claimed
1	Caret or selection being claimed
2	Clipboard being claimed (see §5.2.1)
3-31	Reserved, must be zero

Source

Tasks

Destination

Tasks

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

⚠ FIXME: confirm message definition attributes

A task must determine if it is gaining, or merely repositioning the caret/selection by whether any other task has broadcast a Message_ClaimEntity with bits 0 or 1 set, since the last time the task in question broadcast such a message. Note in particular, this means that a task must not consider the caret to have been lost when the Wimp caret is grabbed by a non- cut-and-paste / drag-and-drop task.

When a task receives Message_ClaimEntity with either one or both of bits 0 and 1 set, it must act as though both the caret and selection have been claimed - and therefore redraw any selection as a shaded

selection, and either redraw the caret as a shadow caret, or remove the caret entirely (the latter will be done automatically if the task was using a Wimp-drawn user caret).

Note that the Wimp does not issue this message when positioning either the caret or a selection within a menu structure.

5.1.2. Clipboard Module

The Clipboard is not involved in the selection process. However, any programs planning to rely entirely on the Clipboard to manage its cut-and-paste / drag-and-drop data transfer must still claim the caret and selection as described in §5.1.1.

5.1.3. Writable Icons

5.1.3.1. Wimp_SetCaretPosition API

Wimp_SetCaretPosition will be extended to allow the following entities to be created:

- Carets in writable icons that are not necessarily centred when the text is wider than the icon.
- "User" ghost carets - i.e. ghost carets not in an icon. *
- Ghost carets in writable icons (not necessarily centred). (See §5.4.3.)
- Selections in writable icons, centred when the text is wider than the icon. *
- Selections in writable icons, not necessarily centred when the text is wider than the icon.

Those entities above marked with an asterisk will also be made available to tasks. Any calls using the API for the others will be ignored, unless called using the internal Wimp routine. Below is the complete new Wimp_SetCaretPosition API, including the existing functionality, in a more digestible form than in the RISC OS 3 PRM. This includes the calls for internal use only; although these are internally distinguished by flags bit 28 being set, as far as the outside world is concerned, both bits 28 and 29 remain "reserved, must be zero".

Wimp_SetCaretPosition (SWI &400D2)

Set up the data for a new caret, ghost caret or selection position, and redraw it there

On entry

R0 - R6 = contents varies by operation

On exit

R0 preserved
R1 preserved
R2 preserved
R3 preserved
R4 preserved
R5 preserved
R6 undefined

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant

This is SWI and has many use cases. The table below shows the complete list of operations.

Action	Description
0	
1	
2	
3	
4	
5	
6	
7	

The versions of the indexes into the string held internally, after an icon caret is positioned by index, will in future be restricted to the range { 0 <= index <= length } rather than just { index >= 0 }. This is essentially a bugfix, and will affect the values returned by Wimp_GetCaretPosition.

The Caret Flag

Flag bits 30/31 determine which entity the call refers to, and also affect the other flag bit meanings:

Value	Entity	Meaning	
0	Caret	**Bit(s)**	**Meaning**
		0-15	height in OS units (0-65535)
		16-23	colour (bits 20-23 ignored if a Wimp colour number)
		24	use a Wimp-drawn caret rather than the Font Manager caret
		25	do not draw the I-beam (caret is invisible)
		26	use bits 16-23 for colour (else defaults to colour 11)
		27	colour is a GCOL, otherwise a Wimp colour number
		28	use both R2 and R5 to position the caret in an icon and override centring behaviour (internal use only, must be zero for external callers)
		29	Reserved, must be zero
1	Ghost Caret	**Bit(s)**	**Meaning**
		0-15	height in OS units (0-65535)
		16-23	not used; must be zero (palette entry &80808000 always used)
		24	use a Wimp-drawn caret rather than the Font Manager caret
		25-27	not used; must be zero (cannot be invisible, colour is fixed)
		28	use both R2 and R5 to position the ghost caret in an icon and override centring behaviour (internal use only, must be zero for external callers)
		29	Reserved, must be zero

Value	Entity	Meaning	
2	Selection	**Bit(s)**	**Meaning**
		0-25	not used; must be zero (height and colour determined by icon properties and bit 26)
		26	use shaded selection colour scheme (also implies that the window containing the selection should not be awarded the input focus as the result of this call)
		27	the window containing the selection should not be awarded the input focus, even if it is not shaded
		28	use both R2 and R5/R6 to position the selection in an icon and override centring behaviour (internal use only, must be zero for external callers)
		29	Reserved, must be zero
3	Reserved		

Related SWIs

SWI Wimp_GetCaretPosition (on page 51)

Wimp_SetCaretPosition 0 Remove (SWI &400D2)

To remove the caret / ghost caret / selection

On entry

R0 = -1
R2 = "TASK"
R3 = caret flags (bits other than 30 and 31 reserved, must be zero)
R4 = use bits 30 and 31 of R4 to determine which entity to remove, otherwise remove the caret

On exit

R0 preserved
R2 = 0
R3 preserved
R3 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

To remove the caret / ghost caret / selection.

Related SWIs

Wimp_SetCaretPosition 1
SetUserCaretOrUserGhostCaret
(SWI &400D2)

To set a user caret / user ghost caret:

On entry

R0 = window handle
R1 = x-offset of caret / ghost caret, relative to work area origin
R2 = y-offset of caret / ghost caret, relative to work area origin
R3 = caret flags (bits other than 30 and 31 reserved, must be zero)
R4 = height of caret, and flags

On exit

R0 preserved
R1 preserved
R2 preserved
R3 preserved
R4 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant

To remove the caret / ghost caret / selection.

Related SWIs

SWI Wimp_GetCaretPosition (on page 51)

Wimp_SetCaretPosition 2
SetIconCaretByIndex
(SWI &400D2)

To set an icon caret, centred if possible, by known index into the
string

On entry

R0 = window handle
R1 = icon handle
R4 = -1
R5 = index of caret into string (must be >= 0)

On exit

R0 preserved
R1 preserved
R4 preserved
R5 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant.

To set an icon caret, centred if possible, by known index into the
string

Related SWIs

Wimp_SetCaretPosition 3 SetIconCaretAndFlags (SWI &400D2)

To set an icon caret and override the default Y position, size or flags

On entry

R0 = window handle
R1 = icon handle
R3 = y-offset of caret, relative to work area origin
R4 = height of caret, and flags (bits 28-31 all clear)
R5 = index of caret into string (must be >= 0)

On exit

R0 preserved
R1 preserved
R3 preserved
R4 preserved
R5 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant.

To set an icon caret and override the default Y position, size or flags

Related SWIs

Wimp_SetCaretPosition 4
SetIconCaretByScreenPosition
(SWI &400D2)

To set an icon caret, centred if possible, by approximate current position on screen

On entry

R0 = window handle
R1 = icon handle
R2 = current x-offset of desired position, relative to work area origin
R3 = current y-offset of desired position, relative to work area origin
R5 = -1

On exit

R0 preserved
R1 preserved
R2 preserved
R3 preserved
R5 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant

To set an icon caret, centred if possible, by approximate current position on screen.

Note that if positioning the caret there causes the icon to scroll, the final caret position may be very different to the specified position.

Related SWIs

[SWI Wimp_GetCaretPosition (on page 51)](#)

Wimp_SetCaretPosition 5
SetIconCaretOrGhostCaret
(SWI &400D2)

To set an icon caret / icon ghost caret, not necessarily centred

On entry

R0 = window handle
R1 = icon handle
R2 = new value of caret scrollx
R3 = 0 (reserved for future expansion)
R4 = height of caret, and flags (bit 28 set, bit 30 set for ghost caret or clear for caret, bit 31 clear)
R5 = index of caret into string (must be >= 0)

On exit

R0 preserved
R1 preserved
R2 preserved
R3 preserved
R3 preserved
R5 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant

To set an icon caret / icon ghost caret, not necessarily centred. *

Related SWIs

Wimp_SetCaretPosition 6
SetIconSelectionCentred
(SWI &400D2)

To set an icon selection, centred if possible

On entry

R0 = window handle
R1 = icon handle
R4 = flags (bit 28 clear, bit 30 clear, bit 31 set)
R5 = index of lower boundary into string
R6 = index of upper boundary into string

On exit

R0 preserved
R1 preserved
R4 preserved
R5 preserved
R6 undefined

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant

To set an icon selection, centred if possible (or if the selection is
wider than the icon, right-aligned within the icon).

Note: no action is taken if R5 >= R6.

Related SWIs

[SWI Wimp_GetCaretPosition (on page 51)](#)

To set an icon selection, not necessarily centred

On entry

R0 = window handle
R1 = icon handle
R2 = new value of caret scrollx
R3 = 0 (reserved for future expansion)
R4 = flags (bit 28 set, bit 30 clear, bit 31 set)
R5 = index of lower boundary into string
R6 = index of upper boundary into string

On exit

R0 preserved
R1 preserved
R2 preserved
R3 preserved
R4 preserved
R5 preserved
R6 undefined

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant

To set an icon selection, not necessarily centred.

Note: no action is taken if R5 >= R6.

Related SWIs

5.1.3.2. Wimp_GetCaretPosition API

The complimentary SWI will be extended to allow for
Wimp_SetCaretPosition's new functionality:

Wimp_GetCaretPosition (SWI &400D3)

Returns details of the state of the caret, ghost caret or writable icon selection

On entry

R0 = if R2 = "TASK", this is the entity to inspect (0 => caret, 1 => ghost caret, 2 => selection)
R1 = block to fill with entity state
R2 = "TASK" => fill block with state of entity specified by R0 and R3, else fill block with caret state
R3 = if R0 = 2 and R2 = "TASK", this is either the handle of the window to inspect, or -1 to inspect the window which currently has the input focus and therefore also the only un-shaded selection

On exit

R0 corrupted
R1 preserved
R2 = 0 if it was "TASK" on entry
R3 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant

This call returns details of the state of the caret, ghost caret or writable icon selection.

If the caret or ghost caret state is being returned, the block is filled as follows:

Offset	Contents
R1+0	window handle (or -1 if there is no [ghost] caret)
R1+4	icon handle (or -1 if a user [ghost] caret)
R1+8	x-offset of [ghost] caret, relative to work area origin
R1+12	y-offset of [ghost] caret, relative to work area origin
R1+16	height of [ghost] caret and flags (bit 28 clear)
R1+20	index of [ghost] caret into string (undefined if a user [ghost] caret)

If the selection state is being returned, the block is filled as follows:

Offset	Contents
R1+0	window handle (or -1 if there is no writable icon selection)
R1+4	icon handle (>= 0)
R1+8	x-offset of lower boundary of selection, relative to work area origin
R1+12	width of selection
R1+16	y-offset of selection, relative to work area origin
R1+20	height of selection and flags (bit 28 clear)
R1+24	index of lower boundary into string
R1+28	index of upper boundary into string

Related SWIs

5.2. Cut and Copy

5.2.1. Protocol

When a cut or copy operation is requested of an application, it must copy the selected data to the clipboard. Under the raw protocol, each task is responsible for allocating (and deallocating) the memory necessary to store the clipboard. The clipboard must hold the data in a form from which it can be translated to the maximum number of other data types, which usually means it must be held in the application's internal format.

In order to ensure that only one clipboard is active globally at a time, it is necessary that when a cut or copy operation is performed, the cutting/copying task broadcasts a Message_ClaimEntity (see §5.1.1) with bit 2 set. Accordingly, when a task receives such a message, it must deallocate the memory used to store its own clipboard (unless of course, its own clipboard was not in use). The message must not be sent if the same task already owned the clipboard.

5.2.2. Clipboard Module

One of the Clipboard's functions is to allocate and manage memory to store the clipboard data for any participating tasks, following a cut or copy operation.

However, the Clipboard has no intrinsic knowledge of how to translate data between different formats, so it is essential that no task uses the Clipboard for this purpose if it is able to translate data on export. For example, none of Edit, Paint or Draw can exclusively use the Clipboard for clipboard storage - Edit could export Basic programs as a tokenised file, or as text; Paint can export both sprites and palettes; and Draw can export text, sprites, JPEGs and PostScript as well as DrawFiles.

Despite this, the raw protocol messaging involved at the pasting stage is still not completely trivial, and so an alternative method will be provided, whereby the task is still responsible for storing the clipboard and translating the data when required, but the Clipboard handles all the associated Wimp messaging. This also allows some code sharing with the data-send end of the drop operation.

Clipboard_Put
(SWI &4E000)

Puts data on the clipboard, or initiates the data-send of a drop

On entry

R0 = flags:

Bit(s)	Meaning
0	Clear the clipboard (must be used when the application exits, unless another task has since claimed the clipboard using a Message_ClaimEntity 4)
1	Do not store the data, just the data length (and the task handle) - when the data is required, the Clipboard will send the clipboard-owning task a Message_PutRequest stating the required data type, see §5.3.2
2	R1 is a pointer to a data type list, otherwise R1 is the data type
3-30	Reserved, must be zero
31	Flag reply messages as for the attention of the Wimp

R1 = depending on flags bit 2, either the data type (in bits 0-11), or a pointer to non-null list of data types that the task can translate the data to (in no particular order), terminated by -1

R2 = pointer to data (if flags bit 1 is clear)

R3 = data length

R4 = pointer to proposed leafname of data, null-terminated

R5 = my_ref of Message_PutRequest which this is a reply to, or 0 if this isn't a reply

On exit

R0 - R5 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI can be used for three main purposes:

- passing clipboard data to the Clipboard module to handle on the application's behalf
- passing enough information about the clipboard to the Clipboard so it can advertise on our behalf (or proxy) and get back to the application if and when a paste operation happens
- passing selection data to the Clipboard

The first and second cases can be initiated by the application, often in response to a Ctrl-C or Ctrl-X keypress. In this case, R5 will be 0. The first and third cases should be called in response to a Message_PutRequest, which is sent to the application by the Clipboard module if it called SWI Clipboard_Put (second case) or SWI Clipboard_StartDrag respectively, and a paste or drop operation (respectively) has been performed by the user.

Deleting the data in the main document following a cut operation remains the task's responsibility. If a task is maintaining its own clipboard storage area, it must release the memory when it receives a Message_ClaimEntity 4 broadcast. When a task exits, if it is maintaining the clipboard, or if Clipboard is maintaining the clipboard for it, the task must call Clipboard_Put with flags bit 0 set, for consistency with applications that do everything themselves.

The Clipboard broadcasts a Message_ClaimEntity 4 (unless the Clipboard owns the clipboard already), and takes a copy of the data, the leafname and the data type list, as appropriate. An error is generated if any of the pointers are invalid.

Related SWIs

5.2.3. Writable Icons

The Wimp itself is not a Wimp task. One of the consequences of this is that it has no task handle, and is as such not well suited to handling Wimp messages. Because of this, it will make heavy use of the Clipboard.

When the user types Ctrl-C or Ctrl-X in a writable icon, the Wimp will call Clipboard_Put with all flags clear and a data type of &FFF. The data will not be terminated; only the data length word will determine

the extent of the data.

5.3. Paste

5.3.1. Protocol

The application must first check to see if it owns the clipboard, and use the data directly if so. If it does not own it, it must broadcast a Message_DataRequest (message type 18):

Message_DataRequest (&00010)

Broadcast by a task when it wishes to paste data from a clipboard maintained by another task

Message

Offset	Contents
R1+12	your_ref: 0
R1+20	destination window handle
R1+24	internal handle to indicate destination of data; may be icon handle (see below)
R1+28	destination x co-ordinate
R1+32	destination y co-ordinate
R1+36	flags: All other bits are reserved and must be clear

Bit(s)	Meaning
2	Send data from clipboard

Offset	Contents
R1+40	list of data types in receiver's order of preference, terminated by -1 (may be null)

Source

Task

Destination

Task

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

⚠ FIXME: confirm message definition attributes

Flags bit 2 must be set when the message is sent. If the message is received with flags bit 2 clear, the message must be ignored.

If an application receiving the message owns the clipboard, it must choose the earliest data type in the list that it can provide, and if

none are possible (or the list is null) it must provide the data in its original (native) format. It must reply using the normal Message_DataSave protocol. Bytes 20-35 of the DataSave block must be copied directly from the corresponding bytes of the Message_DataRequest block (despite the discrepancy between icon handle and internal handle), while the estimated size, data type and leafname must be filled appropriately. The your_ref of the Message_DataSave must be the my_ref of the Message_DataRequest.

Be aware that if the Wimp sees an incoming DataSave with a valid icon handle at bytes 24-27 (i.e. less than the number of icons created in the windows), it will assume that it is a request to paste into that icon. Consequently, an application must be careful how it allocates its internal handles for use in this message. For example, it could use pointers into application space, which will be above &8000 and therefore very unlikely to clash with an icon handle.

When the task that initiated the paste receives the Message_DataSave, it must check the data type to ensure that it knows how to deal with it - it may be the clipboard owner's native format. If it cannot, it may back out of the transaction by ignoring the message. Otherwise, it must continue with the conventional DataSave protocol, preferably using memory data transfer.

If an application needs to find out whether there is data available to paste, but does not actually want to receive the data (e.g. in order to determine whether a "Paste" menu option should be shaded), it must broadcast a Message_DataRequest as described above. If no task replies (i.e. the message bounces) then there is no clipboard data available. If a Message_DataSave is received, then the application must ignore it (i.e. fail to reply), which will cause the operation to be silently aborted by the other task. The data type field of the Message_DataSave can then be used to determine whether the data being offered by the other task is in a suitable format to be pasted.

Related SWIs

Related messages

5.3.2. Clipboard Module

5.3.2.1. The Complete Paste Process

During the paste process, the Clipboard behaves to conventional drag-and-drop tasks exactly like any other clipboard owner, and responds to Message_DataRequests as described above.

It also provides an alternative interface to the pasting process, involving much less messaging. It involves SWI Clipboard_Get and the messages Message_PutRequest and Message_Paste. However, as before, if a task is managing its own clipboard, it must use the data directly in preference (although this will now only be in cases where the data can be translated on export).

Clipboard_Get
(SWI &4E001)

Requests data from the clipboard, using the Clipboard as a proxy

On entry

R0 = flags:

Bit(s)	Meaning
0-30	Reserved, must be zero
31	Flag reply messages as for the attention of the Wimp (this bit must only be set by the Wimp)

R1 = destination window handle
R2 = destination icon handle (-1 if none)
R3 = destination x co-ordinate
R4 = destination y co-ordinate
R5 = pointer to list of data types that the task is interested in receiving, in order of preference, terminated by -1 (may be a null list if the native format is required)

On exit

R0 - R5 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant

The Clipboard takes an internal copy of the data type list. If it owns the clipboard itself, it replies immediately with a Message_Paste. If a task has registered itself with the Clipboard using a bit-1-set SWI Clipboard_Put, the Clipboard sends a Message_PutRequest to the clipboard owner, and uses the data copied from the details in the following SWI Clipboard_Put to construct a Message_Paste. Alternatively, if a conventional drag-and-drop task owns the clipboard, the Clipboard will send a Message_DataRequest and handle all the Message_DataSave etc. messaging, before sending a Message_Paste to the pasting task, thus creating a uniform interface.

Related SWIs

Related messages

Message_PutRequest (&4E000)

Requests that clipboard or selection data be sent to the Clipboard

Message

Offset	Contents
R1+12	your_ref: 0
R1+20	flags:

Bit(s)	Meaning
0	Flags bit 0 to use in Clipboard_Put
1	Flags bit 1 to use in Clipboard_Put
2	Flags bit 2 to use in Clipboard_Put (note this also determines whether a single data type, or a data type list pointer is required in R3)
3	Send the clipboard (otherwise send the selection)
4	Delete the selection after sending the data
5-30	Reserved, must be zero
31	Message is for the attention of the Wimp, other tasks must ignore it

Offset	Contents
R1+24	destination window handle
R1+28	destination icon handle (-1 if none)
R1+32	destination x co-ordinate
R1+36	destination y co-ordinate
R1+40	pointer to list of data types that the destination task is interested in receiving, in order of preference, terminated by -1 (may be a null list if the native format is required) - now held in the Clipboard's workspace

Source

Clipboard

Destination

Task

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

⚠ FIXME: confirm message definition attributes

This message requests that clipboard or selection data be sent to the Clipboard.

Message_PutRequest is sent exclusively by the Clipboard. Any task receiving the message must reply before the next Wimp_Poll using SWI Clipboard_Put with, preserving flags bit 0-2 and 31.

The message is used both for providing the data in a paste operation (when data translation has to be delayed until paste-time) and in a drop operation, so care must be taken to send the data from either the internal clipboard or the selection, respectively. Bit 4 caters for move-drags (see §5.4.2).

The data type chosen must be the first one in the list that it can provide, or the native data type if none (or if the list is null). The data must be translated prior to calling Clipboard_Put (unless bit 2 is set), as it is at that stage that the Clipboard takes an internal copy of the data. The leafname must be built as described in §4.5.1.1.

Related SWIs

SWI Clipboard_Get (on page 60)
SWI Clipboard_Put (on page 54)

Related messages

Message_DataRequest (on page 57)
Message_DataTypeIs (on page 70)

Message_Paste (&4E001)

Informs the task being pasted to or dropped upon that the data is ready to be received

Message

Offset	Contents
R1+12	your_ref: 0
R1+20	flags:

Bit(s)	Meaning
0	Clipboard couldn't find any clipboard after a Clipboard_Get call - take no further action
1-30	Reserved, must be zero
31	Message is for the attention of the Wimp, other tasks must ignore it

Offset	Contents
R1+24	destination window handle
R1+28	destination icon handle (-1 if none) or internal handle if initiated by a Message_DataRequest
R1+32	destination x co-ordinate
R1+36	destination y co-ordinate
R1+40	data type
R1+44	pointer to data, or 0 if flag bit 0 set
R1+48	data length
R1+51	pointer to proposed leafname of data, null-terminated, or 0 if flag bit 0 set

Source

Clipboard

Destination

Task

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message informs the task being pasted to or dropped upon that the data is ready to be received.

This message is also sent exclusively by the Clipboard. The data type must first be checked for suitability, then the data must be copied before the next Wimp_Poll, as the Clipboard will free up the memory unless it was itself the owner of the clipboard (the task must not attempt to determine whether this is the case).

Note that the Clipboard stores data in its own application slot, and this is where the pointer at R1+44 lies, so in order to copy the data, you must use Wimp_TransferBlock. The Clipboard's task handle (required by Wimp_TransferBlock) may be obtained from R1+4.

Related SWIs

SWI Clipboard_Get (on page 60)
SWI Clipboard_GetDataType (on page 68)
SWI Clipboard_Put (on page 54)
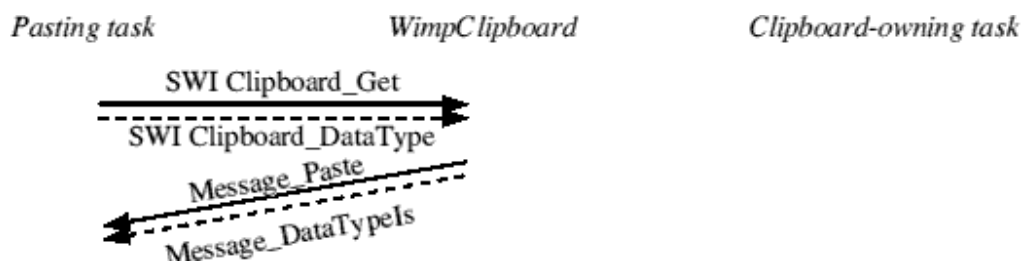
Related messages

Message_DataTypeIs (on page 70)
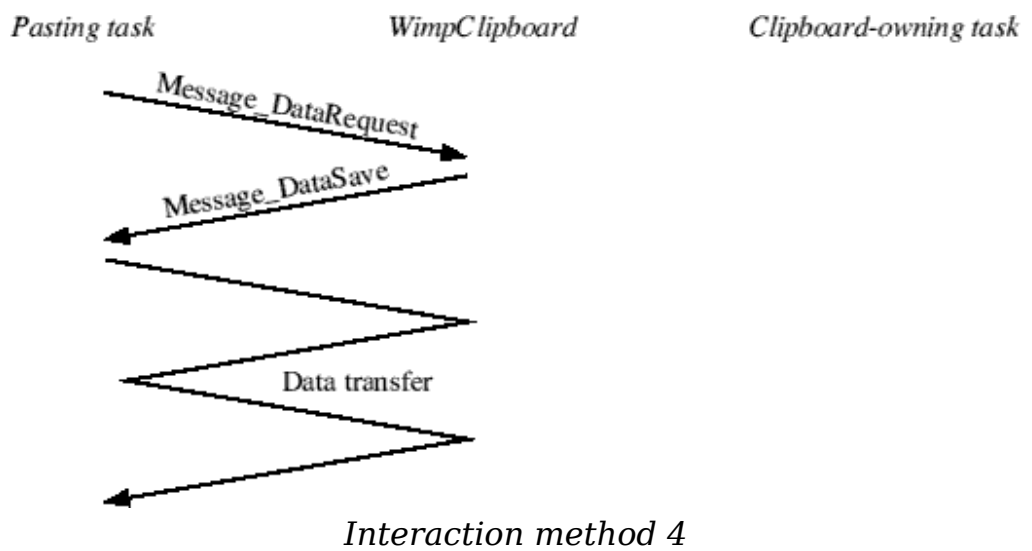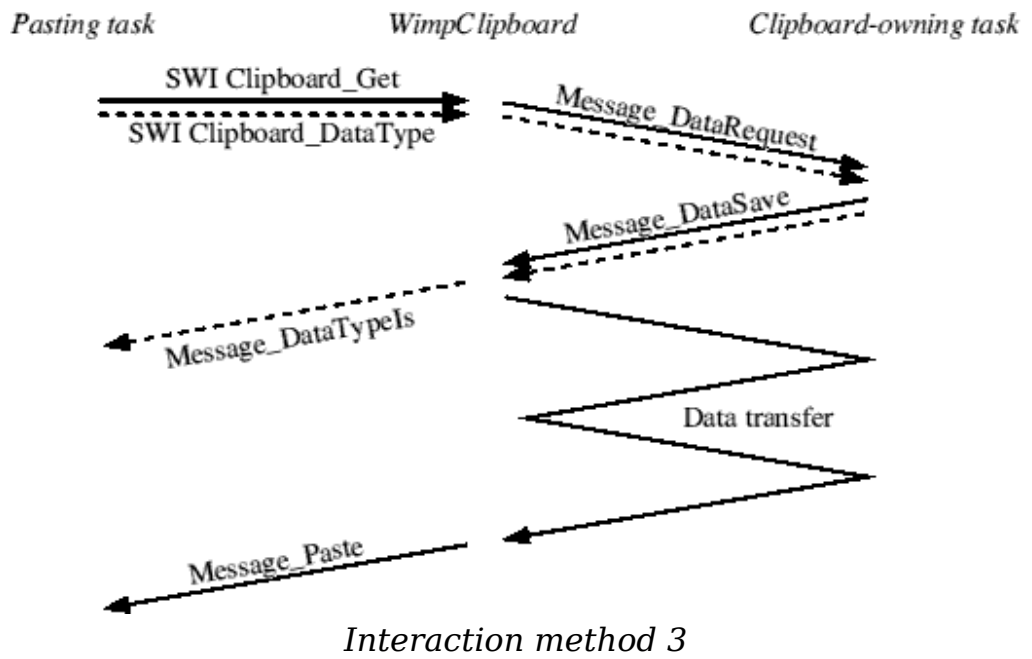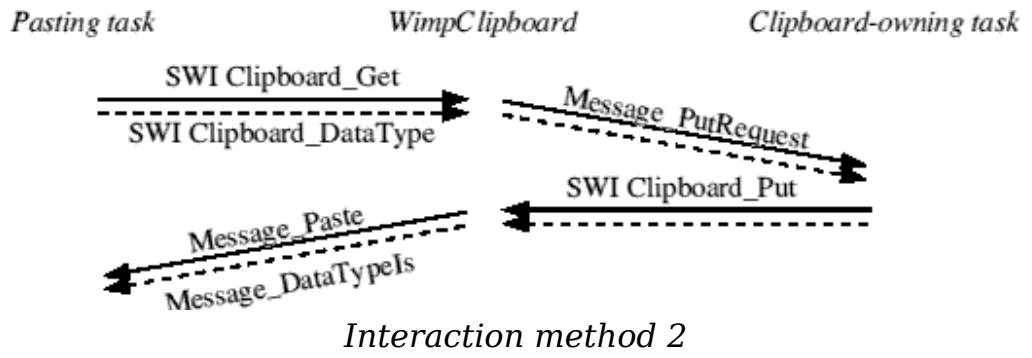Message_PutRequest (on page 62)

5.3.2.2. Interactions

The five possible interactions during a paste operation are outlined below. The solid lines refer to the complete paste process, and the dotted lines refer to clipboard data type determination, as described in §5.3.2.3. Lines are diagonal where a task switch is performed (i.e. for the sending of messages rather than the use of SWIs). Note that the "clipboard-owning task" is the task that most recently performed a cut or copy operation - strictly speaking, if the task is cooperating with the Clipboard, the Clipboard is the clipboard owner.

Interaction method 1

Pasting task   WimpClipboard   Clipboard-owning task

SWI Clipboard_Get

SWI Clipboard_DataType

Message_PutRequest

SWI Clipboard_Put

Message_Paste

Message_DataTypeIs

Interaction method 2

Pasting task   WimpClipboard   Clipboard-owning task

SWI Clipboard_Get

SWI Clipboard_DataType

Message_DataRequest

Message_DataSave

Message_DataTypeIs

Data transfer

Message_Paste

Interaction method 3

Pasting task   WimpClipboard   Clipboard-owning task

Message_DataRequest

Message_DataSave

Data transfer

Interaction method 4

Pasting task   *WimpClipboard*   *Clipboard-owning task*

Message_DataRequest

Message_PutRequest

Message_DataSave

SWI Clipboard_Put

Data transfer

Interaction method 5

In the fifth case, it is necessary for the Clipboard to ack the Message_DataRequest so that it doesn't bounce while the Message_PutRequest is delivered to the clipboard-owning task.

5.3.2.3. Clipboard Data Type Determination

Since the Clipboard is responsible for performing the traditional data transfer protocol, tasks that use the Clipboard need another way in which to determine whether they can use the current clipboard data. This will be provided by the Clipboard using the SWI Clipboard_GetDataType and the message Message_DataTypeIs.

Clipboard_GetDataType (SWI &4E002)

Requests data type of the clipboard, using the Clipboard as a proxy

On entry

R0 = flags:

Bit(s)	Meaning
0-30	Reserved, must be zero
31	Flag reply messages as for the attention of the Wimp (this bit must only be set by the Wimp, even though there are currently no plans for it to do so at present)

R1 = destination window handle

R2 = destination icon handle (-1 if none)

R3 = destination x co-ordinate

R4 = destination y co-ordinate

R5 = pointer to list of data types that the task is interested in receiving, in order of preference, terminated by -1 (may be a null list if the native format is required)

On exit

R0 - R5 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant

Requests data type of the clipboard, using the Clipboard as a proxy.

The Clipboard takes an internal copy of the data type list. If it owns the clipboard itself, it replies immediately with a

Message_DataTypeIs. If a task has registered itself with the Clipboard using a bit-1-set SWI Clipboard_Put, the Clipboard sends a Message_PutRequest to the clipboard owner, and uses the (single) data type returned in the following SWI Clipboard_Put to construct a Message_DataTypeIs. Alternatively, if a conventional drag-and-drop task owns the clipboard, the Clipboard will send a Message_DataRequest, but fail to reply to the subsequent Message_DataSave; the data type from the Message_DataSave is used in the Message_DataTypeIs, thus creating a uniform interface.

Related SWIs

SWI Clipboard_Get (on page 60)

Related messages

Message_DataTypeIs (on page 70)

Message_DataTypeIs
(&4E002)

Informs a task of the data type of the clipboard

Message

Offset	Contents
R1+12	your_ref: 0
R1+20	flags:

Bit(s)	Meaning
0	Clipboard couldn't find any clipboard after a Clipboard_GetDataType call
1-30	Reserved, must be zero
31	Message is for the attention of the Wimp, other tasks must ignore it

Offset	Contents
R1+24	destination window handle
R1+28	destination icon handle (-1 if none)
R1+32	destination x co-ordinate
R1+36	destination y co-ordinate
R1+40	data type

Source

Clipboard

Destination

Task

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

⚠ FIXME: confirm message definition attributes

This message informs a task of the data type of the clipboard (subject to the data type list passed to the preceding SWI Clipboard_GetDataType).

Related SWIs

5.3.3. Writable Icons

The Wimp will use the Clipboard to obtain data when it needs to be pasted. It will (effectively) install a temporary post-poll filter on the task owning the icon, in order to detect the flags-bit-31-set Message_Paste that follows a Clipboard_Get SWI call. The message event will not be claimed, so that a task can be kept informed about what is being done to its icons - but the task must not change the contents of the icon, because the Wimp will already have done so.

Since the Wimp will only export text from writable icons, and as such will have used the bit-1-clear version of Clipboard_Put, it will not have to respond to Message_PutRequests (except as a result of a drag-and-drop, see §5.4.2).

The Wimp will not call Clipboard_GetDataType, so need not respond to Message_DataTypeIs.

5.4. Drag and Drop

5.4.1. Protocol

During a traditional drag operation, no messaging takes place until the drop. However, during a drag-and-drop drag, a dialogue is set up between the dragging (sending) task and the potentially-receiving (claiming) task - which, in general, is the task owning the window under the pointer at any given time.

5.4.1.1. Responsibilities

The sending task's responsibilities include:

- checking the status of the Shift key at the beginning of the drag
- setting the pointer shape to ptr_drop at the beginning of the drag, resetting the pointer shape to ptr_drop when the claiming task has finished with using an alternative pointer shape, and setting it back to ptr_default at the end of the drag
- calling Wimp_DragBox, DragASprite_Start/Stop or DragAnObject_Start/Stop, as appropriate, at the beginning and end (abortion or completion) of the drag, and whenever the claiming task starts or stops requiring that the dragged object be removed from view (during such a period, a type 7 Wimp_DragBox "dragpoint" must be used instead)
- contacting the claiming task every 0.25 seconds, stating the

screen position and "real life" bounding box of the data and the data types in which it is available

- initiating the drop when the drag ends
- deleting the selected data after a successful drop if (drag was within the same window AND Shift was not held down) OR (drag was between windows or to a type-15 writable icon in any window AND Shift was held down) OR the destination is a trashcan application
- aborting the drag (and informing the claiming task as such) when the user presses Escape (which means the sending task must retain the input focus throughout the drag)

The claiming task's responsibilities include:

- updating the ghost caret according to the data passed from the sending task, provided at least one available data type can be used (and telling the sending task to remove the dragged object if a ghost caret is being displayed)
- automatically scrolling the window if the pointer is paused near the edge of a document window (and changing the pointer to reflect as such at the beginning of the autoscroll - changing it back at the end is the sending task's responsibility)
- letting the sending task know its preferred ordering of data types, so that the sending task can work out which data type to send during the drop
- letting the sending task know if it the claiming task is a trashcan (i.e. that the source data must be deleted)

A task must only claim the drag if it can do something useful with the handles and co-ordinates passed to it - typically a response would be made when the pointer is over a text area or in the autoscrolling pause zone, but not when over a "dead" area like a page border, and not when over a writable icon (except that being over an autoscrolling pause zone takes precedence over being over a dead zone or icon).

In practice, the claiming task can choose to continue to be involved in the dialogue, even when the pointer is no longer over one of its windows. This is to allow autoscrolling to continue, even when the pointer is dragged outside the window being autoscrolled (although this must not occur if the pointer has been moved smoothly over the window boundary without pausing over the window's autoscrolling activation zone). In fact, the default behaviour is for the dialogue to continue between the same two tasks, until the claiming task bows out by letting the sending task's message bounce. The claiming task, being the one handling the autoscrolling, is of course the one that knows best when this is necessary.

Message_Dragging (&00011)

This message is sent by a sending task to a (prospective) claiming task at intervals of 0.25 second, carrying context-sensitive information about the drag

Message

Offset	Contents
R1+12	your_ref: my_ref of last Message_DragClaim (or 0 if there was no claimant last time, or if this is the first Message_Dragging)
R1+20	destination window handle (constructed from Wimp_GetPointerInfo)
R1+24	destination icon handle (constructed from Wimp_GetPointerInfo)
R1+28	destination x co-ordinate (constructed from Wimp_GetPointerInfo)
R1+32	destination y co-ordinate (constructed from Wimp_GetPointerInfo)
R1+36	flags:

Bit(s)	Meaning
1	Sending data from selection (for information only, receiver must ignore)
2	Sending data from clipboard - i.e. from a clipboard-displaying window (for information only, receiver must ignore)
3	Source data will be deleted (for information only, and unfortunately is incorrect when generated by DataPower; receiver must ignore)
4	Drag is being aborted, do not respond to this message All other bits are reserved and must be clear
31	All other bits are reserved and must be clear

Offset	Contents
R1+40	xmin, ymin, xmax, ymax (4 bytes each): bounding box of data, relative to pointer, measured in millipoints (1/72000th of an inch), not scaled according to the zoom factor(s) of the source window; xmin > xmax means data has no bounding box, or bounding box is unknown
R1+56	list of available data types in no particular order, terminated by -1 (must not be null)

Source

Task

Destination

Task

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

⚠ FIXME: confirm message definition attributes

This message is sent by a sending task to a (prospective) claiming task at intervals of 0.25 second, carrying context-sensitive information about the drag.

The sending task sends a Message_Dragging, and the claiming task replies with a Message_DragClaim, as follow

Related messages

Message_DragClaim (&00012)

This message is sent by a claiming task to the sending task in response to a Message_Dragging, carrying context-sensitive information about the drag

Message

Offset	Contents
R1+12	your_ref: my_ref of last Message_Dragging
R1+20	flags:

Bit(s)	Meaning
0	A pointer shape other than ptr_drop is in use
1	Claiming task requires the absence of the Wimp dragbox / DragASprite sprite / DragAnObject object
3	Claiming task is a trashcan application, so the source data must be deleted irrespective of Message_Dragging's flags bit 3 (else deletion of the source data is determined by sending task)
	All other bits are reserved and must be clear

Offset	Contents
R1+24	list of available data types in receiver's order of preference, terminated by -1 (may be null)

Source

Task

Destination

Task

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is sent by a claiming task to the sending task in response to a Message_Dragging, carrying context-sensitive information about the drag. It must only be issued if the claiming task is interested in at least one available data type.

Related messages

Tasks are free to use internal routines to keep track of drags within or between windows that it owns, avoiding the performance overhead of all the messaging, as long as the user interface is indistinguishable from that which would result otherwise. Note in particular, that if the pointer is found to be over a type-15 writable icon, messaging must be turned back on as though the pointer was over a window owned by another task; this is so that the Wimp can collaborate in the dragging dialogue. Fortunately, the only application which currently uses this optimisation is Easi/TechWriter, where all the writable icons are in transient (or pseudo-transient) dialogue boxes, so this problem will never be visible, provided new implementations of drag-and-drop in applications follow these revised guidelines.

5.4.1.3. Use

In event-driven terms, the sending and claiming tasks must follow the behaviour outlined below in order to implement every aspect of the protocol. Explanatory comments are italicised (and are all new in this specification).

"Message type 17/18" means "use message type 17 unless drag_finished is 'true', when you must use message type 18". This is an optimisation, because we're not interested if Message_Dragging bounces from a non-drag-and-drop task, until the end of the drag, when we will want to send the data by simple data transfer.

Sending task:

- At drag start,
    - enable null events every 0.25 seconds;
    - call Wimp_DragBox (with a drag type of 5), DragASprite_Start or DragAnObject_Start as appropriate (remember to use the dragbox if CMOS states that dragged sprites/objects must not be used);
    - program pointer shape to ptr_drop;
    - set shift_pressed to indicate current status of the Shift keys;
    - set claimant to 'none' (-1 is a suitable invalid task

- handle for this purpose);
- set drag_finished to 'false';
- set drag_aborted to 'false';
- set lastref to 'none' (0 is suitable for this purpose).
- At drag abort (when Escape pressed during a drag),
  - disable null events;
  - call Wimp_DragBox -1, DragASprite_Stop or DragAnObject_Stop as appropriate (or Wimp_DragBox -1 if old_dragclaim_flags has bit 1 set);
  - program pointer shape to ptr_default;
  - set drag_finished and drag_aborted to 'true';
  - proceed as for a null event...
- At drag end (when User_Drag_Box event is received),
  - disable null events;
  - if necessary, call DragASprite_Stop or DragAnObject_Stop;
  - program pointer shape to ptr_default;
  - set drag_finished to 'true';
  - proceed as for a null event...
- At null events,
  - construct Message_Dragging using data from Wimp_GetPointerInfo and the value of drag_aborted;
  - if claimant is 'none', send message type 17/18 to window owner with your_ref = 0;
  - else, send message type 18 to claimant with your_ref = lastref.
- When Message_DragClaim is received,
  - if drag_finished is 'true',
    - drag has ended successfully while a claim is in force
    - if drag_aborted is 'false' (this comparison is not strictly necessary, since the claiming task is not supposed to reply when the drag is being aborted),
      - initiate enhanced drop operation (send Message_DataSave to claimant, using first possible data type in the list, or the native data type if none are possible, and using your_ref = my_ref of the Message_DragClaim, then delete the source data if shift_pressed and the new window/icon handles (or the trashcan flag bit in Message_DragClaim) indicate as such.
  - else,
    - drag is continuing AND (a claim is in force, or a claim is starting)
    - if lastref != 'none',
      - claim is continuing, not just starting

- if old_dragclaim_flags bit 0 is set, but the new Message_DragClaim flags bit 0 is clear, program the pointer shape to ptr_drop;
- if old_dragclaim_flags bit 1 is set, but the new Message_DragClaim flags bit 1 is clear, call Wimp_DragBox (with a drag type of 5), DragASprite_Start or DragAnObject_Start as appropriate.
- if old_dragclaim_flags bit 1 is clear, but the new Message_DragClaim flags bit 1 is set, call DragASprite_Stop or DragAnObject_Stop if necessary, then call Wimp_DragBox with a drag type of 7.
- else,
  - claim is just starting
  - if Message_DragClaim flags bit 1 is set, call DragASprite_Stop or DragAnObject_Stop if necessary, then call Wimp_DragBox with a drag type of 7.
- set claimant to task handle in Message_DragClaim;
- set lastref to my_ref of Message_DragClaim;
- set old_dragclaim_flags to flags word from Message_DragClaim.
- When Message_Dragging bounces,
  - if claimant is not 'none',
    - claimant is releasing claim (including when claimant doesn't reply because the drag is aborting)
    - if drag_finished is 'false',
      - if old_dragclaim_flags bit 0 is set, program the pointer shape to ptr_drop;
      - if old_dragclaim_flags bit 1 is set, call Wimp_DragBox (with a drag type of 5), DragASprite_Start or DragAnObject_Start as appropriate.
    - set claimant to 'none';
    - set lastref to 'none';
    - resend Message_Dragging as message type 17/ 18 to the window owner (preserving the flags, and with your_ref = 0).
  - else,
    - (no claimant is in effect AND drag has finished) OR the drag is aborting
    - if drag_finished is 'true' (this comparison is not strictly necessary, since drag_aborted also implies drag_finished),

- **if drag_aborted is 'false',**
  - initiate simple drop operation (send Message_DataSave to window owner, using native data type and your_ref = 0).

Claiming task:

- **At initialisation,**
  - set claiming to 'false'.
- **When Message_Dragging is received,**
  - **if claiming is 'false',**
    - **if flags bit 4 is clear,**
      - start claim
      - set claiming to 'true' and autoscrolling to 'false';
      - **if pointer is in the autoscroll pause zone,**
        - set pausing to 'true';
        - set old_pointer_x, old_pointer_y and old_pointer_time to the x and y from Message_Dragging and the current monotonic time;
        - program pointer to autoscroll-pause shape, and set pointer_altered to 'true';
      - **else,**
        - set pausing to 'false' and pointer_altered to 'false';
      - if the data type is suitable, draw the ghost caret (I-beam or bounding box) and set ghost_caret to 'true', else set ghost_caret to 'false';
      - reply with Message_DragClaim (message type 17), using pointer_altered and ghost_caret to determine the flags.
  - **else,**
    - **if flags bit 4 is clear AND (claiming task owns the window/icon handle in Message_Dragging OR autoscrolling is 'true'),**
      - update claim
      - **if pausing is 'true',**
        - if current pointer x or y differs from old_pointer_x or old_pointer_y, set old_pointer_x, old_pointer_y and old_pointer_time to the current pointer x and y and monotonic time;
        - if pointer has left autoscroll

pausing zone,
- set pausing and pointer_altered to 'false';
- else,
  - if (current monotonic time - old_pointer_time) >= pause_time (typically 0.5 seconds), set autoscrolling to 'true' and pausing to 'false', and program the pointer to its autoscroll-active shape;
- else if autoscrolling is 'false',
  - if pointer is in the autoscroll pause zone,
    - set pausing to 'true';
    - set old_pointer_x, old_pointer_y and old_pointer_time to the x and y from Message_Dragging and the current monotonic time;
    - program pointer to autoscroll-pause shape, and set pointer_altered to 'true';
  - else,
    - set pausing to 'false' and pointer_altered to 'false';
- else (i.e. autoscrolling is 'true'),
  - if pointer is over the window, but not in the autoscroll pause zone,
    - set autoscrolling and pointer_altered to 'false';
  - else,
    - scroll the window by an amount proportional to the distance from the pointer to the inside edge of the autoscroll pause zone;
    - if the window cannot be scrolled any further in this direction (or can be scrolled in neither direction if a 2D scroll), set autoscrolling to 'false', set pausing to 'true' and program the pointer to its autoscroll-pausing shape;
- if ghost_caret is 'true', update ghost

- caret - unless the work-area-relative position is unchanged, undraw the old ghost caret and draw the new ghost caret;
  - reply with Message_DragClaim (message type 17), using pointer_altered and ghost_caret to determine the flags.
- else,
  - release claim
  - set claiming to 'false';
  - if ghost_caret is 'true', undraw the old ghost caret;
  - let Message_Dragging bounce (i.e. don't reply to it).
- When Message_DataSave is received,
  - if you_ref != 0,
    - if claiming is 'true',
      - this was an enhanced (full drag-and-drop) drop - the claim was never released set claiming to 'false';
      - if ghost_caret is 'true', undraw the old ghost caret;
      - import data to the last ghost caret position using conventional data transfer protocol (preferably using memory data transfer).
    - else,
      - this is part of the paste protocol
      - continue as for simple drop...
  - else,
    - this was a simple drop
    - import data to position from Message_DataSave using conventional data transfer protocol (preferably using memory data transfer).

5.4.2. Clipboard Module

5.4.2.1. Use

The Clipboard module, in conjunction with the SWI Wimp_AutoScroll, will reduce the coding required to implement drag-and-drop to the following, a great improvement on §5.4.1.3. Note that, unlike the clipboard maintenance and paste protocols, the drag and drop protocols use one-to-one messages rather than broadcast messages, so the Clipboard needs to make use of filters in order to translate between the protocols. (An assumption has been made that the receiving task wishes to use an I-beam ghost caret - this does not have to be the case, but Wimp_SetCaretPosition's new facility for

drawing ghost carets requires simpler but different code from that in §5.4.1.3.)

Sending task:

- At drag start,
    - ensure sending window has the input focus;
    - call SWI Clipboard_StartDrag.
- When Message_PutRequest is received,
    - if flags bits 3 and 31 are clear,
        - this is a PutRequest for the selection, rather than the task-managed clipboard
        - translate selected data to the first possible data type in the list, or leave as the native data type if none are possible;
        - call Clipboard_Put to send the data;
        - if flags bit 4 of the Message_PutRequest was set, delete the selection.

Claiming task:

- At initialisation,
    - set claiming to 'false'.
- When Message_Dragging is received,
    - if claiming is 'false',
        - if flags bit 4 is clear,
            - start claim
            - set claiming to 'true';
            - call Wimp_AutoScroll;
            - call Wimp_SetCaretPosition to position the ghost caret if at least one available data type is suitable;
            - reply with Message_DragClaim (message type 17), with flags bit 0 clear, and flags bit 1 set if a ghost caret is being displayed.
    - else,
        - if flags bit 4 is clear AND (claiming task owns the window handle in Message_Dragging OR Wimp_AutoScroll indicates scrolling is in progress),
            - update claim
            - call Wimp_SetCaretPosition to reposition the ghost caret if at least one available data type is suitable;
            - reply with Message_DragClaim (message type 17), with flags bit 0 clear, and flags bit 1 set if a ghost caret is being displayed.
        - else,

- release claim
- set claiming to 'false';
- call Wimp_AutoScroll to deactivate autoscrolling;
- call Wimp_SetCaretPosition -1 to remove the ghost caret;
- let Message_Dragging bounce (i.e. don't reply to it).
- When Message_DataSave is received,
  - if claiming is 'true',
    - this was an enhanced (full drag-and-drop) drop - the claim was never released
    - set claiming to 'false';
    - call Wimp_AutoScroll to deactivate autoscrolling;
    - call Wimp_SetCaretPosition -1 to remove the ghost caret;
    - copy window handle, icon handle, x and y offsets from last Message_Dragging over the Message_DataSave block equivalents, and call Clipboard_CatchDrop.
  - else,
    - this was a simple drop
    - call Clipboard_CatchDrop.

5.4.2.2. Messaging

The details of the new SWIs introduced are:

Clipboard_StartDrag
(SWI &4E003)

Starts a drag-and-drop drag, using the Clipboard as a proxy

On entry

R0 = flags:

Bit(s)	Meaning
1	As Message_Dragging (= sending from selection)
2	As Message_Dragging (= sending from clipboard)
14-15	Proxy Drag Method:

Value	Meaning
0	use rotating-dash fixed-size Wimp dragbox
1	use DragASprite
2	use DragAnObject
3	reserved

Bit(s)	Meaning
16	As DragAnObject_Start, if applicable (R1 is a pointer to a routine rather than a SWI number)
17	As DragAnObject_Start, if applicable (if bit 16 is set and bit 18 clear, enter routine with R10 below R13 - note this was previously misdocumented as the routine being entered in SVC mode rather than USR mode)
18	As DragAnObject_Start, if applicable (if bit 16 is set and DragAnObject is version 0.09 or later, enter routine in USR mode rather than SVC mode)
31	Flag reply messages as for the attention of the Wimp (this bit must only be set by the Wimp)

All others are reserved and must be clear

R1 = sprite area or renderer (if DragASprite or DragAnObject, respectively)

R2 = pointer to sprite name or register/parameter block (if DragASprite or DragAnObject, respectively)

R3 = source window handle (used in combination with the Shift key state to determine when the source data needs deleting afterwards)

R4 = pointer to word-aligned block containing three bounding boxes, each made up of four 32-bit quantities held in the order xmin, ymin, xmax, ymax, where the minima are inclusive and the maxima are exclusive:

- bounding box to apply to the pointer, in OS units from the screen origin; if xmin > max then the pointer is constrained to the screen
- initial position of the dragbox/sprite/object being dragged, in OS units from the screen origin
- "real" position and size of the data to use to render the ghost caret, in millipoints (1/72000ths of an inch) relative to the pointer; if xmin > xmax then the size is unknown or undefined

R5 = data length, bytes

R6 = pointer to non-null list of data types that the task can translate
the data to (in no particular order), terminated by -1
R7 = pointer to proposed leafname of data, null-terminated

On exit

R0 - R7 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant

Starts a drag-and-drop drag, using the Clipboard as a proxy.

The Clipboard takes a copy of the data pointed to, and performs the
actions described in §5.4.1.3 (sending) on behalf of the task. In order
to achieve this, it forces the required Wimp events to be unmasked
using a pre-poll filter, then performs its main actions using a post-poll
filter; it also calls Wimp_AddMessages, so it not necessary for the
task to register interest in Message_DragClaim etc. at initialisation.
During the drag, the task will not see any user_drag_box events,
key_pressed events (except for Escape), or any DragClaim,
RAMFetch, DataSaveAck or DataLoadAck messages. If null events
were enabled in the poll mask before it was massaged by the pre-poll
filter (and, if it was a call to Wimp_PollIdle, the required time has
passed) they will also pass through to the task once the post-poll
filter has done its work.

When the drag ends (successfully or not), the filters are removed.
When the drag ends successfully, the task's cooperation is required in
order to translate the data to the required data type; this is
accomplished by the Clipboard sending it a Message_PutRequest
with flags bit 3 clear, as described in §5.3.2.1.

Related SWIs

Clipboard_CatchDrop
(SWI &4E004)

Request the Clipboard to act as a proxy for data transfer during a drop

On entry

R0 = flags:

Bit(s)	Meaning
31	Flag reply messages as for the attention of the Wimp (this bit must only be set by the Wimp)
	All others are reserved and must be clear

R1 = pointer to DataSave message block (or DataLoad message block if initiated by the Filer) that needs replying to

On exit

R0 - R1 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

⚠ FIXME: confirm irqs, fiqs, processor-mode, re-entrant

The Clipboard handles the data transfer for the task, trying memory data transfer first if possible. When the transfer is complete, the Clipboard sends a Message_Paste to the task that called this SWI, so as to appear identical to a paste operation.

It will also detect if it sent the DataSave message itself - in other words, if the sending task was using the Clipboard as a proxy too - if so, no further messaging will occur, and the Clipboard will simply use the pointer to its copy of the data made during Clipboard_Put in the

Message_Paste.

Related SWIs

5.4.3. Writable Icons

The Wimp will handle drags to and from writable icons as in §5.4.2.

The Wimp will install an internal message filter in order to listen for Clipboard messages (Message_PutRequest and Message_Paste) with flags bit 31 set; these are handled by the Wimp and not passed on to the task. Similarly, it will intercept and not pass on Message_Dragging throughout any period when it is claiming the drag. However, some tasks (such as FrontEnd) do perform useful functions when they receive a Message_DataSave, so Message_DataSave is always passed through to the task, and the Wimp will only call Clipboard_CatchDrop at the Wimp_Poll following the delivery of Message_DataSave if the task didn't call Wimp_SendMessage, Wimp_UpdateWindow or Wimp_ForceRedraw.

6. Data Formats

No new data formats are introduced.

7. Dependencies

The new Wimp provides facilities not available in earlier versions. Although the source release means that it is possible for applications that use them to require users of older Wimps to upgrade them, the authors may choose to implement them manually in cases where an older Wimp is detected. However, it is encouraged that the new Wimp facilities be utilised when possible, to accommodate future GUI changes, to allow system-wide configuration of autoscroll behaviour and so on.

The Wimp will require the Clipboard to enable functioning of drag-and-drop to or from writable icons. Without it, selections will be able to be made, but no operations (other than deletion) can be performed on them, and no selections from other applications will be inserted when dropped on a writable icon.

8. Acceptance Test

8.1. Clipboard Module

8.1.1. Compatibility

The most advanced features of RISC OS that the Clipboard will make use of are Dynamic Areas and the DragAnObject module, both introduced at RISC OS 3.50. Provision may be made for further development work to be done to support versions back to RISC OS 3.10 (by using RMA rather than Dynamic Areas, and defaulting to rotating-dash boxes when a DragAnObject drag is requested), but for the purposes of the initial release, a modern OS may be assumed.

8.1.2. Reliability/Robustness

The module must be able to handle at least a thousand consecutive operations without crashing. Null-length data, extremely long data and data close in length to a multiple of the page size must not cause problems. Bad parameters (e.g. illegal sprite area pointers) must be handled as well as possible - in the example, a rotating-dash box must be used instead.

8.1.3. Performance

Performance will unfortunately continue to be slow in certain key situations - for example, when transferring data to a conventional task by memory data transfer, where the receiving task has specified too small a buffer. Memory transfer will be slower during drops, since the data is copied twice, once to the Clipboard's application slot, and once from it. The incidences of scrapfile transfer will however be reduced, resulting in speed gains.

8.1.4. Memory Usage

The module itself must not exceed 32kB in length. Stored global clipboard data and transient data (during a drop operation) are stored in the Clipboard's application slot so that the only size limits are those of the application slot size (not a big issue with modern memory maps) and the amount of physical RAM available. The application slot shall grow and shrink so that it is no larger than the combined size of the data stored, rounded up to the next page boundary. The RMA shall be used for general heap storage (linked lists etc.).

8.2. Wimp Writable Icon Code

8.2.1. Compatibility

The writable icon code will function correctly for all tasks that follow the revised guidelines in §5 and all tasks that do not support the drag-and-drop protocol, but it may lack complete functionality (although not to the extent of rendering it useless) for up to 10% of the writable icons in existing applications written to the old application note guidelines.

8.2.2. Reliability/Robustness

The writable icon cut-and-paste / drag-and-drop code must be at least as reliable as the Clipboard module.

8.2.3. Performance

Redraw of writable icons, especially when delimiting a selection with autoscrolling active, must not cause flicker. Data transfer operations must not be appreciably slower than the Clipboard routines that are actually doing the work.

8.2.4. Memory Usage

Writable icon data is usually held in application workspace, and will not increase in size by virtue of these enhancements. A negligible amount of extra module workspace will be required to hold the details of the Wimp selection and ghost caret, this should typically be no more than 32K.

9. Non Compliances

No attempt will be made to develop an selection-drawing algorithm that can cope with overlapping icons. The appearance of such icons after scrolling and redrawing is not defined.

10. Development Test Strategy

Test applications will be written to exercise the Clipboard SWIs.

Drags to and from the existing drag-and-drop applications (e.g. DataPower, EasiWriter and TechWriter) work seamlessly. These applications will therefore be important testing tools. Also for testing purposes, a drag-and-drop trashcan application and simple clipboard-display application will be written.

(Conventional data transfer (as for example, when dropping a selection on to a Filer window) is not expected to cause significant problems, as the protocol has been clearly defined for a long time, unlike the protocol in the application notes.)

A test suite will be written to exercise the functions of the Clipboard through exercise of writable icons' cut-and-paste / drag-and-drop facilities. For example: setting of writable icon selections will be tested repeatedly, involving operations that change one or both ends of a selection (or neither) at the same time, both with and without the presence of a ghost caret. This will be done by direct calling of Wimp_SetCaretPosition. And text files of differing length, of differing line terminator and files that have been accidentally mistyped as text will be saved on to writable icons of differing validation strings, using conventional data transfer, pasting and dropping.

11. Product Organisation

This document, and the code it describes, form part of the Shared Source RISC OS release.

The APIs and messages should ideally be included in a new version of the Programmer's Reference Manual. Use of the raw protocol rather than the Clipboard module will be deprecated.

The Clipboard module can be softloaded, but must also be capable of being built into ROM.

12. Future Enhancements

None planned.

13. Glossary

Term(s) used in Document	Meaning
AND conj.	Logical AND.
Caret n.	The position in a document where typed characters or pasted clipboard contents will be placed. Many pre-drag-and-drop applications also use this position as the insertion point for dropped data, but drag-and-drop applications must use the ghost caret for this purpose instead. In textual documents, the caret is often shown by a red I-beam, but other representations of the caret may be more appropriate for other kinds of data. Some editors, such as !Draw, do not have a visible insertion point, but still "grab the caret" and mark it as invisible, in order to gain the input focus so that they may receive keystroke events."
Clear v.	The operation by which a selection is undone."
Clipboard n.	A hidden, temporary storage area that holds any type of data while the user is copying or moving it using the cut-and-paste protocol, whether internal to one application, or between applications. Conceptually, there is only one clipboard, but the actual storage area may actually be managed by different applications or modules, depending upon the circumstances. The term may also be used to refer to the Clipboard module, although in this eventuality, the initial letter will be in upper case.
Copy v.	The operation by which the current selection is replicated in the clipboard, overwriting any existing data in the clipboard.
Cut v.	As copy, but the selection is subsequently deleted from its original location.
Data type n.	A value equivalent to a filetype, but not necessarily referring to a file.
Drag v.	The operation by which the user indicates where they wish a selection to be copied or moved to by dragging a representation of the data from the selection to the destination.
Drop v.	At the end of a drag, the actual data transfer process. This combines the functionality of a paste operation with either a cut or copy operation, as appropriate.
Ghost caret n.	During a drag operation, the position in a document where the data would be inserted, were the user to release the mouse button. In textual documents, the ghost caret is often

Term(s) used in Document	Meaning
	shown similarly to a normal caret, but coloured grey, and "snapped" to the nearest character boundary. Other documents might better display the ghost caret as the bounding box of the data, scaled according to the destination window's zoom factor(s).
OR conj.	Logical inclusive OR (i.e. not EOR) - used where the 'or' would have an ambiguous meaning, for example in English text.
Input focus n.	The defining attribute of the window where keystroke events will be delivered. The user may be able to see a caret or a selection, or possibly neither, in the window that has the input focus; the window border will be coloured in an alternative colour (conventionally cream). Any parent nested windows (recursively), and any non-pane window behind a pane window, will also have their title bars recoloured.
Paste v.	The operation that the user performs to copy the clipboard contents into a document, at the caret.
Selection n.	The portion of a document which the user has chosen as the target for subsequent operations. This may be a contiguous selection (as in the case of selected text) or a non-contiguous selection (as in the case of a number of selected files in the Filer). The rendering of the selection is media-dependent, but typically may be shown by inversion of the colours of the selected region, or alternatively by the drawing of a bounding box around the selection(s). A shaded selection, which ought be rendered to match the Wimp's rendering of shaded selections in writable icons, indicates the location of a selection after another selection has been made in another window - but not when a caret or selection is made in a non-drag-and-drop application.
Shadow caret n.	The equivalent of a shaded selection, but for carets. A shadow caret must not be rendered in such a way that it can be mistaken for a ghost caret. It is optional, because applications are expected normally to use a Wimp-drawn caret, and the Wimp does not support shadow carets. However, shadow carets can be useful, especially if the application draws its own caret anyway (as, for example, if an I-beam is an unsuitable),

Term(s) used in Document	Meaning
	because they fix an insertion point for a drop, whenever one or both of the sending and receiving tasks uses pre-drag-and-drop data transfer protocol. The shadow caret is also the position to which the caret will be returned if the user Adjust-clicks on the window, or clicks in a "dead" region of the window, such as a page border; this is particularly useful in cases where repositioning the caret would be time-consuming or fiddly, for example if the caret is in a deep "layer" of a document.

14. References

[1]: Support Group Application Note 240: The RISC OS Selection Model and Clipboard

[2]: Support Group Application Note 241: The RISC OS Drag-and-Drop System

[3]: RISC OS 3 PRM 3 §53: The Window Manager, pp 3-249 - 3-256

[4]: RISC OS Style Guide, issue 3, §11: Handling selection, pp 77-82

[5]: Document Ref 1309,413/FS: Ursula Window Manager Changes Functional Specification

15. History

Document information

History:	Revision	Date	Author	Changes
	0.00	12 Sep 1997	BJGA	**Started**
	0.01	13 Oct 1997	BJGA	**First release for comment**
	0.02	14 Oct 1997	BJGA	**Released for review**
	D	19 May 1998	BJGA	**Prepared for D.O**
	E	26 Feb 1999	BJGA	**Started reworking document for Java 1.2 project, didn't get far before cancelled again**
	E	16 Oct 2007	BJGA	**Finally finished integrating the Ursula review comments and 8 years' worth of mental notes, for initial release alongside shared source code**
	F	22 Feb 2015	RPS	**Updated the page references in the Style Guide**
	G draft	22 Feb 2015	Ben Avison	**Shared Source RISC OS release (formerly Ursula and Java 1.2) Ref: 1309,419/FS**
	0.08A	28 Aug 2021	Alan Robertson	**Initial version in PRMinXML format** ● Formatting of text removed from document (italic, underlined, bold) ● Added related links to swi and message definitions

Related: [(PDF format, 8K).](#)

URI Handler Functional Specification

Overview

This document addresses the recognised lack of existing RISC OS specifications that describe a standard method for different applications to communicate URIs (of which URLs are an example) between themselves; for example, to provide for an address book requesting that a Web browser display someone's home page.

The first part of this requirement addressed is the provision of a mechanism for applications to pass URIs between themselves in a uniform manner. To date, several third party developers have independently solved this problem in a variety of different ways, as there was no centrally published, universally available standard for developers to work to. This is such a standard.

This 'central resource broker' will be extended in the future to provide mechanisms to enable more efficient handling of URIs. For example, data may be passed to an appropriate application based on the type of data as opposed to simply the method specified for retrieval of the data, as is often the case with URLs. This too will be via a service interface to the central broker.

Deliverable 'product'

This document describes the API created to fulfil the above stated requirement, and relates to existing software providing the underlying functionality.

The software takes the form of a RISC OS relocatable module, entitled 'AcornURI'. This is a generic, OS-level software component that could as equally sit beneath a text editor which was aware of the form of URIs as sit beneath a Web browser or mail / news reader. Distributed alongside the module are four sprite definitions for URI files.

The module is suitable for RISC OS 3.10 upwards, and should be stored in `!System.310.Modules.Network` as 'URI'.

An archive containing the module, sprites, a text version of this specification and a brief ReadMe describing the component versions can be downloaded here (ZIP format).

Programmer's interface

The application programmer's interface to the services provided by the Acorn URI handler is detailed in the following sections. This interface will be enhanced in the future, as outlined in the overview, to provide a more comprehensive set of services; so it's worth emphasising that only those details and features of the interface specified in the following sections should be considered to be supported. Any behaviour which is not specified below should be considered to be an implementation feature of a particular version of the software, and as such liable to change, alteration or omission without notice.

The following have been allocated for the use of the Acorn URI handler:

Type	Allocated
Module name	AcornURI
SWI prefix	URI
SWI chunk	&4E380
WIMP message chunk	&4E380
Error code chunk	&810A00
Service Call	&A7
FileType	&F91

All environment variables containing the string _URI_ (i.e. matching *_URI_*)

URI 'handles' are utilised to identify a specific URI request when communicating with the URI handler; tasks may assume nothing about these handle values, other than that they identify a particular URI to the handler for the period of their validity.

URI SWIs

URI_Version (SWI &4E380)

return the URI handler module's version number

On entry

R0 = flags:

Bit(s)	Meaning
0-31	Reserved, must be zero

On exit

R0 = current version $\times$ 100

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

Not defined

Use

This SWI is used to inquire of the URI handler module's version number, and should be used to check for a suitable version being present before using the facilities provided.

The number returned is of the form (major version $\times$ 100) + minor version.

Related APIs

None

pass a URI string to the handler for dispatch, or checking for the presence of a potential servicer

On entry

R0 = flags:

Bit(s)	Meaning
0	inform caller of result (=>R2 valid)
1	check only, don't process (R0:0 must be set)
2	don't attempt external process startup
3-31	Reserved, must be zero

R1 = pointer to 0 terminated URI string
R2 = 0, or source task handle if bit R0:0 is set and the caller is a WIMP task

On exit

R0 = flags:

Bit(s)	Meaning
0	request rejected, URI won't be dispatched
1-31	Reserved, must be zero

R1 preserved
R2 = task handle of URI handler
R3 = handle of this URI (request identifier)

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used by an application to pass a URI string to the handler for dispatch, or checking for the presence of a potential servicer. Dispatch provides for optional requesting of a success/failure indication (R0:0 set) via a WIMP message (Message_URI_MReturnResult (on page 125)) or service call reason code (Service_URI 3 (on page 120)) ⚠ FIXME: change this to use-reasonname, when available - necessary since the dispatch of the URI occurs asynchronously.

If R0:0 is set, module clients must signal that a URI_MReturnResult message is not necessary by setting R2 to 0. In this case, only the service call will be sent out. Conversely, WIMP task clients must specify a valid task handle in R2 - in this case, only the WIMP message will be sent out.

When requesting a check only (R0:1 set), it is an error not to set R0:0 and fill in R2 as described above.

The URI will be copied to the URI handler's workspace, optionally transformed (future enhancement, such as canonicalisation), then relocatable modules will be offered the chance to handle the URI via service call &A7 with an appropriate reason code (Service_URI 2 (on page 119)) ⚠ FIXME: change this to use-reasonname, when available ; if the service call is unclaimed, then a User_Message_Recorded WIMP message will be broadcasted (Message_URI_MProcess (on page 123)), offering other tasks the chance of handling the URI; if neither of these mechanisms elicits a response, then the request will be deemed to have failed (in so far as active tasks are concerned).

If R0:2 is clear, then the 'fallback' position of checking a subset of the environment variables will be used to attempt to start a suitable task to handle the URI. The handle ceases to be valid at this point if notification has not been requested, irrespective of whether or not the URI has processed.

If R0:0 is set, the originating task will be informed of the results of the dispatch process (via a User_Message_Recorded WIMP message URI_MReturnResult if R2 contains a valid task handle, or service call Service_URI_ReturnResult if R2 is zero). If the message is not acknowledged or service call claimed, the handle will cease to be valid; otherwise, the originating task becomes responsible for indicating that it no longer needs the URI by calling SWI SWI URI_InvalidateURI (on page 114).

Related SWIs

Related services

Related messages

return size of buffer required to hold specified URI, or to return the URI via the buffer

On entry

R0 = flags:

Bit(s) Meaning

0-31 Reserved, must be zero

R1 = pointer to buffer to hold URI or 0 to read required size
R2 = length of buffer or unused (if R1 = 0)
R3 = URI handle

On exit

R0 preserved
R1 preserved
R2 = offset into buffer of terminating null, or size of buffer required (if R1 = 0 on entry)
R3 preserved

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to inquire what size of buffer is required to hold the specified URI (if R1 is zero on entry), or to pass details of a buffer into which your task desires the URI to be copied.

If this is successful, then R2 should be equal to the size of the buffer: if the buffer specified on entry is not large enough, then R2 will be returned negative (indicating the number of unreturned characters), and the string returned in the buffer will still be zero-terminated i.e. buffersize-1 characters of the string are returned.

Related APIs

None

mark the specified URI as being invalid

On entry

R0 = flags:

Bit(s)	Meaning
0-31	Reserved, must be zero

R3 = URI handle

On exit

R0 preserved
R3 preserved

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to mark the specified URI as being invalid.

Related APIs

None

URI service calls

Service call &A7 has been allocated for the use of the URI handler; the following sub-reason codes are defined for the use of external applications. All other service call reason codes are reserved: a module may assume nothing about these, and should always ignore unrecognised reason codes - never claim such service calls.

A deliberate degree of similarly exists between the WIMP messages and the service calls, since both provide essentially the same functionality; clearly, messages will be convenient in environments where service calls are not and vice versa, hence the duplication of functionality between the two.

events issued by URI handler

On entry

R0 = reason code:

Value	Meaning
0	URI handler started (on page 117)
1	URI handler dying (on page 118)
2	process or check URI (on page 119)
3	return result of a dispatch (on page 120)

All other values are reserved, and must not be used

R1 = service call number
R2 = flags
R3 - R4 = dependant on reason code

On exit

R0 - R3 = dependant on reason code

Use

Related APIs

None

URI handler started

On entry

R0 = 0 (reason code)
R1 = &A7 (service call)
R2 = flags:

> **Bit(s) Meaning**
>
> 0-31 Reserved, must be zero

On exit

R0 - R2 preserved

Use

This service call indicates that the URI handler has started. It is intended for more specific use defined in future versions of this specification.

This service call must be passed on.

Related APIs

None

URI handler dying

On entry

R0 = 1 (reason code)
R1 = &A7 (service call)
R2 = flags:

> **Bit(s) Meaning**
>
> 0-31 Reserved, must be zero

On exit

R0 - R2 preserved

Use

This service call indicates that the URI handler is dying. It is intended for more specific use defined in future versions of this specification.

This service call must be passed on.

Related APIs

None

process or check URI

On entry

R0 = 2 (reason code)
R1 = &A7 (service call)
R2 = flags:

Bit(s)	Meaning
0	check URI only, do not process
1-31	Reserved, must be zero

R3 = pointer to URI string (readonly access)
R4 = handle of this URI

On exit

R0 preserved
R1 = preserved, or 0 to claim
R2 - R4 preserved

Use

This service call indicates that the URI handler has been requested to dispatch the given URI for either processing (R2:0 clear), or just checking (R2:0 set). The URI string is held in the URI handler's workspace; this buffer must not be written to - if it is, behaviour is undefined. It is intended that modules should inspect the string at the given address, and if they decide they can process the given URI, claim the service call. If R2:0 is set, this is all that is required.

However, if R2:0 is clear, i.e. process URI, then a call to SWI SWI URI_RequestURI (on page 112) to obtain a local copy to work with must be made; this step may NOT be omitted, since the internal buffer is not guaranteed to remain valid after return from the service handler.

If a module cannot process the given URI, it must pass the call on with all registers preserved to allow the remainder of the dispatch mechanism to function.

Related SWIs

return result of a dispatch

On entry

R0 = 3 (reason code)
R1 = &A7 (service call)
R2 = flags:

Bit(s)	Meaning
0	Clear: URI was claimed for processing
	Set: URI was not claimed for processing
1-31	Reserved, must be zero

R3 undefined
R4 = handle of this URI

On exit

None

Use

This service call is used by the URI handler to return result status information to a requesting module. The module requests the service call when it calls the SWI URI_Dispatch (on page 109) SWI; it must set R0:0 and R2=0 on entry. Such modules must remember the URI handle returned in R3 by this SWI or they cannot later determine if the service call was meant for them or another client; any client setting R0:0 on entry to URI_Dispatch must see if it recognises the URI handle in R4, and if so, claim the service call. If it does not recognise the handle, it must not claim the service call. Any clients which never set R0:0 on entry to URI_Dispatch can ignore the service call.

Only success or failure is indicated, though this is likely to be enhanced in future.

Related SWIs

SWI URI_Dispatch (on page 109)

Message_URI_MStarted (&4E380)

URI handler started

Message

Offset	Contents
R1+20	flags:

	Bit(s)	Meaning
	0-31	Reserved, must be zero

Offset	Contents
R1+24	undefined (reserved)

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is broadcast (User_Message) to indicate that the URI handler has started up. It must not be acknowledged - information only.

Related APIs

None

Message_URI_MDying
(&4E381)

URI handler dying

Message

Offset	Contents
R1+20	flags:

Bit(s)	Meaning
0-31	Reserved, must be zero

Offset	Contents
R1+24	undefined (reserved)

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is broadcast (User_Message) to indicate that the URI handler is shutting down. It must not be acknowledged - information only.

Related APIs

None

process or check URI

Message

Offset	Contents
R1+20	flags:

	Bit(s)	Meaning
	0	check URI only, do not process
	1-31	Reserved, must be zero

Offset	Contents
R1+24	pointer to URI string (URI internal buffer)
R1+28	URI handle
R1+32	undefined (reserved)

Delivery

Message must be broadcast (destination 0)

Message must be sent recorded delivery (reason code 18)

Use

This message is broadcast (User_Message_Recorded) to indicate that the URI handler has been requested to dispatch the given URI for processing, or check if any task can process the URI.

The URI string is held in the URI module's workspace; this buffer must not be written to - if it is, behaviour is undefined.

It is intended that applications which can process URIs should inspect the string at the given address to determine if they can process the URI. If R0 bit 0 is clear, you must then call SWI [SWI URI_RequestURI (on page 112)](#) to obtain a copy to work with - this step may not be omitted, since the buffer given is not guaranteed to remain unaltered.

If an application is able to check or process the given URI, then it should acknowledge the broadcast by sending a [Message_URI_MProcessAck (on page 126)](#) message to the URI handler, thus preventing it being passed on to other applications, otherwise it must not acknowledge the message.

Related SWIs

[SWI URI_RequestURI (on page 112)](#)

Related messages

[Message_URI_MProcessAck (on page 126)](#)

Message_URI_MReturnResult (&4E383)

return result of a dispatch

Message

Offset	Contents
R1+20	flags:

Bit(s)	Meaning
0	Clear: URI was claimed for processing
	Set: URI was not claimed for processing
1-31	Reserved, must be zero

Offset	Contents
R1+24	URI handle
R1+28	undefined (reserved)

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is used by the URI handler to return result status information to a requesting task. Only success or failure is indicated, though this is likely to be enhanced in future.

Related APIs

None

Message_URI_MProcessAck (&4E384)

acknowledge URI_MProcess

Message

Offset	Contents
R1+20	flags:

Bit(s)	Meaning
0	Check URI only, do not process
1-31	Reserved, must be zero

Offset	Contents
R1+24	pointer to URI string (URI internal buffer)
R1+28	URI handle
R1+32	undefined (reserved)

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is used by clients of the URI handler to indicate to the URI handler that they can claim or process a given URI, thus preventing it being passed on to other applications. Claimants just change the message type to &4E384 (URI_MProcessAck) and copy the supplied my_ref field into your_ref, then send the message back to its originator (ie. the URI handler).

Related APIs

None

* Commands

*Desktop_AcornURI

starts the URI handler

Syntax

*Desktop_AcornURI

Parameters

None

Use

Desktop_AcornURI starts the Acorn URI handler. Do not use
*Desktop_AcornURI, use *Desktop instead.

Help Text:*Do not use *Desktop_AcornURI, use *Desktop
instead.
Syntax: *Desktop_AcornURI

Examples

*Desktop_AcornURI Use *Desktop to start AcornURI

Related APIs

None

*URIinfo

display information about the URI handler

Syntax

*URIinfo

Parameters

None

Use

*URIinfo produces status information from the Acorn URI handler.

HelpText:

URIinfo produces status information from the Acorn URI handler.
Syntax: *URIinfo

Examples

*URIinfo

URI_taskhandle: 4b4016d8
URI chain start: 021cc844
URI handle: 022b60d4 (action:00020000) 'http://www.acorn.com/'

Related APIs

None

*URIdispatch

try to launch a URI

Syntax

*URIdispatch *<uri>*

Parameters

<uri> - the uri to be launched

Use

*URIdispatch tries to lauch a given URI. No indication is given of whether or not the launch succeeded.

Help Text:

```
URIdispatch tries to launch a URI.
Syntax: *URIdispatch <uri>
```

Examples

```
*URIdispatch http://www.acorn.com/
```

Related SWIs

URI handler errors

Defined errors

The URI handler has a error chunk base of &810A00. Currently defined errors are:

Name	Number	Cause
Error_URI_NoMemory	Base + 1	There is not enough memory to complete an operation.
Error_URI_BadURI	Base + 2	An empty URI string is supplied (e.g. to URI_DispatchURI).
Error_URI_BadHandle	Base + 3	A bad URI handle has been supplied.
Error_URI_BadFile	Base + 4	Reported when there is an error accessing a URI file.

Error generators

Generators of the errors are as follows:

Generator	Returns
URI_DispatchURI	Error_URI_NoMemory Error_URI_BadURI
URI_RequestURI	Error_URI_BadHandle
URI_InvalidateURI	Error_URI_BadHandle
*URIDispatch	Error_URI_NoMemory

Finally, the WIMP task may generate (through a standard WIMP error box) `Error_URI_NoMemory` and `Error_URI_BadFile`.

Use of the URI filetype

URI files have the filetype &F91, with the text equivalent 'URI'. The URI handler will deal with such files appropriately when the file is double-clicked upon (currently, it dispatches the URI inside the file - see the file format description below). Applications must not set an `Alias$@RunType` variable for the URI filetype, nor must they deal with DataOpen messages for this filetype. Applications may respond to DataLoad messages for the filetype as they see fit.

Suitable sprites exist (four; medium and high resolution file sprites, small and large variants). These are the only sprite definitions acceptable for use in this context. The sprites should always be distributed alongside the module.

URI files consist of a series of lines of characters. Lines are ended by any number of control code characters (ASCII code less than 32) or the end of the file. All lines in a file do not have to end in the same way provided each individual line ends in a valid manner. Other white space is not ignored, hence a single space character (ASCII code 32) followed by ASCII code 9 does count as a line containing a single space followed by a line end marker.

URI files support comments. Comment lines start with a '#' (ASCII 35) and end in the same way as all other lines. Comment lines are not counted; any file reader that happened to keep track of the line number it was on should not increment the counter for a comment line. A URI file may contain any number of comment lines, but automatic file generators are encouraged to keep comments to a bare minimum to keep file sizes down. Generator code must never create special comment lines which mean something to accompanying reader code - comment lines are always skipped by the reader code and never parsed, beyond identifying them as comments.

The line ending type of a URI file is not fixed as a specific control code or sequence of control codes (e.g. CR+LF) to allow simple generation from a variety of sources, including manual authoring. Given this latter possibility, it is important to stress that unlike, say, HTML, the URI file format is rigorously defined and must be adhered to. Incorrectly formed files are not guaranteed to work correctly with

either the Acorn URI handler or applications which support it.

That said, the use of ASCII code 13 followed by ASCII code 10 (CR+LF) to end lines is strongly encouraged as this is a common line ending type supported by many different editors on many platforms. ASCII code 9 (tab) could also be used to give the file a better visual appearence in the editor - it is still an end of line as far as the file reader is concerned. This convention provides the potential for greater convenience for the end-user, but must NOT be assumed in file reading code!

Currently defined formats:

Line number	Contents
1	'URI' - this must be present before any comments or other information
2	Text equivalent of the earliest module version number (as returned by URI_Version) that would fully understand the file contents; e.g. '5' for v0.05 (any number of preceeding '0's are also valid). So if lines were added to this file format to produce a version 6 file, this implies that URI v0.06 is required to understand those extra lines, even though v0.05 would still understand lines 1 to 4 The first general release version of the URI handler will adopt a version number of 1.00, so the first URI files will start with '100' in this line
3	A fully specified URI; v0.05 of the URI handler does not attempt to canonicalise URIs, though future versions may. If this line contains only one character with ASCII code 42 ('*'), the file does not contain a URI and should be ignored (this is to allow future file formats to hold non-fully specified URIs on later lines that could be canonicalised by the URI module, without breaking legacy file reading code) Lines 1 to 3 are required in a minimal URI file. Any other lines may or may not appear
4	A title string to associate with the URI. Again, if this line contains only one character with ASCII code 42 ('*'), the file does not contain a title string. Processors wishing to display title information alongside a URI may well use the URI itself instead, in this case

You can find some examples of URI files in a SparkFS format archive here.

Future file formats will be backwards compatible with this one, so clients should only check the version number of the file to know what sort of contents to expect. So for example, if a version 100 aware

application encounters a later version file, it can assume that the first 4 lines of the file are as described for the version 100 file; though there may be other lines which clearly it cannot understand, and must ignore.

For example, the file format rationale may be easier to understand given the possibility of a future format - version 101, say - which allowed non-fully specified URIs in line 5 which can be canonicalised, and a preferred external process to start in line 6. The file could look like this:

```
-Start of file-
URI
6

*
Acorn Group PLC
www.acorn.com

<Browse$Dir>.!Run
-End of file-
```

Use of URI environment variables

Currently defined variables are of the form:

```
Alias$Open_URI_<scheme> <file_to_run>
```

for example,

```
Alias$Open_URI_http <Browse$Dir>.!Run
Alias$Open_URI_ftp <FTPClient$Dir>.!Run
```

If a variable such as the above is defined, then the task it names will be run. If this is successful, the URI will be redispatched in the normal way, so the task has the opportunity of dealing with it.

A comma separated list of handlers may be specified, so applications must always add to the contents of the variables. At present, only the first item in the list is used, though this may change in future versions.

For compatability with existing applications, the URI handler will support a similar scheme of system variables defined by ANT Ltd. Details of these are at the time of writing freely available on the ANT Support web site.

Performance targets

Final code size of version 1.00 should be about 26K. Quiescent memory usage should be no more than 512 bytes. When active, the main storage requirement for each URI being processed is storage of the URI itself. This is, then, indeterminate, but unlikely to be more than 2K (not that the URI handler will have any such hard coded limits). An additional overhead of no more than 128 bytes per URI is also required.

Document information

		wording changes here and there; performance targets and development test strategy sections added
1307,260/ FS_7	21 Jun 1997	● Reworded away from future tense to form an externally releasable specification
1307,260/ FS_8	10 Dec 1997	● A couple of implied future tense references missed in Draft 7, Following review of draft 5
		● some minor wording changes here and there
		● performance targets and development test strategy sections addednow corrected
		● some minor rewording associated with this
-	11 Dec 1997	**(General release of 1307,260/FS)**
		● No longer draft
		● settled on WIMP rather than Wimp; couple of minor typos corrected
-	05 Jan 1998	● Few more typos fixed ('21', '23', 'URI_ProcessAck' and 'URIProcessAck' instead of 'R2', '32', 'URI_MProcessAck' and again 'URI_MProcessAck' respectively)
-	05 Feb 1998	● Minor tweaks to fit in with the rest of the Acorn Internet site (now uses a small style sheet like everything else, site map link added, and so-on). No changes to the content of the specification
-	19 Feb 1998	● A few HTML style changes to make some of the section headings a bit clearer; no content

-	23 Feb 1998		change ● Colours changed to blue; now back to green again
1215,215/ FS_1	02 Mar 1998		**(General release of 1215,215/FS)** ● Document number now 1215,215/FS ● Updated history, and navigation links in the page footer now include the specifications section; no other content changes
1215,215/ FS_2	01 Sep 1998		● Corrected table listing allocated items in the Programmer's Interface section - module name is 'AcornURI', not 'URI' ● Issue numbers for 1215,215/FS in this table are now in <strong> to match the 1307,260/FS numbers. ECO 4102 allocated for these changes
1215,215/ FS_3	08 Sep 2021	Alan Robertson	**Initial version in PRMinXML format** ● No major changes to text. Removed the 'Document Status' section as information captured in 'Document Information' section ● Added related links to definitions and various parts throughout document ● Prefixed the Acorn Functional Specification Document Number to each Issue revision in original ● Removed links to zip files

Disclaimer: This document first appeared as 1307,260/FS and went through issues 1 to 8, with 8 being published outside of Acorn. The document number was later changed to 1215,215/FS.

Acorn URL Fetcher API Specification

Overview

The URL (Universal Resource Locator) module is a general purpose module for fetching data from various Internet services. This specification reflects the behaviour of version 0.42 or later of the URL_Fetcher module. 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 (hereafter just "protocol 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. The system structure is shown in figure 1 below.
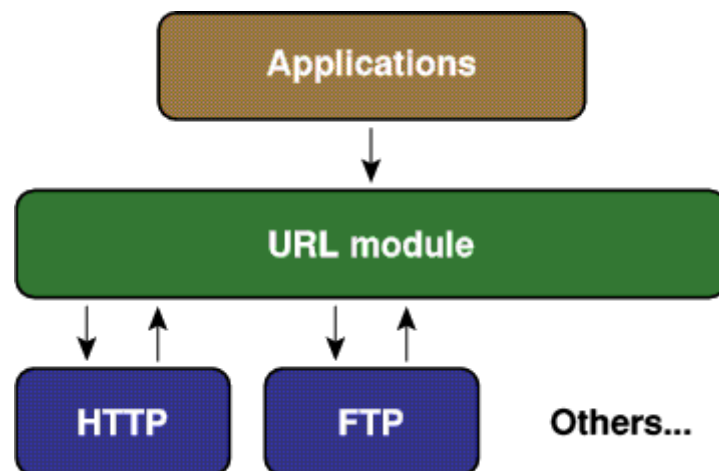
*Figure 1: URL Fetching
system structure*

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 [SWI URL_Register (on page 141)](#))
- Start fetching an object (SWI [SWI URL_GetURL (on page 143)](#))
- Repeatedly, whilst multi-tasking if in the desktop environment:
    1. Read blocks of data (SWI [SWI URL_ReadData (on page 148)](#))
    2. Process that data
- Discard session (SWI [SWI URL_Deregister (on page 154)](#))

If an application decides it requires a premature termination (eg. the user asked the application to quit whilst an object was being downloaded), then the application calls SWI [SWI URL_Stop (on page 152)](#) immediately and then discards the session with SWI [SWI URL_Deregister (on page 154)](#). 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 [SWI URL_GetURL (on page 143)](#)) 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/1.0 or HTTP/1.1 Response-Line of the format: "HTTP/1.0 200 OK" followed by various headers identifying the content-type of the retrieved data, followed by an empty line, followed by the data itself.

Outstanding issues

There are no outstanding issues.

Client to URL module interface

A typical client would be an application, such as a Web Browser. The following SWI calls provide the interface for an application to control and transfer data via the URL module.

URL_Register
(SWI &83E00)

Initialise a client session with the URL module

On entry

R0 = Flags: All bits are currently reserved (must be zero)

On exit

R0 = Reserved - currently zero
R1 = Session identifier

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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.

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.

Related SWIs

Instigate data transfer from / to a resource server

On entry

R0 = Flags:

Bit(s)	Meaning
0	If set, R6 is valid
1	If set, R5 holds length of data in R4 specified buffer, otherwise a single NULL terminated string in buffer
2-31	Reserved, must be zero

R1 = Session identifier

R2 = Bitfield:

Bit(s)	Meaning
0-7	Method (on page 145) (8-bit value, held in bits 0-7). This is protocol dependent
8-15	Method dependent
16-31	Reserved, must be zero

R3 = URL - the document we are after, including the protocol. For example "http://www.acorn.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 R4 data block
If R0:1 is clear, must be 2

R6 = User Agent - Pointer to string to use as 'User Agent' identifier in request header if R0:0 is set. (NULL pointer or NULL string implies use default identifier - see below)

⚠ FIXME: original links to middle of third paragraph below!

On exit

R0 = Protocol status (see SWI SWI URL_Status (on page 146), below)

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to instigate a transfer of data to or from (mainly from) a resource server. When this SWI has been called, the URL module checks the per-session and global proxy settings, looking for a match (see SWI [SWI URL_SetProxy (on page 150)](#) 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 was 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 was found, then all client registers are passed onto the protocol module via the [SWI Protocol_GetData (on page 176)](#) SWI call with the exceptions stated above for proxy handling. On exit, R0 will hold the status code returned by the protocol module.

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/1.0" 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:
```
User-Agent: Acorn Browse/2.06 AcornHTTP/0.82
```

where the client only specified `"Acorn Browse/2.06"`.

Table of method numbers

No.	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 to Acorn - do not use
7	n/a	n/a	Reserved to Acorn - 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

Applications for new method codes should be made to Developer Support. The range 128-254 is reserved for private non-distributed modules. Method numbers 0 and 255 are reserved and must not be used.

The list of methods specific to FTP quoted above are fully implemented in version 0.28 of the FTP Fetcher module. The list of methods specific to HTTP quoted above are fully implemented in version 0.82 of the AcornHTTP module.

Related SWIs

Obtain information on a session

On entry

R0 = Flags: All bits currently reserved (must be zero)
R1 = Session identifier

On exit

R0 = Status word:

Bit(s)	Meaning
0	Connected to server
1	Sent request
2	Sent data
3	Initial response received
4	Transfer in progress
5	All data received
6	All data received
7-31	Reserved, must be zero

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

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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.

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 SWI [SWI Protocol_Status (on page 178)](#) for the protocol module concerned with the request.

Related SWIs

[SWI URL_Register (on page 141)](#)
[SWI URL_Deregister (on page 154)](#)
[SWI Protocol_Status (on page 178)](#)

URL_ReadData
(SWI &83E03)

Read data pending from a request

On entry

R0 = Flags: All bits currently reserved (must be zero)
R1 = Session identifier
R2 = Client buffer for receiving data
R3 = Size of buffer pointed to by R2

On exit

R0 = Status word (see SWI [SWI URL_Status (on page 146)](#))
R1 preserved
R2 = Preserved. Contents of buffer modified
R3 preserved
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

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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. R1 always returns the protocol status code. In the event of all the data being read (R5 = 0 on exit), a call to SWI URL_Stop (on page 152) is not required as this is performed automatically when SWI URL_Deregister (on page 154) is called for the client session. Once all data has been read a call to SWI URL_Status (on page 146) 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 SWI Protocol_ReadData (on page 180) SWI for the protocol module concerned with the request.

Related SWIs

SWI URL_Register (on page 141)
SWI URL_GetURL (on page 143)
SWI URL_SetProxy (on page 150)
SWI URL_Status (on page 146)
SWI URL_Deregister (on page 154)
SWI Protocol_GetData (on page 176)
SWI Protocol_ReadData (on page 180)

URL_SetProxy
(SWI &83E04)

Set up a proxy server for a session with the URL module

On entry

R0 = Flags: All bits currently reserved (must be zero)
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 =

Value	Meaning
0	Proxy request
1	Don't proxy request
	All other values are reserved

On exit

R0 = Status word (see SWI SWI URL_Status (on page 146))

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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 SWI URL_ProtocolRegister (on page 171) regarding how the default port number is derived.

R3 is a pointer to a buffer containing the initial part of the URL to proxy - the URL scheme (eg "http:", "ftp:"). This system has the advantage that requests to certain hosts can be proxied and not others (eg 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 SWI URL_GetURL (on page 143) request is received, the proxy settings are evaluated in the following order:

Order	Description
1	Client no-proxy
2	Client proxy
3	Global no-proxy
4	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.

Related SWIs

SWI URL_Register (on page 141)
SWI URL_GetURL (on page 143)
SWI URL_Deregister (on page 154)

URL_Stop
(SWI &83E05)

Abort a request placed with the URL module

On entry

R0 = Flags: All bits currently reserved (must be zero)
R1 = Session identifier

On exit

R0 = Status word (see SWI [SWI URL_Status (on page 146)](#))
R1 preserved

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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 [SWI URL_Deregister (on page 154)](#).

When this SWI is called, the URL module invokes the [SWI Protocol_Stop (on page 182)](#) SWI for the protocol module concerned with the request.

Related SWIs

URL_Deregister
(SWI &83E06)

Deregister a client session with the URL module

On entry

R0 = Flags: All bits currently reserved (must be zero)
R1 = Session identifier

On exit

R0 = Status word (see SWI [SWI URL_Status (on page 146)](#))
R1 preserved

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

This call deregisters the client session from the URL module, freeing up any information the URL module may have kept about the client session (eg proxy information). The session identifier ceases to be valid and becomes available for re-issue on a subsequent call to SWI [SWI URL_Register (on page 141)](#).

When this SWI is called, the URL module invokes the [SWI Protocol_Stop (on page 182)](#) SWI for the protocol module concerned, if it has not already done so (e.g. during the processing of [SWI URL_Stop (on page 152)](#)).

Related SWIs

URL_ParseURL
(SWI &83E07)

Parse URLs to / from their constituent parts

On entry

R0 = Flags:

Bit(s)	Meaning
0	If set, R5 contains number of words in data block, else a default of 10 words is assumed.
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.
2-31	Reserved, must be zero

R1 = Reason code:

Value	Meaning
0	Return component buffer requirements (on page 158)
1	Return component data in specified buffers (on page 160)
2	Construct full URL from component buffers (on page 162)
3	'Quick parse' (on page 164)

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 exit

R0 = Flags: All bits currently reserved (must be zero)
R1 preserved
R2 preserved
R3 preserved
R4 = preserved. Data block at R4 is updated in line with entry reason code
R5 preserved

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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.

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:

Offset	Usage
+0	Fully canonicalised URL
+4	URL protocol (e.g. "http", "ftp") forced to lower-case
+8	Hostname (e.g. "www.acorn.com") forced to lower-case
+12	Port (e.g. "80")
+16	Username - used for FTP authentication and mailto
+20	Password - for FTP
+24	Account - for FTP
+28	Path (e.g. "pub/riscos/releases") (See note)
+32	Query - for HTTP, things after a query character
+36	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 [SWI URL_ProtocolRegister (on page 171)](#) 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.

If R3 is non-NULL on entry, 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 entry reason codes are described below.

Related SWIs

[SWI URL_ProtocolRegister (on page 171)](#)

Work out space required for URL components

On entry

⚠ FIXME: need someone to double-check entry and exits

R0 = Flags:

Bit(s)	Meaning
0	If set, R5 contains number of words in data block, else a default of 10 words is assumed
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
2-31	Reserved, must be zero

R1 = 0 (reason code)
R2 = Pointer to base URL
R3 = Pointer to URL relative to base URL (or NULL if none)
R4 = Pointer to data block
R5 = If R0:0 set, size of R4 block in words

On exit

R4 = Data block updated with sizes of each component

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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.

Related SWIs

Split a URL into its component parts

On entry

⚠ FIXME: need someone to double-check entry and exits

R0 = Flags:

Bit(s)	Meaning
0	If set, R5 contains number of words in data block, else a default of 10 words is assumed
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
2-31	Reserved, must be zero

R1 = 1 (reason code)
R2 = Pointer to base URL
R3 = Pointer to URL relative to base URL (or NULL if none)
R4 = Pointer to data block
R5 = If R0:0 set, size of R4 block in words

On exit

R4 = Data block updated with pointers to each component requested

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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.

Related SWIs

Combine the components of a URL

On entry

⚠ FIXME: need someone to double-check entry and exits

R0 = Flags:

Bit(s)	Meaning
0	If set, R5 contains number of words in data block, else a default of 10 words is assumed.
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.
2-31	Reserved, must be zero

R1 = 2 (reason code)
R2 = Pointer to base URL
R3 = Pointer to URL relative to base URL (or NULL if none)
R4 = Pointer to data block
R5 = If R0:0 set, size of R4 block in words

On exit

R4 = Data block updated with full URL

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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 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.

Related SWIs

Quickly obtain a fully resolved URL

On entry

⚠ FIXME: need someone to double-check entry and exits

R0 = Flags:

Bit(s)	Meaning
0	Reserved, must be zero
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.
2-31	Reserved, must be zero

R1 = 3 (reason code)
R2 = Pointer to base URL
R3 = Pointer to URL relative to base URL (or NULL if none)
R4 = Pointer to buffer
R5 = Size of buffer in R4

On exit

R4 = Data block updated with fully resolved URL
R5 = Size of buffer remaining (negative if it was too small)

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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:

```
length(base URL) + length(relative URL) + 4
```

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 earler 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.

Related SWIs

URL_EnumerateSchemes
(SWI &83E08)

On entry

R0 = Flags: All bits currently reserved (must be zero)
R1 = Context (0 for first call)

On exit

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)

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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 [SWI URL_ParseURL (on page 155)](#) 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 [9].

⚠ FIXME: Add link to Acorn URI Handler Functional Specification

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).

Related APIs

None

URL_EnumerateProxies (SWI &83E09)

Enumerate proxies or no-proxy URLs

On entry

R0 = Flags:

Bit(s)	Meaning
0	If set, enumerate the no-proxy list
1-31	Reserved, must be zero

R1 = Session identifier, or zero for global proxies / no-proxies)

R2 = Context (0 for first call)

On exit

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 a 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 ia not -1)

If R0:0 set: Corrupted, contains no useful information

Interrupts

Interrupts are undefined

Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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 when the setting was made.

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

Protocol module to URL module interface

This section defines the calls provided by the URL module to enable a fetcher protocol module to interact with it.

URL_ProtocolRegister
(SWI &83E20)

Register a protocol module with the URL module

On entry

R0 = Flags:

Bit(s)	Meaning
0	If set, R5 contains protocol flags word
1	If set, R6 contains the default port number
2-31	Reserved, must be zero

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 ⚠ FIXME: Add link

R6 = Default port number, if R0:1 set. See below ⚠ FIXME: Add link

On exit

R0 = Flags: All bits currently reserved (must be zero)

R1 - R6 preserved

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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 following section. 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 (on page 189) command(or 0 if no descriptive text is provided).

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 (on page 187) 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(s)	Meaning
0	Path is not UNIX-like
1	No parsing should be performed on this scheme
2	Scheme allows "user@" to precede the hostname component
3	Hash (ASCII 35) allowed in hostname (e.g. for file: URLs)
4	No hostname component (e.g. mailto: URLs)
5	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:

1. mailto:
2. telnet:
3. finger:
4. file:

5. filer_opendir:
6. filer_run:
7. local:
8. gopher:
9. ftp:
10. http:
11. https:
12. whois:

It is not necessary for modules implementing those protocols to set either flag bit and hence no need for them to set R5 or R6.

Related SWIs

[SWI URL_ProtocolDeregister (on page 174)](#)

Related services

[Service_URLProtocolModule_ProtocolModule (on page 187)](#)

URL_ProtocolDeregister
(SWI &83E21)

Deregister a protocol module from the URL module.

On entry

R0 = Flags: All bits currently reserved (must be zero)
R1 = Protocol module's SWI base

On exit

R0 = Flags: All bits currently reserved (must be zero)
R1 = Number of client sessions that were using this module

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

SWI is not re-entrant

Use

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.

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 (on page 187) to inform any interested modules.

Related SWIs

SWI URL_ProtocolRegister (on page 171)

Related services

Service_URLProtocolModule_ProtocolModule (on page 187)

URL module to protocol module interface

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 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.

Protocol_GetData
(SWI URLFetcherProtocol+&00)

Start retrieving data

On entry

R0 = Flags:

Bit(s)	Meaning
0-30	As specified by client in SWI URL_GetURL (on page 143)
31	R7 is valid

R1 = Session identifier
R2 = Method (on page 145)
R3 = URL (including fetch scheme)
R4 = Pointer to block of data in addition to URL
R5 = Protocol dependent
R6 = Protocol dependent
R7 = If R0:31 is set, proxy URL information. See below

On exit

R0 = Protocol status word (see SWI SWI URL_Status (on page 146) for details)

Interrupts

Interrupts are protocol module dependent
Fast interrupts are protocol module dependent

Processor mode

Processor is in undefined mode

Re-entrancy

protocol module dependent

Use

This call is used to start retrieving data. The protocol module should raise any events for the client via the session identifier provided in R1. The URL module calls this SWI in response to one of its clients calling SWI SWI URL_GetURL (on page 143).

The proxy URL information specified in R7 (if R0:31 is set) gives the

location of the proxy to be used in the format of a URL. For example, "http://www-cache.demon.co.uk:8080/". This information is supplied by the URL module and not the client. The protocol module must note that on a proxied request, the target URL indicated by R3 may not have the same fetch scheme. For example, it might be an ftp: URL being proxied through an HTTP proxy service.

All other registers are protocol dependent.

⚠ FIXME: This text was originally in 'On Exit'

Related SWIs

Protocol_Status
(SWI URLFetcherProtocol+&01)

Monitor data transfer

On entry

R0 = Flags: All bits currently reserved (must be zero)
R1 = Session identifier

On exit

R0 = Protocol status word (see SWI [SWI URL_Status (on page 146)](#) for details)

R1 = preserved ⚠ FIXME: I added this

R2 = As [SWI URL_Status (on page 146)](#)
R3 = As [SWI URL_Status (on page 146)](#)
R4 = As [SWI URL_Status (on page 146)](#)

Interrupts

Interrupts are protocol module dependent
Fast interrupts are protocol module dependent

Processor mode

Processor is in undefined mode

Re-entrancy

protocol module dependent

Use

This SWI is used to monitor the transfer of data from the remote service. It is protocol independent, with the exit status bits of R0 being common to all fetcher services. R2 should contain the remote server's most recent response code where possible; note that even in the case of, for example, an HTTP 400 (Forbidden) response, some explanatory data may be received, and thus R3 may be non-zero. If the client is unknown to the protocol module then an error should be returned. If the client's last request has finished, but the client session has not yet been deregistered, then the protocol module should return the status code as of the time that the request finished (ie bit 6 or 5 will be set along with another combination if relevant).

The URL module calls this SWI in response to one of its clients

calling SWI [SWI URL_Status (on page 146)](#).

Related SWIs

Protocol_ReadData
(SWI URLFetcherProtocol+&02)

Read data pending from a request

On entry

R0 = Flags: All bits currently reserved (must be zero)
R1 = Session identifier
R2 = Address of client's data buffer
R3 = Size of client's data buffer

On exit

R0 = Protocol status word (see SWI [SWI URL_Status (on page 146)](#) for details)
R1 = preserved ⚠ FIXME: I added this
R2 = As [SWI URL_ReadData (on page 148)](#)
R3 = As [SWI URL_ReadData (on page 148)](#)
R4 = As [SWI URL_ReadData (on page 148)](#)
R5 = As [SWI URL_ReadData (on page 148)](#)

Interrupts

Interrupts are protocol module dependent
Fast interrupts are protocol module dependent

Processor mode

Processor is in undefined mode

Re-entrancy

protocol module dependent

Use

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. The register usage and description is the same as for SWI [SWI URL_ReadData (on page 148)](#). The URL module calls this SWI in response to one of its clients calling SWI [SWI URL_ReadData (on page 148)](#).

Related SWIs

Protocol_Stop
(SWI URLFetcherProtocol+&03)

Abort a current request

On entry

R0 = Flags: All bits currently reserved (must be zero)
R1 = Session identifier

On exit

R0 = Protocol status word (see SWI [SWI URL_Status (on page 146)](#)
for details)

Interrupts

Interrupts are protocol module dependent
Fast interrupts are protocol module dependent

Processor mode

Processor is in undefined mode

Re-entrancy

protocol module dependent

Use

This call aborts a current request if there is one associated with the
session identifier. The URL module calls this SWI in response to one
of its clients calling SWI [SWI URL_Deregister (on page 154)](#) or SWI
[SWI URL_Stop (on page 152)](#).

Related SWIs

[SWI URL_Stop (on page 152)](#)
[SWI URL_Deregister (on page 154)](#)
[SWI URL_ProtocolRegister (on page 171)](#)
[SWI URL_ProtocolDeregister (on page 174)](#)

URL module service calls

The URL fetcher system has been allocated a block of 256 service calls (&83E00-&83EFF). Two are currently defined. The other 254 are reserved by Acorn for future use.

Service_URLProtocolModule
(Service Call &83E00)

Communicate important events to protocol modules

On entry

R0 = Reason code indicating type of event:

Value	Name of event
0	URLModuleStarted (on page 185)
1	URLModuleDying (on page 186)

All other reason codes are reserved to Acorn and must not be used

R1 = &83E00 (Service_URLProtocolModule)

On exit

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 which they do not understand.

Use

The various reason codes are described below.

Related APIs

None

Service_URLProtocolModule 0
UrlModuleStarted
(Service Call &83E00)

URL module has initialised

On entry

R0 = 0 (URLModuleStarted)
R1 = &83E00 (Service_URLProtocolModule)
R2 = Version number of URL module × 100

On exit

None

Use

Upon receiving this service call, protocol modules should re-register with the new URL module by issuing SWI SWI URL_ProtocolRegister (on page 171) as usual. It must assume that any previous registration is no longer valid.

This service call must not be claimed.

Related APIs

None

Service_URLProtocolModule 1
UrlModuleDying
(Service Call &83E00)

URL module is dying

On entry

R0 = 1 (URLModuleDying)
R1 = &83E00 (Service_URLProtocolModule)
R2 = Version number of URL module × 100

On exit

None

Use

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 call arrives.

This service call must not be claimed.

Related APIs

None

Service_URLProtocolModule_ProtocolModule (Service Call &83E01)

A protocol module has registered or deregistered

On entry

R0 = Reason code:

Value	Meaning
0	URLProtocolModuleStarted (A protocol module has just registered)
1	URLProtocolModuleDying (A protocol module has just deregistered)
	All other reason codes are reserved

R1 = &83E01 (Service_URLProtocolModule_ProtocolModule)
R2 = URL fetch scheme (e.g. "http:", "ftp:")
R3 = SWI base chunk of protocol module
R4 = Description of module as shown by *URLProtoShow (on page 189)

On exit

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 which they do not understand.

Use

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 URLProtocolModule/URLModuleStarted (on page 185) service call arrives.

This service call must not be claimed.

Related APIs

None

URL module *-commands

The URL module provides a single *-command.

*URLProtoShow

Shows all the current protocols known and their SWI bases

Syntax

*URLProtoShow

Parameters

None

Use

Display information on currently registered protocol modules.

Help Text: "*URLProtoShow shows all the current protocols known and their SWI bases."

Examples

```
*URLProtoShow

Base URL SwiBase   Version  Comment
=====================================================================
  ---      0x83e00    038     URL © Acorn 1997-8 (Built: 07 May 1998)
gopher:  0x508c0    010     Gopher Fetcher © Acorn 1997-8 (Built: 17
ftp:     0x4bd00    028     FTP Fetcher © Acorn 1997-8 (Built: 19 Ma
file:    0x83f40    038     File Fetcher © Acorn 1997-8 (Built: 04 J
http:    0x83f80    082     Acorn HTTP © Acorn 1997-8 (Built: 07 May
```

Related SWIs

[SWI URL_EnumerateSchemes (on page 166)](#)

URL errors

The URL module is allocated two ranges of error numbers, each range being 256 long. The first 32 errors are reserved to the URL module and the rest are reserved to Acorn protocol modules.

Module	Error range
URL	&80DE00 - &80DE1F
HTTP	&80DE20 - &80DE3F
MAILTO	&80DE40 - &80DE5F
File	&80DE60 - &80DE7F
FTP	&80DE80 - &80DE9F
Gopher	&80DEA0 - &80DEBF
WhoIs	&80DEC0 - &80DEDF
Finger	&80DEE0 - &80DEFF
WAIS	&81EF00 - &81EF1F
HTTPS	&81EF20 - &81EF3F
News	&81EF40 - &81EF5F

Error numbers &81EF60-&81EFFF are reserved for Acorn use only. The URL module errors are:

Error no.	Meaning
&80DE00	Session ID not found. A client passed an unknown session ID in R1 to one of the URL module's SWIs
&80DE01	URL ran out of memory
&80DE02	No matching fetcher for the URL could be found
&80DE03	SWI not found (URL Module). URL attempted to call a fetcher's SWI and received a SWI not known error
&80DE04	Session already has had an object fetch performed in it. You cannot re-use this session
&80DE05	No fetch in progress for this session ID. You have called URL_ReadData or URL_Status having already terminated the fetch
&80DE06	SWI Method already exists. URL already knows of a module which provides this method for fetching - another cannot register
&80DE07	No fetch in progress for this session ID. You have not called URL_GetURL before URL_Stop,URL_ReadData or URL_Status
&80DE08	Message not found in Messages file
&80DE09	(No longer used)
&80DE0A	Unable to parse URL

Error numbers for protocol modules are not within the scope of this specification.

Performance targets

Final code size of the version described by this document should be about 25K. When fetches are active, more memory will be claimed from the RMA to record details of the session. The amount claimed depends on the URL being fetched plus the small overhead for the session information.

Temporary workspace is claimed from the RMA as required for URL resolution equivalent to three times the total combined length of the base and relative URLs involved.

Workspace is claimed from the RMA to store details of registered proxies.

All session-specific memory, including proxy information, is freed when the session is terminated.

Glossary

Term	Description
FTP	File Transfer Protocol - an application level protocol for the transfer of files between a remote host computer and a local client, as defined by RFC 959 [6]
HTTP	HyperText Transfer Protocol - a protocol designed to transfer resources ("documents") from a remote server machine to a local client, as defined by RFC 1945 (version 1.0 [4]) and RFC 2068 (version 1.1 [5])
HTTPS	Secure HyperText Transfer Protocol - HTTP protocol over a communication channel encrypted using SSL
URL	Uniform Resource Locator, as defined by RFC 1738 [2], [3] - a subclass of URIs (Uniform Resource Identifiers, defined in RFC 1630 [1]) which map onto network access protocols. More commonly, the addresses of objects on the World Wide Web
NNTP	Network News Transfer Protocol, as defined by RFC 977 [7]
Gopher	The Internet Gopher Protocol - a distributed document search and retrieval protocol
SSL	Secure Sockets Layer. A specification for encryption of communications on networks
WAIS	Wide Area Information Servers, as defined by RFC 1625 [8]

References

The following references may be of interest:

- [RFC 1630 - Uniform Resource Identifiers](#)
- [RFC 1738 - Uniform Resource Locators](#)
- [RFC 1808 - Relative Uniform Resource Locators](#)
- [RFC 1945 - HyperText Transfer Protocol (HTTP) version 1.0](#)
- [RFC 2068 - HyperText Transfer Protocol (HTTP) version 1.1](#)
- [RFC 959 - File Transfer Protocol (FTP)](#)
- [RFC 977 - Network News Transfer Protocol (NNTP)](#)
- [RFC 1625 - Wide Area Information Servers (WAIS) over Z39.50-1988](#)
- [1215,215/FS Acorn URI Handler Functional Specification](#)
  ⚠ FIXME: version I found on Internet

Document information

History:	Revision	Date	Author	Changes (Developers only)
	1215,2201			
	0.16	19 Oct 1997	RCE	**First formal version of specification based on uncontrolled textual programmer's notes (RCE)**
	0.16a	20 Oct 1997	RCE	**Incorporated notes from ADH and SB**
	0.19	17 Nov 1997	SNB	**Incorporated details of service calls**
	0.20	20 Nov 1997	SNB	**Incporated details of URL parsing SWI**
	0.21	11 Jun 1998	SNB	**All other updates incorporated**
	0.22	22 Jun 1998	SNB	**Comments after first review incorporated. Added details of proxy enumeration SWI**
	0.24	04 Aug 1998	SNB	**No longer live. ECO 4082.**
	0.25	12 Nov 1998	ADH	**Multiple changes** ● Four digit years on all dates. ● Tidied up white space. ● Removed smart quotes and n-dashes. ● Added author details to history. ● Corrected references on R0 exit words from URL_ParseURL to URL_Status. ● Added details of bit 1 of flags word in R0 to URL_ParseURL. ● Clarified a few sentences here and there. ECO 4131.
	0.25a	31 Aug 2021	Alan Robertson	**Initial version in PRMinXML format**
	0.25b	01 Sep 2021	Gerph	**Tiny tweaks to formatting**

Acorn Plug-In Protocol Functional Specification

Overview

The World Wide Web is gradually being extended to offer better support for embedding multimedia data inside Web pages. A well-established mechanism known as "helpers" allows a browser to delegate the display of unsupported data types to other applications. However, the helper application displays this data independently, usually in its own window.

The idea of a "plug-in" is to integrate the display of such data into the WWW browser's own window. A number of proposed HTML extensions are being promoted, such as <APPLET> (by Sun for Java), <EMBED> (by Netscape) and <OBJECT> (by W3C).

Outstanding issues

There are no outstanding issues.

Technical background

Navigator™ for the Mac and Windows™ supports plug-ins in the form of dynamically loaded code resources (DLLs). On finding data of a type it cannot display itself, the browser seeks a DLL which is capable of handling it. If it finds one, it calls standard entry points in the DLL to get it to display the data in the browser's window.

This model does not fit well with RISC OS practices. It does not have a standard scheme for DLLs, and the alternative - using relocatable modules - is not practical for very large playback engines for systems like Java and Director. Therefore plug-ins are implemented as separate tasks, with a special message protocol between the browser and the plug-in to permit communication and control.

In order to display the data inside the browser's own window, the plug-in needs to be made responsible for updating a certain portion of the browser's work area. This could be done by the browser instructing the plug-in to redraw parts of the window. However, this approach is rejected because it introduces significant differences between a plug-in and a normal application. Instead, we utilise a new facility

⚠ FIXME: Original document had link to Nested Window Manager

added to the Window Manager, whereby windows can be created "inside" a parent window. The Window Manager takes care of event distribution to the plug-in, and also ensures that the "child" window is in a fixed position relative to the work-area of the parent - so the plug-in's display area will be scrolled within the browser window if the user manipulates the browser window's scrollbars.

User interface

There is no user interface component to this specification.

Programmer interface

A plug-in accepts one or more types of data, specified using normal RISC OS filetypes. It is the responsibility of the browser to map MIME types to RISC OS filetypes.

Just as other RISC OS applications may be "single document" or "multiple document", a plug-in implementor may choose whether to handle multiple items at once or not. Ideally, plug-ins should be able to cope with multiple pieces of data, potentially owned by multiple client applications. However, for ease of implementation it may sometimes be preferred to restrict each instance of a plug-in to displaying one piece of data. In this case, were two such pieces of data to be displayed at once, it would be necessary to invoke the plug-in twice.

Invocation

Having determined the best RISC OS filetype for the data, the browser performs the following sequence of actions:

1. Broadcast Message_PlugIn_Open (on page 208), passing the filename and filetype of the data, and the parent window information. This message also contains an opaque 32 bit value known as the "browser instance handle". This is a word of significance to the browser, and might be different for each instance of a plug-in. The plug-in must always quote the correct browser instance handle to the browser in subsequent messages.
2. If a Message_PlugIn_Opening (on page 210) is received in reply, an existing invocation of a suitable plug-in has agreed to handle the data. The Message_PlugIn_Opening contains an opaque word value, known as the "plug-in instance handle", which together with the task handle of the plug-in task uniquely identifies the piece of data. The browser remembers both of these values for use in future messages.
3. If no task responds to Message_PlugIn_Opening, the browser attempts to launch the appropriate plug-in. This is done by looking for an environment variable called `Alias$@PlugInType_<xxx>` where *<xxx>* is the hexadecimal type value. If this variable is not found, no suitable plug-in is available, and the browser regards the attempt to display the data as unsuccessful. If the variable is found, then the browser launches it by calling Wimp_StartTask.
4. The result of Wimp_StartTask is the task handle of the new invocation of the plug-in. As soon as Wimp_StartTask returns, the browser re-broadcasts the Message_PlugIn_Open message.
5. Normally, the plug-in accepts this message and replies with Message_PlugIn_Opening, containing a plug-in instance handle as described above.

6. If no reply was forthcoming, the browser assumes that for some reason the plug-in was unable to load the data, and it regards the attempt to display this data as unsuccessful. This might be because the data is malformed, erroneous or of an incompatible version to that expected by the plug-in, or it might be because of some unexpected eventuality (out of memory, etc). If a detailed failure message is to be issued to the user, it is the responsibility of the plug-in to do this.

   If further data of the same type needs to be displayed, either simultaneously or sequentially, then the browser should repeat the whole process starting with the broadcast.
7. If the plug-in replies and so requests then the browser opens a data stream for the initial object being embedded and sends this data to the plug-in according to the plug-in stream (on page 203) protocol.
8. The plug-in examines the contents of the file that was named in the Message_PlugIn_Open message. This file contains all of the information from the OBJECT, EMBED or APPLET tag, and is used by the plug-in to initialise itself. The plug-in may have to fetch the contents of more URLs in order to do this; it may get the browser to do this on its behalf by using Message_PlugIn_URL_Access (on page 231).
9. If during startup the plug-in encounters an unrecoverable error it tidies up after itself and sends a Message_PlugIn_Closed (on page 213) to the browser, setting a flag in the message to indicate that this is due to an error. The message may optionally include an error message for the browser to display.

Shutdown

When the browser wishes the data to be forgotten, for example when the user quits the browser or leaves the current page, the following actions are taken. If multiple pieces of data have been farmed out (to the same or multiple plug-ins) the sequence below is performed for each such piece of data.

1. Browser sends Message_PlugIn_Close (on page 212) directly to the plug-in task, passing the plug-in instance handle associated with the data.
2. Plug-in closes and deletes its window, cleans up state and data, etc.
3. Plug-in replies with Message_PlugIn_Closed (on page 213).
4. Plug-in decrements its count of active objects. If the count is zero, it is free to exit if it wishes. A flag in the Message_PlugIn_Close acts as a hint to the plug-in as to whether the browser would like the plug-in to remain running or not, but the plug-in does not have to honour this if it does not want to.

Plug-in death

If the browser receives a Message_Task_CloseDown, it checks to see whether the exiting task was a plug-in that was currently displaying data on behalf of the browser. If so, all data being displayed by that plug-in is marked as undisplayable. The Window Manager has already deleted the child window(s) associated with the task. The browser might not issue any error in this case (for example, the NCBrowser does not); other possibilities are relaunching the plug-in or reporting the exit to the user.

Browser death

If the plug-in receives a Message_Task_CloseDown, it checks to see whether it is displaying data on behalf of the exiting task. If so, it deallocates any state or data associated with that task, and reduces its reference count by the correct amount. If the reference count reaches zero (i.e. the dead task was the only task using the plug-in), then the plug-in may exit if it wants to.

Window events

The Window Manager's nested window mechanism handles all subwindow positioning issues automatically. If a browser window is closed, then the subwindow is removed from view, and is reinstated when the parent window is reopened. If the browser window is scrolled, the Window Manager ensures that the plug-in window stays at the same position relative to the browser's work area, if necessary it repositions the subwindow and clips it if it has scrolled partiallly or entirely out of view. Repositioning is done by the Window Manager without sending Open_Window_Request events to the plug-in.

If the plug-in receives a keypress or mouse button click that it does not want to handle, it must pass it on to the browser by means of Wimp_SendMessage. It must set the window handle field of the message to the handle of its parent window. Note that this should be used instead of Wimp_ProcessKey.

If the browser wishes to forcibly resize or reposition the subwindow, it sends a [Message_PlugIn_Reshape (on page 215)](#) to the plug-in, quoting the plug-in instance handle. The plug-in must honour this request by re-opening itself at the new position. The coordinates in this request are work-area coordinates of the parent window. The parent window handle in this message may be different to the original one. The plug-in should be prepared to check for this, and re-create its window as a child of the new parent if necessary.

If the plug-in wishes to alter its size, it cannot simply resize its window. Instead it must send a [Message_PlugIn_Reshape_Request (on page 217)](#) to the browser. The browser responds by reformatting

the page (if necessary) and then replying with a suitable Message_PlugIn_Reshape. The plug-in must act on this in the normal way.

Data pointers

Many of the strings passed around in this protocol are of unspecified size and may, especially in the case of URLs, be larger than could fit within the body of a Wimp message. Therefore they are defined in this spec as string_values. These are defined as being either offsets from the start of the message body (if less than 256) or as pointers to data held in shared memory (i.e. the RMA or a dynamic area). It is always the responsibility of the sender to free the memory used for any such pointers. The protocol is defined in such a way that there should always be a reply received or the message will be bounced by the Window Manager. In either case it is then safe for the sender to free the memory allocated.

However to avoid memory leaks it is recommended that careful track is kept of such pointers so that they can be freed when a plug-in instance is closed.

All strings must be null terminated but need not start at a word-aligned address.

Stream protocol

Some plug-ins may wish the browser to fetch data from the net for them rather than having to implement their own fetching code. A flexible interface is provided for this based, in part, on the API used in the de facto standard plug-in API created by Netscape, in order to facilitate porting plug-ins to RISC OS.

N.B. Take note of the .

There are several ways a stream can be instigated, as follows

- The browser wishes to transfer the initial data which launched the plug-in
- The plug-in requests some data be fetched for it with
- The plug-in requests some data be posted for it with Message_PlugIn_URL_Access
- The plug-in wishes to write directly to a browser window

Initial transfer

1. Browser fills in flags, mime type, stream data and sends .
2. The plug-in returns the same message

- quoting the reference
- filling in the plug-in stream instance handle
- updating the stream mode (if necessary)

4. If mode is applicable
   1. Browser sends [Message_PlugIn_Stream_Write (on page 225)](#)
   2. Plug-in replies with [Message_PlugIn_Stream_Written (on page 227)](#) giving the number of bytes that is could process

   This is repeated until all data is transferred or an error occurs
3. Browser sends [Message_PlugIn_Stream_Destroy (on page 223)](#) with appropriate reason code

Plug-in requests data be fetched or posted

1. Plug-in sends the [Message_PlugIn_URL_Access (on page 231)](#) message
2. When data starts arriving we continue as initial transfer

Plug-in write to browser

1. Plug-in fills in MIME type, target, plug-in stream instance and sends [Message_PlugIn_Stream_New (on page 221)](#)
2. The browser returns the same message
   - quoting the reference
   - filling in the stream fields
3. The plug-in writes data;
   1. Plug-in sends [Message_PlugIn_Stream_Write (on page 225)](#)
   2. Browser replies with [Message_PlugIn_Stream_Written (on page 227)](#) giving the number of bytes that it could process

   This is repeated until all data is transferred or an error occurs
3. Plug-in sends [Message_PlugIn_Stream_Destroy (on page 223)](#) with appropriate reason code

System variables

For a plug-in *<yyyy>* whose file type is *<xxx>* the variables which the plug-in must set are:
<yyyy>`$Dir` The application directory containing !Boot, !Run etc. files
`PlugIn$Type_`*<xxx>* Name of plug-in for browser menu
`Alias$@PlugInType_`*<xxx>* Command to run plug-in as a stand-alone application, no arguments

The plug-in can optionally set these variables:
`PlugIn$About_`*<xxx>* The directory containing plug-in copyright details

If the plug-in is capable of being launched as a stand-alone

application without the browser involvement it must define these variables:

`File$Type_<xxx>` Up to 8 character name describing file format
`Alias$@RunType_<xxx>` Command to run plug-in as a standalone application, takes filename as an argument

If the plug-in can also be used as a helper application then this variable must also be set:
`Alias$@HelperType_<xxx>` Command to run plug-in as a helper application

For example a sample !Boot file might contain the following:

```
Set Java$Dir          <Obey$Dir>
Set File$Type_AE4     Java
Set PlugIn$Type_AE4   Java
Set PlugIn$About_AE4 <Java$Dir>.About

SetMacro Alias$@RunType_AE4     /<Java$Dir>.!RunImage -standalone
SetMacro Alias$@PlugInType_AE4 /<Java$Dir>.!RunImage -plug-in %%*
```

If a file is embedded with `APPLET`, `EMBED` or `OBJECT` then the `Alias$@PlugInType_<xxx>` variable is used to start the application.

If a file is pointed to with an anchor (eg `<A HREF="applets/myapplet.class">`) then the file is downloaded and the `Alias$@Runtype_<xxx>` variable is used.

The OBJECT tag

Note that plug-ins can be launched from an OBJECT tag as well as EMBED or APPLET. When this happens there are some minor differences to the values in the parameter file (on page 241) The following table also describes how the attribute names in the HTML tag get mapped to the entries in the parameters file:

EMBED	APPLET
The SRC attribute is named DATA	The ALT attribute is named STANDBY The CODE attribute is named CLASSID The value of CLASSID may not have the ".class" suffix The value of CLASSID may have a prefix "java:"

Helper applications

This same interface is also used for helper applications. Helper applications are very like plug-ins except that they open their windows external to the parent rather than embedded in the parent's window. This means that they are not constrained to close down when the parent window is closed (e.g. when the browser follows a link to another page) but can still benefit from the communication protocols with the parent. There is a flag in Message_PlugIn_Opening

to inform the parent whether a window was embedded or not.

When trying to launch a helper application the process described in the Invocation section is used except that if the initial Message_PlugIn_Opening is not claimed the system variable `Alias$@HelperType_<xxx>` is used to start the helper task.

Help protocol

A plug-in may support the Wimp Help protocol. If they do then help mesages are displayed in the browser status bar (if configured). Messages must be limited to at most 40 characters.

About plug-in

A plug-in may display a logo and some associated text (e.g. copyright information) in a browser's window at the user's request. The suggested URL for this is 'about:'. The system variable `PlugIn$About_<xxx>` points to a directory containing text files with optional image (PNG, GIF or JPEG) files.

Each file has a two digit reference number to allow a single plug-in to have multiple logos and copyright entries (e.g. each Replay codec). The file 'About*<yy>*', where *<yy>* is the two digit reference number, contains the text suitable for inclusion inside a table cell of an HTML document. For each About file there is an optional image file, of the name '*<yy><wwww><hhhh>*', where *<wwww>* and *<hhhh>* are each four digits for the size that the image will be scaled to (usually the same as the actual image size). It is strongly recommended that the width and height are specified, but a filename of just '*<yy>*' is accepted.

If the plug-in has a single copyright message and logo, the filename 'About' can be used as a shortcut for 'About00'. The optional logo must still be called '00*<wwww><hhhh>*' or '00'.

It is the browser's responsibility to enumerate all the `PlugIn$About_<*>` system variables and compile an HTML document containing all available plug-in details.

Data interchange

The following new Wimp messages are defined.

⚠ FIXME: All Messages delivery elements to be checked for errors.

Message_PlugIn_Open (&4D540)

Sent by the browser to create a plug-in instance

Message

Offset	Contents
R1+16	Message_PlugIn_Open
R1+20	Flags:

Bit(s)	Meaning
0	Open the file as a helper (else open it as a plug-in)
1-31	Reserved, must be zero

Offset	Contents
R1+24	Reserved, must be zero
R1+28	Browser instance handle (provided by the browser)
R1+32	Parent window handle
R1+36	Bounding box in parent window's work area co-ordinates: Left
R1+40	Bounding box in parent window's work area co-ordinates: Bottom
R1+44	Bounding box in parent window's work area co-ordinates: Right
R1+48	Bounding box in parent window's work area co-ordinates: Top
R1+52	File type
R1+56	Filename (string_value) (on page 203)

Source

Browser

Destination

Plug-in

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

The file specified by "Filename" at R1+56 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 plug-in to understand what is being requested of it. There are more details in the [Data formats (on page 241)](#) section.

If bit 0 of the flags word at R1+20 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 36 to 51) are invalid. The parent window handle may be valid or 0 depending on how the file is launched.

Related messages

[Message_PlugIn_Opening (on page 210)](#)

Message_PlugIn_Opening
(&4D541)

Sent by the plug-in task to say an instance has been created

Message

Offset	Contents
R1+12	my_ref field from Message_PlugIn_Open (on page 208)
R1+16	Message_PlugIn_Opening
R1+20	Flags:

Bit(s)	Meaning
0	Plug-in can accept input focus (else it cannot use input focus)
1	Plug-in wants the code resource fetched for it (else it will fetch this itself)
2	Plug-in wants the data resource fetched for it (else it will fetch this itself)
3	Plug-in will delete the parameters file itself (else the browser should delete this file now)
4	Plug-in has more work to do, keep showing a busy indicator in the browser (if appropriate)
5	Plug-in does understand the PlugIn_Action message beyond only the STOP reason code
6	Plug-in task has actually opened a helper window (else it embedded itself in the parent)
7-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle (invented by the plug-in)
R1+28	Browser instance handle (copied from the Message_PlugIn_Open (on page 208))

Source

Plug-in

Destination

Browser

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This is sent by the plug-in in response to [Message_PlugIn_Open (on page 208)](#). Note that bit 6 of the flags word at R1+20 may indicate that the Plug-in opened a helper window even if the browser requested that it be embedded as a plug-in.

Related messages

[Message_PlugIn_Open (on page 208)](#)

Message_PlugIn_Close (&4D542)

Tell a plug-in instance to close down

Message

Offset	Contents
R1+16	Message_PlugIn_Close
R1+20	Flags:

Bit(s)	Meaning
0	Browser would also like plug-in to exit
1-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle to close
R1+28	Browser instance handle

Source

Browser

Destination

Plug-in

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is sent by the browser to a plug-in, if it wants an instance of a plug-in to be closed down (e.g. because the browser window is being closed, or is moving to a new page). Bit 0 of the flags word at R1+20 may be set if the browser needs urgently to free up memory; it is a hint to the plug-in to free up as much memory itself as it can. Not all plug-ins will read this bit.

Related messages

Message_PlugIn_Closed (&4D543)

A plug-in [instance] has closed down

Message

Offset	Contents
R1+12	my_ref field from [Message_PlugIn_Close (on page 212)](#), unless bit 1 of the flags word at R1+20 is set
R1+16	Message_PlugIn_Closed
R1+20	Flags:

Bit(s)	Meaning
0	Plug-in itself will exit after this message
1	The message is not in reply to a Message_PlugIn_Close (so R1+12 is irrelevant)
2	There is an error message at R1+32 as detailed below
3-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle of the closed instance
R1+28	Browser instance handle of the closed instance
R1+32	If R1+20 Bit 2 is set: Error number
R1+36	If R1+20 Bit 2 is set: Zero terminated message to be displayed by the browser (N.B. this message is always embedded here as the plug-in may be exitting itself)

Source

Plug-in

Destination

Browser

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is usually sent as a reply to a Message_PlugIn_Close (on page 212) from the browser, and confirms that the requested instance has been closed down. It may also be sent if the plug-in should exit for its own reasons without the browser asking. An error which the browser should display will be embedded in the message at R1+32, if bit 2 of the flags word at R1+20 is set.

Related messages

Message_PlugIn_Close (on page 212)

Message_PlugIn_Reshape (&4D544)

Move or resize a plug-in instance

Message

Offset	Contents
R1+12	my_ref field from [Message_PlugIn_Reshape_Request (on page 217)](#) (if applicable)
R1+16	Message_PlugIn_Reshape
R1+20	Flags:

Bit(s)	Meaning
0-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle
R1+32	Parent window handle
R1+36	Bounding box in parent window's work area co-ordinates: Left
R1+40	Bounding box in parent window's work area co-ordinates: Bottom
R1+44	Bounding box in parent window's work area co-ordinates: Right
R1+48	Bounding box in parent window's work area co-ordinates: Top

Source

Browser

Destination

Plug-in

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is sent by a browser to a plug-in. The plug-in should move the specified instance to the specified position; this may involve resizing the embedded window.

Some plug-in types may want to resize the windows themselves (for example, some Java applets do this). In that case, they will send [Message_PlugIn_Reshape_Request (on page 217)](#) to the browser and it should reply with Message_PlugIn_Reshape once it has determined where the plug-in should be moved to (since the resizing may affect page formatting and therefore the coordinates of the embedded plug-in window). A plug-in should therefore not expect an immediate reply to the message.

Related messages

[Message_PlugIn_Reshape_Request (on page 217)](#)

Message_PlugIn_Reshape_Request (&4D545)

A plug-in instance wants to resize

Message

Offset	Contents
R1+16	Message_PlugIn_Reshape_Request
R1+20	Flags:

	Bit(s)	Meaning
	0-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle
R1+32	Width (in OS units)
R1+36	Height (in OS units)

Source

Plug-in

Destination

Browser

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

A plug-in may sometimes want to resize its embedded window. It sends this message to the browser when it does so. The browser should respond with Message_PlugIn_Reshape (on page 215), though it may not do so immediately.

On sending this message a plug-in may immediately resize its window, or it may wait; this is undefined. The browser should not assume either. To be sure that the plug-in embedded window ends up in a sensible position, the browser must eventually reply to the message.

Related messages

Message_PlugIn_Focus (&4D546)

Move the input focus between plug-in and parent

Message

Offset	Contents
R1+16	Message_PlugIn_Focus
R1+20	Flags:

	Bit(s)	Meaning
	0-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle

Source

Browser or plug-in

Destination

Browser or plug-in

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is used to transfer the input focus between a plug-in and its parent. It can be sent in either direction. 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.

Related APIs

None

Message_PlugIn_Unlock (&4D547)

This message call is for internal use only. You must not use it in your own code.

Message_PlugIn_Stream_New (&4D548)

Create a new stream

Message

Offset	Contents
R1+16	Message_PlugIn_Stream_New
R1+20	Flags:

Bit(s)	Meaning
0-3	Stream type field:

Value	Meaning
0	Normal
1	Seek only
2	As file
3	As file only

All other values are reserved, and must not be used

Bit(s)	Meaning
4	Stream is seekable
5-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle
R1+32	Plug-in stream instance handle
R1+36	Browser stream instance handle
R1+40	URL of stream source / destination (string_value (on page 203))
R1+44	End of stream in bytes, or 0 if unknown
R1+48	Last modified date of URL (in Unix time)
R1+52	Notify data
R1+56	MIME type of URL (string_value (on page 203))
R1+60	Window target (string_value (on page 203))

Source

Browser

Destination

Plug-in

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is part of the [Stream protocol (on page 203)](#) as already described.

Related messages

[Message_PlugIn_Stream_Destroy (on page 223)](#)

Message_PlugIn_Stream_Destroy (&4D549)

Destroy a stream

Message

Offset	Contents
R1+16	Message_PlugIn_Stream_Destroy
R1+20	Flags:

Bit(s)	Meaning
0-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle
R1+32	Plug-in stream instance handle
R1+36	Browser stream instance handle
R1+40	URL of stream source / destination ([string_value (on page 203)](#))
R1+44	End of stream in bytes, or 0 if unknown
R1+48	Last modified date of URL (in Unix time)
R1+52	Notify data
R1+56	Reason code:

Value	Meaning
0	Stream finished successfully
1	Stream finished due to an error
2	Stream finished due to user intervention

All other values are reserved, and must not be used

Source

Plug-in

Destination

Browser

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is part of the [Stream protocol (on page 203)](#) as already described.

Related messages

[Message_PlugIn_Stream_New (on page 221)](#)

Message_PlugIn_Stream_Write (&4D54A)

Write data to a stream

Message

Offset	Contents
R1+16	Message_PlugIn_Stream_Write
R1+20	Flags:

	Bit(s)	Meaning
	0-3	Data type field:

		Value	Meaning
		0	String_value (on page 203)
		1	Anchor
		2	File handle
			All other values are reserved, and must not be used

	Bit(s)	Meaning
	4-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle
R1+32	Plug-in stream instance handle
R1+36	Browser stream instance handle
R1+40	URL of stream source / destination (string_value (on page 203))
R1+44	End of stream in bytes, or 0 if unknown
R1+48	Last modified date of URL (in Unix time)
R1+52	Notify data
R1+56	Logical offset in stream of data
R1+60	Length of data
R1+64	Data pointer

Source

Browser

Destination

Plug-in

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is part of the Stream protocol (on page 203) as already described.

Related messages

Message_PlugIn_Stream_Written (on page 227)

Message_PlugIn_Stream_Written (&4D54B)

Accept data that was written to a stream

Message

Offset	Contents
R1+12	my_ref field from [Message_PlugIn_Stream_Write (on page 225)](#)
R1+16	Message_PlugIn_Stream_Written
R1+20	Flags:

Bit(s)	Meaning
0-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle
R1+32	Plug-in stream instance handle
R1+36	Browser stream instance handle
R1+40	URL of stream source / destination ([string_value (on page 203)](#))
R1+44	End of stream in bytes, or 0 if unknown
R1+48	Last modified date of URL (in Unix time)
R1+52	Notify data
R1+56	Length of data consumed; less than zero if the plug-in experienced an error

Source

Plug-in

Destination

Browser

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is part of the [Stream protocol (on page 203)](#) as already described.

Related messages

[Message_PlugIn_Stream_Write (on page 225)](#)

Message_PlugIn_Stream_As_File (&4D54C)

Send stream data as a file

Message

Offset	Contents
R1+16	Message_PlugIn_Stream_Written
R1+20	Flags:

	Bit(s)	Meaning
	0-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle
R1+32	Plug-in stream instance handle
R1+36	Browser stream instance handle
R1+40	URL of stream source / destination ([string_value (on page 203)](#))
R1+44	End of stream in bytes, or 0 if unknown
R1+48	Last modified date of URL (in Unix time)
R1+52	Notify data
R1+56	Filename of stream data (string_value)

Source

Browser

Destination

Plug-in

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

⚠ FIXME: Confirm Source and Destinations for this Message.

This message is part of the [Stream protocol (on page 203)](#) as already described.

Related messages

Message_PlugIn_URL_Access (&4D54D)

Ask the browser to deal with a URL

Message

Offset	Contents
R1+16	Message_PlugIn_URL_Access
R1+20	Flags:

Bit(s)	Meaning
0	Return a [Message_PlugIn_Notify (on page 233)](#) on completion
1	Fetch by POST, else fetch by GET
2	Should be 0 if bit 1 is unset If bit 1 is set, bit 2 means POST a file if set, else POST a block of memory
3-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle
R1+32	URL to access ([string_value (on page 203)](#))
R1+36	Window target ([string_value (on page 203)](#))
R1+40	Notify data to be returned (if bit 0 of the flags word at R1+20 is set)
R1+44	Length of data to be posted
R1+48	If R1+20 bit 2 is set: Filename ([string_value (on page 203)](#)) If R1+20 bit 2 is unset: Pointer to data ([string_value (on page 203)](#))

Source

Plug-in

Destination

Browser

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is sent by a plug-in to the browser, to ask it to deal with
a URL in various ways. The plug-in may ask the browser to send it a
notification message when it has completed whatever action is
required on the URL.

If the window target is non-zero then the URL is fetched to the given
window name. Otherwise, a stream is opened and the data is sent to
the plug-in.

Related messages

Message_PlugIn_Notify (on page 233)

Message_PlugIn_Notify (&4D54E)

Signal completion of handling a URL to a plug-in

Message

Offset	Contents
R1+16	Message_PlugIn_Notify
R1+20	Flags:

Bit(s)	Meaning
0-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle
R1+32	URL accessed (string_value (on page 203))
R1+36	Reason for notify:

Value	Meaning
0	Stream finished successfully
1	Stream finished due to an error
2	Stream finished due to user intervention
	All other values are reserved, and must not be used

Offset	Contents
R1+40	Notify data

Source

Browser

Destination

Plug-in

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This is sent by the browser to the plug-in, because the plug-in requested it through a Message_PlugIn_URL_Access (on page 231).

Related messages

Message_PlugIn_Status (&4D54F)

Send a status message to the browser

Message

Offset	Contents
R1+16	Message_PlugIn_Status
R1+20	Flags:

Bit(s)	Meaning
0-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle
R1+32	Status message (string_value (on page 203))

Source

Plug-in

Destination

Browser

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

Requests that the parent display some information in its status bar, or similar. The message should be reasonably short.

Related messages

Message_PlugIn_Busy (on page 236)

Message_PlugIn_Busy (&4D550)

Signal a plug-in state change to the parent

Message

Offset	Contents
R1+16	Message_PlugIn_Busy
R1+20	Flags:

Bit(s)	Meaning
0	Plug-in is busy
1	Word at R1+32 has some meaning, else ignore it
2-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle
R1+32	If R1+20 bit 1 is set: Plug-in's new state:

Value	Meaning
0	Stop
1	Play
2	Pause
3	Fast Forward
4	Rewind
5	Record

All other values are reserved, and must not be used

Source

Plug-in

Destination

Browser

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

> ⚠ FIXME: Flag to be changed. bit 1 seems to be used as an identifier

Requests that the parent display some indication of business (e.g. spinning logo, etc.). If the plug-in had set the busy bit in its Opening message then it should send this message with bit 0 of the flags word at R1+20 clear when it has finished its loading.

This is also used to notify the parent of any state change by the plug-in in case it needs to update any user interface.

Related messages

Message_PlugIn_Action (&4D551)

Send a command to a plug-in

Message

Offset	Contents
R1+16	Message_PlugIn_Action
R1+20	Flags:

Bit(s)	Meaning
0-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle
R1+32	If R1+20 bit 1 is set: State the plug-in should move to:

Value	Meaning
0	Stop
1	Play
2	Pause
3	Fast Forward
4	Rewind
5	Record
6	Mute
7	Unmute

All other values are reserved, and must not be used

Source

Browser

Destination

Plug-in

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

⚠ FIXME: Flag to be changed. bit 1 seems to be used as an identifier

This message is used for sending specific commands to a plug-in. Not all plug-ins will understand the commands sent.

The new state sent is the state the plug-in 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 (on page 236) confirming the new state, except for the Mute and Unmute actions.

Related messages

Message_PlugIn_Busy (on page 236)

Message_PlugIn_Abort (&4D552)

Stop activity for a plug-in instance

Message

Offset	Contents
R1+16	Message_PlugIn_Abort
R1+20	Flags:

	Bit(s)	Meaning
	0-31	Reserved, must be zero

Offset	Contents
R1+24	Plug-in instance handle
R1+28	Browser instance handle

Source

Browser

Destination

Plug-in

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is sent by the browser when the user clicks on the Stop icon (or performs its equivalent). The plug-in should stop as much of its activity as possible. Specifically, anything that updates the screen, anything that uses significant CPU time and anything that accesses the network.

Note that this message is sent to each plug-in instance individually and should be treated as such.

Related messages

Message_PlugIn_Status (on page 235)

Data formats

The Message_PlugIn_Open (on page 208) contains a filename that refers to a file of parameters and attributes. The plug-in uses this information to locate the correct data, classes, implementation etc.

The file contains the concatenation of one or more binary records of the following form:

Record size	Record contents
4 bytes	Type: 0 = terminator (this is the last word in the file) 1 = data from PARAM 2 = URL from PARAM 3 = object ref PARAM 4 = special parameter from browser
4 bytes	Size of record (without header): (4+n+p) + (4+s+q) + (4+t+r) bytes
4 bytes n bytes p bytes	n = size of name *(unpadded)* Name Padding to word boundary
4 bytes s bytes q bytes	s = size of data *(unpadded)* Data Padding to word boundary
4 bytes t bytes r bytes	t = size of mime type *(unpadded)* Mime type Padding to word boundary

Integers are stored in little-endian order.

Flags (parameters with void value whose presence or absence only is significant) are represented by a parameter of type DATA with zero length.

The parameters include:

- all the attributes of the OBJECT (or other) element that references this plug-in
- all the PARAM elements enclosed within it
- special parameters created by the browser

These parameters are passed exactly as seen in the HTML without any conversions. The data/url/ref distinction is as given in the DTD (for OBJECT attributes) or in the VALUETYPE attribute of the PARAM element.

The plug-in may implement its own URL fetching code, or it may have the browser fetch URLs on its behalf by issuing a Message_PlugIn_URL_Access (on page 231) message to the browser.

Special parameters are created by the browser (rather than being part of the object element). They are:

Parameter	Meaning
BASEHREF	(Mandatory) The full URL of the document containing this object.
USERAGENT	(Mandatory) The name of the browser.
UAVERSION	(Mandatory) Version number of the browser (user agent) in format x.y. If the plug-in needs a specific browser feature it may refuse to initialise if this version is not high enough.
APIVERSION	(Mandatory) Version number (on page 242) of this API in format x.y. Changes in x mean a major incompatible change in formats. If the plug-in doesn't understand this version it should refuse to initialise. Changes in y mean some new functionality introduced in a backwards compatible way.
BGCOLOR	(Optional) The background colour of the page, which can be used by the plug-in as the default background colour. The colour is passed in as a string in the format 'BBGGRR00'.

API Versions

Released versions of this API about are listed below. A plug-in can use this information to alter its behaviour if used with an application supporting an older version of the protocol.

Version 1.00
Original plug-in specification (version 0.09). All messages up to and including STATUS. Supported by NCBrowser 1.06, for example.

Version 1.10
This specification. Adds BUSY, ACTION, ABORT messages and support flags. Adds Helper information. Supported by NCBrowser 1.07 and above, or Browse 1.27 and above, for example.

External dependencies

This specification relies on the existence of a Window Manager with nested window support.

Acceptance test

The protocol must be able to cater for the needs of [Shockwave (on page 247)](#) and [Java (on page 247)](#) plug-ins.

Non-compliances

At the time of writing, neither the NCBrowser nor Browse support all of the variations of the [STREAM protocol (on page 203)](#). They do not support POSTing from a plug-in or streaming data from a plug-in to a browser window.

Development test strategy

The protocol has been tested during the development testing of a browser and during the creation of plug-ins. Each plug-in listed in the [acceptance criteria (on page 244)](#) tested that its interactions with the browser through this protocol performed as expected by this specification. This included deliberately generating errors to check the error recovery of the plug-in and the browser.

Glossary

Term	Description
Aplet	Application Programmer Interface.
API	Small application, usually written in Java, embedded in a web page
ARM	Acorn RISC Machine OR Advanced RISC Machines Ltd
Cache	Area of disk or memory used to store recently accessed files
Caret	Text cursor
Codec	COder-DECoder
Director Player	MacroMedia multi-media animation player
DLL	Dynamically linked library (loaded at runtime)
Frame	An independently scrollable portion of an HTML page
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTML 4	The current base-line HTML standard
Java	Machine independent interpreted programming language
MIME	Multipurpose Internet Mail Extensions
NC	Network Computer
OS	Operating System
Plug-in	A program that extends the browser by handling a particular type of file embedded in an HTML page
PRM	Programmers Reference Manual
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RISC OS	Acorn's operating system, the basis of RISC OS
ROM	Read Only Memory
Shockwave	MacroMedia multi-media browser plug-in player
Sprite	An Acorn proprietary bitmap graphics file format
SWI	Software Interrupt
UI	User Interface
URL	Uniform Resource Locator (HTML link)
Wimp	Colloquialism for Window Manager

References

The following references may be of interest:

Director Player Software Functional Specification

Document reference 2107,711 (covers Shockwave as well as Director Movies). Obtain through Developer Support.

Java Software Functional Specification

Document reference 2107,710. Obtain through Developer Support.

[NC] Browser Software Functional Specification

Obtain through Developer Support.

Acorn Nested Window Manager Functional Specification

Document reference 1215,401/FS.

Wimp message protocol

PRM Volume 3.

Wimp Help protocol

PRM Volume 3.

Document information

2107,740 issue 1			**(Developers only)**
0.01	09 Jan 1997	SJM	**Created from 2103,740 and added BUSY notification protocol**
1.2	06 Feb 1997	SJM	**New format**
1.3	07 Feb 1997	SJM	**Fixed errors in Message numbers**
1.4	14 Feb 1997	SJM	**Added PASSWORDS. Changed API version. Added Glossary**
1.5	18 Feb 1997	SJM	**Changed PASSWORDS to file. Added Message_PlugIn_Action. Added Helper app info**
1.6	24 Feb 1997	SJM	**Added ABORT message to replace some uses of STOP**
1.7	24 Feb 1997	SJM	**Fixed error in states. Changed API info**
1.9	26 Feb 1997	SJM	**Added mute**
1.12	09 Apr 1997	SJM	**Added missing history comments for 1.10 and 1.11, updated with comments from SG. Added glossary, references and development test strategy. Added Helper launching system variable**
1.13	11 Apr 1997	SJM	**Fixed typos after review**
1.14	11 Apr 1997	SJM	**Fixed some links**
2.1	11 Apr 1997		**Signed off, AMR allocated**
3.1	11 Aug 1997		**Few small changes; then signed off, ECO 3995 allocated**
1116,010/ FS issue 1			**(Developers only)**
1.0	26 Jan 1998	PW	**Added 'About Plug-in'**
1.1	06 Feb 1998	PW	**Added BGCOLOR special parameter (PW); AMR 4903 allocated**
1116,010/ FS Issue 2			**(General release)**
1.0			● HTML style changes

			for publishing on the Web; some clarifications here and there in the body content.
			● Various minor 'tweaks' such as changing, for example, "Netscape" to read "Navigator™".
			● There are a few more in-document links to make finding things easier.
			● Some typos corrected (e.g. 'data' changed to 'date').
			● A few history and references bits removed ready for general public release (that's why the revision list given here has gaps in it).
1.1	23 Feb 1998	Simon Middleton (SJM), Piers Wombwell (PW), Andrew Hodgkinson (AH)	**AMR allocation details corrected in this history section**
1.2	23 Feb 1998	SJM, PW, AH	**ECO 4049 allocated**
1.3	26 Mar 1998	SJM, PW, AH	**Created revision 1.3 purely to fix the erroneous reference to the Nested Wimp specification which gave an incorrect drawing number. No ECO allocated for such a trivial change**
1.4a	04 Sep 2021	Alan Robertson	**Initial version in PRMinXML format** ● No major changes to text. Removed the 'Document Status' section as information captured in

'Document Information' section

- Added related links to message definitions

Disclaimer: This document has a fairly long history; originally it was an internal-only specification (2103,740); it later became available to developers in a revised form (2107,740 and recently 1116,010/FS issue 1) and is now on general release (1116,010/FS issue 2).

Acorn Nested Window Manager Functional Specification

Overview

Version 3.97 and 3.98 of the Wimp are beta versions, incorporating extensions required by numerous projects. The main features are:

- Nested windows
- Icon bar auto-fronting
- Icon bar scroll regulation and acceleration
- 24-bit icon colour specification
- Border-less windows
- New filter types
- Redraw optimisation
- Numerous bugfixes and other optimisations

Technical Background

This document documents changes to the Wimp over the version present in RISC OS 3.71, as determined from the PRM volumes 3 and 5a and the old Wimp itself.

The Wimp has been written so that any given version can be built to soft-load on any OS version back to 3.10. Builds of version 3.97 and version 3.98 suitable for RISC OS 3.1x, 3.5x, 3.6x and 3.7x have been released with beta status for external testing, because a nested Wimp is a prerequisite of the browser and Java. The main differences from the RISC OS 3.7x build in those for earlier operating systems are as follows:

In RISC OS 3.6x and earlier, the Wimp

- has no support for StrongARM
- handles task memory management (rather than delegating to OS_AMBControl)

In RISC OS 3.5x and earlier, the Wimp

- always plots sprites using a translation table (ColourTrans module isn't new enough to plot paletted sprites from the palette)

In RISC OS 3.1x, the Wimp

- handles memory management significantly differently (for example, it doesn't use dynamic areas)
- doesn't assume FPEmulator 4.00 or later will be present
- doesn't support mode specifiers

User Interface

Child and Nested Windows

The single biggest enhancement to the Wimp is support for child windows: windows that are linked to and are only displayed within their parent. Each edge, and both scroll offsets of every child window are independently linked to the work area or one edge of the parent window, in whatever way suits the task. Thus, when a parent window is moved, scrolled or resized, any related changes to child windows are dealt with automatically by the Wimp.

Any window may have any number of children, and within each window there is a stack of child windows, whose relative depth can change in the same manner as the top-level window stack. Child windows may change their parent at any time, jumping between stacks. Child windows may themselves have nested children within them, which may in turn have their own children, and so on. Child windows are considered to lie above any icons in their parent window, and above the parent window's caret.

It should be noted that with all this added flexibility comes a potential for badly designed, non-intuitive application front-ends, so care must be taken when designing a user interface which uses the window nesting facilities.

Child Windows Without a Work Area

In the past, it has not been possible to display windows without a work area, for example to implement a scroll bar in a non-standard place. This was due not only to a hard-coded minimum size visible area (see Minimum Sizes (on page 257)) but also due to the ever-present single-pixel border drawn by the Wimp along any edge of a window that lacks window furniture. In fact, it has previously been possible to remove the single-pixel border, but at the expense of the removal of all the window furniture at the same time, leaving an isolated work area. These restrictions have been lifted.

This behaviour is in fact also available with top-level windows, but its usefulness is expected to be limited to child windows.

Furniture Windows

Normally, child windows are clipped according to the visible area of the parent window. However, there are occasions where it is desirable for the child windows to be clipped by the window outline - that is, allowing them to overlap the window furniture. This might, for example, be used to display status information within the scrollbars, or add window furniture buttons. An additional window

flag has been introduced, allowing a child window to overlap the window furniture of its parent in this way.

Further, it is realised that to place such a furniture window at, for example, the bottom left of a window, would obscure the parent's scroll arrow icon. In order to compensate for this, scrollbars are allowed to move to accommodate any child window found to be touching both the outside edge of the scrollbar, and the end of the scrollbar. A window is deemed to be touching the end of the scrollbar if its end outline coordinate is within 1 pixel of the end of the scrollbar, i.e. there must be no gap between the child window outline and the final pixel of the scrollbar. The end of the scrollbar is adjusted so that it just underlaps the other end of the child window outline. The child window should normally be wide enough to cover all the area where the scroll bar would have been, as only a blank area of colour will be drawn there otherwise.

Notice that these constraints allow for four furniture windows within the scrollbars of the parent. If, for example, the developer wanted two child windows in the bottom left, one window would have to be made a child of the other. Under current Wimp behaviour, if two sibling child windows are placed side by side in this manner, the scrollbar will move to accommodate both of them, but only if they are in a certain stacking order; this behaviour is not guaranteed in future versions of the Wimp, and must not be relied upon.

Again, the flexibility offered by these child windows must not be abused: developers must, for example, take steps to ensure that the parent's minimum visible area is large enough for the furniture window never to overlap the parent's adjust size button.

Furniture windows which are independently moveable and/or resizable are beyond the scope of the Nested Window Manager, and any attempt to give them such abilities will result in unpredictable behaviour. Such designs are thus strongly discouraged.

Windows in General

Invalid Rectangle Handling

The way in which the Wimp calculates the invalid and block-copy rectangle lists is optimised over old Wimps, in order to increase the proportion of block-copy operations, which are usually much faster than redraws. This is done by compiling a list of changes between each call to Wimp_Poll, and only then calculating which rectangles are genuinely invalid, and which can be displayed using a quick block-copy operation. The block-copies themselves are re-ordered so that wherever possible, one copy does not invalidate screen area needed as source for another copy operation; in the rare cases where this is not possible, the least damaging alternative is chosen - that is,

the one requiring the smallest invalid area to be redrawn.

The upshot of this is that wherever touching or overlapping windows move together - external and internal panes, and of course, nested windows - the shuffling effect present in earlier Window Managers, where an area is alternately covered by the top window, and exposed and redrawn, is eliminated. There is also some speed gain from combining block-copy operations. In the unusual event that the window opening queue needs to be flushed before the Wimp_Poll, an extension to is provided.

Standard Window Furniture

All window furniture buttons slab in, including close and toggle size.

The bug regarding slabbing-in of the other furniture buttons, which could be unreliable following a Service_InvalidateCache, is fixed.

Single-pixel borders can be removed from windows without removing all the other window furniture at the same time.

For large work area windows, when the scroll sliders start being dragged, they jump less than in previous Wimps, and are displayed more accurately when they reach the end of the document.

A plain-colour background is no longer drawn underneath the solid toolsprites, thereby reducing flicker.

Minimum Sizes

All windows are optionally shrinkable to zero size visible area (subject to the continued visibility of any back, close, toggle size and adjust size buttons). In order to achieve this, scroll bars no longer have a minimum length (except in special cases). After the scroll slider region has been shrunk to zero length, the scroll arrows start to be plotted scaled down until the entire bar shrinks to nothing.

Most conventional programs which rely upon the old behaviour continue to function as before, due to the special-case exemptions in .

Shift-Toggle-Sized Windows

Windows which have been toggle-sized with Shift held down (i.e. made to fill all the screen except the icon bar) are now internally marked as being full size, and the toggle-size button indicates this by switching to the "fulled" sprite. A further click on the toggle size button will then reduce the window's size to its original size, as opposed to the previous, unhelpful behaviour, which was to enlarge

the window to full screen, including covering the icon bar!

Error Report Dialogue Boxes

Since each error button in a Wimp error box can contain user-defined text, it is possible for the text to be wider than the fixed width action buttons used in previous Wimps' error boxes. The Wimp now enlarges each action button if the width of its text (plus 36 OS units to allow for borders) is greater than that of its standard action button.

Icons

A number of long-standing bugs relating to "3-D" icons are fixed: clicking on an action button that uses an antialiased font doesn't reset the Wimp font to the old bitmap system font; multiple selection of action buttons via dragging off one button and Adjust-clicking on another is no longer possible; menu clicks on action buttons no longer cause a flicker; and the 3-D plinths are drawn correctly in EX0 and/or EY0 screen modes and/or when not pixel-aligned. Icon foregrounds and backgrounds can be drawn using any 24-bit specified colour, not just one one of the Wimp colours.

Any 2-, 4-, 16- or 256-colour sprites with palettes within icons are now plotted using the palette directly, rather than via a translation table. The effect of this is better colour reproduction of such sprites in 32 thousand colour and 16 million colour modes.

Line spacing for multi-line text icons, first specified for RISC OS 3.10, via the parameters to the "L" command of an icon's validation string, has finally been implemented.

Menus

Submenus and dialogue boxes opened from "reversed menus" are opened at the correct horizontal position again. However, the automatically-opened position still does not perfectly mirror normal menus for cases where the pointer is held over the "tick" space opposite the arrow.

Icon Bar

The icon bar scrolls at a rate independent of processor speed or loading - the position is determined according to the time elapsed since scrolling started, irrespective of how many screen updates have been possible since.

Also, the speed of scrolling increases linearly over time; it accelerates. This eases navigation of very wide icon bars.

If the pointer is left over the bottom pixel row of the screen for 0.5

seconds, the icon bar now pops to the top of the window stack, much as it would if you had used the Shift-F12 hotkey. The icon bar remains at the top of the stack (and therefore accessible) while the pointer stays over the icon bar and/or there is an icon bar menu open. When this condition is no longer true, the icon bar returns to its original position in the window stack (note that this differs from Shift-F12 behaviour, where the icon bar always becomes a "back" object again).

Panic Redraws

When a panic redraw occurs due to there being too many invalid rectangles for the Wimp to handle, the first thing drawn is a plain background. In a feature dating back to Arthur, this was hard-coded to Wimp colour 15 (now light blue) - hardly appropriate. This is changed to a mid-grey colour.

Also, anticipating the increased number of invalid rectangles made likely by the nested windows system, the number of invalid rectangles allowed before a panic redraw is triggered is raised from 128 to 256.

Programmer's interface

The following SWIs detail the changes from the previous version of the Window Manager. They are not full definitions for each SWI call.

Wimp_Initialise
(SWI &400C0)

On entry

-

On exit

-

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

SWI is not re-entrant

Use

Wimp_Initialise recognises a new version number, in addition to the
established 200, 300 and 310 versions. 380 will, at a sufficiently
advanced version of the Wimp, be necessary in order to activate most
of the features described in this specification. As of Wimp version
3.97, **only** the window foreground colour byte of the window block is
affected by the version passed to Wimp_Initialise, but this must not
be relied upon for future releases of the Wimp; other Nested-Wimp
features may in future become reliant upon having passed 380.

Related APIs

None

On entry

$R1 = $ Window block

Offset Contents

+28   window flags:

Bit(s) Meaning

22 This bit is overwritten by the Wimp, and may be read us
Wimp_GetWindowState. When set, it indicates that the v
is, or *will be*, toggled to full size without covering the ic
Note that this behaviour is different to bit 18, which is s
the window is, or *has been*, toggled to full size including
icon bar.

Toggling behaviour can only be properly resolved after
Wimp_Poll returns an Open_Window_Request reason co
before the subsequent call to Wimp_OpenWindow. Flags
is set, by definition; applications may distinguish betwee
different types of toggle-size clicks using the following t
table:

Operation	Bit 22	Bit 21	Bit 1
To full size	0	1	0
From full size	0	0	1
To semi-full size	1	1	0
From semi-full size	0	0	0

These are the only values that can be returned in combi
with bit 19 being set.

23 If this is a child window, make it a furniture window. (Th
no meaning for top-level windows, so the bit should alwa
cleared in such cases to allow for future expansion).

+32   Title foreground and window frame colour:

Value Meaning

&FF Window has no 1-pixel border components, but furniture
still be present (as controlled by the usual flag bits). Titl
foreground colour defaults to Style Guide standard colou
(Wimp colour 7).

+68   Minium width of window (16 bits).

This used to be the minimum visible width in OS units, unless a g
width was required by either of the following:

●   The top edge furniture - any combination of the back, c
title and (as a special case when the vertical scrollbar i
absent) the toggle-size icons. The width required by the
icon was defined as 8 OS units, except when 0 was use
indicating that the full width of the title text or sprite w

     apply;
●   The bottom edge furniture - the minimum (unsquashed)
horizontal scrollbar (if present), plus any adjust-size bu
the special case when the vertical scrollbar is absent).

The 0 special case retains exactly the same behaviour as before
(horizontal scroll bars, if present, cannot be squashed below a ce
minimum width). Any other values activate the new behaviour: a
horizontal scrollbar can be squashed down to zero width, and the
bar can be squashed down to zero width as long as the back and
buttons are both absent (otherwise, 8 OS units remains the mini
title icon width). A new special case, 1, is introduced, activating
new behaviour, but with an actual minimum window width of 0 r
than 1 (although it is obviously still subject to any non-squashab
furniture width requirements as discussed).

+70   Minium height of window (16 bits).

This used to be the minimum visible height in OS units, unless a
height was required by a minimum (unsquashed) size vertical scr
plus any toggle-size and adjust-size buttons (in the special cases
there is no title or horizontal scrollbar, respectively). It was also
to a restraint that the minimum height could not be less than 2 p
high when the vertical scrollbar was absent.

The value 0 becomes a special case, and retains exactly the same
behaviour as before. That is, the vertical scroll bar, if present, ca
be squashed down below a certain size. All other values activate
new behaviour: any vertical scrollbar may be squashed down to z
height, and the 2-pixel hard-minimum no longer applies. A specia
1, is introduced, activating the new behaviour but with a minimu
window height of zero rather than of 1 (subject to constraints im
by window furniture as described).

On exit

unchanged

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

SWI is not re-entrant

Use

The meanings of certain parts of the window block are altered and
extended as shown.

Related APIs

None

Wimp_CreateIcon
(SWI &400C2)

On entry

unchanged

On exit

unchanged

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

SWI is not re-entrant

Use

The (C)olour validation string command is introduced to allow icon colours to be set from a 24-bit palette. It is typically followed by two hexadecimal numbers (BBGGRR) separated by a /, but either one may be omitted, and the relevant colour from the icon flags (or the F command) will then be used instead. It is suggested that the old-style colours should be specified to something sensible, in case the program gets run on a Window Manager that doesn't support the command.

Validation	Colours produced		
C008000/0080FF	Mid green foreground, orange background	🟩	🟧
CFF00FF	Magenta foreground, background as specified in flags word	🟪	F
C/00FFFF	Foreground as specified in the flags word, yellow background	F	🟨

Note also that the line spacing specified after the (L)ine spacing command is now acted upon.

Bit 20 of the icon flags has not been part of the Exclusive Selection

Group (ESG) number since at least RISC OS 3.10, and should be considered 'reserved'.

Related APIs

None

Wimp_OpenWindow
(SWI &400C5)

On entry

R1 = Pointer to block, or NULL (0 or -1) to flush all pending opens to the screen.

R2 = "TASK" (&4B534154)

This is a 'magic word' to tell Wimp that the extended version of this SWI call is being made.

In the extended call, R3 and R4 are as described below. Otherwise, the previous parent and flags (if any) are reused. The parent defaults to -1 and the flags default to 0, i.e. traditional Wimp behaviour is still the default.

It is important to ensure that R2 does *not* **accidentally get left with this value from a previous call in code which mixes old and new style calls.** This is mostly an issue for C SWI veneers.

R3 = Handle of window to make parent (or -1 to make a top-level window)

R4 = flags

Bit(s)	Meaning
0	Use extended OpenWindow block in R1 (R1 + 32 = window flags).
16-17	left edge of child
18-19	bottom edge of child
20-21	right edge of child
22-23	top edge of child
24-25	x-scroll of child
26-27	y-scroll of child

These flag pairs have the following meanings (as high bit, low bit):

Setting	Action
00	linked to work area of parent
01	linked to left / bottom of visible area of parent
10	linked to right / top of visible area of parent
11	reserved

1-31	Reserved, must be zero

On exit

unchanged

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

SWI is not re-entrant

Use

This call is the key to the nested window system. Changes are as shown.

If R3 is -1, bits 16-27 must all be clear. Otherwise, they specify how certain aspects of the child are aligned with the parent window.

If R4 bit 0 is set, then R1 + 32 holds the new window flags to use. This can be used, for example, to add or remove window furniture without having to delete and re-create the window.

Not all window flags can be altered in this way. In particular, bits 16-22 can only be set or cleared by the Wimp, in order to reflect the window status. The Wimp will also modify the bits relating to the window furniture as follows: if bit 31 is unset (indicating the old-style bits are to be used) then bits 24-30 are updated to reflect the status indicated by bits 0, 2, 3 and 7 (but note that bit 31 itself is left unchanged). If bit 31 is set, however, bits 0, 2, 3 and 7 are cleared. All other bits are preserved (and acted upon) by the Wimp.

Under previous Wimps, the window handle at R1+0 had to be owned by the task calling Wimp_OpenWindow. Because a child window need not belong to the same task as its parent, this restriction has now been lifted; this is the case even for the non-extended form (R2 not equal to the magic word "TASK").

Since RISC OS 2 (and possibly even earlier), Wimp_OpenWindow has had undocumented return conditions: values at R1+4 - R1+24 are updated to represent the actual parameters of the opened window after valid ranges have been taken into account, and the window has

been forced on-screen (if applicable). Rather than continue to have programs waste time following a Wimp_OpenWindow with a Wimp_GetWindowState (except in cases where the new window flags are required), the exit conditions can now be considered official.

Related APIs

None

Wimp_GetWindowState
(SWI &400CB)

On entry

changes detailed below

On exit

unchanged

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

SWI is not re-entrant

Use

This call mirrors Wimp_OpenWindow. If R2 = "TASK" on entry, then on exit R3 and R4 are as [described above (on page 269)](). Note however that Wimp_GetWindowState has always returned the window flags in R1+32, but despite this, R4 bit 0 will always be returned cleared.

Related APIs

None

Wimp_GetWindowInfo
(SWI &400CC)

On entry

unchanged

On exit

changes detailed below

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

SWI is not re-entrant

Use

The returned window block's extended meanings are as for SWI
Wimp_CreateWindow (on page 262).

Related APIs

None

Wimp_ForceRedraw (SWI &400D1)

On entry

R0 = Window handle (as before)
R1 = "TASK" (&4B534154)
    This signals that the extended version of
    Wimp_ForceRedraw is being used, and R2-R4 are as stated
    below.

R2 =

Value	Meaning
+3	Redraw title bar
	Other values are reserved

R3 - R4 = Ignored

On exit

unchanged

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

SWI is not re-entrant

Use

Wimp_ForceRedraw is changed so that it can be applied to windows owned by other tasks, because a child window may belong to another task.

In the past, redrawing the title bar of a window has been accomplished either by working out where the window's title bar is on the screen and calling Wimp_ForceRedraw with R0=-1 to invalidate that area, or alternatively by toggling the input focus in and out of the window to force its borders to be redrawn.

Neither of these methods is particularly satisfactory: the first could

cause other windows on top of the one in question to be redrawn unnecessarily, and the second redraws the rest of the borders as well, and in the case of child windows, would also cause a redraw of the parent's title bar.

So Wimp_ForceRedraw is extended as shown above.

Note: Since the value &4B534154 ("TASK") is far too big to be an minimum x coordinate, it is safe to use as described above.

Related APIs

None

Wimp_GetWindowOutline
(SWI &400E0)

On entry

unchanged

On exit

unchanged

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

SWI is not re-entrant

Use

Previously, a window had to be open and visible on screen for this call to work. It will now work on windows which are closed or not yet visible.

Related APIs

None

Wimp_RegisterFilter
(SWI &400F5)

On entry

R1 = Reason code:

Value	Meaning
4	Register / deregister post-rectangle filter
5	Register / deregister post-icon filter

On exit

unchanged

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

SWI is not re-entrant

Use

Each type of filter may only be registered once using this call, and so it remains for the use of the Filter Manager only (which will normally be responsible for delegating filter calls for specific tasks) unless you want to replace the whole filter system. The SWI is thus, in effect, only really for internal use but since it is documented in the PRMs the extended version is included here for completion.

As far as the Filter Manager is concerned, note that certain filter types require a newer Filter Manager than present in the RISC OS 3.60 / RISC OS 3.71 ROMs.

There are two new reason codes that may be passed to Wimp_RegisterFilter (4 & 5)

Notes:
There is an undocumented entry condition for any registered pre-

filter: R3 points to the poll word if R0 bit 22 was set on entry to Wimp_Poll. On exit, R1 and R3-R10 must be preserved. The PRMs have also forgotten to mention that on entry to the post-filter, R12 holds the value given in R2 when the routine was registered. On exit, R1 and R3-R10 must be preserved. Future documentation will include this information.

The Wimp now calls the post-filter, with a null reason code, whenever Wimp_StartTask returns, even if the child task didn't call Wimp_Poll. In either case, any attempt to claim the null event will now be ignored.

The entry and exit conditions for reason code 2 and 3 filters have not previously been documented, and those for reason codes 4 and 5 are new in Wimp version 3.86, so they are in numerical order by reason code in the section entitled .

Related APIs

None

Wimp_Extend
(SWI &400FB)

On entry

R0 = Reason code (see exit conditions for R1)
R1 = Window handle, or for reason codes in R0 of 7 and 8, a value of -1 to enquire about the top-level stack.

On exit

R1 = The value of R1 depends on the reason code and R1 value supplied on entry:

Value	Meaning
0 - 5	*Internal use only*
6	Parent window
7	Frontmost child window
8	Backmost child window
9	Sibling immediately behind
10	Sibling immediately in front

Interrupts

Interrupts are unchanged
Fast interrupts are unchanged

Processor mode

Processor is in undefined mode

Re-entrancy

Not defined

Use

It is possible to enumerate window stacks using only Wimp_GetWindowState, but it requires that you open a "special" window of your own at the back of each stack to be enumerated, and you can only enumerate the stack from back to front. It may also return rather more information that you actually need, and so may be a little bit slower than it might be.

Consequently, five new index values are added to Wimp_Extend. For each of the following, R1 holds the window handle being queried, or

a value of -1 to enquire about the top-level stack (for index values 7 and 8 only).

Any of the above calls can return R1 = -1 for "no window", indicating that the end of the stack was reached, or the window had no parent or child, or R1 was -1 on entry and R0 was not 7 or 8.

Note also that pane windows are not skipped by any of the above calls.

Related APIs

None

Filter Entry Points

Rectangle Copy Filter

On entry

R2 =	Destination bounding box: min x
R3 =	Destination bounding box: min y
R4 =	Destination bounding box: max x
R5 =	Destination bounding box: max y
R6 =	Source bounding box: min x
R7 =	Source bounding box: min y
R8 =	Source bounding box: max x
R9 =	Source bounding box: max y
R10 =	Window handle, minus one (only in Nested Wimp variants from v3.90 onwards)
R12 =	Value of R2 when registered

On exit

R0 - R1 preserved
R3 - R10 preserved

Interrupts

Interrupts are enabled
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

Entry point is not re-entrant

Use

The rectangle copy filter is called when the Wimp is about to copy a rectangle across the screen, not exclusively due to Wimp_BlockCopy. The current and previous graphics cursor positions are describing the area to be copied, ready for the VDU block copy operation, but the actual operation has not yet been performed.

All bounding boxes (R6-R9 values on entry) are in screen coordinates.

Related APIs

None

Get Rectangle Filter

On entry

R2 = Task handle
R6 = Rectangle to be drawn: min x
R7 = Rectangle to be drawn: min y
R8 = Rectangle to be drawn: max x
R9 = Rectangle to be drawn: max y
R12 = Value of R2 when registered

On exit

R0 - R1 preserved
R3 - R10 preserved

Interrupts

Interrupts are enabled
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

Entry point is not re-entrant

Use

The get rectangle filter is called just before the redrawing of a rectangle begins, before the window background has been filled (if appropriate), and even before the VDU graphics window has been set up. This filter is no longer called when it is only the caret which is being redrawn; the new rectangle filters below never have been.

Note that on entry, R10 is undefined (this may change to match the other rectangle filters, but don't rely on it).

All bounding boxes (R6-R9 values on entry) are in screen coordinates.

Related APIs

None

Post-Rectangle Filter

On entry

R2 = Task handle
R6 = Rectangle to be drawn: min x
R7 = Rectangle to be drawn: min y
R8 = Rectangle to be drawn: max x
R9 = Rectangle to be drawn: max y
R10 = Window handle, minus one
R12 = Value of R2 when registered

On exit

R0 - R1 preserved
R3 - R10 preserved

Interrupts

Interrupts are enabled
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

Entry point is not re-entrant

Use

The post-rectangle filter is called after the window background is filled as part of a rectangle redraw, i.e. shortly before Wimp_GetRectangle or Wimp_RedrawWindow (or their internal equivalents) reset the VDU state and return, unless the call is returning with "no more to do" status. The filter is linked to the filling-in of the window background; redraw loops initiated by Wimp_UpdateWindow never cause this filter to be called, because they do not cause the window background to be redrawn. However, the filter is called after a "transparent" window background would have been filled.

All bounding boxes (R6-R9 values on entry) are in screen coordinates.

Related APIs

None

Post-Icon Filter

On entry

R2 = Task handle
R6 = Rectangle to be drawn: min x
R7 = Rectangle to be drawn: min y
R8 = Rectangle to be drawn: max x
R9 = Rectangle to be drawn: max y
R10 = Window handle, minus one
R12 = Value of R2 when registered

On exit

R0 - R1 preserved
R3 - R10 preserved

Interrupts

Interrupts are enabled
Fast interrupts are enabled

Processor mode

Processor is in svc mode

Re-entrancy

Use

In the past, the rectangle redraw cycle has consisted of the Wimp filling the background and returning control to the application, which then draws whatever it wants and calls Wimp_GetRectangle. The Wimp subsequently draws the icons before moving on to the next rectangle and filling its background, and so on. This did mean that applications never got a chance to draw on top of the icons; the post-icon filter now allows them to.

All bounding boxes (R6-R9 values on entry) are in screen coordinates.

Related APIs

None

References

The Filter Manager

PRM Volume 3, section 56, pp. 303-312 and erratum, Volume 5a, page 668.

The Window Manager: Wimp_RegisterFilter

PRM Volume 3, section 53, pp. 224-225

Acorn Filter Manager v0.18: Functional Specification

Document reference 1215,102/FS.

Document information

History:	Revision	Date	Author	Changes
	1116,011/ FS_1	06 Feb 1998		**Original Version (not released)**
	1116,011/ FS_2	10 Feb 1998		● HTML version completed for publishing on the Web
	1116,011/ FS_3	16 Feb 1998		● Fixed up the HTML a bit ● in Wimp_CreateWindow the title foreground colour defaults to 7, not to 2
	-	23 Feb 1998		● Various HTML tweaks; no content change
	1215,401/ FS_1	02 Mar 1998		**General Release** ● Document number now 1215,401/FS ● Updated history, and navigation links in the page footer now include the specifications section. ● no other content changes
	1215,401/ FS_2	08 Apr 1998		● Added Wimp_RegisterFilter details. ● Some missing spaces added. ● Alphabetic components of the hex SWI numbers in body text capitalised. ● Added References section. ● Used <acronym> tag for acronyms.
	1215,401/ FS_3	21 Sep 2021	Alan Robertson	**Initial version in PRMinXML format** ● No major changes to text. Removed the 'Document Status' section as

- information captured in 'Document information' section
- Prefixed the Acorn Functional Specification Document Number to each Issue revision (where possible) in original
- Removed links to external files
- The Filter Entry Points are now defined within their own section, rather than as part of Wimp_RegisterFilter

Disclaimer: Originally appearing as 1116,011/FS up to issue 3, this specification now has a document number of 1215,401/FS for General Release.

Various authors for original document, including:

- Piers Wombwell
- Kevin Bracey

Later revisions for first formal specification:

- Ben Avison
- Andrew Hodgkinson

CryptRandom

Introduction

CryptRandom is a module for generating cryptographically useful random bytes under RISC OS. It can use a number of sources to provide this information to clients needing secure, or high quality random data.

Overview

Computers are, by their nature, deterministic - so applying the same sequence of inputs to any program is likely to produce the same result. This is a **bad thing** when it comes to cryptography, as if you use a known sequence to encrypt a data stream, next time you turn on your machine you'll use the same known sequence, making the code possible to break. Thus we need a random sequence so that no pattern can be spotted in it. Basic provides a pseudo-random sequence, but this is the same every time the machine is turned on, so is not very good. It is also just a sequence, which will eventually repeat. True randomness is only possible on a computer by attaching it to other devices such as a radioactive source - not very practical.

CryptRandom applies another method, which will produce different values showing to no known pattern, which are different each time you switch the machine on. This is much less secure than using a true random source, but better than using a predictable random number generator like that Basic uses.

The CryptRandom module provides SWI calls which allow access to random data retrieved from a variety of sources.

Installation

CryptRandom is supplied in an archive containing a !System directory. It can be installed by decompressing this archive, then using a !System merge tool - such as that accessible by running !Boot, or !SysMerge for RISC OS 3.1 machines.

CryptRandom provides a service to clients that require it. Such applications should load it in the following way:

```
RMEnsure CryptRandom a.bc RMLoad System:Modules.CryptRand
RMEnsure CryptRandom a.bc Error CryptRandom version a.bc is require
```

where a.bc is the oldest version supporting the features the application requires (see the history file). Note that this version should be at least that of the latest security advisory (if any).

Lineage

CryptRandom is based on code from PuTTY, the Windows SSH client by Simon Tatham (see http://www.chiark.greenend.org.uk/~sgtatham/putty/). It consists of a 'pool' of random data, which is 'stirred' every time a byte is requested, using a complex hashing function to ensure there is no discernible pattern. The pool is supplied by 'entropy' from various sources, designed so that they are different every time they are called. The numerous sources include:

Technical details

- On initialisation of CryptRandom:

  - Unique machine ID
  - Current WIMP tasks
  - Current dynamic areas
  - Disc free space
  - Disc directory listings
  - Previous saved CryptRandom seed

- Every time a byte is requested:

  - Real time clock
  - System interval timer
  - Battery manager data (voltages/temperatures etc)

- Every mouse and key press:

  - Press data
  - Mouse position
  - System timer

Sources are ignored if they don't work (eg a Risc PC doesn't have a battery manager).

Interrupts are disabled on SWIs as mentioned above - this is to allow multiple users to access the pool from interrupt routines (events/ callbacks etc) - this may be subject to change in future versions.

The seed is saved over sessions to preserve the entropy - it'll first look for `CryptRandom$SeedFile`, and if this is set use this as the seed location, otherwise try `Choices:Crypto.CryptRand.Seed` or if `Choices$Path` is unset use `<Wimp$ScrapDir>.Seed`.

I don't claim to prove the security of the hashing process, so I can't guarantee the randomness of the output, but it appears to be white noise - if in doubt, do your own tests. The hash is based on SHA-1, which is believed by the computing community to be secure. Any comments in this respect would be welcomed.

contact

Newer versions (if any) of this software may be found at http://www.markettos.org.uk/ or else by contacting the author at: email theo@markettos.org.uk

Theo Markettos
5 Willow Close
Liphook

Hants
GU30 7HX
UK

I'd also welcome any bug reports or fixes, or any other comments.

Sources

Sources can be obtained from http://www.markettos.org.uk/

To build them you'll need:

- Acorn C v4 or v5 (the Makefiles are designed for Castle's 32bit C compiler, so may need modification otherwise)
- SDLS if have Acorn C v4 http://www.excessus.demon.co.uk/acorn/ssr/
- Syslog (optional) http://www.drobe.co.uk/archives/freenet.barnet.ac.uk/Acorn/freenet/j.ribbens/syslog-0.17.spk (note that Syslog 0.19 appears to have bugs in it which may cause problems)
- Makatic (optional) http://www.mirror.ac.uk/collections/hensa-micros/local/riscos/projects/makatic.zip
- OSLib http://ro-oslib.sourceforge.net/

SWIs

CryptRandom_Byte
(SWI &51980)

Reads a byte from the random pool

On entry

None

On exit

R0 = Random byte value (0-255)

Interrupts

Interrupts are disabled
Fast interrupts are undefined

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI reads a byte from the pool, and subsequently stirs it.

Related SWIs

SWI CryptRandom_Block (on page 297)
SWI CryptRandom_Word (on page 298)

CryptRandom_Stir
(SWI &51981)

Stirs the random pool

On entry

None

On exit

None

Interrupts

Interrupts are disabled
Fast interrupts are undefined

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI stirs the random pool - this should not be necessary in normal use

Related SWIs

[SWI CryptRandom_AddNoise (on page 296)](#)

CryptRandom_AddNoise
(SWI &51982)

Introduce data to the random pool

On entry

R0 = Pointer to block of noise data to add
R1 = Size of data in the block

On exit

None

Interrupts

Interrupts are disabled
Fast interrupts are undefined

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

Adds a block of noise to the random pool - shouldn't be necessary in normal use.

Related SWIs

CryptRandom_Block
(SWI &51983)

Reads multiple bytes from the random pool

On entry

R0 = Pointer to block to fill with random bytes
R1 = Number of bytes to fill into the buffer

On exit

None

Interrupts

Interrupts are disabled
Fast interrupts are undefined

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

Generates a block of random data. Note this is called with interrupts off, so large blocks may cause your machine to hang while they are generated. Note also the entropy generated by this call is likely to be less than multiple SWI CryptRandom_Byte (on page 294) calls (since times/battery status etc are likely to be the same during this call, but not if _Byte calls are spread at different points in your program), so randomness may suffer as a result.

Related SWIs

SWI CryptRandom_Byte (on page 294)
SWI CryptRandom_Word (on page 298)

CryptRandom_Word
(SWI &51984)

Reads a 32-bit word from the random pool

On entry

None

On exit

R0 = Random 32-bit word from the pool

Interrupts

Interrupts are disabled
Fast interrupts are undefined

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This reads a 4 bytes from the pool, and assembles them into a 32-bit word.

Related SWIs

SWI CryptRandom_Byte (on page 294)
SWI CryptRandom_Block (on page 297)

Document information

<table>
<tr><td>Maintainer(s):</td><td colspan="3">Gerph <<u>gerph@gerph.org</u>>
Theo Markettos <<u>theo@markettos.org.</u>></td></tr>
<tr><td>History:</td><td>Revision Date</td><td>Author</td><td>Changes</td></tr>
<tr><td></td><td>0.13</td><td>Theo Markettos</td><td>Text documentation
● Original documentation for the CryptRandom module.</td></tr>
<tr><td></td><td>0.13a</td><td>Gerph</td><td>PRM-in-XML documentation
● Documentation re-written as PRM-in-XML.</td></tr>
</table>

Filing system drive information

Introduction

Filing systems may be based on devices which are able to be changed dynamically in use ('hot pluggable'). These devices may issue an UpCall to indicate that a device is now available, or has become unavailable.

Technical details

Two UpCalls are used to indicate that a filing system path residing on a device is now available, or has been made unavailable:

● UpCall_DriveAdded (on page 302)
● UpCall_DriveRemoved (on page 304)

Not all devices and filing systems issue these UpCalls, so clients should treat them as advisory.

UpCall_DriveAdded
(UpCall &18)

A filing system may be available on a given path.

On entry

R0 = 24 (&18)
R1 = Pointer to a zero-terminated filing system path prefix for a new
    device

On exit

R0 - R1 preserved

Use

This UpCall is issued by a device when filing system path has been
made available. This may happen due to a new disc being inserted, a
device being formatted, or a remote system becoming available.

At the time that the UpCall is issued the filing system path should be
accessible through normal filing system operations. The nature of hot
pluggable systems mean that by the time this call is received, the
device may have already become unavailable, or the filing system on
the device may not be present.

The filing system path takes the form of a filing system name, a disc
name specification and an optional path specification. For some
devices, the disc name may be a number, indicating that no name has
been determined yet, or that there is no name available. For others, a
name may be given. It is recommended that the path be
canonicalised to obtain the correct name of the device.

Example prefix names:

- `ADFS::4`
- `SDFS::0`
- `Share::Storage`
- `HostFS::Host.$.Mountpoint`

This UpCall must not be claimed.

Related upcalls

UpCall_DriveRemoved
(UpCall &19)

A filing system is no longer available on a given path.

On entry

R0 = 25 (&19)
R1 = Pointer to a zero-terminated filing system path prefix for a new
device

On exit

R0 - R1 preserved

Use

This UpCall is issued by a device when filing system path is no longer
available. This may happen due to a disc being removed, a device
being formatted, or a remote system becoming unavailable.

At the time that the UpCall is issued the filing system path will not be
accessible and no further information is available. As such, clients
should attempt to track the paths to which the drive may refer. In
particular, devices may refer to drive numbers, without any name
being canonicalised, and clients may therefore need to track which
drive numbers refer to which canonicalised disc names.

This UpCall must not be claimed.

Related upcalls

Document information

Maintainer(s):	Gerph <gerph@gerph.org>			
History:	**Revision**	**Date**	**Author**	**Changes**
	1	23 Dev 2021	Gerph	**Initial version**

- Created from examples of sources using it.

Related: https://gitlab.riscosopen.org/RiscOS/Sources/FileSys/SDFS/SDFS/-/blob/master/c/service
https://gitlab.riscosopen.org/RiscOS/Sources/FileSys/ADFS/ADFSFiler/-/blob/master/s/ADFSFiler#L1055

Pointer devices (supplement for Pyromaniac)

Introduction and overview

Pointer devices (usually mice) have been extended to provide additional functionality found in more modern devices such as additional buttons and an alternate positioning device (usually provided as a single or dual 'scroll wheel'). Similarly, devices which provide absolute positioning such as tablets and touchscreens are now capable of being serviced by the pointer system. In order to provide these extra functions, a revised form of the PointerV interface has been used.

This builds upon the interface declared in PRM 5a, but moves some functions away from the driver.

The OSPointer module controls pointer movement and will handle these extended functions. Previously the Kernel provided management of the pointer. These functions are now provided entirely by the OSPointer module.

The operation is split into two major parts:

- How drivers provide information to the OSPointer module about the new features
- How programmers access this information

In addition, a separate document details the operation of the WindowScroll module which provides functionality for desktop tasks.

Technical details

PointerV

PointerV has been extended with a new reason codes - [PointerV 4 (on page 315)](#) - in RISC OS Select to support the use of alternative pointing device values. Specifically this allows for the scroll wheel provided by modern mice, and for tablet or touch screen devices.

Driver updates in RISC OS Select

Quadrature mouse driver

The quadrature mouse driver ('Mouse' module) has been updated to provide an additional device type for Stuart Tyrrell's PS2 mouse interface. This interface functions in 'driver' mode to provide alternate device support for single axis devices (primarily vertical scroll wheels).

Dual axis movement is presently not supported.

PS 2 mouse driver

The PS 2 mouse driver ('PS2Driver' module) has been updated to provide support for 'Intellimouse' and 'Intellimouse Pro' devices. These are more commonly known as 'scroll mice' or '5 button mice' respectively.

Dual axis movement is presently not supported.

Touch screen or tablet drivers

No touch screen or tablet driver is supplied with the current version of RISC OS. Developers wishing to implement such a driver should contact their supplier.

OSPointer handling of extended requests

The OSPointer module will issue the [Extended request (on page 315)](#) for versions of the OS which support these new features (RISC OS 4.32 and later). If the call returns unclaimed (R0 having not been set to -1 or 5), the module will issue PointerV 1 (Request) and defer button handling to the driver.

If the call is claimed, the OSPointer module will issue KeyV events for the buttons which have been pressed and apply the change or absolute position to the pointer.

The absolute positioning interface is available from version 0.25 of

the OSPointer module.

Additional buttons

In addition to the base 3 buttons up to 8 buttons are supported by the OSPointer module. 5 button mice are common and the PS 2 driver has been updated to support such devices.

The additional buttons are reported through the extended KeyV interface for mouse buttons. These buttons are detected by the OSPointer module and returned as useful values through the standard interfaces.

Programmers interface

In order obtain position details for the alternate scrolling device, a new reason has been added, SWI OS_Pointer 2 (on page 311).

SWI calls

Read current mouse state

On entry

None

On exit

R0 = X position of the pointer
R1 = Y position of the pointer
R2 = mouse buttons:

Bit(s)	Meaning
0	right button
1	middle button
2	left button
3	fourth button
4	fifth button
5	sixth button
6	seventh button
7	eigth button
8-31	Reserved, must be zero

R3 = time of button chan

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI returns the pointer position from the mouse buffer if events are buffered or the current position if the buffer is empty. It has been extended from the interface described in PRM 1-699 by adding additional buttons.

Related SWIs

SWI Pointer 2 (on page 0)

Related vectors

PointerV 4 (on page 315)

Read alternate position

On entry

R0 = 2 (reason code)
R1 = Register details

On exit

R0 = signed 32bit X position of the alternate device
R1 = signed 32bit Y position of the alternate device

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI returns the position of the alternate positioning device. The device position is unbounded and thus may wrap when the limits of the 32bit representation are reached. Should the device position wrap past a limit, it will be reset to zero. Thus, should the position exceed either &7FFFFFFF or -&80000000 it will be reset. Clients should be aware of this and handle such conditions appropriately.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

RISC OS Pyromaniac   RISC OS ≥ 7.48
Supported

Related SWIs

Related vectors

Software vectors

Scroll event has been triggered by the user

On entry

R0 = reason code (21)
R1 = subreason code (4)
R2 = signed 32bit change in X position
R3 = signed 32bit change in Y position

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This event is generated by the OSPointer module when a scroll event
is triggered by the user. Clients which track mouse movements
should monitor this event. This allows clients to monitor either
changes or the absolute position should they wish to do so. If clients
wish to cause the scroll event to be ignored they should claim the
event.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

RISC OS Pyromaniac   RISC OS ≥ 7.48
Supported

Related SWIs

Related vectors

Request information about the current pointing device position

On entry

R0 = reason code (4)
R1 = pointer type

On exit

R0 = Request state:

Value	Meaning
-1	Extended request claimed for this pointer type, for relative positioning device
4	Extended request not understood
5	Extended request claimed for this pointer type, for absolute positioning device

R1 preserved

R2 = relative device: signed 32 bit change in X position

absolute device: fractional 16 bit X position

R3 = relative device: signed 32 bit change in Y position

absolute device: fractional 16 bit Y position

R4 = relative device: signed 32 bit change in X position of alternate device

absolute device: must be 0

R5 = relative device: signed 32 bit change in Y position of alternate device

absolute device: must be 0

R6 = Mouse buttons:

Bit(s)	Meaning
0	Right button
1	Middle button
2	Left button
3-7	May be provided at the discretion of driver
8-31	Must be 0

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector reason is called by the OSPointer module to determine the position of the pointing device in a similar manner to that of PointerV 1 (Request). Drivers should check the pointer type and if it matches the device being provided details should be returned and the vector claimed. If the pointer type does not match the vector should be passed on.

Unlike PointerV 1 (Request), drivers should not issue KeyV requests for the mouse buttons that they provide. This task will be performed by the OSPointer module based on the button state returned. Drivers wishing to support both the old and new protocol may share code between PointerV 1 (Request) and PointerV 2 (Result) but they must ensure that registers are not corrupted unduly and that the different mouse button processing is performed based on the request type.

Relative devices and absolute devices respond to the same request but provide slightly different responses. The value returned in R0 is used to determine the type of response made.

Absolute devices return a 16 bit value (0-65535) which determines the position of the event. The driver may determine how the event is to be processed and indicate an equivalent button state for the event. This allows devices to provide positioning within the absolute device as distinct from button click events. For absolute devices the meaning of R4 and R5 is undefined and the registers must be returned as 0 for future compatibility. Internally, the absolute position request is scaled by the screen size and converted into a relative position which is applied to the mouse position with an equivalent of a mouse step of 1.

For scroll wheel-like alternate devices the +ve Y direction should be that for pushing the wheel 'away' from the user.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related SWIs

[SWI Mouse (on page 0)](#)
[SWI Pointer 2 (on page 0)](#)

Related vectors

Document information

Icon bar file drags

Introduction

For RISC OS 4 the behaviour of the iconbar device icons was updated to allow files dropped on them to be saed to a specified directory for the device. Usually this would be the root of the device, but is configurable. This avoids the user opening a save box but being unable to drop the file anywhere.

Technical details

File drags from save boxes to icon bar Filer icons will cause the file to be saved in a specified directory of the device, most sensibly the root. The Filer will then open the directory viewer. In combination with the autofronting icon bar in the new window manager, the user will now never face the situation of having a save box open, but nowhere to drag the file to.

For consistency, drags from Filer windows to icon bar icons will cause files to be copied/moved to the directory. As with saves, the Filer will open the directory viewer.

The directory which a file is saved/copied to will be specified by a system variable and will default to the root directory. The system variable will be of the form *<FSName>*`Filer$DefaultPath`. For example `ADFSFiler$DefaultPath` or `NetFiler$DefaultPath`.

Icon bar save protocol

In order for files to be saved to icon bar device icons, the FS Filers will now be required to receive the WIMP message Message_DataSave. They will reply with [Message_DataSaveAck (on page 0)](#) specifying a pathname for the saved file. They will also receive Message_DataLoad on completion of the save and use this as the trigger for opening the Filer window of the directory the file has been saved in.

Icon bar copy protocol

For file copies to work, a different system is necessary. Without modification, when a file is dragged from a Filer viewer to a device icon, the Filer will send a Message_DataLoad to the FS Filer responsible for the device. All the FS filers will be changed to receive this message and then to reply with a new message, [Message_FilerDevicePath (on page 323)](#).

System variables

FSFiler$DefaultPath

Default path for files dropped on the filer icon

Use

The iconbar device filers should use these variables - substituting
their own filing system name in the name - to decide where to save
files when the user drops a file on their iconbar icon.

Related messages

[Message_FilerDevicePath (on page 323)](#)

Wimp messages

<div align="right">

Message_FilerDevicePath (&408)

</div>

Request to Filer to copy a file to a location

Message

Offset	Contents
R1+20	zero-terminated path name to copy to

Source

Icon bar Filer tasks

Destination

Filer

Delivery

Message must be sent directly to task

Message may only be sent normally (reason code 17)

Use

This message is sent by icon bar filer tasks to the Filer in response to a Message_DataLoad to request that it copy the file to a new location.

The path that should be copied to is formed as *<DevicePath>.<path>*.

Where *<DevicePath>* is the root directory of the device eg. `ADFS::HardDisc4.$`, and *<path>* is the expansion of `FSFiler$DefaultPath`. If the variable is unset, the root of the device should be used. By default, these system variables will be unset but will be left to more experienced users to set, as needed.

An example may help to clarify. If the user has set `ADFSFiler$DefaultPath` to be `Files.Junk` and they drag a file from a Filer viewer to the HardDisc4 icon, then the ADFS Filer should return a Message_FilerDevicePath, with the path name `ADFS::HardDisc4.$.Files.Junk` and the Filer will copy the file into that directory (if it exists).

The FS Filer will also prompt the Filer to open the directory viewer for the directory who's path it has just specified, using Message_FilerOpenDir.

If the path name consists of no characters and then the terminator, it is assumed that the root directory is read only.

⚠ FIXME: What does this mean? Does it mean that the variable can be set to an empty string to not perform the save ?

Related system variables

Related messages

Message_FilerOpenDir

Document information

<table>
<tr><td>**Maintainer(s):**</td><td colspan="4">Gerph <gerph@gerph.org></td></tr>
<tr><td>**History:**</td><td>**Revision**</td><td>**Date**</td><td>**Author**</td><td>**Changes**</td></tr>
<tr><td></td><td>A-H</td><td>28 Jan 1988</td><td>RML</td><td>**Initial version**</td></tr>
<tr><td></td><td>2</td><td>24 Jan 2022</td><td>Gerph</td><td>**Conversion to PRM-in-XML**</td></tr>
<tr><td></td><td></td><td></td><td></td><td>● Created from original RISC OS 4 documentation</td></tr>
<tr><td>**Related:**</td><td colspan="4">None</td></tr>
<tr><td>**Disclaimer:**</td><td colspan="4">© Gerph, 2022.</td></tr>
</table>

Icon bordering filters

Introduction

The WindowManager has, since RISC OS 3, been able to render icons with styled borders. In RISC OS 3.0 this was through the 'Z' validation, and with RISC OS 3.1 this settled on the 'R' validation to select the button style. The forms of borders that the WindowManager renders were, at that time, fixed to be 3D-effect rectangular regions. The WindowManager and FilterManager have been updated to provide the ability to offload the rendering of icon borders to third party extensions through Icon border filters.

Icon border filters are able to change the presentation of the icons with styled borders through extension modules. A 'plain' rendering, which matches the original style used by RISC OS 3 onwards, is supplied in the form of the IconBorderPlain. As the first registrant, this is used as a fallback when no other custom form of buttons is available. This results in an experience for users which is unchanged until another customised module is loaded (or activated).

Customisation of the icon borders is intended to allow greater degrees of customisation for the user, and easier development of 'themes' which group many stylistic elements into a cohesive look and feel. RISC OS Select is supplied with a customisable border filter which allows for rounded buttons, and some variation on the standard rectangular button.

Icon border filtering is available in RISC OS Select versions of the WindowManager from 4.64 onwards.

Overview

Icon border filters are provided by relocatable modules, registered through SWI Filter_RegisterIconBorderFilter (on page 334). The WindowManager will call the FilterManager to dispatch requests to draw icon borders to the registered filters. The WindowManager does not contain any code to render the styled borders - and if no filters are registered which can provide the rendering of the requested icons, there will be no border drawn.

Icon border filters are dispatched in most recent registration order. This is the same behaviour as other filters within FilterManager, and of the software vectors. This means that filters may layer their behaviour upon one another if necessary.

Icon border filters can:

- Change the outer border style
- Change the inner fill style
- Change the colouring of borders and filled regions
- Change the position of text within the border
- Look different when selected
- Look different when the mouse is over them

Technical details

Registration

Icon border filters may be registered by modules when they start, using [SWI Filter_RegisterIconBorderFilter (on page 334)](#). When they are finalised they must de-register themselves with [SWI Filter_DeRegisterIconBorderFilter (on page 336)](#).

Modules should be aware of the FilterManager's service calls. They should register themselves if they receive Service_FilterManagerInstalled, and note that they are not registered on receipt of Service_FilterManagerDying

Rendering icons

When an icon with a styled border needs to be redrawn by the Window Manager the following steps are followed:

1. Call [IconBorder_State (on page 345)](#) to determine whether a full redraw is required due to the shape changing.
   If it does require a full redraw, redraw all the content from the window background up to the icon.

2. Call [IconBorder_Colour (on page 343)](#) to determine the colours to use for the icon, supplying the initial colours given by the icon itself.

3. Apply any additional changes to the colour of the icon indicated by its validation:

   - Apply any tinting validation
   - Apply selected icon highlighting
   - Apply shading of the colours

4. If the icon is filled, call [IconBorder_Fill (on page 339)](#) to render the background of the icon.

5. Call [IconBorder_Draw (on page 337)](#) to render the border of the icon.

6. Call [IconBorder_Size (on page 341)](#) to determine the size to set the graphics window to, to render the text and/or sprite within the icon.

Customisable features

Icon borders can customise some of the rendering features of the icon, but are constrained by the existing use of the icons within applications, and the expectations of users. A given filter can change just as much of an icon's rendering as it wants, although if this matches up poorly with other filters, or the applications, it may give

an unappealing look.

In general, there are a few features of icons that icon border filters may wish to check before attempting to render icons.

- Filled flag: Some icons are commonly filled, for example buttons (type 5 and 6) and writable boxes (type 7). Recognising unfilled forms of these icons and rejecting them may prevent unexpected effects.
- Sprite icons: Any of the styled borders which use sprites are likely to have undesireable effects if the icon border deviates far from the expected 3D effect.
- Long text: The 'L' validation to render long text inside the icon may not wrap correctly if the borders of the icon change significantly.
- Oversize icons: For stylistic reasons, styled borders may have been used to create tall regions, or vertical dividers with a very thing icon. Checking the size of the icon is suitable and rejecting the icon if it is unsuitable may ensure that the intended effect is retained for the user.
- Inactive buttons: The button type borders (type 5 and 6) may have been used in cases where the icon's button type is set to a type which does not react to the user's clicks. It may be undesireable to style such buttons as if they are pressable.

Border colouring

Border colours may be changed by the filter. This might be as simple as changing the strength of the 3D effect, or forcing the colours to match a different style. The colouring of the border can be modified at will by the filter, but this may need to be done with care to avoid explicit choices by application authors being overridden in ways that produce unusable interfaces.

For example, forcing the border of all icons to be solid black with no 3D effect would look fine within a regular application, but any application which used a black background with white text would find that the border became indistinguishable from the filled background of the icon. Filters should either declare that they are not suitable for use with applications which do not follow its expectations, or should attempt to cater for non-standard forms. This might mean disabling themselves when colours are not as expected, or providing variations which retain contrast.

Restricting effects to just where the button borders (type 5 and 6) may avoid making too great a set of changes to the buttons.

Fill colouring

The background fill colour can equally be changed as freely, but has a much greater impact on the user's experience as most button icons (type 5 and 6) will be filled. Informational fields which are shown as sunk borders (type 2) are commonly filled, but that form of field is also commonly used as a colour selector region.

Bordering

Although the border is expected by designers and users to be a rectangular border, equally surrounding the text of the icon, the icon border author will find that there is flexibility in how the border is drawn.

The most obvious change that can be made is to use non-rectangular borders. The round borders supplied with RISC OS Select through the IconBorderReound module show that with some degree of freedom for regular button icons (type 5 and 6) the corners can be varied. Any changes to the rendered border width must also be reflected when the size of the text is calculated.

The border shape can be different for a selected and unselected border. This can have useful effects for pressing buttons. For example, when pressed, the button might bulge outwards (although it must still not exceed the bounds given by the icon). If the rendered shape of the border changes when pressed, the border filter must return with bit 0 set in the filter flags when [IconBorder_State (on page 345)](#) is called.

The border is not required to use the colours supplied to it. Whilst using other colours will mean that layered filters will be unable to change the style of a button, it may allow certain styles which are otherwise impossible. For example, an default button might have a more stylised shape which uses more colours than the single 'well colour'. Used with care, this may make for an interesting effect.

Generally it is best to keep to colours based on those supplied. For example, the IconBorderRound implementation allows a graduated fill to be used to make the button appear more rounded. This uses a colour slightly lighter and slightly darker than the supplied colour, with the horizontal mid-line being the colour supplied.

Filling

The fill operation is performed before the text is rendered, and should fill the region within the bordered itself. Not all icons are filled, and those that are not filled will never receive the call to their [IconBorder_Fill (on page 339)](#) entry point.

As with the border, filling is not required to use the colours supplied to it. The problems of selecting different colours for the fill are worse than that of the border as the text must be visible on top of whatever colour is filled.

Generally it is best to keep to colours based on those supplied. For example, the IconBorderRound implementation allows a graduated fill to be used to make the button appear more rounded. This uses a colour slightly lighter and slightly darker than the supplied colour, with the horizontal mid-line being the colour supplied.

Sizing the text

Although the expectation is that the icon will have a symmetric border, this is not required. Varying degrees of success have been found with creating icon borders for buttons which have one size larger than the other. When the border has different sizes, the size which the text can be rendered into must be returned correctly when the IconBorder_Size (on page 341) entry point is called.

The default behaviour when buttons (type 5 and 6) are selected is to change the border rendering. However, for some effects, moving the text to one side when selected may be a useful effect. This can be acheived by changing the size of the region which the text can be rendered into. As most bordered text is centred horizontally, this will generally have the effect of moving the text by half the distance that the border was increased.

Highlighting

Where supported by the WindowManager, the icon borders may be aware of the pointer being placed over the bordered icon. This is indicated by bit 23 (the 'deleted' bit) being set in the icon flags. The highlighting of the icons will only happen when the IconBorder_State (on page 345) call returns with bit 1 set, indicating that the icon supports being highlighted.

Highlighting the icon may be as simple as changing the colours. However, it may mean a completely different border shape, or even a different text position.

Common parameters

The entry points have some common parameters passed through the registers on entry.

Icon flags word

The icon flags word supplied to the border rendering entry points is

the same as that used in the icon block, with a small exception. The 'icon is deleted' bit (bit number 23) is repurposed to indicate that the pointer is currently over the icon. This bit is only set when the entry point has declared that the filter is able to change when the pointer is over the icon.

The inverted and shaded bits in the flags word will be set according to the original icon's state, and may change the rendering of the border. The colours supplied to the border rendering entry points will have been updated by the WindowManager to reflect the icon's state when the IconBorder_Draw (on page 337) and IconBorder_Fill (on page 339) entry points are called.

Icon border rendering box

The icon rendering box is supplied as a parameter to the border entry points to describe the region the icon border should cover. It contains 8 words which describe the box, together with the size of the pixels on the screen. The coordinates are in half-open format (x0 and x1 are inclusive coordinates, and x1 and y1 are exclusive coordinates).

Offset	Contents
+0	x0 coordinate in OS units
+4	y0 coordinate in OS units
+8	x1 coordinate in OS units
+12	y1 coordinate in OS units
+16	x pixel size in OS units
+20	y pixel size in OS units
+24	x pixel size in OS units - 1
+28	y pixel size in OS units - 1

Icon border colour table

The colour table is supplied as a parameter to the border entry points to describe the colours to be used for the icon border regions. The colours are supplied as 32bit palette entries in the form &BBGGRRxx.

Offset	Contents
+0	Foreground colour
+4	Background colour
+8	Selected background colour (for border type 5 and 6)
+12	Well colour (for border type 6)
+16	'Face' colour (usually the light highlight colour)
+20	'Opposite' colour (usually the dark highlight colour)

Icon rendering flags

The icon rendering flags passed to the icon border filter allow the
WindowManager to control additional features of the rendering. This
allows the icon to be rendered consistent with configuration of the
WindowManager by honouring the configuration to dither colours, or
sprites.

Bit(s)	Meaning
0	Dither background colours
1	Dither deep sprites
2	Reserved, must be zero

Configuration

Modules which provide icon border filters should include *Commands
to allow them to be activated and deactivated. They may also provide
the ability to configure their capabilities. It is strongly recommended
that modules initialise in their disabled state. This will allow users to
load multiple filter modules and select which are active.

SWI calls

Filter_RegisterIconBorderFilter
(SWI &4264C)

Register a filter to handle the rendering of icon borders

On entry

R0 = Pointer to zero-terminated string describing the filter (must be static for the lifetime of the filter)
R1 = Pointer to entry point for the filter code
R2 = R12 value to supply to the filter code
R3 = Mask of border types supported by this filter

Bit(s)	Meaning
0	Simple bordered icons (not used in current versions)
1	R1 border supported (Raised region)
2	R2 border supported (Lowered region)
3	R3 border supported (Ridge group)
4	R4 border supported (Channel group)
5	R5 border supported (Action button)
6	R6 border supported (Default button)
7	R7 border supported (Writable box)
8-31	Reserved, must be zero

On exit

R0 - R9 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

The Filter_RegisterIconBorderFilter SWI is used to register a new filter with the FilterManager which is capable of providing rendering of the borders on icons. The new filter code is registered for only certain border types through the use of mask bits. The filter will only be called for those button types which have been registered.

Although the filter may report an interest in a given set of borders, it is not required to actually service any of the requests. For example, a filter might only take effect for a particular size of border, and for all others it can pass on the call to other filters.

Related SWIs

SWI Filter_DeRegisterIconBorderFilter (on page 336)

Filter_DeRegisterIconBorderFilter
(SWI &4264D)

De-register a filter from handling the rendering of icon borders

On entry

    R0 = Pointer to zero-terminated string describing the filter (should be the same pointer as used on registration)
    R1 = Pointer to entry point for the filter code
    R2 = R12 value to supply to the filter code

On exit

    R0 - R9 preserved

Interrupts

    Interrupts are undefined
    Fast interrupts are enabled

Processor mode

    Processor is in SVC mode

Re-entrancy

    SWI is not re-entrant

Use

The Filter_DeRegisterIconBorderFilter SWI is used to remove the registration of border rendering code. The values supplied in R0-R2 must match those that were supplied on registration.

Related SWIs

SWI Filter_RegisterIconBorderFilter (on page 334)

Entry points

IconBorder_Draw (0)

Draw an icon border on behalf of the WindowManager

On entry

R0 = Border type (0-7) of the bordered icon
R1 = Icon flags word (on page 331) of the bordered icon
R2 = Pointer to Icon border rendering box (on page 332) for the icon being drawn
R3 = Pointer to Icon border colour table (on page 332) for the icon being rendered
R4 = Icon rendering flags (on page 333) for the icon
R9 = 0 (reason code)

On exit

R9 = -1 if handled, or preserved to pass to the next filter

Interrupts

Interrupts are enabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is not re-entrant

Use

This entry point is called when the WindowManager wishes to render the border of an icon. The border type will have been filtered by the mask supplied on registration. Filters may decide to handle the border returning with R9 set to -1, or pass it on to other border filters by preserving R9.

The filter may update the icon border rendering box if it is passing on the call. This can be used if the border wishes to handle part of the outer rendering of the border before the next filter handles it.

The filter may use any of the parameters to decide whether it wishes to handle rendering of the border. For example, a filter may only handle certain sizes of boxes, or only icons which have a particular combination of flags set. Care must be taken to ensure that this presents a consistent experience to the user, as icons which change in style may be confusing.

The filter should draw a suitable border within the bounds of the icon border rendering box, and reduce the bounding box size accordingly. To be consistent with the user's configuration, the flags in R4 should be used to decide whether dithering should be used to draw the border.

Related SWIs

SWI Filter_RegisterIconBorderFilter (on page 334)

Related entry points

IconBorder_Fill (on page 339)
IconBorder_Colour (on page 343)
IconBorder_Size (on page 341)
IconBorder_State (on page 345)

Fill an icon border on behalf of the WindowManager

On entry

R0 = Border type (0-7) of the bordered icon
R1 = [Icon flags word (on page 331)](#) of the bordered icon
R2 = Pointer to [Icon border rendering box (on page 332)](#) for the icon being drawn
R3 = Pointer to [Icon border colour table (on page 332)](#) for the icon being rendered
R4 = [Icon rendering flags (on page 333)](#) for the icon
R9 = 1 (reason code)

On exit

R9 = -1 if handled, or preserved to pass to the next filter

Interrupts

Interrupts are enabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is not re-entrant

Use

This entry point is called when the WindowManager wishes to render the fill of a bordered icon. The border type will have been filtered by the mask supplied on registration. Filters may decide to handle the border returning with R9 set to -1, or pass it on to other border filters by preserving R9.

The filter may use any of the parameters to decide whether it wishes to handle rendering of the border. Usually this is the same criteria used to decide whether icon should be handled in the [IconBorder_Draw (on page 337)](#) entry point. For example, a filter may only handle certain sizes of boxes, or only icons which have a particular combination of flags set. Care must be taken to ensure that

this presents a consistent experience to the user, as icons which change in style may be confusing.

The filter should fill the region not covered by the border within the bounds of the icon border rendering box. The bounding box supplied will be that of the icon itself. The filter will only be called icon flag bit is set in the icon flags word which indicates that the icon is filled. To be consistent with the user's configuration, the flags in R4 should be used to decide whether dithering should be used to fill the icon.

Related SWIs

SWI Filter_RegisterIconBorderFilter (on page 334)

Related entry points

IconBorder_Draw (on page 337)
IconBorder_Colour (on page 343)
IconBorder_Size (on page 341)
IconBorder_State (on page 345)

Return the size available for text after rendering the icon border

On entry

R0 = Border type (0-7) of the bordered icon
R1 = Icon flags word (on page 331) of the bordered icon
R2 = Pointer to Icon border rendering box (on page 332) for the icon being drawn
R9 = 2 (reason code)

On exit

R9 = -1 if handled, or preserved to pass to the next filter

Interrupts

Interrupts are enabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is not re-entrant

Use

This entry point is called when the WindowManager wishes to render the text of a bordered icon. The text within an icon will be bounded by the edges of the border. In order that this text be clipped to those edges, this entry point must reduce the size of the icon border reendering box in R2 by the space covered by the border. The border type will have been filtered by the mask supplied on registration. Filters may decide to handle the border returning with R9 set to -1, or pass it on to other border filters by preserving R9.

The filter may use any of the parameters to decide whether it wishes to handle this border. Usually this is the same criteria used to decide whether icon should be handled in the IconBorder_Draw (on page 337) entry point. For example, a filter may only handle certain sizes of boxes, or only icons which have a particular combination of flags set. Care must be taken to ensure that this presents a consistent

experience to the user, as icons which change in style may be confusing.

The filter should increase the x0 and y0 values, and decrease the x1 and y1 values in the bounding box to reflect the region that the text of the icon may use.

Related SWIs

SWI Filter_RegisterIconBorderFilter (on page 334)

Related entry points

IconBorder_Draw (on page 337)
IconBorder_Fill (on page 339)
IconBorder_Colour (on page 343)
IconBorder_State (on page 345)

Update the colours for an icon border on behalf of the WindowManager

On entry

R0 = Border type (0-7) of the bordered icon
R1 = [Icon flags word (on page 331)](#) of the bordered icon
R2 = Pointer to [Icon border rendering box (on page 332)](#) for the icon being drawn
R3 = Pointer to [Icon border colour table (on page 332)](#) for the icon being rendered
R9 = 4 (reason code)

On exit

R9 = -1 if handled, or preserved to pass to the next filter

Interrupts

Interrupts are enabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is not re-entrant

Use

This entry point is called when the WindowManager is about to render a bordered icon, to give the border rendering filters an opportunity to override the icon's own colours. The border type will have been filtered by the mask supplied on registration. Filters may decide to handle the border colours returning with R9 set to -1, or pass it on to other border filters by preserving R9.

The filter may use any of the parameters to decide whether it wishes to handle rendering of the border. Usually this is the same criteria used to decide whether icon should be handled in the [IconBorder_Draw (on page 337)](#) entry point. For example, a filter may only handle certain sizes of boxes, or only icons which have a

particular combination of flags set. Care must be taken to ensure that this presents a consistent experience to the user, as icons which change in style may be confusing.

The filter should update the values in the icon border colour table in R3 if it wishes to override the colours that the icon has selected. The colours in the table have not yet been updated to invert, shade, tint or apply other colour effects to the icon. As such, the colours chosen here are the base colours. After this entry point has returned, the colours will be updated by the WindowManager to reflect the effects that the icon requests.

Related SWIs

SWI Filter_RegisterIconBorderFilter (on page 334)

Related entry points

IconBorder_Draw (on page 337)
IconBorder_Fill (on page 339)
IconBorder_Size (on page 341)
IconBorder_State (on page 345)

Get information about the type of icon border filter

On entry

R0 = Border type (0-7) of the bordered icon
R1 = Icon flags word (on page 331) of the bordered icon
R2 = Pointer to Icon border rendering box (on page 332) for the icon being drawn
R3 = Filter flags word

Bit(s)	Meaning
0	The border changes in shape, so must be redrawn completely on all state transitions
1	The border has a different style when the pointer is over it (the border is 'highlightable')
2	Reserved, must be zero

R9 = 5 (reason code)

On exit

R3 = Filter flags word updated with this filter's behaviour for the icon
R9 = -1 if handled, or preserved to pass to the next filter

Interrupts

Interrupts are enabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is not re-entrant

Use

This entry point is called when the WindowManager needs to know about the filter's behaviour with the icon. The border type will have been filtered by the mask supplied on registration. Filters may decide to handle the border colours returning with R9 set to -1, or pass it on to other border filters by preserving R9.

The filter may use any of the parameters to decide whether it wishes to handle rendering of the border. Usually this is the same criteria used to decide whether icon should be handled in the IconBorder_Draw (on page 337) entry point. For example, a filter may only handle certain sizes of boxes, or only icons which have a particular combination of flags set. Care must be taken to ensure that this presents a consistent experience to the user, as icons which change in style may be confusing.

The entry point is used to decide whether to redraw it fully from the background when a state change happens, or if the icon needs to be drawn at all as the pointer moves over it. The filter should update the flags by OR-ing any new flags into the supplied filter flags word and returning the new value in R3.

Shape changes

If the rendered border changes shape when there are state transitions, then bit 0 should be set on return. This might happen if the icon had rounded edges normally and square edges when selected, or if the outer edges were not drawn in any colour unless the pointer was over the icon.

Highlightable borders

If the rendered border provides a form of highlighting when the pointer is over the icon, then bit 1 should be set on return. The highlighting will only be performed on border types 5 and 6.

Related SWIs

SWI Filter_RegisterIconBorderFilter (on page 334)

Related entry points

IconBorder_Draw (on page 337)
IconBorder_Fill (on page 339)
IconBorder_Colour (on page 343)
IconBorder_Size (on page 341)

Document information

Iconbar priorities

Introduction

Iconbar priorities have been vague at best, with some usages being declared to be the 'easiest way to do things'. In addition to this, some entities such as ADFSFiler have used incorrect iconbar priorities by accident. This particular case means that Floppy discs do not appear where they are documented to appear in the PRMs. Floppies are documented to appear at &60000000. They actually appear at &70000000. ShareFS used a priority of &68000000. This results in a mismatched iconbar, where the three cases of documentation, logical appearance and prior use cannot be resolved simultaneously.

Thus, it has been decided to clarify the usage of parts of the iconbar. This should make for a more logical system, and the possibility of multi-tier iconbars and other such changes.

Technical details

These categorisations provided here amend and expand on documentation provided in the PRMs. The intent is to clarify the system for a logically organised iconbar, with clearly defined positioning for components, and whilst retaining the current state wherever possible.

The iconbar should be viewed as:
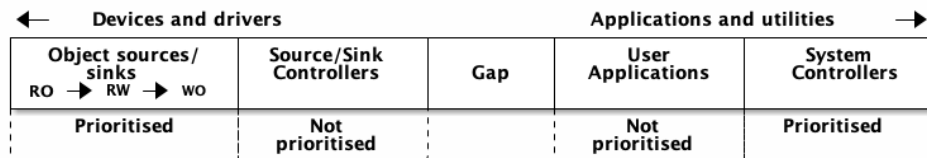
Iconbar layout

Object sources and sinks

These are icons for devices to which objects can be sent or retrieved from. They are ordered logically from read only devices on the left through read/write devices to write only devices towards the right. Alongside write only devices are the volatile devices; those devices whose contents are not likely to remain permanent from session to session (or even within a session).

The full ordering is:

Priority	Name	Meaning
&76000000	Scanners	Read-only device
&74000000	CD-ROM	Read-only device
&70000000	Hard disc	Read-write device Examples: Any fixed RW medium falls into this category
&68000000	Floppies	Read-write device Examples: Any removable RW medium falls into this category
&60000000	Network	Read-write/Read-only device Examples: Any network filing system falls into this category
&40000000	Volatiles	Read-write device Examples: RAMFS, Transient, Trash cans, Memphis, Scrap, etc
&0F000000	Printers	Write-only device
&04000000	Accelerators	Examples: ResourceFS, "Pinned" items, Director, Menon, etc.

All devices in this category should have a name underneath, ideally identifying the medium name with which they are associated. If no medium is associated (eg. a removable, or unconfigured device), they should display the medium name (eg. Zip disc, Printer, LanMan, etc), or a generalised medium identifier (eg. the drive or port number).

These applications should not provide a "Quit" option.

Data source / sink controllers

This is basically a place for internet servers, connection systems and other network utilities to live, as well as local servers. Like the Object sources, these should have their name under them. Samba, TelnetD, Newsbase, InetSuite, WebServe and Netplex would fall into this category. These will grow to the right when the user loads a new controller.

These applications should provide "Quit" options. Most should provide a status window, and many will provide configuration windows.

User applications

This is where user applications appear when loaded. They will grow to the left as they are loaded, taking up the free gap space.

These applications should provide a "Quit" option.

They should not have text placed under them unless they are configured into a particular state that must be described. Such applications are discouraged unless there is a genuine need.

System control applications

This is where system control applications live. These are things that will control the machine, the desktop or the way in which the system works. Initially, this comprises the Task Manager and Display Manager. Because of its high priority, Help lives here too. This may be rationalised in future.

Document information

Maintainer(s):	Gerph <gerph@gerph.org>			
History:	**Revision**	**Date**	**Author**	**Changes**
	1	08 Feb 2000	Gerph	**Initial version**
				● Released as part of Technote 20000502-001.
	2	12 Jan 2022	Gerph	**PRM-in-XML conversion**
				● Created from original Select documentation.
Related:	http://www.riscos.com/support/developers/riscos6/desktop/wimp/iconbarpriorities.html			
Disclaimer:	© Gerph, 2021.			

Hardware timer device driver (TimerManager)

Introduction

The Timer module provides an abstraction of the hardware timers. It is used by the Kernel in order to provide the monotonic timer used for the system clock, interval timer, monotonic timer, and system timed events. The module also provides an interface to allow other hardware timers to be controlled by other components. Timers may run off independant clock sources and so may have different granularity and ranges of rates at which they may generate interrupts.

Each hardware implementation has an independant Timer module implementation specific to the timers which are available to the operating system.

Overview

The number of timers provided by the hardware can be read using [SWI Timer_ReturnNumber (on page 0)](). Timers may be claimed and released by components using [SWI Timer_Claim (on page 0)]() and [SWI Timer_Release (on page 0)](). The rate at which a timer is running can be modified after it has been claimed by using [SWI Timer_SetRate (on page 0)](). The relationship between timer rates and external measurements can be obtained by using [SWI Timer_Convert (on page 0)]().

Technical details

A number of timers may be provided by a TimerManager hardware device driver. These timers can be claimed by a single client at any time. The timer's rate may be defined in a number of different forms, to allow clients to specify the rate in the most natural manner. Not all timers may support the exact rate requested, so clients should expect to handle different forms of timers.

Timers are numbered from 0, and timer 0 is reserved for use by the Kernel as the monotonic timer.

Measurement format

Many of the SWIs take a number of flags to indicate the measurement format of the timer. The measurement format flags take the form of 8 bits:

Bit(s)	Meaning
0-2	Unit scaler:

Value	Meaning
0	Invalid
2	Scaled by 1/1000000
3	Scaled by 1/1000
4	Scaled by 1
5	Scaled by 1000
6	Scaled by 1000000
7	Scaled by 1000000000

Bit(s)	Meaning
3	Reserved, must be zero
4-5	Measurement type:

Value	Meaning
0	Native ticks
1	Frequency (interrupts per second)
2	Period (in seconds)
3	Invalid

Bit(s)	Meaning
6-7	Reserved, must be zero

For example, to request a frequency of 100 Hz one could use a measurement type of 1, a scaler of 4 (scale by 1) and a value of 100. Alternatively, this could be represented as a period of 1/100th second by using a measurement type of 2, a scaler of 3 (scale by 1/1000) and a value of 10.

For SWIs which take an input rate the measurement format flags are held in the bits 0-7 of the SWI flags.

For SWIs which return a rate the measurement format flags are held in bits 8-15 of the SWI flags.

SWI calls

TimerManager_ReturnNumber
(SWI &58B80)

Return number of supported timers

On entry

R0 = Flags (must be 0)

On exit

R0 = Number of timers available

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to find the number of timers available to the
operating system. Timers are numbered from 0, so the maximum
timer number that may be used is the value returned - 1.

Related APIs

None

TimerManager_Claim
(SWI &58B81)

Claim a hardware timer

On entry

R0 = Flags:

Bit(s)	Meaning
0-7	Measurement format for the timer rate
8-15	Measurement format for the returned timer rate
16-31	Reserved, must be zero

R1 = Timer number
R2 = Timer rate, using the measurement format from bits 0-7
R3 = Pointer to function to call on interrupt
R4 = Value to pass in R12 to interrupt function

On exit

R2 = Actual timer rate, using the measurement format from bits 8-15,
or 0 if the rate cannot be represented

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to claim a timer for dedicated use by a client. Only a
single client may claim a timer; subsequent claims will return an
error. The timer specified will be set to the rate given and interrupts
will call the routine specified. The interrupt routine may corrupt
R0-R3 but should preserve all other registers.

An error will be returned if the input measurement format in R0 bits
0-7 is not valid or cannot be provided by the timer.

If the meaurement format used for return in R0 bits 8-15 is invalid the value returned in R2 will be 0, but no error will be raised. The return value is provided as a convenience.

Related vectors

TimerManager_Release
(SWI &58B82)

Release a hardware timer

On entry

R0 = Flags (must be zero)
R1 = Timer number

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to release a previously claimed timer. The IRQ will
no longer cause the specified routine to be called.

Related vectors

TimerManager_Claim (on page 0)

TimerManager_SetRate
(SWI &58B83)

Change the rate used by a hardware timer

On entry

R0 = Flags:

Bit(s)	Meaning
0-7	Measurement format for the timer rate
8-15	Measurement format for the returned timer rate
16-31	Reserved, must be zero

R1 = Timer number
R2 = Timer rate, using the measurement format from bits 0-7

On exit

R2 = Actual timer rate, using the measurement format from bits 8-15, or 0 if the rate cannot be represented

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to change the rate used by a timer. Only timers which have been claimed can have their rate changed; unclaimed timers will return an error. The timer specified will be set to the rate given.

An error will be returned if the input measurement format in R0 bits 0-7 is not valid or cannot be provided by the timer.

If the meaurement format used for return in R0 bits 8-15 is invalid the value returned in R2 will be 0, but no error will be raised. The

return value is provided as a convenience.

Related vectors

TimerManager_Convert
(SWI &58B84)

Convert between rate formats used by a hardware timer

On entry

R0 = Flags:

Bit(s)	Meaning
0-7	Measurement format for the timer rate
8-15	Measurement format for the returned timer rate
16-31	Reserved, must be zero

R1 = Timer number
R2 = Timer rate, using the measurement format from bits 0-7

On exit

R2 = Timer rate in form specified by R0 bits 8-15

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to convert between timer rate formats. The values converted will be checked to ensure that the timer is capable of those rates.

An error will be returned if the input measurement format in R0 bits 0-7 is not valid or cannot be provided by the timer.

An error will be returned if the meaurement format used for return in R0 bits 8-15 is invalid.

Related vectors

Document information

NVRAM vector

Introduction

The RISC OS system uses memory which is preserved whilst the power is off to store configuration information. This allows the system to start with the correct settings such as hardware configuration and user preferences. Examples of hardware configuration are settings such as the filing system to boot from and which drive should be used. User preferences include the type and volume of system beep.

Technical details

Under earlier versions of the operating system the non-volatile RAM ('NVRAM', also referred to as CMOS RAM, or battery backed RAM) was handled entirely by the Kernel. From Kernel 9.48, the handling of NVRAM is delegated to hardware support modules. The Kernel communicates with these modules through the vector NVRAMV.

Driver modules which provide the NVRAMV vector should be initialised with the early initialisation flag (module flags bit 1) set. This allows the modules to be started before the configuration for unplugged modules is required.

Terminology

The configuration data handled by NVRAMV has generally been termed 'CMOS' or 'CMOS data'. Historically, the configuration information was used CMOS technology to store the contents of the memory, but this is not required. The name 'non-volative RAM' is a more general term which does not imply the use of a particular technology, so is used to describe the mecanishm for storing the configuration data.

Software vectors

Vector NVRAMV (Vector &3E)

Operations on non-volatile memory used for configuration

On entry

R0 = reason code:

Value	Meaning
0	Populate the cache with NVRAM data (on page 368)
1	Read a single value from NVRAM (on page 369)
2	Write a single value to NVRAM (on page 370)

On exit

R0 = -1 if handled, preserved if not handled
R1 - R9 = dependant on reason code

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called by the Kernel to control the configuration data stored in the NVRAM.

Related SWIs

SWI OS_Byte 161SWI OS_Byte 162

Populate the cache with NVRAM data

On entry

R0 = 0 (reason code)
R1 = pointer to cache block to fill
R2 = number of bytes to fill

On exit

R0 = -1 (operation complete)
R1 preserved
R2 = number of bytes populated

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called by the Kernel to fill in its cache of NVRAM values. A cache is provided in order to reduce the impact of repeated reading of configuration data by clients. Clients should write 0 to the cache for unsupported values. The number of bytes to fill may take any value. The total amount of NVRAM should be returned, not the amount of NVRAM filled. Only the first 240 bytes of NVRAM will be used by the Kernel initially.

Related APIs

None

Read a single value from NVRAM

On entry

R0 = 1 (reason code)
R1 = byte to read

On exit

R0 = -1 (operation complete)
R1 = value read, or 0 if byte is out of range

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called by the Kernel to read a single value. It will usually only be used before the NVRAM cache has been populated during system initialisation.

Related SWIs

SWI OS_Byte 161

Write a single value to NVRAM

On entry

R0 = 2 (reason code)
R1 = byte to read

On exit

R0 = -1 (operation complete)
R1 = value read, or 0 if byte is out of range

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called by the Kernel to write a single value to the
NVRAM.

Related SWIs

SWI OS_Byte 162

Document information

Maintainer(s):	Gerph <gerph@gerph.org>			
History:	**Revision**	**Date**	**Author**	**Changes**
	1	31 May 2023	Gerph	**Initial version**
				● Created from Select technical documentation.
Related:	http://www.riscos.com/support/developers/riscos6/ hardware/nvramv.html			
Disclaimer:	© Gerph, 2023.			

Real Time Clock

Introduction

The Real Time Clock has previously been handled by the Kernel. With Kernel 8.64 and later the clock is managed by the RTC module. This communicates with the hardware driver through a vector (RTCV) whose default claimant is the RTCHW module. The RTCHW module provides implementations for the RiscPC, A7000-series, RiscStation, and A9.

The RTC module provides the Kernel SWI OS_ResyncTime and all the OS_Word 14 and 15 operations to control the clock. A new reason code has been added to SWI OS_Word 15 (on page 0) for setting the clock's 5 byte time directly.

A separate section describes the RTCV vector.

Service calls

Service_RTCSynchronised (Service Call &DD)

Real time clock has been synchronised

On entry

R1 = &DD (reason code)

On exit

R1 preserved

Use

This service is issued by the RTC module to inform clients that the software and hardware clocks have been synchronised. It may indicate that an indeterminate period of inactivity has taken place, such as after returning from a suspend state. Where possible, timed events should be synchronised and where necessary appropriate action taken to ensure that queued events take place.

This service should never be claimed.

Related SWIs

SWI OS_ResyncTime (on page 375)

SWI calls

OS_Word 15, 5
(SWI &7)

Set real time clock to UTC time as a 5-byte value

On entry

R0 = 15 (reason code)
R1 = Pointer to time values:

Offset	Contents
+0	5 (sub-reason code)
+1	5 bytes of time value as centiseconds since 1900 in UTC

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This call is used to set the Real Time Clock to a time value as a UTC time. It avoids the requirement to convert a UTC time to a locale-specific time string first.

This call was new to RISC OS 4.

Related APIs

None

OS_ResyncTime
(SWI &6C)

Synchronisation operations for RTC

On entry

R0 = Reason code:

Value	Meaning
0	Synchronise with hardware clock
other	Reserved

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to cause the software clock to be resynchronised with the hardware clock, where available. When changed, a service call Service_RTCSynchronised (on page 373) will be issued.

This call was new to RISC OS 4.

Related services

Service_RTCSynchronised (on page 373)

Document information

Real Time Clock Vector

Introduction

The Real Time Clock Vector is a means for alternate clock implementations to be provided for differing hardware. The calls which are provided by the Operating System (either within the Kernel or through an extension module) for setting the system time - OS_Word 14 calls - are decoded and passed to this vector for processing.

The time values passed to the RTCV handlers will be specified as UTC time. That is, they will have had any time zone or daylight saving time adjustments removed from them. Handlers should use the values directly with the suitable hardware.

Certain clock implementations, for example the RiscPC clock chip, do not provide sufficient information to allow the values to be read directly from the chip and returned. In such cases additional steps may be taken by the hardware driver.

The RiscPC clock chip driver code is limited by only storing a year value in the range 0-3. As such, two NVRAM bytes have been allocated to augment this (byte &80 and &81). Should similar adjustments be required for other hardware it is recommended that these bytes be used. During the vector call it is safe to issue NVRAM OS_Byte calls.

Software vectors

Abstracted interface to the real time clock

On entry

R0 = reason code:

Value	Meaning
0	[Read time from hardware Real Time Clock (on page 379)](#)
1	[Update hardware Real Time Clock with a new value (on page 381)](#)

R0 - R8 = Dependant on the reason code

On exit

R0 = -1 if reason claimed

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

The Real Time Clock vector is called to manipulate the real time clock.

Related APIs

None

Read time from hardware Real Time Clock

On entry

R0 = 0 (reason code)

On exit

R0 = -1 if reason claimed
R1 = centiseconds (0-99)
R2 = seconds (0-59)
R3 = minutes (0-59)
R4 = hours (0-23)
R5 = day of month (1-31)
R6 = month (1-12)
R7 = year (0-99)
R8 = century (19-21)

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This reason code is called when it is necessary to synchronise the internal clock with the hardware-supplied time source. For devices which take a significant time to be read the driver may initiate a request of the time from the hardware and pass on the call. When the hardware has been read such a driver should call SWI OS_ResyncTime request to update the Operating System with the new values and notify other clients of an updated internal clock.

Where inaccurate values are available from hardware the driver should return the middle value for the relevant range.

Related vectors

Update hardware Real Time Clock with a new value

On entry

R0 = 1 (reason code)
R1 = centiseconds (0-99)
R2 = seconds (0-59)
R3 = minutes (0-59)
R4 = hours (0-23)
R5 = day of month (1-31)
R6 = month (1-12)
R7 = year (0-99)
R8 = century (19-21)

On exit

R0 = -1 if reason claimed
R1 - R8 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This reason code is called when a request is made to set the hardware clock to a specific value. The Operating System's internal representation will not yet have been updated to reflect these values. Any of the values passed in R1-R8 value may be -1 to indicate that it is not to be modified.

Related vectors

Document information

Maintainer(s):	Gerph <gerph@gerph.org>			
History:	**Revision**	**Date**	**Author**	**Changes**
	1	2006	Gerph	**Initial version** ● Released as RISC OS Select documentation.
	2	30 Dec 2021	Gerph	**PRM-in-XML conversion** ● Created from original Select documentation
Related:	http://www.riscos.com/support/developers/riscos6/time/rtcv.html			
Disclaimer:	© Gerph, 2021.			

System clock

Introduction

The system clock, as accessed by OS_Word 1 and 2, is no longer a pair of incrementing timers. Because of this the 'timer switch' state (OS_Byte 243) is no longer used and will return a constant value.

The system clock is not related to the interval timer (OS_Word 3,4) or to the monotonic time (OS_ReadMonotonicTime) except that all are triggered at 100Hz.

Document information

ShareFS

Introduction

ShareFS provides a simple mechanism for accessing files on locally networked RISC OS systems. The system uses Freeway to distribute details of the shared discs. This allows any Freeway reachable system (usually those on the local network, but may include any NetI accessible networks), to access the shared files. Although the objects are known as 'shared discs' they may refer to parts of a filing system. Under Select 1, and later, the Filer menu offers the option to share sub-directories.

In the past the SWI calls for ShareFS have been undocumented. They are presented here to fill in this gap, but may be extended and modified without notice. The flags on the SWIs are inconsistent for legacy reasons.

From ShareFS 3.97 onward, the ShareFS Filer can be disabled by setting the ShareFS$Filer variable to 'no'. The filer can be re-enabled by setting it to any other value.

System variables

ShareFS$Filer

Whether the ShareFS Filer is enabled

Use

Controls whether the ShareFS Filer icon is displayed. It can be disabled by setting the ShareFS$Filer variable to 'no'. The filer can be re-enabled by setting it to any other value.

Related APIs

None

Service calls

<div align="right">

Service_Sharing
(Service Call &801C8)

</div>

Change to shared directories

On entry

R0 = pointer to zero-terminated filing system name ('ShareFS' in our case)
R1 = &801C8 (reason code)
R2 = Share state: 0 if object is unshared, 1 if object is shared
R3 = pointer to zero-terminated directory name being shared
R4 = pointer to zero-terminated name of the shared object
R5 = private data (filesystem specific)

On exit

R0 - R5 preserved

Use

This service is issued when a path is shared or unshared by a filing system. It should not be claimed.

Related SWIs

SWI ShareFS_CreateShare (on page 389)
SWI ShareFS_StopShare (on page 391)

SWI calls

Share a directory through ShareFS

On entry

R0 = Flags:

Bit(s)	Meaning
0	Share is protected
1	Share is read only
2	Share is hidden
3	Share is a 'sub directory'
4	Share is a CD ROM
5	Share is authenticated (use R3 as key)
6-31	Reserved, must be zero

R1 = Pointer to zero-terminated share name
R2 = Pointer to zero-terminated directory name
R3 = Authentication key number (if bit 5 of the flags is set)

On exit

R0 - R3 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to share a directory.

Related SWIs

SWI ShareFS_StopShare (on page 391)

Related messages

Message_FileShareDir (on page 395)

ShareFS_StopShare
(SWI &47AC1)

Stop sharing a directory through ShareFS

On entry

R0 = Flags (reserved, must be 0)

R1 = Pointer to zero-terminated share name, or directory name

On exit

R0 - R1 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to stop sharing a directory.

Related SWIs

SWI ShareFS_CreateShare (on page 389)

Related messages

Message_FileShareDir (on page 395)

ShareFS_EnumerateShares
(SWI &47AC2)

List the currently shared directories

On entry

R0 = Flags:

Bit(s)	Meaning
0	Share is protected
1	Share is read only
2	Share is hidden
3	Share is a 'sub directory'
4	Share is a CD ROM
5-30	Reserved, must be zero
31	Share is authenticated (use R5 as key)

R4 = Opaque value for enumeration, starting from 0
R5 = Authentication key number (if bit 5 of the flags is set)

On exit

R1 = Pointer to zero-terminated disc name
R2 = Pointer to zero-terminated directory name
R3 = Flags used for the share
R4 = New opaque value, or -1 if no more details

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to enumerate the shared discs.

Related SWIs

ShareFS_IdentifyShare
(SWI &47AC3)

Identify a shared disc

On entry

R0 = Flags:

Bit(s)	Meaning
0	Set: R1 contains share name
	Clear: R1 contains directory name
1-31	Reserved, must be zero

R1 = Pointer to zero-terminated share name or directory name
R2 = Pointer to buffer for data
R3 = Length of buffer

On exit

R0 = Flags for share (see SWI ShareFS_CreateShare (on page 389))
R3 = Length of data written to buffer, or -ve length if the name would not fit

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to identify a share given its name or directory name.

Related SWIs

SWI ShareFS_CreateShare (on page 389)
SWI ShareFS_EnumerateShares (on page 392)

Wimp messages

Message_FileShareDir (&408)

Request a dialogue for sharing directories

Message

Offset	Contents
R1+20	Flags:

Bit(s)	Meaning
0	Share is protected (public access attributes are obeyed)
1	Share is read only
2	Share is hidden (doesn't appear in display)
3	Share is authenticated (blank password initially)
4	Share is a CD ROM (Read only, with a different icon)
5-29	Reserved, must be zero
30	Open window at position given
31	Reserved, must be zero

Offset	Contents
R1+24	x co-ordinate to open at (if bit 30 set)
R1+28	y co-ordinate to open at (if bit 30 set)
R1+32	zero-terminated directory name to share

Source

Tasks

Destination

ShareFS Filer task

Delivery

Message must be broadcast (destination 0)

Message must be sent recorded delivery (reason code 18)

Use

This message will cause ShareFS to open a dialogue box showing the share details requests, or the live share details if the directory is already shared.

It should be sent by an application which wishes to present the user with a set of options for sharing a directory. A window will be opened either around the pointer, or at the position requested.

Related SWIs

SWI ShareFS_CreateShare (on page 389)

Related messages

Message_FileShareDir (on page 395)

Document information

Internet address collisions

Introduction

The Internet module now issues a service when it detects another
system on the network with the same address. Components may
recover from this by reconfiguring the interface.

Service calls

Service_InternetStatus (Service Call &B0)

Duplicate Internet address detected

On entry

R0 = 8 (subreason code)
R1 = &B0 (reason code)
R2 = pointer to zero-terminated interface name, eg 'ea0'
R3 = pointer to Driver Information Block for interface
R4 = IPv4 address which has been duplicated (network byte order)
R5 = pointer to hardware address of system with a duplicate IP address

On exit

R0 - R5 preserved
R0 = 0 to claim service when duplicate address has been resolved, or preserved to shut down the Internet module

Use

This service call is issued by the Internet module (version 5.08 or later) when it detects a machine using a duplicate IP address. This is normally detected when an incoming ARP packet is received with our IP address but a different hardware address.

As a probe, whenever an interface is reconfigured, the Internet module sends out an ARP request for our IP address to make any such machines reply. That will then trigger this service call.

Normally, the Internet module will shut down outright as a safety measure if this happens. However, if this service call is claimed it will continue operation. It is expected that anyone claiming this service call should take appropriate action; for example the DHCP module might remove our IP address, send a DHCPDECLINE message and go back into the DHCP INIT state.

Related APIs

None

Document information

DCI Driver Link Status

Introduction

Under the DCI, network device drivers must announce their presence through Service_DCIDriverStatus. It is assumed that devices announcing themselves in this way are available for use. The device may become unavailable, most likely due to link loss (such as a cable being disconnected) or memory shortage. No indication is available to the user as to the state of the device from the driver.

In order to allow notifications of such states to be provided to the user (and to other clients who may need to be aware of network infrastructure changes), it is proposed that the service call be extended. Authors should consult the DCI driver specification or the Internet chapter within PRM 5a for more details of the current interface. In summary, Service_DCIDriverStatus is issued by drivers to announce startup (reason 0) and shutdown (reason 1) of a driver and its associated DIB (Device Information Block).

This has been extended to include announcement of link status changes ([Service_DCIDriverStatus 2 (on page 402)](#)). Two new reason codes are to be used for this purpose.

Service calls

Service_DCIDriverStatus 2
LinkActive
(Service Call &9D)

Notification that the link provided by a DCI driver has become active

On entry

R0 = Pointer to Device Information Block
R1 = &9D (reason code)
R2 = 2 (sub-reason code)
R3 = DCI version supported

On exit

R0 - R3 preserved

Use

This service is issued to announce changes to a Device Driver. An 'active link' indicates that the device driver is capable of sending and receiving data. An 'inactive link' will never send or receive data. This mirrors the use of the DCI statistics flag. For compatibility with devices which are not aware of these new reason codes, all modules should assume that a newly started device driver has an active link. It follows that any device which starts up and is aware of these new reason codes must issue the 'link inactive' (reason code 3) service if its link is not available.

Expected uses for this service:

- Dynamic address configuration in presence of new network infrastructure (eg ZeroConf address re-announcement, DHCP lease renewal)
- User notification of network absence (eg Notifier protocol)

Non-module clients, and module clients wishing to obtain more information about the state of the link should query the statistics for the DCI driver in the usual manner.

Drivers may, but are not required to, defer announcement of inactive links if their physical state is such that transient failures (in the order of seconds) may occur. Drivers which can detect the physical nature of the network to which they are connected must signal a link state change when they detect such a change (eg configuration changes to

a wireless network SSID, encryption key, channel, etc).

Clients should expect that any 'link inactive' notification may indicate that the previous network connection is invalid. On 'link active' notification, clients may attempt to re-establish connections to remote systems.

Clients should attempt to use the existing connections before restarting a lengthy negotiation or configuration process. Where confidential information is involved, clients should not attempt to re-establish any connection without first confirming the action with the user.

Related services

[Service_DCIDriverStatus 3 (on page 0)](#)

Document information

Maintainer(s):	Gerph <gerph@gerph.org>			
History:	**Revision**	**Date**	**Author**	**Changes**
	1	2006	Gerph	**Initial version** ● Released as RISC OS Select documentation.
	2	30 Dec 2021	Gerph	**PRM-in-XML conversion** ● Created from original Select documentation
Related:	http://www.riscos.com/support/developers/riscos6/ networking/dcidriverlink.html			
Disclaimer:	© Gerph, 2021.			

RouterDiscovery

Introduction

The RouterDiscovery module implements RFC1256 Router Discovery for multiple interfaces as hosts or routers. The action of the RouterDiscovery must be triggered by the user in order to be used. It is expected that address configuration clients will perform this trigger when appropriate. ZeroConf would be expected to trigger RouterDiscovery if no other address has been configured. DHCP would be expected to trigger RouterDiscovery if the relevant options are returned in the DHCP packets from the configuration server.

The module will monitor interface changes and resend solicitations or advertisements as appropriate. Non-availability of the router system is not currently checked for.

Multiple interfaces are supported.

Service calls

Service_InternetStatus &40
(Service Call &B0)

RouterDiscovery has changed its host behaviour for an interface

On entry

R0 = &40 (sub-reason code)
R1 = &B0 (reason code)
R2 = New state:

Value	Meaning
0	No longer monitoring interface
1	Starting soliciting on interface
2	Starting monitoring interface

R3 = Pointer to zero terminated interface name

On exit

R1 - R3 preserved

Use

This service call is issued by the RouterDiscovery module when it starts monitoring an interface for router advertisement packets. The module will start by issuing solicitations. Once an advertisement is received the module will modify the default route appropriately.

This service should never be claimed.

Related APIs

None

Service_InternetStatus &41
(Service Call &B0)

RouterDiscovery has changed its router behaviour for an interface

On entry

R0 = &41 (sub-reason code)
R1 = &B0 (reason code)
R2 = New state:

Value	Meaning
0	Ending advertisements
1	Starting advertisements

R3 = Pointer to zero terminated interface name
R4 = Number of routes being advertised
R5 = Pointer to router/preference pairs for routers being advertised

On exit

R1 - R5 preserved

Use

This service call is issued by the RouterDiscovery module when it starts issuing advertisements on an interface. The module will initially issue a few advertisements, before settling into a much more leisurely advertisement every 10 minutes. If a solicitation is received from a host, an advertisement will be made.

This service should never be claimed.

Related APIs

None

Service_InternetStatus &42
(Service Call &B0)

RouterDiscovery has changed the route

On entry

R0 = &42 (sub-reason code)
R1 = &B0 (reason code)
R2 = Pointer to zero terminated interface name
R3 = IP address of gateway through which packets will be routed, or 0 if the default route has been deleted due to non-responsiveness.

On exit

R1 - R3 preserved

Use

This service call is issued by the RouterDiscovery module when it changes the default route based on information provided from RouterDiscovery operations.

This service should never be claimed.

Related APIs

None

SWI calls

RouterDiscovery_Control
(SWI &57D80)

Control the operation of the RouterDiscovery module

On entry

R0 = Reason code:

Value	Meaning
0	Activate Host mode for the interface (on page 410)
1	Activate Router mode for the interface (on page 412)
2	Deactivate control of interface (on page 414)

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to control the operation of the RouterDiscovery module.

Related APIs

None

RouterDiscovery_Control 0
ActivateHost
(SWI &57D80)

Activate Host mode for the interface

On entry

R0 = 0 (reason code)
R1 = Pointer to zero terminated interface name to activate on
R2 = IPv4 Aaddress to use for solicitations or special value:

Value	Meaning
&0	use appropriate address based on interface
&FFFFFFFF	use broadcast address
&E0000002	use 'all routers' multicast group

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to allow the RouterDiscovery module to control the operation of an interface as a Host. Solicitations will be sent when the interface changes state and a default route will be configured based on those addresses.

Interface names will not be validated, allowing the interfaces to become available at a future point. Absent interfaces will cause the module to become quiescent until the interfaces become available.

Related SWIs

Activate Router mode for the interface

On entry

R0 = 1 (reason code)
R1 = Pointer to zero terminated interface name to activate on
R2 = IPv4 Aaddress to use for advertisements or special value:

Value	Meaning
&0	use appropriate address based on interface
&FFFFFFFF	use broadcast address
&E0000001	use 'all hosts' multicast group

R3 = Minimum advertisement interval in seconds, or 0 for default
R4 = Maximum advertisement interval in seconds, or 0 for default
R5 = pointer to a list of router/preference pairs, terminated by a 0 word. A pointer of 0 will mean that the address of the interface will be used, however the interface must be present for this to function.

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to allow the RouterDiscovery module to issue advertisements of router addresses on an interface. Advertisements will be sent regularly, as specified, or when the interface changes state.

Interface names will not be validated unless the pointer in R5 is 0. Absent interfaces will cause the module to become quiescent until the interfaces become available.

Related SWIs

SWI RouterDiscovery_Control 2 (on page 414)

RouterDiscovery_Control 2
Deactivate
(SWI &57D80)

Deactivate control of interface

On entry

R0 = 2 (reason code)
R1 = Pointer to zero terminated interface name to deactivate

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to stop an interface being monitored by the
RouterDiscovery module.

An interface being killed will not implicitly cause this to happen in
order that interfaces can be restarted without affecting the operation
of RouterDiscovery.

Related SWIs

RouterDiscovery_Status
(SWI &57D81)

Return information about the RouterDiscovery module

On entry

R0 = Reason code (none defined)

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is not implemented.

Related APIs

None

Document information

Maintainer(s):	Gerph <gerph@gerph.org>			
History:	**Revision**	**Date**	**Author**	**Changes**
	1	2006	Gerph	**Initial version**
				● Released as RISC OS Select documentation.
	2	09 May 2022	Gerph	**PRM-in-XML conversion**
				● Created from original documentation for RISC OS Select.
Related:	http://www.riscos.com/support/developers/riscos6/ networking/routerdiscovery.html			
Disclaimer:	© Gerph, 2022.			

DHCPClient

Introduction

The DHCPClient module provides an implementation of the 'Dynamic Host Configuration Protocol'. This allows a server to allocate addresses to a client based on its internal ethernet 'MAC' address.

The module is able to control multiple interfaces simultaneously. Information about the DHCP configuration process is recorded to the DHCP log.

Service calls

Service_InternetStatus 4
BootPReply
(Service Call &B0)

Response received for BootP/DHCP request

On entry

R0 = 4 (sub-reason code>
R1 = &B0 (reason code)
R2 = Pointer to zero-terminated interface name
R3 = Pointer to Device Information Block for interface
R4 = Pointer to BootP/DHCP reply message buffer
R5 = Size of BootP/DHCP reply

On exit

R0 preserved
R1 = 0 to claim service, else preserved
R2 - R5 preserved

Use

This service call is issued by the Internet module (version 5.28 or later) when a BOOTP/DHCP reply is received. Clients may inspect the contents of the buffer to extract any configuration information.

If you want to alter information in the buffer you may do so, but you must then claim the service call by setting R1 to zero on exit. If the service call is claimed the Internet module will reprocess the buffer as if it had just arrived from the network. Another Service_InternetStatus 4 will arrive in due course.

You should not normally claim this service call.

Related APIs

None

DHCPOffer has been received

On entry

R0 = 5 (sub-reason code>
R1 = &B0 (reason code)
R2 = Pointer to zero-terminated interface name
R3 = Pointer to Device Information Block for interface
R4 = Pointer to DHCPOFFER message buffer
R5 = Size of DHCPOFFER message

On exit

R0 preserved
R1 = 0 to claim service, else preserved
R2 - R5 preserved

Use

This service call is issued by the DHCPClient module whenever it receives an offer of an IP address lease which is better than its current best choice (or if it is the first acceptable offer). You may inspect the buffer, but it must not be modified.

If clients choose to retain information about offers they MUST use the value of OPTION_SERVERIDENTIFIER as an opaque key to identify which offer has been chosen.

If you claim this service the DHCPClient module will not accept the offer, but will wait for another offer to be made.

You should not normally claim this service call.

Related APIs

None

Service_InternetStatus 48
DHCPLeaseGained
(Service Call &B0)

DHCP address has been configured on an interface

On entry

R0 = 48 (sub-reason code>
R1 = &B0 (reason code)
R2 = Pointer to zero-terminated interface name
R3 = IP address assigned to interface

On exit

R0 - R3 preserved

Use

This service call is issued by the DHCPClient module (after 0.37)
when it has successfully configured an interface with an address
leased from a DHCP server. If the interface is reconfigured, the
server releases the lease, the server fails to renew the lease, a
duplicate address is identified, or the network stack is stopped, the
lease will be lost and InternetStatus_DHCPLeaseLost will be issued.
This service will not be reissued for renewals of the lease.

This service should never be claimed.

Related services

Service_InternetStatus 49
DHCPLeaseLost
(Service Call &B0)

DHCP address has been removed from an interface

On entry

R0 = 49 (sub-reason code>
R1 = &B0 (reason code)
R2 = Pointer to zero-terminated interface name
R3 = IP address that was assigned to interface

On exit

R0 - R3 preserved

Use

This service call is issued by the DHCPClient module (after 0.37) when it has lost the DHCP server leased address allocated to an interface. A new address may be established by the DHCPClient if the server responds, or the interface may be manually reconfigured (however, this service may have been issued because of a manual reconfiguration). See Service_InternetStatus 48 (on page 420) for details of circumstances in which this service will be issued.

This service should never be claimed.

Related services

Service_InternetStatus 49 (on page 421)

SWI calls

DHCPClient_Control (SWI &55E00)

Controls the DHCPClient interface management

On entry

R0 = Reason code:

Value	Meaning
0	Add interface
1	Remove interface
2	Renew lease/re-try obtaining a lease on an interface

R1 = Pointer to zero-terminated interface name

On exit

R0 - R1 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to add or remove an interface from the DHCPClient's control. Once placed under the control of the DHCPClient the interface will continue to operate according to the DHCP protocol until either the interface is configured manually or it is removed from the module's control by being reconfigured.

Related * commands

DHCPClient_State
(SWI &55E01)

Reads the status of a DHCPClient managed interface

On entry

R0 = Pointer to zero-terminated interface name
R1 = Pointer to a list of information types as words, terminated by -1.
Information types:

Value	Meaning
0	Interface state (1 word):

Value	Meaning
0	sleeping
1	initreboot
2	init
3	rebooting
4	selecting
5	requesting
6	bound
7	renewing
8	rebinding

1	Bound address - 'yiaddr' (1 word)
2	Server address - 'siaddr' (1 word)
3	Gateway address - 'giaddr' (1 word)
4	lease period in centiseconds (1 word)
5	T1 period in centiseconds (1 word)
6	T2 period in centiseconds (1 word)
7	DHCP start (8 bytes; 5 bytes time, 3 bytes padding)
8	Lease start (8 bytes; 5 bytes time, 3 bytes padding)
9	Lease end (8 bytes; 5 bytes time, 3 bytes padding)
10	T1 end (8 bytes; 5 bytes time, 3 bytes padding)
11	T2 end (8 bytes; 5 bytes time, 3 bytes padding)

R2 = Pointer to buffer for returned data
R3 = Size of the output buffer

On exit

R0 preserved
R1 = Pointer to invalid option, or -1 if all types are valid
R2 = Pointer to first free byte in the output block
R3 = Space left, or negative space needed if data would not fit

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to read the current DHCP client state for an interface. R1 points to a list of types which will be returned in the output buffer in the order in which they were supplied. If the block was not large enough, a 'Buffer overflow' error will be returned, with R3 set to the -ve size required. If the type of information requested was invalid, an error will return and R1 will point to the invalid entry.

Related * commands

DHCPClient_Enumerate
(SWI &55E02)

Enumerates names of interfaces controlled by DHCPClient

On entry

    R0 = Pointer to zero-terminated interface name of the last interface, or 0 initially
    R1 = Pointer to buffer for returned data
    R2 = Size of the output buffer

On exit

    R0 = Number of state transitions
    R1 = pointer to buffer on entry, or 0 if no interfaces remain
    R2 = Space left, or negative space needed if data would not fit

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI enumerates the interfaces under DHCPClient control.

Related SWIs

SWI DHCPClient_State (on page 423)

*Commands

<div align="right">

*DHCP

</div>

Modify the DHCP control of an interface

Syntax

*DHCP [-+] *<interface>*

Parameters

<interface> - Name of the interface to change management of.

Use

This command is used to control whether the DHCPClient module will configure the network automatically using the DHCP protocol.

If no prefix is applied to the interface name the interface will be added to the list of those controlled by the DHCP module.

If a '-' prefix is used, the interface name will be removed from those controlled by the DHCP module and any address which is in use will be removed.

If a '+' prefix is used, an existing DHCP lease on that interface will be renewed, or a new attempt to obtain a lease will be made.

Examples

*DHCP eh0

Related APIs

None

*DHCPStatus

Display information on DHCP controlled interfaces

Syntax

```
*DHCPStatus
```

Parameters

None

Use

This command is used to display information about interfaces controlled by the DHCPClient module.

Examples

```
*DHCP eh0
```

Related * commands

*ShowStat

Document information

Maintainer(s):	Gerph <gerph@gerph.org>			
History:	**Revision**	**Date**	**Author**	**Changes**
	1	2006	Gerph	**Initial version** ● Released as RISC OS Select documentation.
	2	30 Dec 2021	Gerph	**PRM-in-XML conversion** ● Created from original Select documentation
Related:	http://www.riscos.com/support/developers/riscos6/ networking/dhcpclient.html			
Disclaimer:	© Gerph, 2021.			

Chapter Title

Introduction

The ZeroConf module deals with Link-Local zero-configuration network address assignment. This module has been present since Select 3. The implementation follows that of RFC3927. The module can only handle a single interface. It will be automatically configured by the InetConfigure module when 'Dynamic' network addressing is configured.

The ZeroConf module will always configure alias 9 of an interface, for example 'eh0:9'.

Conformance

The ZeroConf module and other components of the stack follow the protocol laid down by this RFC with certain caveats:

- Link-local addresses assigned to interfaces with routable addresses will continue to be advertised by the Internet stack through the SIOCGIFCONF interface. (1.9 rule 2)
- No operational changes have been made to prevent the issuing of link-local packets to a router, or forwarding by a router if so configured. (2.6.2, 2.7, 7)
- The use of subnetting is not prevented. (2.8)
- DNS addresses supplied by external sources may be cached for link-local addresses. (2.9)
- DNS server may provide locally known link-local addresses. (2.9)
- Operation where multiple interfaces use link-local addresses is not supported by the ZeroConf module and, where manual configuration occurs is not expected to route correctly. (3)

Select 3 ZeroConf implementation follows draft 7 of the link-local standard and had the following differences from the released RFC:

- 4 probes will be sent initially (RFC now states 3).
- The maximum number of conflicts before rate limiting was 60 (RFC now states 10).

Service calls

Service_InternetStatus 32 ZeroConfAddressAcquired (Service Call &B0)

Address has been acquired by the ZeroConf module

On entry

R0 = 32 (sub-reason code>
R1 = &B0 (reason code)
R2 = Pointer to zero-terminated aliased interface name
R3 = IP address assigned to the interface

On exit

R1 - R3 preserved

Use

This service call is issued by the ZeroConf module when it has successfully configured an interface with a link-local address. This address may be used just like any other address. This address may be changed (and the appropriate services issued) if collisions occur or if manually modified.

This service should never be claimed.

Related services

Service_InternetStatus 33 (on page 431)

Service_InternetStatus 33
ZeroConfAddressLost
(Service Call &B0)

Address has been lost by the ZeroConf module

On entry

R0 = 33 (sub-reason code>
R1 = &B0 (reason code)
R2 = Pointer to zero-terminated aliased interface name
R3 = IP address that was assigned to the interface

On exit

R1 - R3 preserved

Use

This service call is issued by the ZeroConf module when it has lost the link-local address allocated to an interface. A new address may be reestablished by the ZeroConf module if the reason for the address loss was due to a collision.

This service should never be claimed.

Related services

Service_InternetStatus 32 (on page 430)

SWI calls

ZeroConf_Control
(SWI &56A00)

Controls the ZeroConf interface management

On entry

R0 = Reason code:

Value	Meaning
0	Places an interface under management by ZeroConf (on page 433)
1	Releases an interface from management by ZeroConf (on page 434)

R1 - R8 = Dependant on reason code

On exit

R0 - R8 = Dependant on reason code

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to control the operation of the ZeroConf module.

Related APIs

None

ZeroConf_Control 0
ZeroConfAddInterface
(SWI &56A00)

Places an interface under management by ZeroConf

On entry

R0 = 0 (reason code)
R1 = Pointer to zero-terminated interface name

On exit

R0 - R1 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to add an interface to those that the ZeroConf module controls. Only a single interface can be controlled by the ZeroConf module. An error will be returned if the interface cannot be added.

Related APIs

None

ZeroConf_Control 1
ZeroConfRemoveInterface
(SWI &56A00)

Releases an interface from management by ZeroConf

On entry

R0 = 0 (reason code)
R1 = Pointer to zero-terminated interface name

On exit

R0 - R1 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to remove an interface from those that the ZeroConf
module controls. If the interface named is not controlled by
ZeroConf, an error will be returned.

Related APIs

None

ZeroConf_Status
(SWI &56A01)

Reads the status of the ZeroConf module

On entry

R1 = Status type:

Value	Meaning
0	Reads the current configuration status (on page 436)

R1 - R8 = Dependant on reason code

On exit

R0 - R8 = Dependant on reason code

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to read the status of the ZeroConf module.

Related APIs

None

ZeroConf_Status 0 ConfigurationState (SWI &56A01)

Reads the current configuration status

On entry

R1 = 0 (reason code)

On exit

R0 = State of the ZeroConf module

Value	Meaning
0	idle
1	probing for address
2	announcing address assignment
3	configured
4	configured, defending against address collision

R1 = Pointer to zero-terminated interface name

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to read the state of the operation of the ZeroConf module.

Related APIs

None

Document information

Maintainer(s):	Gerph <gerph@gerph.org>			
History:	**Revision**	**Date**	**Author**	**Changes**
	1	2006	Gerph	**Initial version** ● Released as RISC OS Select documentation.
	2	30 Dec 2021	Gerph	**PRM-in-XML conversion** ● Created from original Select documentation
Related:	[http://www.riscos.com/support/developers/riscos6/ networking/zeroconf.html](http://www.riscos.com/support/developers/riscos6/networking/zeroconf.html)			
Disclaimer:	© Gerph, 2021.			

Graphics Mode Specification

Introduction and Overview

Graphics modes can be specified in a number of ways, which have been added to with each iteration of the Operating System. Originally only mode numbers were allowed, but hardware improved and more flexibility was required, so the mode specification was extended.

Graphics modes may be supplied to a number of interfaces, most of which will eventually come down to a call to OS_ReadModeVariable. Some of the interfaces that you may find using mode specifications are:

Interface	Usage
OS_ReadModeVariable	Read values for a given mode.
OS_CheckModeValid	Return whether the mode specified can be selected.
OS_ScreenMode	Operations on the graphics mode
ColourTrans_*ForMode	Colour operations for a given mode
OS_SpriteOp	Sprite creation operations may be supplied modes
Sprite Header	Defines the type of data within a sprite
Wimp_SetMode	Selects the mode used by the Window Manager

Technical details

Mode specifiers

Mode specification is always through a single 32bit word value
known as a mode specifier. This allows it to be supplied in many of
the places that a mode number was used in earlier interfaces. This
mode specifier can represent a number of ways of describing a mode.
The following mode specifier formats are defined:

- Mode number
- Sprite mode word
- Sprite pointer
- Mode selector

These can be distinguished by the following algorithm:

- If the mode specifier is < 256:
    - This is a mode number, and shadow bank
      selection.
    - The mode number is in the low 7 bits, and shadow
      bank selection is given in bit 7.
    - If the mode number is not recognised
      Service_ModeExtension is issued to determine the
      mode's parameters.
    - Modes up to 7 are supported from the BBC
      onwards.
    - Shadow modes are supported from the Master
      onwards (although they are less reliable from
      RISC OS 3.6 onwards)
- If the mode specifier has bit 0 set, this is a sprite mode
  word:
    - Sprite mode words are given in the sprite header,
      but may also be supplied to many of the mode
      functions (except for display selection).
    - They only contain the DPI (and thus eigenfactors),
      and type of data within the sprite. No resolution
      information is available.
    - Sprite mode words are supported from RISC OS
      3.5 onwards.
- If the mode specifier has bit 0 and 1 clear, this is a pointer
  to data, whose meaning is differentiated by the value of the
  first word.
    - If the first word has bit 0 clear, the data is a sprite
      (the mode specifier is a sprite pointer):
        - Sprite pointers allow information about
          the width and height to be included in
          the information, and allow the use of
          palette data as well. These types of mode
          specification are usually only used with

ColourTrans_*operations.
- Sprite pointers are supported from RISC OS 3 onwards.
- If the first word has bit 0 set, the data is a mode selector:
    - Mode selectors expose the base specifications for the mode and modifications to mode variables.
    - Mode selectors allow for extended formats, but only a single format is currently defined.
    - Mode selectors are supported from RISC OS 3.5 onwards.
- If the value has bit 0 clear and bit 1 set, this is an invalid mode specification.

Mode numbers

Mode numbers may be extended through the Service_ModeExtension interface. This allows new numbered modes to be defined, either completely or based on other modes.

Sprite mode words

Sprite mode words allow some of the parameters of the mode to be determined, but because they do not include resolution information they cannot be selected. Sprite mode words are only supported from RISC OS 3.5 onwards.

The sprite mode word format has undergone a few revisions. The current definition of the sprite mode word is:

Bit(s)	**Meaning**
0	Set (indicator that this is a new format sprite, together with set bits in bits 27-31)
1-13	Horizontal dots per inch, should be 180, 90, 45, 23/22, 11
14-26	Vertical dots per inch, should be 180, 90, 45, 23/22, 11
27-30	Sprite type :

Value	**Meaning**
0	Old format mode word (mode is a standard number)
1	1 bpp
2	2 bpp
3	4 bpp
4	8 bpp
5	15 bpp in 16bit values
6	24 bpp in 32bit values
7	CMYK
8	24 bpp compact format (allocated but not used)
9	JPEG data (allocated but not used)
10-15	Reserved

31	Set:	Alpha channel data present. May not be set for type 0 sprites
	Clear:	Binary mask data present

For sprite types 1-4, the palette is only supported from RISC OS 3.6 onwards.

Although the DPI value should be the values defined above, values outside these may be supported. Certain interfaces, such as PlotSpriteTransformed, may use this information to render the sprites to the correct size for the display. Other interfaces, such as OS_ReadModeVariable and PlotSpriteScaled, may quantise these DPI values to the closest eigenfactor.

CMYK format sprites are supported from Select 2 onwards. JPEG data has been supported by third party extensions.

Compatibility

RISC OS < 3.5
    Does not support sprite mode words.

RISC OS ≥ 3.5
    Sprite types 0 to 6 supported, but does not support palettes on types 1-4.

RISC OS ≥ 3.6
    Supports palettes on sprite types 1-4.

RISCOS Ltd   RISC OS ≥ Select 2
    Sprite types 0-6 and 7 (CMHK) supported.

RISCOS Ltd   RISC OS ≥ Select 3
    Supports alpha channel data in addition to the types supported by Select 2.

RISC OS Pyromaniac   RISC OS ≥ 7.19
    Sprite types 0-6 supported.

Gerph   JPEGSprites
    Adds support for sprite type 9 (JPEG) to those supported by RISC OS 3.5.

Mode selectors

A mode selector is a word-aligned structure that defines the properties of a mode. This includes its resolution, numbers of colours, frame rate and other mode variables.

A mode selector has the following format:

Offset	Contents
+0	mode selector flags:

Bit(s)	Meaning
0	1 (this differentiates it from a sprite pointer)
1-7	mode specifier format (0 for this format)
8-31	other flags (reserved - must be zero)

Offset	Contents
+4	x-resolution (in pixels)
+8	y-resolution (in pixels)
+12	colour data format and depth:

Value	Meaning
0	1 bpp
1	2 bpp
2	4 bpp
3	8 bpp
4	15 bpp in 16 bit values
5	24 bpp in 32 bit values

Offset	Contents
+16	frame rate (in Hz); -1 => use highest rate available
+20	pairs of [mode variable index, value] words; there may be any number of these, including zero
+n	-1 (terminator)

Mode variables may be given in any order, although it is recommended that they be supplied in ascending order. Repeating a variable definition has undefined behaviour.

Compatibility

RISC OS < 3.5
Does not support mode selectors.

RISC OS ≥ 3.5
Supports mode selectors as described.

RISC OS Pyromaniac   RISC OS ≥ 7.19
Supports mode selectors as described.

Mode strings

To allow modes to be described within a string specification, a mode string is able to be supplied to various interfaces. Mode strings must be converted to a mode specifier before they can be used with many interfaces. OS_ScreenMode allows these mode strings to be converted to and from mode specifiers.

The mode string takes the form of a space or comma separated list of parameters. Each parameter is a sequence of alphabetic characters defining the parmeter, followed by a number sequence and possible qualifiers.

The mode string parameters have the following format:

Parameter	Meaning
X#	X resolution in pixels
Y#	Y resolution in pixels
C#	Number of colours (# = 2, 4, 16, 64, 256, 32T, 32K, 16M)
G#	Number of greys (# = 4, 16, 256)
T#	Teletext mode, with specified number of colours (# as C)
EX#	X eigen factor (# = 0, 1, 2, 3)
EY#	Y eigen factor (# = 0, 1, 2, 3)
F#	Frame rate in Hz
TX#	Teletext display width in characters
TY#	Teletext display height in characters

Up to RISC OS Select 3, the X and Y resolution must be values from 100-9999. From Select 3 inwards, any value other than 0 may be supplied, although support for resolutions above 16384 may not be reliable.

Teletext mode selection and character width/height is supported from RISC OS Select 3 onwards.

Selection of modes with 64 colours results in an old-style VIDC 1 mode selection of a 256 colour mode with 192 derived colours. Prior to Select 3, selection of 'C256' would result in a the old-style VIDC 1 mode being selected.

The OS_ScreenMode interface for converting and selecting mode strings is supported from RISC OS Select 3 onwards.

*WimpMode supports selecting mode strings from RISC OS 3.5 onwards.

Compatibility

RISC OS ≥ 3.5
> Supports mode string specifications X, Y, C, G, EX, EY and F, but only through '*WimpMode'.

RISCOS Ltd   RISC OS ≥ Select 3
> Supports specifications for T, TX and TY in addition to those supported by RISC OS 3.5. C256 will select a full 256 colour palette, whilst C64 will support a VIDC 1 palette.
> OS_ScreenMode mode string processing supported.

RISC OS Pyromaniac   RISC OS ≥ 7.19
> Supports specifications for T, TX and TY in line with RISC OS Select 3. C64 will emulate old VIDC 1 palettes. OS_ScreenMode mode string processing supported.

Document information

Maintainer(s):	Gerph <gerph@gerph.org>			
History:	**Revision**	**Date**	**Author**	**Changes**
	1	22 Nov 2020	Gerph	**Initial version**
				● Created from PRM and Select technical documentation.
	2	19 May 2023	Gerph	**Compatibility with Pyromaniac**
				● Added details about the compatibility of the interfaces with RISC OS Pyromaniac and RISC OS Select.
Disclaimer:	© Gerph, 2020-2023.			

The Image File Renderer

Introduction and Overview

A number of graphics formats are supported natively by RISC OS. JPEG, DrawFiles and Sprites are directly renderable, and PNGs are supported through a number of conversion calls. Each of these formats, however, is rendered using slightly different calls. The ImageFileRender module simplifies rendering these (and potentially other third party) image files.

All graphics formats have two things in common :

- They cover a region (even empty files must say what space they cover).
- They have a resolution at which they are drawn.

The region they cover is known as the 'bounding box'. For many graphics formats, this will be aligned with the origin - for example a bitmap graphic. For others, this bounding box may be elsewhere in the image - for example vector formats such as DrawFiles.

The resolution at which they have been drawn describes how accurately the images is stored. Usually this is stored in 'dots per inch' (DPI) along with the image itself. Screen resolution is usually - this depends on the eigenfactors for the screen mode in use - treated as 90 DPI. Some formats may use much more accurate internal representations than this; for example DrawFiles are stored at 2048 DPI.

For the purposes of rendering the image file, we ignore the colour depth because the rendering process will generate its results in the most accurate manner possible for output depth.

Images may be rendered using a number of transformation types, allowing them to be rendered to fit a region, to a scale, or using a more general transformation.

Within each image file there may be a number of individual images. These can be accessed by a sequence number which indicates their logical location within the file. The images may be related - as would be the case with frames of an animation - or they may be unrelated - as would be the case with a collection of resources.

When accessing images, additional information may be provided to the renderer which may perform specific operation on the image.

This extra data is specific to the renderer and cannot be handled generically.

Technical Details

Sequence numbers

Graphics files may contain multiple logical images which may either be frames of an animation, alternate versions, or other image resources. These images are accessed through a sequence number which must be supplied to all images. A sequence number of 0 will render the 'default' image within the file. This may be the first image in some formats, the last in others, or some arbitrary image. A sequence number higher than that of the last image should be treated as the last image. A sequence number of 1 indicates the first image should be processed.

Rendering quality

Image files may contain data which is more accurate than can be represented by the display. This is usually the case for bitmap images at high colour depths and almost always the case for bitmap images. In order to allow some control over the quality of the rendered image (and usually the rendering speed) a 'quality' parameter can be provided to the renderer. This is a value from 1 to a renderer specific limit (with a maximum of 15) and will be bounded to the maximum that the renderer supports. Thus, if the highest quality is required, a value of 15 should be supplied. If the lowest quality is required, a value of 1 should be supplied. In the majority of cases, however, the 'default' will be required. This is a value which the renderer feels is suitable for most operations and does not require excessive processing to complete. To request the default quality, a value of 0 should be specified as the quality.

Transformation types

Graphics files may be transformed in a number of ways. This allows us to provide a simpler interface for rendering based on the requirements of the application. At present, there are three transformation types provided by the module:

Value	Meaning
0	Render to fit
1	Render scaled
2	Render transformed

For all rendering types an x and y origin are supplied from which all operations will be based. This allows the same details to be used for the fit, scale or transform regardless of the images location on the screen.

Render to fit

When rendering to fit, a width and height must be supplied by the application. The image file will be scaled to fit within this region. In addition, a border and angle may be provided to specify an area around the image which should be left clear, and to specify the angle through which the image should be rotated.

Rotation is performed anti-clockwise. The centre of the rotation is not strictly relevant to this operation because the image is always scaled to fit the width and height supplied.

The 'fit' block has the following structure:

Offset	Contents
+0	width (in OS units)
+4	height (in OS units)
+8	border (to apply to all edges)
+12	angle (in degrees clockwise, as a 16.16 fixed point value)

As the shape is scaled to fit the size specified, the point about which rotation occurs is not important. It can be considered to be the centre of the image.

Render scaled

When rendering scaled, a pair of multiplication and division factors should be supplied which describe the scale at which the image should be rendered. The scale block is a standard RISC OS scale block (as used by SpriteExtend)

The scaling block has the following structure :

Offset	Contents
+0	X multiplication factor
+4	Y multiplication factor
+8	X division factor
+12	Y division factor

Render transformed

Rendering images through a transformation matrix is the most flexible method of rendering that the ImageFileRender module provides. Transformation matrices are provided in standard RISC OS tranformation blocks (as used by SpriteExtend, Draw, DrawFile and others).

The transformation has the following structure:

Offset	Contents
+0	m00
+4	m10
+8	m01
+12	m11
+16	m20
+20	m21

where the matrix is constructed:

{ m00, m01, 0 }

{ m10, m11, 0 }

{ m20, m21, 1 }

m00, m01, m10, and m11 are 16.16 fixed point values.

m20 and m21 are 24.8 fixed point values.

Arbitrary transformations

Not all image formats support arbitrary transformations. Because of this, certain formats will be unable to render when a complex transformation is in use. A typical example of such limitations is that of JPEGs. The internal renderer can only render JPEGs as a scaled object. If rotation, or other complex transformations are applied to files which are not capable of those transformations, an error will be returned.

Clipping

All images will be clipped to the standard graphics rectangles. If an image must not pass outside a region, a graphics window should be used. This can be set through a VDU 24 sequence.

Image file origins

Whilst most images are based at the origin, some images will have a bounding box which are not. When the image is rendered 'to fit', the image origin is implicitly ignored. When scaling and transforming however, the origin is maintained and will be scaled with the image itself. Because this can make manipulating such images more complex, this origin offset can be negated by the ImageFileRender module. In this mode, the image can be treated as if it does not have any offset from the origin.

Colour mapping

In order to provide highlighting and other colour manipulation on the image, the ImageFileRender module can use colour mapping functions (as used by SpriteExtend, DrawFile, and ColourTrans). These allow the colours in the image to be manipulated to provide effects such as highlighting or shading.

Extensions for more complex colour mapping

The operations that can be provided in a generic manner by the ImageFileRender module do not cover the full range of operations that might be applied to every image file format. Because of this, extension data may be provided which is specific to the renderer in use. Because each renderer may provide specific data to enable it to render images, and there may be multiple providers of rendering facilities, a 'magic' identifier is allocated to each renderer. This is ensures the the renderer is not given data in a form which it does not understand.

Where a magic identifier is supplied and a suitable renderer is available, it will be used. If no suitable renderer can be found, the last registered renderer will be used. This ensures that the where extension data is used it is passed to the appropriate renderer, and falls back to using the most recent renderer installed.

The extension data block must be word aligned, and the first word contains the magic identifier for the render that it is intended for. The remainder of the extension data block is specific to the renderer in use.

The magic identifier may be any 32bit value, but we recommend that these are registered with RISCOS Ltd to ensure that there are no duplicated identifiers. At present, allocations are of the form &6699ccii, where cc indicates the company or individual producing the renderer, and ii is some image format number at the company or individual's discretion.

Sprite file extensions

When rendering sprite files, by default the first sprite is rendered from the file. This covers the majority of the situations that it will be required, but where different sprites are required, the extension block describes which to use. The identifier for the RISCOS Ltd sprite renderer is &66990101. The named sprite will only be used when the sequence number is left as 'default'.

Offset	Contents
+0	&66990101
+4	Sprite name, up to 12 characters

Renderers

Custom renderers

Custom renderers may be registered with the ImageFileRender module. These renderers can provided additional rendering facilities for third party filetypes, or provide additional facilities over those of the standard renderers.

Renderers have four components:

1. A routine which calculates the bounding box and resolution of an image
2. A routine which renders an image
3. A routine which declares fonts in a document (may be omitted)
4. A routine which returns information about an image

In addition, they provide a number of informational fields which describe the renderer's capabilities:

- The filetype that the renderer applies to
- The name of the renderer (including the version and author)
- A flags word that describes the renderers capabilities
- The renderers 'magic' identifier (or 0 if it provides no special operations)

Renderer name

The renderer name provides details about the renderer in order that diagnostics may be performed and information about the installed renderers is available. The renderer name consists of three, tab (ASCII 9) separated, fields:

- The renderer name
- The version number in the form x.xx
- The authors (or publishers) name

Renderer flags

Not all renderers have the same capabilities, as stated earlier. The flags provide details to ImageFileRender of the capabilities of the renderer. This is a bit field, structured:

Bit(s) Meaning

0-1 Renderer transformation capabilities:

Value Meaning

0 Renderer cannot draw anything but identity scaling and translation

1 Renderer can translate and scale, but scaling must be by identical factors

2 Renderer can translate and scale by any values in both axes

3 Renderer supports any form of transformation

These bits should be set to the capabilities of the renderer.

Attempts to render files of which the capabilities word indicates are not possible by the renderer will be faulted by ImageFileRender module.

2 Renderer supports colour mapping.

This bit should be set if the renderer can perform colour mapping. If unset, attempts to use colour mapping on this file type will be faulted by ImageFileRender.

3 Renderer can draw irregular shapes so must be called to calculate bounding boxes.

This bit should be set if transforming a shape using a complex matrix (eg skew or rotate) may result in a different bounding box than that which would be generated for a rectangular area. If unset, the renderer will be called to calculate the bounds of an identity transform only. ImageFileRender will perform the remainder of the calculations.

If a renderer can only render rectangular areas then leaving this bit clear simplifies the implementation.

4-7 Maximum number of 'quality' levels supported (1-15).

The highest quality level which is supported by the renderer.

If the quality level requested by a client exceeds this, the renderer will be called with this value.

8-11 Default 'quality' level to use (1-15).

Where quality settings are omitted (ie when 'default' quality

Bit(s)	**Meaning**
	is selected) the default quality will be passed to the renderer. A value of 0 means that quality levels are ignored.

Service calls

Service_ImageFileRender_Started (Service Call &80D40)

ImageFileRenderer has initialised

On entry

R0 = API version (102 at present)
R1 = &80D40

On exit

None

Use

This service is issued after the ImageFileRender module has initialised. Renderers should register themselves with the module.

Related services

Service_ImageFileRender_Dying (on page 457)

Service_ImageFileRender_Dying
(Service Call &80D41)

ImageFileRenderer about to finalise

On entry

R0 = API version (102 at present)
R1 = &80D41

On exit

None

Use

This service is issued as the ImageFileRender module finalises to
notify clients that it is no longer providing rendering facilities.

Related services

Service_ImageFileRender_Started (on page 456)

Service_ImageFileRender_RendererChanged (Service Call &80D42)

A renderer has initialised or finalised

On entry

R0 = API version (102 at present)
R1 = &80D42
R2 = Filetype affected

On exit

None

Use

This service is issued when a renderer registers or deregisters with the ImageFileRender module. Clients which have cached details of other renderers should re-read any renderer values necessary after checking whether the filetype matches those which they are interested in.

Related SWIs

SWI ImageFileRender_Register (on page 471)

SWI calls

ImageFileRender_Render
(SWI &562C0)

Render an image

On entry

R0 = Rendering flags:

Bit(s)	Meaning
0-2	Transformation type:

Value	Meaning
0	Render to fit
1	Render scaled
2	Render transformed
3-7	Reserved

Bit(s)	Meaning
3	Colour mapping function supplied
4	Ignore document origin
5	Reserved, must be zero
6-9	Quality to render at:

Value	Meaning
0	Use default quality
1	Lowest quality
2-14	Renderer specific values
15	Highest quality

Bit(s)	Meaning
10-16	Reserved, must be zero
17-31	Reserved, must be zero

R1 = Filetype
R2 = Pointer to data to render
R3 = Length of data
R4 = Pointer to extension data, or 0 if none
R5 = Image sequence number, or 0 for default image
R6 = X coord for origin
R7 = Y coord for origin
R8 = Transformation data:

Value	Name	Transformation type in R0
0	Pointer to size	Image file origin is ignored
1	Pointer to scale block	

Offset	Contents
+0	X mult
+4	Y mult
+8	X div
+12	Y div

Value	Name	Transformation type in R0
2	Pointer to transformation matrix	Standard draw transformation matrix format

R9 = Pointer to colour map descriptor

On exit

None

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is re-entrant

Use

This SWI is used to render an image file.

Related entry points

[IFR_Render (on page 493)](#)

ImageFileRender_BBox
(SWI &562C1)

Calculates an image's bounding box

On entry

R0 = Rendering flags:

Bit(s)	Meaning
0-2	Transformation type:

Value	Meaning
0	Render to fit
1	Render scaled
2	Render transformed
3-7	Reserved

Bit(s)	Meaning
3	Reserved, must be zero
4	Ignore document origin
5	Return in OS units (otherwise bounding box will be returned in draw units
6-31	Reserved, must be zero

R1 = Filetype
R2 = Pointer to data to render
R3 = Length of data
R4 = Pointer to extension, or 0 if none
R5 = Image sequence number, or 0 for default image
R6 = Pointer to transformation data (see above)
R7 = Pointer to bounding box to fill in

On exit

None

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is re-entrant

Use

This SWI is used to calculate the bounding box for a transformation operation.

Related entry points

ImageFileRender_Transform
(SWI &562C2)

Return transformation matrix for render operation

On entry

R0 = Rendering flags:

Bit(s)	Meaning
0-2	Transformation type:

	Value	Meaning
	0	Render to fit
	1	Render scaled
	2	Render transformed
	3-7	Reserved

Bit(s)	Meaning
3	Reserved, must be zero
4	Ignore document origin
5-31	Reserved, must be zero

R1 = Filetype
R2 = Pointer to data to render
R3 = Length of data
R4 = Pointer to extension, or 0 if none
R5 = Image sequence number
R6 = Pointer to transformation data
R7 = Pointer to output transformation block to fill in

On exit

None

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is re-entrant

Use

This SWI is used to calculate the transformation matrix that would be used for an operation without performing that operation. Where clients wish to combine a transform matrix with the operation applied by the scaling specified, this call can obtain the transformation matrix which ImageFileRender will use.

Related APIs

None

ImageFileRender_DeclareFonts (SWI &562C3)

Declare fonts prior to printing

On entry

R0 = Flags (reserved, must be 0)
R1 = Filetype
R2 = Pointer to data to render
R3 = Length of data
R4 = Pointer to extension data, or 0 if none
R5 = Image sequence number
R6 = Flags to pass to PDriver_DeclareFont

On exit

R0 - R6 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is re-entrant

Use

This SWI should be used when printing images using the
ImageFileRender module before any printing operations begin. Refer
to the section 'Declare the fonts your document uses' in the chapter
on Printing for more details.

Related entry points

IFR_DeclareFonts (on page 497)

ImageFileRender_Info
(SWI &56264)

Discover miscellaneous image information

On entry

R0 = Flags (reserved, must be 0)
R1 = Filetype
R2 = Pointer to data to render
R3 = Length of data
R4 = Pointer to extension data, or 0 if none
R5 = Image sequence number
R6 = Query type:

Value	Meaning
&00000000	Base details
&00000001-&00000FFF	Reserved for system use
&00001000-&0000FFFF	Reserved for developers
&00FF0000-&00FFFFFF	Reserved for private use

Others are reserved for future expansion.
R7 = Pointer to query block
R8 = Length of query block

On exit

R8 = If successful, R8 returns the length of block used.

If the block was too small, R8 returns a negative value showing how much space was required. If another error occurs, R8 will be positive.

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is re-entrant

Use

This SWI should be used to find out information which is not provided by the generic APIs. It may be used (for example) to read the time between frames for a custom renderer, or to read additional information about the image which would otherwise not be available.

The base details query returns the following:

Offset	Contents
+0	Sequence number
+4	X DPI
+8	Y DPI
+12	Colour type:

Value	Meaning
0	Unspecified colour type (usually 'free' colour selection)
1	1bpp RGB
2	2bpp RGB
3	4bpp RGB
4	8bpp RGB
5	16bpp RGB
6	24bpp RGB
7	CMYK
Others	Reserved

Offset	Contents
+16	Image flags:

Bit(s)	Meaning
0	If set, the image is solid and covers the entire bounding box described. If clear, the image may have sections which reflect the background colour.
1-31	Reserved, must be zero

The base query is used to get generic information on an image in the file which was not necessary for the rendering of the file. This call is most commonly used to find the sequence number of the default and last logical image within a file. The sequence number may be set to &FFFFFFFF to indicate that the sequence number is not known. This might be the case if the format has no indication of the number of images present.

The image flags provide additional information about the image which might be useful to renderers. The only defined flag at present is that indicating if the image is 'solid' or not. This can be used by clients to decide whether drawing a background behind the image is neccessary or not.

Related entry points

ImageFileRender_RendererInfo
(SWI &56265)

Discover information on the renderer

On entry

R0 = Flags (must be 0)
R1 = Filetype
R2 = Magic identifier

On exit

R0 = Pointer to renderer definition block (read only)
R1 = Pointer to renderer name

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is re-entrant

Use

This SWI is used to return information about a renderer.

Related SWIs

SWI ImageFileRender_EnumerateRenderers (on page 474)

ImageFileRender_Register
(SWI &56266)

Register a renderer

On entry

R0 = Flags (reserved, must be 0)
R1 = Pointer to definition (all will be copied):

Offset	Contents
+0	API version (102 at present)
+4	[Renderer flags (on page 453)](#)
+8	Filetype
+12	Magic value, or 0 if none
+16	Pointer to name to be copied, in the format: `<name><tab><version x.xx><tab><author>`
+20	Workspace value for R12
+24	Pointer to start entry point ([IFR_Start (on page 489)](#))
+28	Pointer to stop entry point ([IFR_Stop (on page 491)](#))
+32	Pointer to render entry point ([IFR_Render (on page 493)](#))
+36	Pointer to bounding box entry point ([IFR_BBox (on page 495)](#))
+40	Pointer to declare fonts entry point ([IFR_DeclareFonts (on page 497)](#))
+44	Pointer to information entry point ([IFR_Info (on page 499)](#))

Or use 0 to get the current API version

On exit

R1 = API version (even if an error occurred)

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is re-entrant

Use

This SWI is used to register a new renderer.

Related SWIs

[SWI ImageFileRender_Deregister (on page 473)](#)

ImageFileRender_Deregister (SWI &56267)

Deregister a renderer

On entry

R0 = Flags (reserved, must be 0)
R1 = Filetype
R2 = Pointer to name used on registration
R3 = Magic value to match (must be the same as when registered

On exit

R0 - R3 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is re-entrant

Use

This SWI is used to deregister a renderer.

Related SWIs

SWI ImageFileRender_Register (on page 471)

ImageFileRender_EnumerateRenderers (SWI &56268)

Enumerate the active renderers

On entry

R0 = Flags (reserved, must be 0)
R1 = Last filetype, or -1 for first call
R2 = Magic value, or 0 for first call

On exit

R0 = Pointer to renderer definition block (read only)
R1 = Filetype of this renderer, or -1 if there are no more
R2 = Magic value of this renderer

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is re-entrant

Use

This SWI is used to enumerate the renderers which have been registered with the ImageFileRender module.

Related SWIs

Error Messages

Error_IFR_BadTransformType
(Error &81A800)

Bad transform type

Use

This error is returned when the transformation type specified is invalid.

Error_IFR_Reserved
(Error &81A801)

Reserved flags set for ImageFileRender SWI

Use

This error is returned when a SWI has been called with flags set
which have been defined as reserved. Where possible, this will be
returned to allow clients to use new features when they are available,
and to fall back to older methods where the features requested are
not avialable.

Error_IFR_ReservedRendererFlags (Error &81A802)

Reserved flags set for ImageFileRender renderer

Use

This error is returned during renderer registration when the flags specified in the renderer definition has flags set which have defined as reserved.

Error_IFR_Memory
(Error &81A803)

Not enough memory for ImageFileRender

Use

This error is returned when there is not enough memory for the rendering (or other) operation.

Error_IFR_NoSuchRendererToRemove (Error &81A804)

Renderer not known

Use

This error is returned when the renderer being deregistered is not known to the ImageFileRender module.

Error_IFR_NoRenderr
(Error &81A805)

No renderer registered for that filetype

Use

This error is returned when an operation is attempted on a filetype for which no renderer has been registered.

Error_IFR_BadAPI
(Error &81A806)

Bad API version

Use

This error is returned when an operation is attempted for which the
renderer API is not understood by the renderer. This will most likely
not be seen by external clients. Clients who proxy their rendering
through another renderer may see this if the APIs provided do not
match between the proxy and the client.

Error_IFR_CantTransform (Error &81A807)

Transformation type not supported by filetype

Use

This error is returned when the rendering operation cannot be performed because the renderer does not support the transformation requested by the client. The most likely cause for this error is attempted to skew or rotate a filetype which cannot be skewed or rotated (for example JPEGs).

Error_IFR_NoColourMap
(Error &81A808)

Colour mapping not supported by filetype

Use

This error is returned when a rendering operation cannot be performed because the renderer does not support colourmapping and colourmapping has been requested by the client.

Error_IFR_BadInfoQuery (Error &81A809)

Query type not recognised

Use

This error is returned when the ImageFileRender_Info query type has not been recognised by the renderer.

Error_IFR_BadInfoLength
(Error &81A80A)

Bad query length

Use

This error is returned when the ImageFileRender_Info query type has
been recognised by the renderer, but the length supplied was not
understood.

Error_IFR_BadSpriteMode (Error &81A810)

Bad sprite mode

Use

This error is returned by the sprite renderer when the image being rendered uses a mode which is not understood by the system.

Error_IFR_BadSpriteFile
(Error &81A811)

Sprite file corrupt or contains unrecognised data

Use

This error is returned by the sprite renderer when the image being rendered is malformed or contains data which is not understood.

Error_IFR_NoSuchSprite
(Error &81A812)

Sprite not found

Use

This error is returned by the sprite renderer when it cannot locate
the sprite named in the extension data.

Entry Points

Initialisation routine for ImageFileRender

On entry

R0 = API version * 100 (102 in this version)
R1 = Pointer to image descriptor:

Offset	Contents
+0	Pointer to data to render
+4	Length of data
+8	Pointer to extension data, or 0 if no data
+12	Image sequence number
+16	Private image data, 0 initially

R2 = 0
R12 = Workspace value on entry to ImageFileRender_Register

On exit

None

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is re-entrant

Use

The 'start' routine is called before any operations are applied to an image. This allows clients to cache information relevant to the image such that subsequent calls do not have to re-read the data. If the image data is not recognised, it should be faulted. Errors should be reported by setting V and returning an error block in R0.

Clients may fill in the private word with cached data. Usually this is a pointer to some workspace specific to this image.

Related SWIs

Related entry points

Finalisation routine for ImageFileRender

On entry

R0 = API version * 100 (102 in this version)
R1 = Pointer to image descriptor:

Offset	Contents
+0	Pointer to data to render
+4	Length of data
+8	Pointer to extension data, or 0 if no data
+12	Image sequence number
+16	Private image data

R2 = 0
R12 = Workspace value on entry to ImageFileRender_Register

On exit

None

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is re-entrant

Use

The 'stop' routine is called after all operations are applied to an image. This allows clients to release space allocated for cache information relevant to the image. If there is any internal error, the client should tidy up as best it can and return an error. Errors should be reported by setting V and returning an error block in R0.

Related SWIs

Related entry points

Rendering routine for ImageFileRender

On entry

R0 = API version * 100 (102 in this version)
R1 = Pointer to image descriptor:

Offset	Contents
+0	Pointer to data to render
+4	Length of data
+8	Pointer to extension data, or 0 if no data
+12	Image sequence number
+16	Private image data

R2 = Pointer to rendering descriptor:

Offset	Contents
+0	Flags :

Bit(s)	Meaning
0-2	Reserved, must be zero
3	Colour mapping function supplied
4-5	Reserved, must be zero
6-9	Quality to render at:

Value	Meaning
0	Use default quality
1	Lowest quality
2-14	Renderer specific values
15	Highest quality

10-31	Reserved, must be zero

Offset	Contents
+4-24	Transformation matrix to apply (standard format)
+28	Minimum X clipping rectangle in external coordinates
+32	Minimum Y clipping rectangle in external coordinates
+36	Maximum X clipping rectangle in external coordinates
+40	Maximum Y clipping rectangle in external coordinates
+44	Pointer to colour mapping routine
+48	Workspace for colour mapping routine

R12 = Workspace value on entry to ImageFileRender_Register

On exit

None

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is re-entrant

Use

The rendering routine is called to render an image using a given transformation. If the image data is not recognised, it should be faulted. Errors should be reported by setting V and returning an error block in R0.

The clipping rectangle passed represents the graphics rectangle as external coorinates (OS units) which is currently in use. It is provided for information such that rendering can take advantage of fast rejection of regions which do not need to be redrawn.

Related SWIs

SWI ImageFileRender_Register (on page 471)
SWI ImageFileRender_Deregister (on page 473)
SWI ImageFileRender_Render (on page 459)

Bounding box function for ImageFileRenderer

On entry

R0 = API version * 100 (102 in this version)
R1 = Pointer to image descriptor:

Offset	Contents
+0	Pointer to data to render
+4	Length of data
+8	Pointer to extension data, or 0 if no data
+12	Image sequence number
+16	Private image data

R2 = Pointer to bounding box descriptor:

Offset	Contents
+0	Flags (0)
+4-24	Transformation matrix to apply (standard format)
+28	Minimum X position in Draw coordinates
+32	Minimum Y position in Draw coordinates
+36	Maximum X position in Draw coordinates
+40	Maximum Y position in Draw coordinates

On exit

None

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is re-entrant

Use

The bounding box routine is called to calculate the bounding box for a given transformation. If the image data is not recognised, it should be faulted. Errors should be reported by setting V and returning an error block in R0. The bounding box should be returned in draw coordinates for the images extent. That is, OS units * 256. Resolution values should be provided for information. If no DPI information is available, 180 (the screen resolution) should be returned.

If bit 3 of the renderer flags was clear on registration, the transformation matrix will be an identity matrix and can effectively be ignored. The scaling to the clients required size will be performed by ImageFileRender module based on the bounding box returned.

Related SWIs

SWI ImageFileRender_Register (on page 471)
SWI ImageFileRender_Deregister (on page 473)
SWI ImageFileRender_BBox (on page 462)

Declare fonts function for ImageFileRenderer

On entry

R0 = API version * 100 (102 in this version)
R1 = Pointer to image descriptor:

Offset	Contents
+0	Pointer to data to render
+4	Length of data
+8	Pointer to extension data, or 0 if no data
+12	Image sequence number
+16	Private image data

R2 = Pointer to declare fonts descriptor:

Offset	Contents
+0	Flags (0)
+4	Flags to pass to PDriver_DeclareFont

On exit

None

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is re-entrant

Use

The font declaration routine need only be provided by renderers which use fonts. The renderer should call SWI PDriver_DeclareFont with the names of all fonts and the flags passed in R4. If the image data is not recognised, it should be faulted. Errors should be reported by setting V and returning an error block in R0.

Related SWIs

Information function for ImageFileRenderer

On entry

R0 = API version * 100 (102 in this version)

R1 = Pointer to image descriptor:

Offset	Contents
+0	Pointer to data to render
+4	Length of data
+8	Pointer to extension data, or 0 if no data
+12	Image sequence number
+16	Private image data

R2 = Pointer to information descriptor:

Offset	Contents
+0	Query type
+4	Query data length
+8	Pointer to data block to take details from / fill in

On exit

R0 = If V flag set, a pointer to an error block, or a special error code :

Value	Meaning
1	Invalid query type - the query was not understood.
2	Invalid query length - the query was understood but its length was invalid.

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is re-entrant

Use

The information routine should be provided by renderers to query information about the images. The routine should fault invalid queries and invalid query lengths.

Related SWIs

*Commands

*ImageFileRenderers

List renderers registered with ImageFileRender

Syntax

```
*ImageFileRenderers
```

Parameters

None

Use

*ImageFileRenderers is used to list the renderers known to the
ImageFileRender module. This can be used to check which file
formats are available for use with ImageFileRender from the
command line.

Examples

```
*ImageFileRenderers &695 00000000 ConvertGIF 0.08 RISCOS
Ltd (via IFC)
&69c 00000000 ConvertBMP 0.05 RISCOS Ltd (via IFC)
&69e 00000000 ConvertPNM 0.02 RISCOS Ltd (via IFC)
&aff 00000000 ImageFileRender 0.25 RISCOS Ltd
&b60 00000000 ConvertPNG 0.09 RISCOS Ltd (via IFC)
&b61 00000000 ConvertXBM 0.06 RISCOS Ltd (via IFC)
&c85 00000000 ImageFileRender 0.25 RISCOS Ltd
&d94 00000000 IFR Artworks 0.08 RISCOS Ltd
&fc9 00000000 ConvertSun 0.05 RISCOS Ltd (via IFC)
&ff9 66990101 ImageFileRender 0.25 RISCOS Ltd
```

Related SWIs

SWI ImageFileRender_EnumerateRenderers (on page 474)
SWI ImageFileRender_Register (on page 471)

*ImageFileViewer

Sets the default viewer to use for files known to ImageFileRender

Syntax

*ImageFileViewer [<*command*>]

Parameters

None

Use

*ImageFileViewer is used to register a command which can be used to view files known to ImageFileRender. The `Alias$@RunType_XXX` variables will be set for filetypes known to ImageFileRender which have not already been set. If no parameter is passed to the command, the default viewer wil be cleared and all the variables will be unset.

The effect of issuing this command is that any files know to ImageFileRender which are not recognised by running applications when double-clicked in Filer (or run explicitly) will cause the command specified to be run, passing the filename of the file run as the first parameter.

Examples

*ImageFileViewer /<ImgViewer$Dir>.!Run -file %*0

Related APIs

None

Document information

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

History:	Revision	Date	Author	Changes
	pre-1		AMH	**Pre-release** ● Converted from original text to XML
	pre-2		AMH	**Pre-release** ● Modified ImageFileRender_BBox to add sequence number. ● Modified service call names to Render rather than Renderer. ● Updated the base details query.
	pre-3	20 Nov 2002	ROL	**Validated** ● XML validated. ● Corrected sections which were undefined (IRQ, FIQ, Reentrancy).
	pre-4	21 Jan 2003	ROL	**Misc corrections** ● Corrected the DPI size given for drawfiles. ● Added details of start and stop operations. ● Finished off remaining undefined sections. ● Added error message definitions.
	pre-5	31 Jan 2003	ROL	**Added Enumerate SWI** ● Added documentation about EnumerateRenderers SWI.
	pre-6	21 Jan 2003	ROL	**Misc corrections** ● Correction for ImageFileRender_Transform description.
	pre-7	15 Feb 2003	ROL	**New commands** ● Added documentation of ImageFileRenderers and ImageFileViewer commands.
	pre-8	06 Apr 2003	ROL	**Misc corrections** ● Added magic error values for IFR_Info. ● Corrected documentation of IFR_BBox parameter block. ● Added hyperlinks to

			ImageFileRender_Register and a few related references.
pre-8	01 May 2004	ROL	**Table correction**
			● ImageFileRender_Render's R8 table has been clarified.
1	17 Oct 2020	Gerph	**Backported text file changes**
			● Backported changes from the text version.
			● Bitfields for the quality were incorrectly specified one bit short (should be bits 6-9)
			● Range for query types didn't actually match up consistently.
			● Renderer name wasn't specified in the documentation, although it was expected for presentation.

Video drivers (supplement for RISC OS Pyromaniac)

Introduction and Overview

The video system was traditionally been part of the RISC OS Kernel. However, this overly complicated the assembler portion of the Kernel, made new hardware harder to support, and meant that providing more flexible and faster rendering was significantly impeded. In RISC OS Select 3, the video system was moved out of the Kernel and became a set of regular modules.

There are a number of parts of the video system which were handled by the RISC OS Kernel, and which have been made available through a standard RISC OS vector. These parts are, from lowest level to highest level:

- Mode and frame buffer initialisation. These are handled by a hardware driver such as VideoHWVIDC, VideoHWVF or VideoHWPL110.
- Pointer operations. These are usually handled by the hardware driver.
- VDU 4 text. These are, by default, handled by the software driver, VideoSW, but may be accelerated.
- Graphics operations. These are also handled by the software driver.
- Sprite operations. These are still handled by the Kernel, but are already vectored through SpriteV.
- Teletext operations. These are handled by the VideoTTX module.

This separation makes the maintenance of the video system much easier, and allows runtime modifications to its behaviour. In additioon, the video system has been extended to allow for multiple displays by allowing a separate driver to take over the graphics system. Although in RISC OS Select this was a limited operation, allowing only a single active display at any time, the framework provided allows for greater flexibility in the future.

Technical details

The graphics system has been split up in modern versions of RISC OS. The intention of the division of the system is to allow for accelerated graphics drivers. Graphics operations will be passed to drivers using the new [VideoV (on page 510)](#) vector (&2C). The operation to be performed is passed in a fixed register to the vector.

The reason codes for the vector are grouped into the major regions that they cover:

- &000 - &00F - Text (VDU 4) operations
- &010 - &1FF - Graphics (OS_Plot and similar) operations
- &200 - &2FF - Pointer operations
- &300 - &3FF - Mode and display driver operations
- &400 - &4FF - Teletext operations

Text and graphics operations are provided by the VideoSW module on the VideoV vector. Should there be no accelerated handler earlier on the vector the VideoSW driver will provide the operation.

Text operations

Text operations may be accelerated by the driver, or if no implementation is provided they will be provided by the VideoSW module. Drivers should pay attention to the current display start as set with [VideoV 18 (on page 529)](#) in order to know whether their display, or a sprite output has been selected.

Graphics operations

Graphics operations may be accelerated by the driver, or if no implementation is provided they will be provided by the VideoSW module. Drivers should pay attention to the current display start as set with [VideoV 18 (on page 529)](#) in order to know whether their display, or a sprite output has been selected.

Coordinates

All coordinates passed to the functions have taken account of the eigen-factors for the output. They describe the pixels from the bottom left corner of the output. Many interfaces will require that these coordinates be inverted by subtracting them from the screen height to get the offset from the top of the screen. Coordinates are signed.

Colour operation

The graphic operations may use a special values in R6 to indicate the type of colour operation being performed. Whilst these may be a

fixed 'OR-EOR' pattern (see and for more details), they may also take one of 4 special values. Clients may use these values as a short hand notation to remove the need to check on every graphic operation the whether the operation can be accelerated or not. The values are :

Value	Meaning
0	No effect - the operation can just return
1	Use the last set Colour 1
2	Use the last set Colour 2
3	Invert destination

Any value other than these special values is a pointer to an ECF. If either of bit 0 or 1 is set on these pointers it should be ignored by the driver. This allows for future expansion.

Graphics context

The graphics operations may use a block in R7 to determine the current graphics context. This contains a number of values which may vary between calls. Clients should check these against each operation.

Offset	Contents
+0	Graphics clipping window x-min (inclusive)
+4	Graphics clipping window y-min (inclusive)
+8	Graphics clipping window x-max (exclusive)
+12	Graphics clipping window y-max (exclusive)
+16	Function to call to render a
+20	Function to call to render a
+24	Function to call to render a
+28	Function to call to render a

The functions for bounded and unbounded point rendering have the same interface, but the bounded entry points should clip the rendering to the supplied clipping window.

Pointer operations

Pointer operations are generated by the OSPointer module. The hardware driver should provide an implementation which does not affect the screen buffer (for example, by hardware overlay).

Mode operations

The mode operations are generally only handled by hardware drivers. Each driver will usually decide whether to handle the operation based on the display number. The only exception to this is the display selection entry point (on page 587) which must be handled by all clients in order to determine whether the display is selected.

Teletext operations

Teletext operations are provided as a software supported device driver. Once a teletext mode has been selected, the teletext operations will be passed through the vector in place of the standard text operations. A few of the operations have been modified in order to provide more specialised operations in the teletext modes. Only a single teletext mode is ever in use at any time. Sprite redirection does not allow for teletext within sprite images.

Display device registration

Display devices should register themselves with the Operating System using OS_ScreenMode 255, and deregister when they have been terminated with OS_ScreenMode 254.

The order of operations for a display driver on initialisation should be along the lines of:

- Set private display number variable to -1.
- Initialise any video hardware to a functioning, but disabled state. This may include setting up a display buffer.
- Claim VideoV vector.
- Construct a display device descriptor for the hardware.
- Call OS_ScreenMode 255 to register the display.
- If an error was returned, release all resources and exit with the error.
- Set private display number variable to the value returned.
- Store the VSync dispatcher and its workspace pointer for use later
- If R3 was set to 1, issue OS_ScreenMode 11 to select the display number supplied.
- If an error was returned from the display selection, attempt to select a known supported mode with OS_ScreenMode 0. If a further error is returned, release all resources and exit with the error.
- Complete initialisation and return with no error.

During finalisation it is important that the device shut itself down safely. The following sequence is recommended:

- Set the private display number variable to -1, such that no

vector calls will be interpreted and that VSyncs will no longer be triggered by the driver.

- Release the VideoV vector to prevent any other calls being serviced.
- Disable VSyncs for the hardware.
- Call OS_ScreenMode 254 to deregister the display.
- Release other hardware resources, shutting the hardware down to its most quiescent state.
- Release any other claimed resources.
- Complete finalisation and return with no error.

During the reset sequence, Service_PreReset will be issued. As with other hardware drivers, the hardware should be placed into a quiescent state, and interrupts disabled where necessary. The driver should place itself in a state similar to that of finalisation, except that the VideoV vector may be called and operations on the display buffer may still be performed by other components. Hardware may not have any reset lines asserted or similar within the system and it must be possible for the initialisation sequence to successfully start the hardware from the state which Service_PreReset placed it.

Details of OS_ScreenMode 255, 254 and the display device descriptor can be found in the OSScreenMode documentation.

Software vectors

Vector VideoV
(Vector &2C)

Graphics operation abstraction

On entry

R0 - R6 = Dependant on reason code
R7 = Display number (where relevant)
R8 = Reason code

Value	Meaning
0	[Notifies the text system when redirection occurs (on page 515)](#)
1	[Defines the bitmap of a text character (on page 517)](#)
2	[Change the colour used for rendering text (on page 518)](#)
3	[Render a character on the screen (on page 520)](#)
4	[Render a cursor on the screen (on page 522)](#)
5	[Clear a region of the screen for text (on page 524)](#)
16	[Selects a colour to use as the primary drawing colour (on page 526)](#)
17	[Selects a colour to use as the secondary drawing colour (background) (on page 528)](#)
18	[Notifies the graphics system when redirection occurs (on page 529)](#)
19	[Notifies the graphics system that the destination base has changed (on page 531)](#)
20	[Read primitive operations to use for the current output (on page 532)](#)
21	[Render a rectangle (on page 534)](#)
22	[Render a triangle (on page 536)](#)
23	[Render a parallelogram (on page 538)](#)
24	[Copy a rectangle (on page 540)](#)
25	[Render the outline of a circle (on page 542)](#)
26	[Render a filled circle (on page 544)](#)
27	[Render the outline of an circle arc (on page 546)](#)
28	[Render a filled segment of a circle (on page 548)](#)
29	[Render a filled sector of a circle (on page 550)](#)
30	[Render the outline of an ellipse (on page 552)](#)
31	[Render a filled ellipse (on page 554)](#)
32	[Fill a line right from a position (on page 556)](#)
33	[Fill a line left and right from a position (on page 558)](#)
34	[Flood fill a region (on page 560)](#)
35	[Fill multiple horizontal lines (on page 562)](#)
512	[Define a pointer shape (on page 564)](#)
513	[Select a pointer for use (on page 565)](#)
514	[Updates the location of the pointer on the screen (on page 566)](#)
515	[Removes the pointer from the screen (on page](#)

On exit

R0 - R7 = Dependant on reason code

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

The vector should be claimed when it is handled entirely within the driver.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

RISC OS Pyromaniac   RISC OS ≥ 7.47
Only teletext calls are issued by the graphics system.

Related APIs

None

Notifies the text system when redirection occurs

On entry

R0 = pointer to the base of the destination (DisplayStart)
R1 = line length for this destination (LineLength)
R2 = maximum height in text lines (ScrBRow)
R3 = destination log2bpp depth (Log2BPP)
R4 = mode flags for destination (ModeFlags)
R8 = 0 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called a new destination has been selected for output. In particular, it will be called on mode change and sprite redirection. Clients should initialise any variables which are necessary for their operation. Clients wishing to handle the entire Text interface may claim this call point, but it is strongly recommended that the call be passed on.

The mode flags will indicate any special features which are present in the mode specified. In particular:

Bit(s)	Meaning
2	'Gap mode', indicating that characters will be spaced 9 rows apart, rather than 8 - leaving a single line which will not be written to. This flag is deprecated and support is not required of any clients.
3	'BBC gap mode', should be treated as identical to Bit 2. This flag is deprecated and support is not required of any clients.
5	Double height-VDU characters, indicating that characters should be written 16 units high, rather than 8 This flag is deprecated and support is not required of any clients.

This vector should never be claimed.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
   Supported

Related vectors

[VideoV (on page 510)](#)

Defines the bitmap of a text character

On entry

R0 = the character to define (32-255)
R0 = pointer to word aligned 8 bytes to use as the character
R8 = 1 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to define a character. Clients should normally make a note of the changes and pass this call on.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

VideoV (on page 510)

Change the colour used for rendering text

On entry

R0 = foreground screen colour
R1 = background screen colour
R2 = BPP of the current mode
R8 = 2 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called whenever the text colour has been changed and text output is about to be performed. It is expected that clients initialise any cached data they require in order to render characters in the new colours. The colours supplied are values to be written to the screen for that pixel, for example, in 2 BPP modes, a foreground colour of 3 would be used to indicate that all pixels set for that character would use the value 3 in memory. Clients may need to replicate these bits across a word in order to perform operations more rapidly, however implementation details are left to the client's discretion.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

Render a character on the screen

On entry

R0 = character to write (32-255)
R1 = pointer to address of top left pixel to write (word aligned) on the screen
R2 = character x (origin top left)
R3 = character y (origin top left)
R8 = 3 (reason code)

On exit

R0 - R8 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to render a character on the screen at a text position. Characters are 8x8 pixels in all modes and should be rendered in the colours specified by VideoV 2 (on page 518). Unset pixel data for the character should use the background colour, and set pixel data for the character should use the foreground colour. Registers R2 and R3 are provided for clients which require the actual pixel position to render the character.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

Render a cursor on the screen

On entry

R0 = composite cursor position (&SS0000EE)
R1 = pointer to address of top left pixel for the character (word aligned) on the screen
R2 = character x (origin top left)
R3 = character y (origin top left)
R4 = offset of start of the cursor, from R1
R5 = offset of line after the end of the cursor, from R1
R8 = 4 (reason code)

On exit

R0 - R8 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to render a 'cursor' at a text position. This should be an inverted rectangle which spans the lines of the character requested by R0, or R4 and R5. The composite cursor mask is provided for clients which use the pixel position of the character to render the shape. The cursor should be rendered at the character specified by inverting the current contents of that location. The vector will be called repeatedly in order to 'flash' the cursor. In 'split' editing mode, the vector will be called to render whichever cursor requires redrawing.

The composite cursor position indicates the start and end lines of the

cursor by the SS and EE values. The EE value is the last line that should be drawn. Thus, SS=6, EE=7 would invert lines 6 and 7 within the character. Similarly, SS=8, EE=9 would invert lines 8 and 9 within the character. Line 8 and 9 are only applicable to gap-modes.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
    Supported

Related vectors

[VideoV (on page 510)](#)

Clear a region of the screen for text

On entry

R0 = pointer to box description

Offset	Contents
+0	left char x
+4	bottom char y
+12	right char x
+16	top char y
+20	top left address to start at (might be byte aligned)
+24	number of bytes to fill per line (might be byte aligned; usually will be the same as LineLength)
+28	number of lines to fill
+32	character line size (8, 10, 16, 20)
+36	fill word for first 8/16 lines of each character
+40	fill word for subsequent lines (gap fill)

R8 = 5 (reason code)

On exit

R0 - R8 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to clear a region of the text window. The supplied description block gives the character positions, and the memory positions which require clearing. The clear operation should be performed using the fill words given in +36 and +40. These words will be the same unless the user is in a BBC-style gap mode. Such modes are not expected to be used by clients in the future and support may be omitted from accelerated modules. The software implementation provides support for all combinations.

As with other text operations, the character positions are specified with their origin at the top left.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
    Supported

Related vectors

[VideoV (on page 510)](#)

Selects a colour to use as the primary drawing colour

On entry

R0 = action type, for information purposes
R1 = pointer to OR-EOR pattern to use

Offset	Contents
+0	Value to OR in word for line 0
+4	Value to EOR in word for line 0
+8	Value to OR in word for line 1
+12	Value to EOR in word for line 1
+16	Value to OR in word for line 2
+20	Value to EOR in word for line 2
+24	Value to OR in word for line 3
+28	Value to EOR in word for line 3
+32	Value to OR in word for line 4
+36	Value to EOR in word for line 4
+40	Value to OR in word for line 5
+44	Value to EOR in word for line 5
+48	Value to OR in word for line 6
+52	Value to EOR in word for line 6
+56	Value to OR in word for line 7
+60	Value to EOR in word for line 7

R8 = &10 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to set the primary colours to use for plotting graphics operations. Clients should record the details. They may pre-cache the values after determining whether they can handle the operation type.

Lines are measured from the top left of the screen and should be ANDed with 7. Consult the example VideoSW code for more details (s/GrafPoint gives an obvious use).

Whilst many store and invert operations will be simple to accelerate within drivers, the more complex operations may be deferred to the software driver where necessary. In particular, drivers should be aware that the operation pattern may not correspond directly to any PLOT reason code. Users may manually select different colour operations for different bit regions or lines.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
   Supported

Related vectors

VideoV (on page 510)
VideoV 16 (on page 526)

Vector VideoV 17
Graf_SetColour2
(Vector &2C)

Selects a colour to use as the secondary drawing colour (background)

On entry

R0 = action type, for information purposes
R1 = pointer to OR-EOR pattern to use, as for VideoV 16 (on page 526)
R8 = &11 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to set the secondary colour used for graphics operations. This is commonly called the 'background colour'. The operation is identical to that of VideoV 16 (on page 526).

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

VideoV (on page 510)
VideoV 16 (on page 526)

Notifies the graphics system when redirection occurs

On entry

R0 = pointer to context information

Offset	Contents
+0	Mode flags
+4	Text screen width-1
+8	Text screen height-1
+12	Number of colours
+16	X-eigen factor
+20	Y-eigen factor
+24	Line length
+28	Total output size
+32	Base of the output buffer
+36	Log2 Bits Per Pixel
+40	Log2 Bits Per Addressable Pixel (hang-over from double-width pixel modes)
+44	Graphics screen width-1
+48	Graphics screen height-2

R1 = 0 if the destination is the screen, otherwise the destination is a sprite

R8 = &12 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called whenever the output changes destination, usually due to a mode change or sprite redirection. Drivers should cache the context information for use within other calls. This vector should never be claimed.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

[VideoV (on page 510)](#)

Notifies the graphics system that the destination base has changed

On entry

R0 = pointer to base of the buffer
R8 = &13 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called when the output destination base address has changed. It may be called under a number of circumstances, including sprite deletion, mask changes and hardware scrolling. Drivers should make a note of the new address and base all their calculations from it.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

VideoV (on page 510)

Graf_ReadPrimitives
(Vector &2C)

Read primitive operations to use for the current output

On entry

R0 = pointer to where to store HLine handler (r12, function)
R1 = pointer to where to store Point handler (r12, function)
R2 = pointer to where to store VLine handler (r12, function)
R8 = &14 (reason code)

On exit

R0 = new pointer to where to store HLine details at
R1 = new pointer to where to store Point details at
R2 = new pointer to where to store VLine details at
R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to read the primitive operations which can be
used to render a horizontal line, a point and a vertical line. Since
these operations are expected to be called regularly it is important
that they be fast. Rather than using the vector dispatch for every line
or point to be rendered, a single function dispatch can be used -
obviating the need for a SWI call, and subsequent vector dispatch.
Consult the Graf_ReadPrimitives source for more details of the
operation.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

[VideoV (on page 510)](#)

Render a rectangle

On entry

R0 = pointer to coordinate block:

Offset	Contents
+0	x-min (inclusive)
+4	y-min (inclusive)
+8	x-max (exclusive)
+12	y-max (exclusive)

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &15 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to fill a rectangle. Clients should bound the coordinates if necessary and then plot the rectangle.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

Render a triangle

On entry

R0 = pointer to coordinate block:

Offset	Contents
+0	x0
+4	y0
+8	x1
+12	y1
+16	x2
+20	y2

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &16 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to fill a triangle. Clients should plot the triangle, bounded by the coordinates given.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
    Supported

Related vectors

[VideoV (on page 510)](#)

Render a parallelogram

On entry

R0 = pointer to coordinate block:

Offset	Contents
+0	x0
+4	y0
+8	x1
+12	y1
+16	x2
+20	y2

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &17 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to fill a parallelogram. Clients should plot the parallelogram, bounded by the coordinates given. The coordinates given are in no particular order. The fourth verex can be calculated from the other three.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
 Supported

Related vectors

[VideoV (on page 510)](#)

Copy a rectangle

On entry

R0 = pointer to coordinate block:

Offset	Contents
+0	source x0
+4	source y0
+8	source x1
+12	source y1
+16	destination x0
+20	destination y0
+24	destination x1
+28	destination y1

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &18 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to copy a rectangle from part of the display elsewhere. No context or colour operation is provided. The region has already been bounded to the screen and graphics window (both the source and destination are guarenteed to be within the buffer). The source data, where not overwritten, should be unaffected by the copy operation.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
    Supported

Related vectors

[VideoV (on page 510)](#)

Render the outline of a circle

On entry

R0 = pointer to coordinate block:

Offset	Contents
+0	x centre
+4	y centre
+8	x edge
+12	y edge

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &19 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

Render a filled circle

On entry

R0 = pointer to coordinate block:

Offset	Contents
+0	x centre
+4	y centre
+8	x edge
+12	y edge

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &1A (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to fill a circle. Clients should plot the circle, bounded by the coordinates supplied if necessary.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

Render the outline of an circle arc

On entry

R0 = pointer to coordinate block:

Offset	Contents
+0	x centre
+4	y centre
+8	x start
+12	y start
+16	x end
+20	y end

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &1B (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to draw an arc. Clients should plot the arc, bounded by the coordinates supplied if necessary. The coordinate order and meaning is identical to that used by the PLOT arc operation. That is, first coordinate pair indicates the centre of the circle, the second pair provides the start position on the edge of the circle, and the final pair provides the end position as a point on a line that intersects the circle. The arc should be drawn anti-clockwise.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
   Supported

Related vectors

[VideoV (on page 510)](#)

Render a filled segment of a circle

On entry

R0 = pointer to coordinate block:

Offset	Contents
+0	x centre
+4	y centre
+8	x start
+12	y start
+16	x end
+20	y end

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &1C (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to draw a segment. Clients should plot the segment, bounded by the coordinates supplied if necessary. The coordinate order is the same as VideoV 27 (on page 546). A segment fills the arc shape, closing it with a line from the start position to the intersection of the end position line and the circle.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

[VideoV (on page 510)](#)

Render a filled sector of a circle

On entry

R0 = pointer to coordinate block:

Offset	Contents
+0	x centre
+4	y centre
+8	x start
+12	y start
+16	x end
+20	y end

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &1D (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to draw a sector. Clients should plot the sector, bounded by the coordinates supplied if necessary. The coordinate order is the same as VideoV 27 (on page 546). A segment fills the arc shape, closing it with a pair of lines from the start position to the centre of the circle, and from the intersection of the end position line and the circle to the centre of the circle. The shape is often known as a 'pie' from the common shape produced by removing sections from pies.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
   Supported

Related vectors

VideoV (on page 510)

Render the outline of an ellipse

On entry

R0 = pointer to coordinate block:

Offset	Contents
+0	x centre
+4	y centre
+8	x width
+12	ignored
+16	x limit point
+20	y limit point

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &1E (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to draw an ellipse. Clients should plot the ellipse, outline bounded by the coordinates supplied if necessary. The coordinate order is the same as the equivilent PLOT ellipse operation. That is, the first pair gives the centre position of the ellipse, the second pair gives the width of the ellipse at centre line (the y coordinate is ignored), and the final pair gives the position of the top-most, or bottom-most limit of the ellipse.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
  Supported

Related vectors

[VideoV (on page 510)](#)

Render a filled ellipse

On entry

R0 = pointer to coordinate block:

Offset	Contents
+0	x centre
+4	y centre
+8	x width
+12	ignored
+16	x limit point
+20	y limit point

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &1F (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to draw an ellipse. Clients should plot the solid ellipse, bounded by the coordinates supplied if necessary. The coordinate order is as for the VideoV 30 (on page 552) operation.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
    Supported

Related vectors

[VideoV (on page 510)](#)

Fill a line right from a position

On entry

R0 = x start
R1 = y start
R2 = delimiting colour
R3 = delimiting condition:

Value	Meaning
0	fill to delimiting colour
&80000000	fill to non-delimiting colour

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &20 (reason code)

On exit

R0 corrupted
R1 = right x position where fill ended
R2 corrupted
R3 = 1 if anything was filled, 0 if nothing was filled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to fill a line right until a given condition is met
(denoted by R2 and R3).

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

[VideoV (on page 510)](#)

Fill a line left and right from a position

On entry

R0 = x start
R1 = y start
R2 = delimiting colour
R3 = delimiting condition:

Value	Meaning
0	fill to delimiting colour
&80000000	fill to non-delimiting colour

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &21 (reason code)

On exit

R0 = left x position where fill ended
R1 = right x position where fill ended
R2 corrupted
R3 = 1 if anything was filled, 0 if nothing was filled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to fill a line left and right until a given condition
is met (denoted by R2 and R3).

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

[VideoV (on page 510)](#)

Flood fill a region

On entry

R0 = x start
R1 = y start
R2 = delimiting colour
R3 = delimiting condition:

Value	Meaning
0	fill to delimiting colour
&80000000	fill to non-delimiting colour

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &22 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to perform a flood fill operation, stopping at a given condition (denoted by R2 and R3).

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

Fill multiple horizontal lines

On entry

R0 = pointer to buffer of horizontal line segments:

Offset	Contents
+0	number of horizontal lines to draw
+4	1st horizontal line segment: x left position
+8	1st horizontal line segment: y position
+12	1st horizontal line segment: x right position
+16...	Subsequent horizontal line segments

R6 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)
R8 = &23 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to perform many horizontal line fill operations as a single step. The block supplied must be copied if the driver is intending to buffer these operations. Because this call is always issued through the vector, clients should try to ensure that it is worthwhile calling using this interface rather than the direct HLine interface (see OS_ReadVduVariables).

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

[VideoV (on page 510)](#)

Define a pointer shape

On entry

R0 = width
R1 = pointer to pointer data
R2 = height
R6 = pointer number to define (0-3)
R7 = display number
R8 = &200 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to define one of the 4 pointer shapes. Pointer shapes are pre-compensated for the 'active' position.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

VideoV (on page 510)

Select a pointer for use

On entry

R0 = pointer number
R7 = display number
R8 = &201 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to select one of the pointers for use. Only one
pointer is ever displayed at a time.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

VideoV (on page 510)

Updates the location of the pointer on the screen

On entry

R0 = height of the pointer
R1 = y coordinate
R2 = x coordinate
R3 = screen height
R7 = display number
R8 = &202 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to update the location of the pointer on the screen.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

[VideoV (on page 510)](#)

Removes the pointer from the screen

On entry

R3 = screen height
R7 = display number
R8 = &203 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to remove the pointer from the screen.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

VideoV (on page 510)

Pointer_SetPalette (Vector &2C)

Set a colour used by the pointer

On entry

R0 = colour number to update
R1 = colour number in form &00BBGGRR
R7 = display number
R8 = &204 (reason code)

On exit

R0 - R3 corrupted

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to set the colour of the pointer.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

Check the validity of a mode

On entry

R0 = pointer to a VIDC type 3 table
R1 = memory required for the mode
R7 = display number
R8 = &300 (reason code)

On exit

V   If set, indicates an error. The usual error pointer may be
flag   supplied in R0, or if R0 is set to 0 this indicates that a generic
       error will be returned.

       If clear, indicates the mode is acceptable.

R7 = -1 if handled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to check the validity of a mode prior to selecting
it. Drivers should reject modes which they are incapable of
displaying.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

Select a screen mode for use

On entry

R0 = pointer to a VIDC type 3 table
R1 = memory required for the mode
R7 = display number
R8 = &301 (reason code)

On exit

R0 = Pointer to base of screen area
R7 = -1 if handled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to select a screen mode for use. It has already
passed the vetting procedure above so should be able to be selected.

Drivers should perform the following actions on a mode change:

- Default to the first screen bank for display and driver
- Release any claimed memory for alternate screen banks
- Clear the palette to black

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

Hardware scroll of the display

On entry

R0 = number of scanlines to move down by (may be negative to scroll up)
R1 = number of bytes to which this equates
R2 = background colour (word) to fill lines with
R3 = background colour (word) to fill 'gaps' with
R7 = display number
R8 = &302 (reason code)

On exit

R0 = address of screen base, or 0 if hardware scroll is not supported. The address may be the same as the current base if the operation can be performed with acceleration (ie a 'move' operation). The address should be in the lower mapping of the screen buffer if a doubly mapped area is in use.
R1 = State of exposed region:

Value	Meaning
1	the exposed region has been cleared to the requested background
0	the exposed region has not been cleared and must be cleared by the OS

R2 - R3 corrupted
R7 = -1 if handled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to request a hardware scroll of the device. The scroll may be a change of the base of the area, as used by the VIDC driver, or an accelerated copy operation. Drivers can just ignore this operation, or return 0 for both R0 and R1 to indicate that the hardware scroll is not supported. The Kernel can perform the necessary operations if they are not supported by the driver.

Note: (R0 AND NOT 7) lines should be filled with R2.
(R0 AND 7) lines should be filled with R3.

This allows the system to provide gap modes in the same manner as the BBC did.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
    Supported

Related vectors

Change displayed colours in paletted modes

On entry

R0 = 0-255 for regular palette entries
256 for 'border'

R1 = colour in form &0sBBGGRR, where 's' indicates a supremacy
nibble which may be used for hardware masking operations. Its
default will be 0.
R7 = display number
R8 = &303 (reason code)

On exit

R0 - R3 corrupted
R7 = -1 if handled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to change the displayed colours. The operation
of regular palette entries is defined within paletted modes only.
Drivers or hardware not supporting the screen 'border' colour should
ignore the operation. Drivers which cannot provide the full 24bit
colour specification should make best effort to match the colours
requested. The colours should be transformed by any RGB tables
specified in a separate call if the hardware does not support any
specific transformation configuration.

Where RGB tables are not supported by the hardware, it will be

necessary to take a copy of the palette entries.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

[VideoV (on page 510)](#)

Enable display hardware

On entry

R7 = display number
R8 = &304 (reason code)

On exit

R0 - R3 corrupted
    R7 = -1 if handled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to enable the display hardware. This operation is intended to allow components to restart the display hardware for client defined reasons. The Portable module may use this to control the power to the hardware when requested.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

VideoV (on page 510)

Disable display hardware

On entry

R7 = display number
R8 = &305 (reason code)

On exit

R0 - R3 corrupted
R7 = -1 if handled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to disable the display hardware. This operation is intended to allow components to shut down and restart the display hardware for client defined reasons. The Portable module may use this to control the power to the hardware when requested.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

VideoV (on page 510)

Select a power saving mode for the display

On entry

R0 = power saving state:

Value	Meaning
-1	D0 DPMS state - Normal operation, display enabled
0	D0 DPMS state - No DPMS but display disabled if possible (rest of the system will blank palette in this state; VIDC will disable disable refresh and DMA in this state)
1	D1 DPMS state - 'Standby'
2	D2 DPMS state - 'Suspend'
3	D3 DPMS state - 'Active off'
4	DPMS state supplied in modes VIDC type 3 table (ie 'default')
other	will not be used

R7 = display number
R8 = &306 (reason code)

On exit

R0 - R3 corrupted
R7 = -1 if handled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to select an immediate power-save mode should be entered. Regardless of whether this call is handled or not, the palette will be set to black. Drivers should expect to get palette operations whilst blanked. They may wish to ignore these.

The state values (with the exception of -1) are directly modelled after the DPMS states. -1 can be treated as equivilent to 0 for the purposes of H/V sync control.

If mapping directly to HSync and VSync lines, these values are:

Value	Meaning
0	HSync On, VSync On
1	HSync Off, VSync On
2	HSync On, VSync Off
3	HSync Off, VSync Off

Hardware may promote or demote these settings as it sees fit, but this should be documented in hardware documentation as appropriate.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
    Supported

Related vectors

[VideoV (on page 510)](#)

Modify RGB mapping tables (gamma tables)

On entry

R0 = pointer to 768 byte table of R, G, B mappings

R7 = display number
R8 = &307 (reason code)

On exit

R0 - R3 corrupted
R7 = -1 if handled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to modify the RGB mapping tables, through which the palette entries will be transformed. The palette should be affected immediately. Where the hardware does not support RGB mapping tables it will be necessary to translate all palette operations through the tables. It is expected that unpaletted modes (16bpp, 32bpp) support this operation, even though they do not use palette entries.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

Configure acceleration options

On entry

R0 = configuration flags, or -1 to read current state:

Bit(s)	Meaning
0	suspend cached screen until mode change
1	suspend automatic screen cleaning until mode change
2	disable all hardware acceleration
3-30	reserved, must be 0
31	must be set

R1 = automatic screen cleaning level, 1-3 or -1 to read current state
R7 = display number
R8 = &308 (reason code)

On exit

R0 = new or current flags, modified to acceptable values
R1 = new or current automatic cleaning level

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to control the generic acceleration options provided by the video driver. The core acceleration options are to provide a CPU cached screen. One effect of CPU cached screen is an undesirable delay on data being written to the physical screen memory from the processor. This data should be flushed by the driver regularly, if it detects that data has been written to the screen. It is usual to use abort trapping, domain mapped dynamic areas to manage this process (consult the example VIDC driver).

Bit 0 and 1 can be used to suspend this operation until the next mode change. Clients will probably use this for specialised tasks to ensure that their output is immediately visible, such as within games or high responsiveness applications.

Bit 2 is intended to allow clients to disable all hardware acceleration. Whilst this is generally not desirable, it may be useful to clients who either believe there to be faults within the hardware acceleration or who wish to determine the difference made by hardware acceleration. Hardware acceleration is should not be re-enabled by a mode change as the client may wish to examine mode change timings, or it may be believed that mode change acceleration is faulty.

This vector call is triggered by OS_ScreenMode 4.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
    Supported

Related vectors

[VideoV (on page 510)](#)

Immediate control operations for acceleration

On entry

R0 = operation:

Value	Meaning
0	clean cache immediately, if necessary and not suspended
1	clean cache immediately, if necessary
other	reserved

R1 - R6 = dependant on reason code
R7 = display number
R8 = &309 (reason code)

On exit

R1 - R6 = dependant on reason code
R7 = display number

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to cause the screen to be made up to date if necessary such that the data written to it is visible to the user. It is likely that these operations will be used by games. The WindowManager will use this call at the end of a series of redraw operations.

This vector call is triggered by OS_ScreenMode 5 and 6 at present.

Compatibility

RISCOS Ltd RISC OS ≥ Select 3
    Supported

Related vectors

[VideoV (on page 510)](#)

Select a display for use

On entry

R7 = display number
R8 = &30A (reason code)

On exit

R7 = display number

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to inform display device drivers of a device selection. Drivers should claim VSync interrupts when they are enabled and begin generating VSyncs through the VSync dispatch entry point returned when they registered with OS_ScreenMode 255. Drivers should release the VSync interrupts when another device is selected.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

[VideoV (on page 510)](#)

Read number of supported screen banks

On entry

R0 = size of current mode, in bytes
R7 = display number
R8 = &320 (reason code)

On exit

R0 = number of banks available (1 if only a single bank available)
R1 - R3 corrupted
R7 = -1 if handled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to request the number of screen banks that can be supported by the current configuration of the display. The driver should return the maximum number of banks that may be supported at the instant that the request was made. Subsequent driver operations may cause this value to change.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

Change the displayed screen bank

On entry

R0 = bank to display (0 .. max-1)
R1 = size of current mode, in bytes
R7 = display number
R8 = &321 (reason code)

On exit

R0 = bank we are displaying
R1 = pointer to address of bank
R7 = -1 if handled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to request a change of display bank. The display bank is presented to the user. Both display and driver banks should be available in memory simultaneously once selected and may both be written to directly. They may be coincident. Once requested (and the request honoured), the Driver should maintain the memory allocated for that bank until the next mode change. Drivers are not required to page all available, or used, banks into logical memory at any time. At the drivers discretion it may remove mappings a bank which is not the driver or display from logical memory. Clients are not expected to access banks which are not selected. Drivers are not expected to support hardware scrolling of any bank but the display bank. Selection of a pre-existing bank should not clear its contents.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

[VideoV (on page 510)](#)

Change the screen bank used by VDU drivers

On entry

R0 = bank to update through VDU drivers (0 .. max-1)
R1 = size of current mode, in bytes
R7 = display number
R8 = &322 (reason code)

On exit

R0 = bank we are displaying
R1 = pointer to address of bank
R7 = -1 if handled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to request a change of driver bank. The driver bank is presented to the video drivers for writing to. Both display and driver banks should be available in memory simultaneously once selected and may both be written to directly. They may be coincident. Once requested (and the request honoured), the Driver should maintain the memory allocated for that bank until the next mode change. Drivers are not required to page all available, or used, banks into logical memory at any time. At the drivers discretion it may remove mappings a bank which is not the driver or display from logical memory. Clients are not expected to access banks which are not selected. Drivers are not expected to support hardware scrolling of any bank but the display bank. Selection of a pre-existing bank should not clear its contents.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

[VideoV (on page 510)](#)

Copy a screen bank

On entry

R0 = source bank number (0..max-1)
R1 = destination bank number (0..max-1)
R2 = size of current mode, in bytes
R7 = display number
R8 = &323 (reason code)

On exit

R2 - R3 corrupted
R7 = -1 if handled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called when a client requests a copy of a screen bank be made. If the destination bank has not yet been allocated it should be allocated. The banks requested need not be paged in to logical memory. Drivers which require the banks to be paged in to logical memory for the copy to take place should take the necessary steps to achieve this. This operation must not affect any other bank by the destination. Clients are expected to perform a bank switch to update the display bank, rather than a bank copy.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

Related vectors

[VideoV (on page 510)](#)

Initialise teletext mode

On entry

R0 = requested maximum text columns (ScrRCol)
R1 = requested maximum text rows (ScrBRow)
R2 = maximum horizontal pixel coordinate (XWindLimit)
R3 = maximum vertical pixel coordinate (YWindLimit)
R4 = x eigen factor
R5 = y eigen factor
R7 = display number
R8 = &400 (reason code)

On exit

R0 = acceptable text columns
R1 = acceptable text rows
R2 - R8 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to initialise teletext mode. The values supplied in
R0 and R1 are the requested number of columns and rows for the
mode. The driver should either fit those values within the graphics
limitations of the mode, or modify the values to be acceptable before
return. The frame buffer does not need to be initialised.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
     Supported

RISC OS Pyromaniac   RISC OS ≥ 7.47
     Supported

Related vectors

Clear a region of the display

On entry

R0 = left char x
R1 = bottom char y
R2 = right char x
R3 = top char y
R7 = display number
R8 = &401 (reason code)

On exit

R0 - R8 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to clear a region of the display. It is used after mode selection and for CLS operations. The frame buffer does not need to be updated.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
   Supported

RISC OS Pyromaniac   RISC OS ≥ 7.47
   Supported

Related vectors

Update the frame buffer with teletext changes

On entry

R7 = display number
R8 = &402 (reason code)

On exit

R0 - R8 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to update the frame buffer with the previously applied changes. This is the primary point at which the frame buffer should be modified.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

RISC OS Pyromaniac   RISC OS ≥ 7.47
Supported

Related vectors

VideoV (on page 510)

Write a character to the teletext screen

On entry

R0 = character
R1 = x position
R2 = y position
R7 = display number
R8 = &403 (reason code)

On exit

R0 - R8 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to change the character at a given position. The frame buffer does not need to be updated. The change of the character may change not only the representation of the character at that position, but also all subsequent character on that line and (potentially) the characters on the line below.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

RISC OS Pyromaniac   RISC OS ≥ 7.47
Supported

Related vectors

Scroll a region of the teletext buffer

On entry

R0 = left char x
R1 = bottom char y
R2 = right char x
R3 = top char y
R4 = change in x position (negative = left, positive = right)
R5 = change in y position (negative = up, positive = down)
R7 = display number
R8 = &404 (reason code)

On exit

R0 - R8 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to scroll a region of the teletext buffer. The frame buffer does not need to be updated. The change of the region may also result in surrounding characters being changed in their representation.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
> Supported

RISC OS Pyromaniac   RISC OS ≥ 7.47
> Supported

Related vectors

[VideoV (on page 510)](#)

Change the flash state of the teletext buffer

On entry

R0 = Flags:

Bit(s)	Meaning
0	Flash characters visible
1-31	Reserved, must be zero

R7 = display number
R8 = &405 (reason code)

On exit

R0 - R8 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to update the state of the display with respect to flashing characters. The frame buffer does not need to be updated. This entry point may be called under interrupts.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
   Supported

RISC OS Pyromaniac   RISC OS ≥ 7.47
   Supported

Related vectors

Read a character from the teletext buffer

On entry

R0 = x position
R1 = y position
R7 = display number
R8 = &406 (reason code)

On exit

R0 = character read
R1 - R8 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to read a character from the display at a position. It is used during the character copying routines, via the OS_Byte mechanisms.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

RISC OS Pyromaniac   RISC OS ≥ 7.47
Supported

Related vectors

Invert the text cursor in the teletext screen

On entry

R0 = composite cursor position (&SS0000EE)
R2 = x position (origin top left)
R3 = y position (origin top left)
R7 = display number
R8 = &407 (reason code)

On exit

R0 - R8 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called to invert a cursor on the screen. It is used to flash the cursor whilst updating the display. It may be called under interrupts.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

RISC OS Pyromaniac   RISC OS ≥ 7.47
Supported

Related vectors

Change the quality of teletext rendering

On entry

R0 = Flags:

Bit(s)	Meaning
0	High quality requested
1-31	Reserved, must be zero

R7 = display number
R8 = &408 (reason code)

On exit

R0 - R8 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called in response to VDU23,18,16,flags| to set the quality of the display. Display drivers may use it to provide a higher quality (and potentially slower) implementation.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
Supported

RISC OS Pyromaniac   RISC OS ≥ 7.47
Supported

Related vectors

Change the reveal state for hidden characters

On entry

R0 = Flags:

Bit(s)	Meaning
0	Concealed characters visible
1-31	Reserved, must be zero

R7 = display number
R8 = &409 (reason code)

On exit

R0 - R8 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Vector is not re-entrant

Use

This vector is called in response to VDU23,18,2,flags| to change the reveal state. Display drivers may update the hidden characters to be visible immediately or defer this until an update.

Compatibility

RISCOS Ltd   RISC OS ≥ Select 3
   Supported

RISC OS Pyromaniac   RISC OS ≥ 7.47
   Supported

Related vectors

Entry points

VideoV_Context_HLine

Draw horizontal line

On entry

R0 = X left coordinate
R1 = Y coordinate
R2 = X right coordinate
R3 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)

On exit

R0 - R7 preserved

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

Not defined

Use

This entry point is provided as part of the graphics context. It may be used by any of the operations called by VideoV to effect a horizontal line. This allows a VideoV claimant who has been requested to draw a shape to render without knowing the specifics of how to access the screen.

The VideoV system will pass the request to draw the line to the registered drivers as a primitive rendering operation.

The functions for bounded and unbounded line rendering have the same interface, but the bounded entry points should clip the rendering to the supplied clipping window.

Related entry points

VideoV_Context_Point

Plot a point

On entry

R0 = X coordinate
R1 = Y coordinate
R2 = Colour operation (on page 506)
R7 = pointer to Graphics context (on page 507)

On exit

R0 - R7 preserved

Interrupts

Interrupts are undefined
Fast interrupts are undefined

Processor mode

Processor is in undefined mode

Re-entrancy

Not defined

Use

This entry point is provided as part of the graphics context. It may be used by any of the operations called by VideoV to effect a single point. This allows a VideoV claimant who has been requested to draw a shape to render without knowing the specifics of how to access the screen.

The VideoV system will pass the request to plot the point to the registered drivers as a primitive rendering operation.

The functions for bounded and unbounded point rendering have the same interface, but the bounded entry points should clip the rendering to the supplied clipping window.

Related entry points

VideoV_Context_HLine (on page 615)

Document information

Maintainer(s):	Gerph <gerph@gerph.org>			
History:	**Revision**	**Date**	**Author**	**Changes**
	1	28 Oct 2006	Gerph	**Initial version**
				● Released as part of the Video SDK
	2	30 Mar 2023	Gerph	**Updated for PRM-in-XML**
				● Transferred content to PRM-in-XML format
	3	17 May 2023	Gerph	**Updated for RISC OS Pyromaniac**
				● Updated with details of compatibility
				● Fixed some validity problems
Disclaimer:	© Gerph, 2006-2023.			

PathUtils

Introduction

The PathUtils module provides an interface to manipulate system variables used as path variables by FileSwitch. That is, variables ending '$Path' which are used as references to multiple paths in filenames.

SWI calls

PathUtils_EnumeratePath
(SWI &53B80)

Enumerate the components of a path variable

On entry

R0 = Flags:

Bit(s)	Meaning
0	Set: Return all components of the path recursively
	Clear: Return only leaf components of the path
1-31	Reserved, must be 0

R1 = Pointer to path to process
R2 = Pointer to output buffer
R3 = Maximum length of the buffer, or 0 to request length
R4 = Opaque context value, or 0 for the first call

On exit

R0 - R2 preserved
R3 = Number of spare bytes in the buffer
R4 = Context value, or -1 if complete (and the other registers are invalid)
R5 = Variable type that the value was expanded from
R6 = Depth the value was expanded from

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to enumerate the components of a path variables. The path variable is expanded recursively. If R0 bit 0 is set, each path component will be returned in the results, even it is not terminal itself.

Related APIs

None

PathUtils_JoinPath
(SWI &53B81)

Join a new path to a path variable

On entry

R0 = Flags:

Bit(s)	Meaning
0	Set: Append the supplied path
	Clear: Prepend the supplied path
1-31	Reserved, must be 0

R1 = Pointer to variable name to modify
R2 = Pointer to path component to join

On exit

R0 - R2 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to join a path to an existing path variable. If the component already exists in the path variable, it will not be added.

Related * commands

PathUtils_RemovePath
(SWI &53B82)

Remove a path from a path variable

On entry

R0 = Flags:

Bit(s)	Meaning
0-31	Reserved, must be 0

R1 = Pointer to variable name to modify

R2 = Pointer to path component to remove

On exit

R0 - R2 preserved

Interrupts

Interrupts are undefined

Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to remove a path from an existing path variable. If the component is not present, the variable will not be modified.

Related * commands

*Commands

*AppPath

Append a path component to a path variable

Syntax

*AppPath *<path-variable> <path-component>*

Parameters

<path-variable> - name of the path variable to append to

<path-component> - name of the path to append to the variable

Use

This command appends a given path component to a path variable. If the path is already present, it has no effect.

Examples

*AppPath Run$Path $.Library.

Related * commands

[*PrepPath (on page 625)](#)

Related SWIs

[SWI PathUtils_JoinPath (on page 622)](#)

*PrepPath

Prepend a path component to a path variable

Syntax

*PrepPath *<path-variable>* *<path-component>*

Parameters

<path-variable>	- name of the path variable to append to
<path-component>	- name of the path to insert at the start of the path variable

Use

This command prepends a given path component to a path variable, inserting the path at the start of the variable's value. If the path is already present, it has no effect.

Examples

*PrepPath Run$Path $.Library.

Related * commands

[*AppPath (on page 624)](#)

Related SWIs

[SWI PathUtils_JoinPath (on page 622)](#)

*RemPath

Remove a path component from a path variable

Syntax

*RemPath *<path-variable>* *<path-component>*

Parameters

<path-variable> - name of the path variable to change

<path-component> - name of the path to remove from the path variable

Use

This command removes a given path component from a path variable. If the variable is not present, the variable is not changed.

Examples

*RemPath Run$Path $.Library.

Related SWIs

Document information

Index (Commands)

Index (SWIs)

Index (SWIs by number)

Index (UpCalls)

Index (UpCalls by number)

Index (Messages)

Index (Messages by number)

Index (Services)

Index (Services by number)

Index (Vectors)

VideoV

Index (Vectors by number)

&2C

Index (SysVars)

Index (Entry points)

Index (Errors)

Index (Errors by number)

Index (VDU codes)

Index (TBox methods)

Index (TBox methods by number)

Index (TBox messages)

Index (TBox messages by number)