Ursula Window Manager Changes Functional Specification

Confidential Information

Document Ref:	1309,213/FS
Project:	Ursula
Revision:	G *** LIVE ***
Date:	24 March 1998
Author(s):	Ben Avison

Contents:

1. Overview

This specification builds upon the *Nested Window Manager Functional Specification* [1], and describes additional features to be added, plus the Ursula-specific features not mentioned therein.

Work will progress roughly in the order:

- Features required by other parts of Ursula (eg §4.1.1)
- Configuration
- Flicker reduction
- Everything else

3. Technical Background

A knowledge of the use of the Window Manager, including the extensions added up to the beta release Nested Wimp, v 3.98, is assumed. For the differences between the RISC OS 3.7x release of the Wimp (v 3.68) and the Nested Wimp, please see [1].

4. User Interface

Within each section below, work is presented in approximate order of priority.

4.1. Windows

4.1.1. Foreground Windows

The concept of *foreground windows* will be introduced. This is intended for use by screen saver applications, see [2].

To facilitate this, the windows within any given stack (ie each stack of sibling child windows, as well as the top-level window stack) will be segregated into three layers: foreground windows, normal windows and background windows. All foreground windows are kept above all normal windows, and all normal windows are kept above all backwindows. (The only exception to this rule is when a window is iconized, where it will be opened behind the Wimp's backwindow irrespective of its category.)

Opening a window behind handle -1 will open it at the front of the relevant category, and opening a window behind handle -2 will open it at the back of the relevant category.

Necessarily, backwindows (windows with window flag bit 11 set) will behave subtly differently from in the past, in that they can no longer be opened in front of a normal window. Note that for backwards compatibility reasons, opening a backwindow behind handle -2 will open it at the *front* of the backwindows.

Foreground windows will be identified by having window flag bit 23 set. This bit is already used by the Nested Wimp to flag furniture windows (ie that the window may encroach upon its parent's border), and therefore has a dual meaning for child windows. However, this is not a problem, because furniture windows should always be kept above their non-furniture siblings anyway (in the same way that the window border is normally kept "above" any child windows).

The error box will be made a foreground window, so that it may be opened on top of screensaver windows.

4.1.2. Error Box Auto-growth

Error report windows will automatically be widened until they are large enough to contain the error message. This is to support long pathnames.

A number of error box bugs will be fixed:

- Stretched error buttons will shrink back for their next use.
- Action buttons will always be seen to slab in (to match RISC OS 3.7 Wimp).
- No block-copying will be allowed when the error box is redisplayed at a different width (this was causing unnecessary flicker).
- The rightmost action button will always be the one activated by the Return key, and will always have a R6-type border.
- Escape will always select the second-from-right action button (except where there is only one, when Escape will duplicate the effect of Return). This will be consistent, regardless of whether the buttons are standard (Continue/Cancel), Describe or user-defined ones.
- Keypresses will have a consistent effect in "xxx may have gone wrong" error boxes, irrespective of the action buttons available in the error box that would be displayed if "Describe" were clicked.

Additionally, the screensaver will be deactivated when an error box is displayed (see [2]), the Escape character will be forced to &1B during error box display, and the buffer size for holding action button labels will be increased from 64 characters to 256 characters.

4.1.3. Iconise Button

An iconise button will be added to all windows, excluding child windows.

The iconise button will be configurable on/off on a system-wide basis; the default will be on. The configuration will be held in a CMOS bit.

4.1.4. Release-type Furniture Buttons

Window furniture buttons will be changed to have an icon button type of 4 (release). This better demonstrates the new depressed button code. This option will not be configurable.

4.1.5. Autoscrolling

The Wimp will offer to perform autoscrolling on behalf of aware tasks; for the API, see §5.7. The pre-scroll pause time (if appropriate to the type of autoscroll) will be configurable using CMOS, over a range from 0 to 1.5 seconds in steps of 0.1s, or from 2.0 to 15.0 seconds in steps of 1.0s.

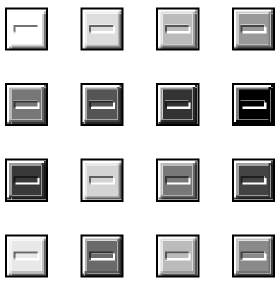
4.1.6. Window Furniture Colouring

One of the remaining cases of flicker in Wimp rendering is that of title bar redraw, most noticeable during window resize.

This is because the redraw is a three-pass process: first an area of plain colour is drawn (indicating whether the window has the input focus, for example), then a masked sprite is tiled across the title bar, and then the text/sprite icon contents are drawn on top.

This flicker will be reduced by reducing the process to two passes: non-masked sprite tiling and text/sprite plotting. However, we still need to represent the input focus, so this will instead be achieved by using a translation table to map the greyscale part of the tool sprite palette (except for scroll well/slider sprites) to an appropriate colour scale.

This approach also allows for Java 24-bit window colours to be implemented. As an example, here is the iconise button, recoloured, using a simple linear algorithm, to each of the standard Wimp colours:



4.1.7. Bugfixes

The following known bugs will be fixed:

- The title text clipping rectangle is incorrectly calculated for minimum-width windows with the "no borders" window flags set.
- If a window is opened with a minimum-x coordinate of -1, it thereafter leaves a trail behind it whenever it is moved or resized.

- Unnecessary whole-screen redraw can be observed when a window containing a reasonably large number of children is scrolled or re-sized, but only if some of them are partially or wholly outside the parent's visible area.

4.1.8. Sticky edges

Options already exist to prevent or allow windows from moving off the edges of the screen. A further option will be provided to adjust the Wimp's interpretation of "prevent windows from moving off-screen in this direction" to only apply to attempted window-opens where the edge overlaps the edge of the screen by an amount less than 64 OS units. The effect of this is to make the edges of the screen "sticky", making it easier to align windows neatly along the edge of the screen, without preventing them from being moved out of the way when required.

Only top-level windows will be affected by the sticky edges.

One extra CMOS bit will be necessary to flag "force on-screen" directions to be "sticky" directions instead.

4.1.9. Multi-window Closing and Iconising

A Ctrl-Alt-Select click on a close or iconise button will close or iconise all the open windows at that nesting level and (if applicable) having the same parent.

4.2. Icons

4.2.1. ROM Sprite Priority

The Wimp has two sprite pools, the ROM sprite pool and the RAM sprite pool; where sprites of the same name exist in both pools, the RAM pool sprite was used.

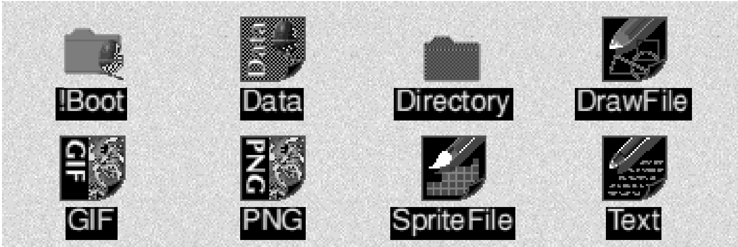
To enforce Ursula's New Look, the ROM sprites will take precedence over the RAM sprites. However, this would prevent deliberate wholesale look-and-feel redesign, so the relative priority of the two pools will be configurable (using another one bit of CMOS).

When ROM priority is in effect, any duplicate RAM sprites will be deleted, in order to save memory. This process will actually occur at the first sprite plot after the Wimp RAM sprite pool is altered, and may therefore result in a slight delay at this stage if there are a large number of duplicates to be deleted (eg if an attempt has been made to replace the entire set of ROM sprites).

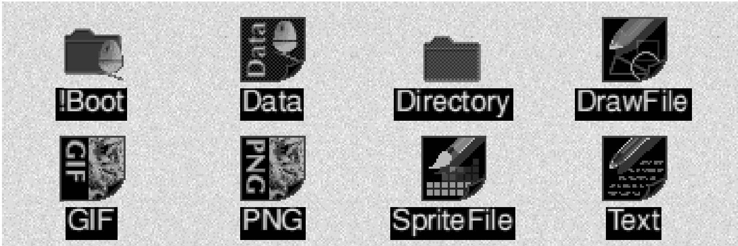
4.2.2. New Look Sprite Highlighting

The sprite colour inversion algorithm will be changed to better suit the 256-colour sprites designed for Ursula.

Old algorithm:



New algorithm:

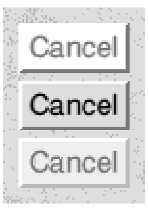


The old algorithm treated greys and colours differently, which causes problems where there is a blend from a grey to a colour, as is more common in pseudo-photographic images. The new algorithm is closely related to the photographic negative, but retains the hue of the colour.

4.2.2. Icon Rendering Consistency and Bugfixes

The rendering of icons has been added to gradually over the years, with the result that certain combinations of flags produce bugged or useless icons (for example, the anti-aliasing scale is wrong for shaded-and-selected icons; or, any attempt to produce a text-plus-sprite action button gives what is probably not what you required). Now, with the addition of 24-bit colour support for Java, we even have different results possible for each combination of icon flags, depending on whether the Wimp font is configured to be an outline font or not, and whether the colours are specified in 24 bits or not. This situation is documented in Appendix 1; the three old cases will be fixed and unified into the one new scheme also presented in the appendix.

Note also that shading of 24-bit-colour icons has not been considered so far: the icon shading algorithm involves EORing the (Wimp) colours of the icon with 2. Not only is this unsuitable for application to 24-bit colours, it wastes the expanded colour capability of Phoebe. Therefore, shading of icon colours will be changed to use a variation on the sprite-shading algorithm, whereby the colour range is "squashed" into a smaller colour space - in this case, between 50% grey and white. For example, the result of shading an action button will be as shown below:



- Appearance using old algorithm (new algorithm is similar in 16-colour modes)

- Appearance when not shaded

- Appearance using new algorithm (cf. shaded option buttons)

Work on Java icon colours is, in fact, not yet complete: the colours of selected action button backgrounds, and of the 3-D plinth sides, are not yet specifiable in 24-bit colour. This will also be addressed.

4.2.3. Kerning and Blending of Text

All Wimp text will be kerned. Blending of text in icons will be supported by a new validation string command.

Note, where a text-plus-sprite icon has text, the Wimp traditionally fills a rectangle behind the text for it to antialias with. This would make the blending command useless in these cases, so the rectangle will not be drawn when the blending command is present.

In fact, the filled rectangle is not very useful when an alternative highlighted sprite is supplied, as the text itself is never inverted. Therefore, the rectangle will be removed from this category of icon altogether. This will improve the appearance of option and radio buttons.

4.3. Icon Bar

4.3.1. Autofronting

This will be made configurable on/off (using one CMOS bit), and will have a configurable delay of 0 - 1.5 seconds in steps of 0.1s or 2.0 - 15.0 seconds in steps of 1.0s (using 5 CMOS bits).

Autofronting will be disabled while Wimp AutoScrolling is active (see §5.7).

4.3.2. Autoscrolling

The initial speed, and the acceleration rate, of icon bar scrolling will be made configurable. Each will be available as 0, 20, 50, 100, 200, 500, 1000 or 2000 units, where speed is measured in OS units s^{-1}, and acceleration is measured in OS units s^{-2}. In this way, the CMOS requirement is kept down to 6 bits.

(Note that the exact values may be subject to change, once we have a feel for what these values mean.)

4.3.3. Depth Constraints

The icon bar will only ever be either a "back object", or at the front of the window stack (excluding menu structures), irrespective of whether it has been moved by Shift-F12 or by autofronting. If the icon bar is at the front of the window stack and another window is opened in front of it, it will jump to the back of the stack and become a back object again.

4.4. Miscellaneous

4.4.1. Mouse Configuration

To make room in CMOS RAM for all the additional configuration options, the available ranges of mouse times (drag delay, double-click delay, submenu open and close) will be reduced to 5 bits (cf. autofronting and autoscrolling delays already described), and similarly for mouse distances.

4.4.2. Automatic Hourglass

The hourglass will be turned on at exit from Wimp_Poll, or when exiting from Wimp_StartTask back to the parent task. The hourglass will remain on until control passes through WrchV, or until the task calls Wimp_Poll again, whichever is sooner. (When WrchV is called, the routine removes itself from the vector so as not to unduly slow the redraw process; if necessary, it is put back on the vector the next time the autohourglass is turned on.)

The effect of this will be that when a task takes overly long to return control to the Wimp, the hourglass will be displayed. (The hourglass is automatically time-lagged, so it will not become visible for quick tasks.)

The time before the automatic hourglass starts will be 1 second - longer than the default for the hourglass. (This will also require a minor change to the Hourglass module.)

4.4.3. Windows '95 Keys

The pair of similarly-marked keys unique to "Windows '95" keyboards will be assigned Wimp keycodes, unless we decide to use them as shifting keys.

5. Programming Interface

In general: wherever a SWI's extended abilities are flagged by a "TASK" magic word, the appropriate register will be cleared on exit from the SWI. This will affect the implementations of Wimp_CloseDown (since RISC OS 2), and Wimp_OpenWindow, Wimp_GetWindowState and Wimp_ForceRedraw (in their Nested Wimp forms), and the new form of Wimp_DragBox (see below).

5.1. Wimp_OpenWindow (SWI &400C5)

Window flags bit 23 gains the foreground window meaning described in §4.1.1.

An attempt to open a window with the icon bar as its parent (window handle -2) will be faulted.

5.2. Wimp_Poll (SWI &400C7)

The Key_Pressed event will return new keycodes when a "Windows '95" key is pressed.

New scroll "directions" for the Scroll_Request event are defined for use by Wimp_AutoScroll (see §5.7).

5.3. Wimp_DragBox (SWI &400D0)

If R2 is "TASK", then R3 will become a flags word. This is primarily to support autoscrolling.

On entry:

R3	Flags:	Bit 0 set	=>	Keep fixed end of rubber-box drags in line with work area of the window whose handle is given in [R1,#0]
		Bit 1 set	=>	Clip the dragbox so that it is only displayed within the work area of the window whose handle is given in [R1,#0]
		Other bits are reserved and must be clear.		

On exit:

R0 corrupted.

If R2 was "TASK" on entry, it is cleared.

All other registers preserved.

5.4. Wimp_ReadSysInfo (SWI &400F2)

A new reason code will be added: if R0 is 16 on entry, then on exit R0 points to the low-priority Wimp sprite pool, and R1 points to the high-priority sprite pool.

This is necessary in order to properly support the variable sprite pool priority feature (see §4.2.1) in the time period between when new sprites are added to the Wimp's RAM sprite pool with *IconSprites, and when duplicate sprites are deleted from the Wimp's RAM sprite pool (which does not occur until a sprite is next plotted).

For example, some applications might want to locate a sprite in memory, in order to read from it, prior to plotting it. This would have been done by using Wimp_BaseOfSprites, then trying the returned sprite area in R1 followed by the one returned in R0 if the sprite could not be found in the first sprite area. You should now use this instead:

```
MOV     R0, #16
SWI     XWimp_ReadSysInfo
SWIVS   XWimp_BaseOfSprites     ; support old Wimps
STMFD   R13!, {R0}              ; remember low-priority pool, in case
...                             ; the sprite we want isn't in the
...                             ; high-priority pool
...
LDMFD   R13!, {R1}
...
```

5.5. Wimp_RegisterFilter (SWI &400F5)

The window handle will be correctly passed to GetRectangle filters (see [3]).

5.6. Wimp_Extend (SWI &400FB)

A new reason code, 11, will be introduced.

On entry:

R0	11
R1	Pointer to 100-byte block
[R1,#0]	Window handle, or 0 to return generic values for a window with line borders enabled

On exit:

R0, R1	Preserved
[R1,#0]	Preserved
[R1,#4]	Left border width (1 pixel, or 0 if absent)
[R1,#8]	Bottom border width (horizontal scroll height, or 1 pixel if no scroll, or 0 if absent)
[R1,#12]	Right border width (vertical scroll width, or 1 pixel if no scroll, or 0 if absent)
[R1,#16]	Top border width (title bar height, or 1 pixel if no title, or 0 if absent)
[R1,#20]	Back width, or 0 if absent
[R1,#24]	Close width, or 0 if absent
[R1,#28]	Reserved for forwards compatibility - will be 0 for Ursula Wimp
[R1,#32]	Title width, or 0 if absent (corrupt if no window handle was given on entry)
[R1,#36]	Reserved for forwards compatibility - will be 0 for Ursula Wimp
[R1,#40]	Iconise width, or 0 if absent (ie. no close, or configured off, or a child window)
[R1,#44]	Toggle-size width, or 0 if absent
[R1,#48]	Toggle-size height, or 0 if absent
[R1,#52]	Vertical scrollbar upper-gap, or 0 if no vertical scrollbar
[R1,#56]	Up-arrow height, squashed if necessary, or 0 if no vertical scrollbar

[R1,#60]	Vertical scroll well height, or 0 if no vertical scrollbar (corrupt if no window handle was given on entry)
[R1,#64]	Down-arrow height, squashed if necessary, or 0 if no vertical scrollbar
[R1,#68]	Vertical scrollbar lower-gap, or 0 if no vertical scrollbar
[R1,#72]	Adjust-size or blank height, or 0 if neither
[R1,#76]	Adjust-size or blank width, or 0 if neither
[R1,#80]	Horizontal scrollbar right-gap, or 0 if no horizontal scrollbar
[R1,#84]	Right-arrow width, squashed if necessary, or 0 if no horizontal scrollbar
[R1,#88]	Horizontal scroll well width, or 0 if no horizontal scrollbar (corrupt if no window handle was given on entry)
[R1,#92]	Left-arrow width, squashed if necessary, or 0 if no horizontal scrollbar
[R1,#96]	Horizontal scrollbar left-gap, or 0 if no horizontal scrollbar

All measurements are in OS units; the size of a pixel is determined by the current screen mode. If [R1,#0] is 0 on entry, the sizes of all furniture items will be given assuming they are present and unsquashed (except the iconise button, if it has been configured off), while the title width and scroll well lengths are undefined.

The title bar and scroll well lengths include the 1-pixel borders at each end; all the other sizes include the width of one of the two borders. (This is because each separating line is shared by two pieces of furniture, except for the ones at each end.)

Note that scroll wells can be negative in length, eg if a furniture window obscures the entire scrollbar.

Also note that summing [R1,#20] through [R1,#44] gives the window outline width (provided there is a title bar), and similarly for the other sides of the window. The window outline width and height may also be calculated by adding the border widths/heights at [R1,#4] - [R1,#16] to the visible area width/height.

For example, to open a window so that the title bar is a given length, take the required length, add on [R1,#20], [R1,#24], [R1,#28], [R1,#36], [R1,#40] and [R1,#44], subtract [R1,#4] and [R1,#8], and use the result as the visible area width of the window.

5.7. Wimp_AutoScroll (SWI &400FD)

On entry:

R0	Flags:	Bit 0 set	=>	Enable horizontal autoscrolling
		Bit 1 set	=>	Enable vertical autoscrolling

Bit 2 set => Delegate actual window repositioning to task (ie. send Scroll_Request events rather than Open_Window_Requests)

Bit 7 set => Just read the current states of all flags

Other bits are reserved and must be clear.

If R0 bits 0 or 1 are set,

R1 Pointer to block:

[R1,#0] Window handle of window to scroll

[R1,#4] Left pause zone width (in OS units), eg. 80, ignored if flag bit 0 is clear

[R1,#8] Bottom pause zone height (in OS units), eg. 80, ignored if flag bit 1 is clear

[R1,#12] Right pause zone width (in OS units), eg. 80, ignored if flag bit 0 is clear

[R1,#16] Top pause zone height (in OS units), eg. 80, ignored if flag bit 1 is clear

[R1,#20] Pause duration (in centiseconds), or 0 for no pause, or -1 for configured pause length

[R1,#24] State change handler:

0 => none

1 => use inbuilt Wimp pointer shape routine

Other values are interpreted as a pointer to a routine.

[R1,#28] Value to pass in R12 to state change handler routine (if applicable)

On exit:

R0 Bits 0-2 are preserved or read, according to bit 7, and also:

Bit 7 is cleared

Bit 8 set => Autoscrolling has commenced (this bit is set after a pause is completed, and remains set when a scroll has "hit the stops", but is cleared again when the pointer re-enters the centre zone)

Bit 9 set => Pointer is outside the window's visible area

Bit 10 set => Pointer is within one or two pause zones

Bit 11 set => Pointer is within the centre zone

Bit 12 set => Pointer is left of the centre zone

Bit 13 set => Pointer is below the centre zone

Bit 14 set => Pointer is right of the centre zone

Bit 15 set => Pointer is above the centre zone

Bit 16 set => There is work area to the left of the visible area (always cleared if bit 0 is clear)

Bit 17 set => There is work area below the visible area (always cleared if bit 1 is clear)

Bit 18 set => There is work area to the right of the visible area (always cleared if bit 0 is clear)

Bit 19 set => There is work area above the visible area (always cleared if bit 1 is clear)

Other bits must be ignored.

If R0 bit 7 was set on entry, and R0 bits 0 or 1 were set on exit, then the block at R1 is updated on exit, to hold the current autoscroll parameters.

Any state change handler must not be held in application space, as the Wimp will not page in the relevant application before calling it. This means it must be in RMA. It is called as follows:

On entry:

R0 New autoscroll state (bit meanings as for R0 on exit from Wimp_AutoScroll)

R1 Previous autoscroll state (bit meanings as for R0 on exit from Wimp_AutoScroll)

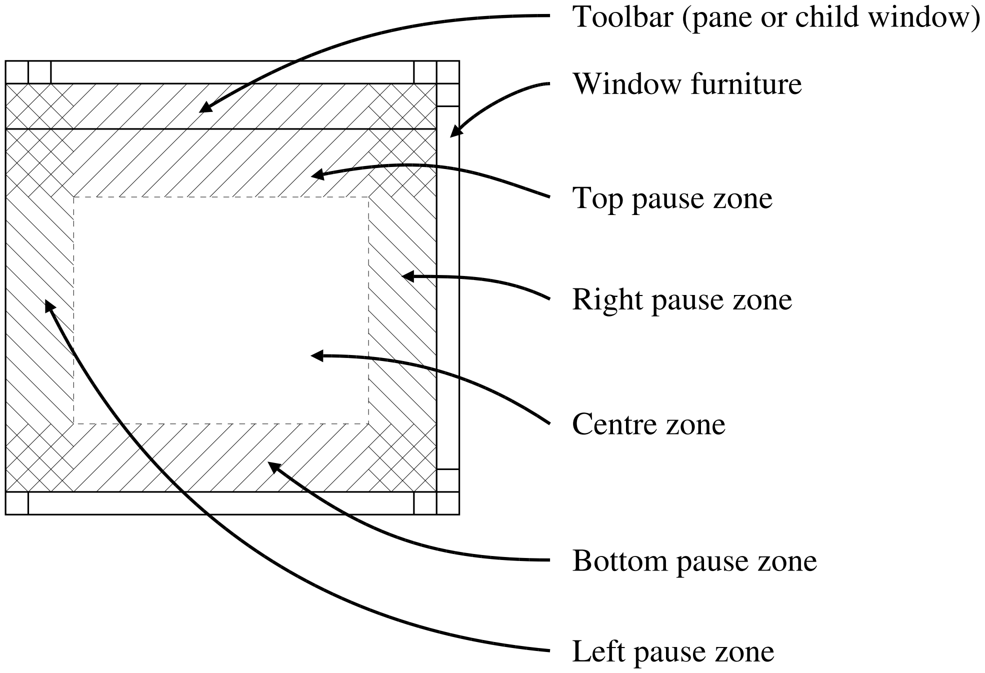
R12 Workspace pointer, as passed to Wimp_AutoScroll in [R1,#28]

Processor is in SVC mode.

On exit:

All registers must be preserved.

This is how the *pause zones* and *centre zone* are defined for a typical window:

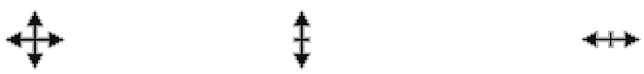


In the most general case, autoscrolling works in an environment where the user may drag an object out of a window (by moving the pointer smoothly over the boundary), or drag the object to elsewhere in the same window's work area; the autoscrolling is necessary to reveal hidden parts of the work area, and is activated by pausing the pointer briefly over one of the pause zones. Once autoscrolling has activated, the speed of scrolling is proportional to the distance of the pointer from the inside edge of the pause zone, and the pointer may even be moved outside the window's visible area for even faster scrolling, if so desired. If the pointer is outside a corner of the centre zone (eg the cross-hatched areas above), the scrolling is performed in two directions at once.

Other types of autoscrolling may be seen as specialised cases of the above. For example, a pause time of 0 is required when a rubber dragbox is used to make a selection (dragging out of the window has no meaning here).

A typical task would call Wimp_AutoScroll 1, 2 or 3 (as appropriate to the circumstances) when starting a drag, and then call Wimp_AutoScroll 0 when finishing it (ie when a User_Drag_Box event or an Escape keypress are received). However, there are circumstances (eg. when implementing scrolling toolbars, or when following the RISC OS drag-and-drop protocol) where calls to the SWI may be necessary at other times.

The Wimp will handle autoscrolling for the given window, including detection of entering and leaving the various zones, and timing any pause of the pointer. It will also normally be responsible for prompting the task to re-open of the window, with new scroll offsets, at regular intervals during any scroll, and also for reprogramming the pointer shape when necessary. For simplicity, the Wimp will simply use one of the pointers shown below (programmed into pointer number 1), then restore the previous pointer number and colours when appropriate. If the previous pointer number was 1, it will be reprogrammed to ptr_default at such points.



In order to prevent undesired further pointer reprogramming, the pointer will be (internally) considered to have left all windows when Wimp_AutoScroll is called, and won't return until Wimp_AutoScroll 0 is called. This is equivalent to the behaviour during Wimp_DragBox, and is for the same reason. As a result, a Pointer_Leaving_Window event will be sent soon after Wimp_AutoScroll is called; if the task wishes to use a special pointer during the time when the Wimp's autoscroll pointers are not in use, it must therefore ignore the Pointer_Leaving_Window event, and instead restore pointer number 1 immediately after calling Wimp_AutoScroll 0. If the pointer is still over the relevant window, the task will shortly receive a Pointer_Entering_Window event again.

Bit-2-set autoscrolls will work like this: when the Wimp detects that scrolling should start, it will send a Scroll_Request with both x and y scroll directions of 0. This allows the task to initialise any necessary timers. At subsequent intervals, it will send a Scroll_Request with one or both scroll directions set to ±3, as appropriate to the required scroll direction. The task must adjust the returned scroll offsets as required, then call Wimp_OpenWindow, as usual.

Autofronting and autoscrolling of the icon bar will be disabled while a Wimp_AutoScroll state is in effect.

5.8. UpCalls

UpCall 1 and 2 strings will be accepted correctly if they are terminated by ASCII 0, 10 or 13, rather than just the (strictly correct) 0.

The Wimp will also support UpCall 19 (MediaSearchEndMessage), which is like UpCall 4, but will update the message in the error box with the string pointed to by R1 prior to removing it. This is useful in cases where it may still be some time before the screen is updated (notably during printing).

6. Data Interchange

(See the *Ursula Pinboard Changes Functional Specification* [4].)

7. Data Formats

No new data formats are introduced.

8. Dependencies

The Ursula Wimp memory management changes and cached screen support are designed to interact with the Ursula kernel. GetRectangle filters requires the Ursula Filter Manager. Correct functioning of the automatic hourglass requires the Ursula version of the Hourglass module. ROM sprite priority requires the Ursula DragASprite module.

15. References

[1]: Document Ref 1215,401/FS: *Nested Window Manager Functional Specification*

[2]: Document Ref 1309,215/FS: *Ursula Screensaver Functional Specification*

[3]: Document Ref 1309,217/FS: *Ursula Filter Manager Changes Functional Specification*

[4]: Document Ref 1309,210/FS: *Ursula Pinboard Changes Functional Specification*

[5]: Document Ref 1309,200/FS: *Ursula Top Level Functional Specification*

[6]: Document Ref 1309,216/FS: *Ursula Drag-and-Drop Functional Specification*