Ursula !Configure Changes Functional Specification

SUBJECT TO CHANGE WITHOUT NOTICE

Document Ref:	1309,214/FS
Project:	Ursula
Revision:	C *** LIVE ***
Date:	2 February 1998
Author(s):	Ben Avison
Change:	Revised following second review and majority of implementation.

Contents:

1. Overview

This document covers the changes required to the *Configure* application in Ursula over the version present in RISC OS 3.71. The application has remained largely unchanged since it was revamped for RISC OS 3.50, the last time there was a significant hardware change. Since the Phoebe hardware will again be radically different, with different configuration needs, we will take the opportunity to perform a major overhaul on the application. The proposed changes can be roughly categorized thus; priority for each aspect which gets the go ahead will be roughly equal, except for the plug-in support, which is a pre-requisite of the plug-ins:

- Introduction of modularity (along the lines of the current *InetSetup*, but more seamlessly, and without the hard-coded links between applications)
- Support for new hardware and software (hardware autodetection, the screen saver, enhancements to the Pinboard and the Wimp, and changes to the keyboard and mouse drivers)
- New system configuration sections (the boot sequence and the Filer)
- Changes to current behaviour (System and Fonts merge code)
- User interface consistency - so all gadgets are Style-guide compliant, and all windows have Set and Cancel buttons, rather than just some of them

The intention is to provide an application that is similar enough to the old *Configure* so as not to frighten users who have upgraded, but different enough to embrace all the new configuration options in Ursula without producing an unintuitive user interface.

2. Outstanding Issues

2.1. Discs and Sound Windows

The exact details of these windows are pending stabilisation of the relevant parts of the OS.

3. Technical Background

It is assumed that the reader understands the Risc PC boot structure and is familiar with conventional configuration procedures using *Configure* and `*Configure`. Since the issues raised are primarily GUI ones, any relevant technical background is covered alongside the appropriate GUI issue.

4. User Interface

4.1. User Interface in General

It is important that we resolve the inconsistency of user interface that currently exists: some windows (all of which are really dialogue boxes) have remained in a pseudo-RISC OS 2 style, with instant effect buttons and close icons; others comply with the Style Guide and have separate "Set" and "Cancel" action buttons at the bottom of the window.

To be consistent, and to sidestep the difficulties of implementing adjust-close behaviour to match the old *Configure*, windows will, in general, be given "Set" and "Cancel" buttons, and where appropriate, a "Default" one to restore sensible factory defaults. The "Help" button, currently in *InetSetup*, which starts up the *Help* application, will be relocated to an item on the main window's menu, since all the others windows have interactive help messages as well, which may not be obvious to the user.

Mouse button	Action button	Action to be taken
Adjust	Set	Configure the current selections
Select	Set	As above, but also close the window
Adjust	Cancel	Reset the selections to the configured values (ie reset the window to the way it was when last Adjust-Set or when it opened, whichever is more recent)
Select	Cancel	Just close the window - next time it is opened, it will be displaying the configured values anyway
Adjust	Default	Reset the selections to the delete-power-on defaults for Ursula (note these may differ from the defaults for the same modules soft-loaded on a pre-Ursula machine)
Select	Default	Selecting the defaults without configuring them, and then closing the window, would just appear to duplicate the Cancel button, so instead this will be the same as an Adjust click on "Default"

Each window will be handled by a separate plug-in application, and each plug-in will be written to use the User Interface Toolbox. This will resolve the past inconsistencies between *Configure* and the Style Guide.

All windows, when first opened, must gain the input focus so that Return and Escape keypresses can be acted upon; if the window has no writable icons, the window must gain an invisible caret. The "Set" action button (or equivalent) must in all cases have a "default action button" (cream well) 3D border to indicate the action to be taken on a Return keypress.

When an already-opened window is forced to the front of the window stack by a click on its icon in the main window, it will simply be reopened at the front of the window stack, without altering the selections therein. (This resolves an inconsistency in the old *Configure*).

Interactive help messages will be standardised across all plug-ins. The text we can use is governed by the Toolbox's handling of interactive help; in particular, only one help message can be assigned per gadget (which may consist of many icons, and in the case of a string set, a menu as well), and gadgets stop providing help when they are faded.

Wherever possible, the interactive help messages shall take one of these forms, where "***" should be replaced with whatever is appropriate under the circumstances, and "|M" is the usual sequence telling !Help to insert a line break:

Plug-in activation icon (in main window)	Click SELECT to open the *** configuration window.
Plug-in work area	This window allows you to configure various aspects of the ***.
Preceding labels	This is the ***.
Numeric ranges (writable type)	You can adjust the *** using either the keyboard or the arrows.
Numeric ranges (display type)	You can adjust the *** using the arrows.
Writable icons	You can enter the *** here.
String sets	You can choose a *** from the menu.
Sliders	You can adjust the *** using the slider.
Following labels	The *** is measured in ***. *[Use the non-abbreviated unit name, and follow by any necessary explanation of the unit, within () if fairly brief.]*
Option buttons	This indicates if ***.|MClick SELECT to select or deselect this option.
Radio buttons	This indicates if ***.|MClick SELECT to select this alternative.
Drop zones	You can drop a *** here.
"Try" action buttons	Click SELECT to sample the selected ***.
"Default" action buttons	Click SELECT to reset the selections to the factory defaults.
"Cancel" action buttons	Click SELECT to close this window without configuring the selections.|MClick ADJUST to reset the selections to the configured settings.
"Set" action buttons	Click SELECT to configure the selections and close this window.|MClick ADJUST to configure the selections without closing this window.

4.2. Changes Sorted by Section

The following changes have been suggested by people within Acorn. Some are based upon various public domain configuration utilities, and some are just a GUI for existing or new command line or CMOS configuration options.

4.2.1. Main Window

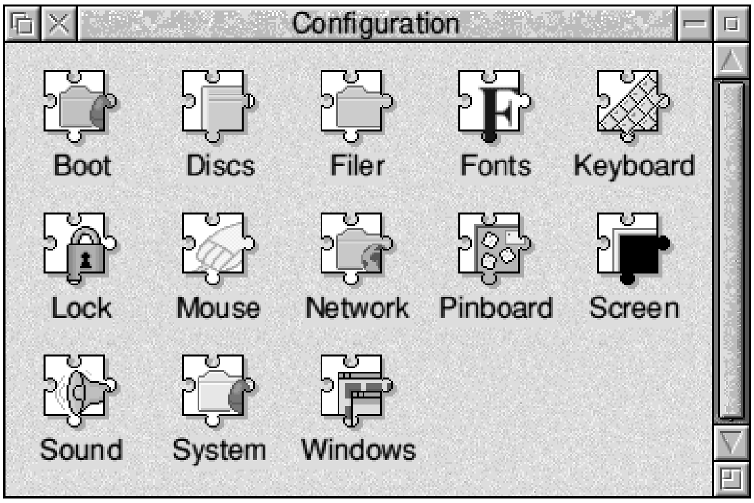
The main window will have to adjust itself to compensate for any available plug-ins - the plug-ins will be sorted by alphabetical order of the plug-in's descriptive text. The dimensions when the window is first opened will allow for up to 5 columns of icons, and up to 4 rows of icons; any more and the work area will extend below the visible area. The window will be resizable similar to a directory display.

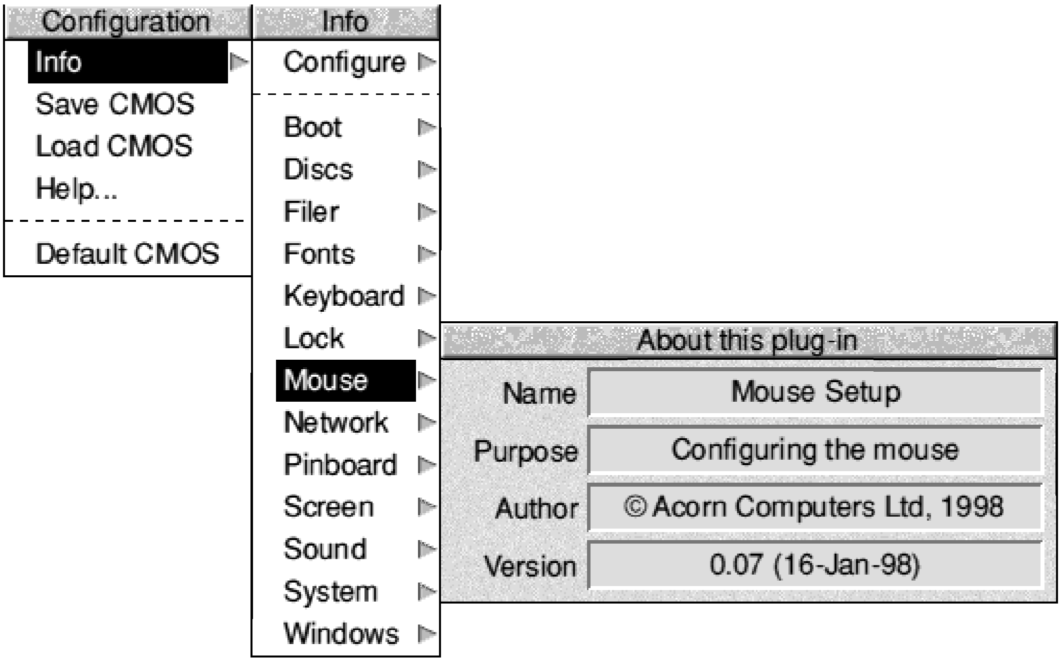
The menu available from the Main window will lose the "Quit" item. The "Info" item will lead to a submenu, offersing info boxes for each of the top-level plug-ins, as well as for the main application. The "Save" submenu will be replaced by "Save CMOS" and "Load CMOS" items for saving and restoration of the CMOS RAM to and from a fixed location, <Choices$Write>.CMOS. Also added will be a "Default CMOS" item, for loading CMOS RAM from the file Choices:ResetCMOS (supplied by Acorn), and a "Help..." item to initiate interactive help.

The !Boot file of *Configure* will no longer claim the RunType for CMOS files, as the old technique of setting recognised CMOS bytes is no longer appropriate.

When <Choices$Write> is on a FSLocked filing system, or when the machine has been remotely booted and <Choices$Write> is not publicly writable, all the main window's icons will be faded. However, when FSLock is active, the Lock icon will not be faded, so that the user has a chance to unlock the machine. The three CMOS menu items will be faded when read and write operations from and to the relevant CMOS files are impossible, eg due to FSLock.

The resulting window and menu structure will look like this:





Help messages:

Work area, case 1	Configure allows you to control various preferences of operation and startup.|MClick SELECT on the icons to open the individual configuration windows.
Work area, case 2	Configure allows you to control various preferences of operation and startup.|MThe icons are faded because the hard disc is locked.|MClick SELECT on the Lock icon if you wish to unlock the hard disc.
Work area, case 3	Configure allows you to control various preferences of operation and startup.|MThe icons are faded because you do not have permission to save your choices on the remote machine.
Icons in work area	*These are determined by the relevant plug-in's Messages file*
Menu item 1	Move the pointer right to view information about the configuration system.
Menu item 2, case 1	Click SELECT to save all the CMOS RAM as a file, from where you can restore it later.
Menu item 2, case 2	You cannot save the CMOS RAM while FSLock is active.
Menu item 3, case 1	Click SELECT to load all the CMOS RAM from your saved CMOS file.
Menu item 3, case 2	You do not have a CMOS RAM file to load from.
Menu item 3, case 3	You cannot load the CMOS RAM while FSLock is active.

Menu item 4	Click SELECT to load interactive help, for guidance on the use of Configure.
Menu item 5, case 1 (cases 2/3 as item 3)	Click SELECT to restore all the CMOS RAM to the factory default settings.

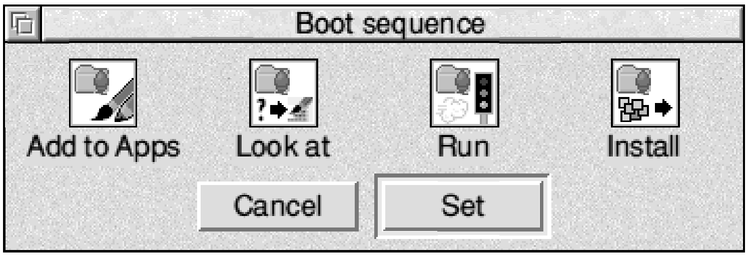
4.2.2. Boot Windows

Various aspects of the boot sequence will be addressed in *Configure*. The user will be able to specify additional items to be added to Resources:$.Apps, booted and/or run during the boot sequence, and a generalised installation facility (a superset of the *SysMerge* and *FontMerge* facilities) be provided.

The boot and run facilities will replace part of the functionality of the Desktop Boot File (<Choices$Write>.Boot.Tasks.!Boot), so the Task Manager will be modified to filter out, during saving of a Boot File, all commands that can be identified as requests to run or boot files or applications (lines beginning with "/", or where the first word is "Run", "Filer_Run", "Filer_Boot" or contains a "." or ":"). This will prevent the problems often experienced under current behaviour, where a given program is run twice, once by PreDesktop or Desktop, and once by the Desktop Boot File - typically causing a string of error messages on startup, complaining that "Alarm is already running" or similar.

4.2.2.1. The Main Boot Window

To display all the information, we will provide a main Boot window and a number of windows leading from it to handle different aspects of functionality (in a similar style to InetSetup). The main window will look like this:



The subsidiary windows will actually be second-level plug-ins, to allow them to be supplied separately if that becomes necessary in future. Communications with these will be via a variation upon the main application's plug-in protocol.

The "Cancel" button will cause all the subsidiary windows to be closed (they will have an opportunity to complain if they have modified data, the same as with top-level plug-ins). Unless Adjust was used, the main boot window will be closed as well.

The "Set" button will have the same effect, but will tell the plug-ins that they should save any modified data, rather than complaining.

Help messages:

Add to Apps	Click SELECT to open the window allowing changes to the applications added to the Apps folder at startup.
Look at	Click SELECT to open the window allowing changes to the applications which will be seen by the Filer at startup.
Run	Click SELECT to open the window allowing changes to the files and applications which will automatically be run or loaded at startup.
Install	Click SELECT to open the window allowing you to merge new information into the boot sequence.|MYou will need to have a "skeleton" !Boot application to provide the information.

4.2.2.2. "Add to Apps", "Look at" and "Run" Boot Windows

These will be achieved using scrolling list-based windows. The scrolling list will contain the leafnames for the action (AddApp, boot or run, depending on the window in use) to be performed upon. Directories will also be valid leafnames, referring to directories full of objects for the action to be applied to using the Repeat command.

The lists will be sensitive to files, applications or directories being dropped on to them, drags within them (with autoscrolling of the pane) to change the order, and clicks on the accompanying "Remove" action button. "Select all" and "Clear selection" buttons will also be processed.

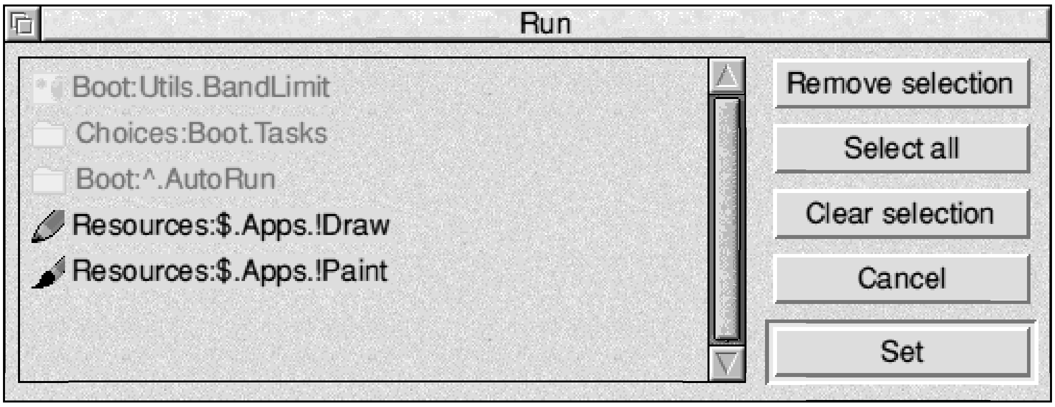
The three different windows will save the chosen commands in one of these, as appropriate:

- "Add to Apps": in <Choices$Write>.Boot.PreDesktop, at the end of the Acorn entries in the "ResApps" section;
- "Look at": in <Choices$Write>.Boot.Desktop, in the Acorn part of a new section named "Boot", to come immediately after the "Auto tasks" section;
- "Run": in <Choices$Write>.Boot.Desktop, in the Acorn part of a new section named "Run", to come immediately after the "Boot" section, or after "Auto tasks" if "Boot" does not exist.

The insertion of the data into these files will be handled by the new Installer module (see §5.2).

Any pre-existing commands in either <Choices$Write>.Boot.PreDesktop or <Choices$Write>.Boot.Desktop that look like suitable commands, but which are not bounded by the appropriate `|Start Acorn Configure x.xx` *`section`* and `|End` comments will be included in the list, but will be faded, and will not be selectable or edittable through the GUI. This is to ensure, for example, that ADFS::4.$.Apps always gets AddApped, and for the use of system administrators, for example to install network applications. This way, users are reminded that they are augmenting the existing applications and files in the list, not replacing them. New items cannot be placed between or before the early faded items (if any), nor between or after the late faded items (if any).

Non-faded items will always be listed in the window in their fully-canonicalised forms, but in their stored form, they will be converted to be relative to Boot: or Boot:^ if possible.



Help messages for "Run" boot window (the others will be the same, but without the references to files):

Remove selection	Click SELECT to remove all the selected files, applications and directories from this window.
Select all	Click SELECT to select all the files, applications and directories.
Clear selection	Click SELECT to deselect all the selected files, applications and directories.
Pane work area	Drag files, applications and directories to this window to add them to the list.
Un-faded items in pane	Drag this around within the window to change its position within the sequence.|MDrag other files, applications and directories to this window to add them to the list.

4.2.2.3. "Install" Boot Window (*BootMerge*)

This will be a generalised facility for installing anything into !Boot, similar in function and user interface to the font installation window and the System window (for more information, see §4.2.13).

The dropped item must be named and structured the same as !Boot. The directory will be scanned for any type of file, and they will then be merged into objects at the same location within <Boot$Dir>. Backups and logs will be kept in <Boot$Dir>.Backup and <Boot$Dir>.Log, unless the destination file falls within <System$Dir>, in which case the *SysMerge* locations will be used. The <Choices$Write>.Boot.PreDesktop and <Choices$Write>.Boot.Desktop files will also be treated as special cases; these will be processed using `*Install_Merge` rather than `*Install_Update`.

4.2.3. Discs Window

Design is pending the determination of auto-detection techniques.

This window will include the functionality of the old Discs and Floppies windows (if needed) except for the IDE spindown delay, which will no longer be configurable. The initial RAM disc size will be configurable here, as will ADFSDirCache, ADFSbuffers and CDROMbuffers.

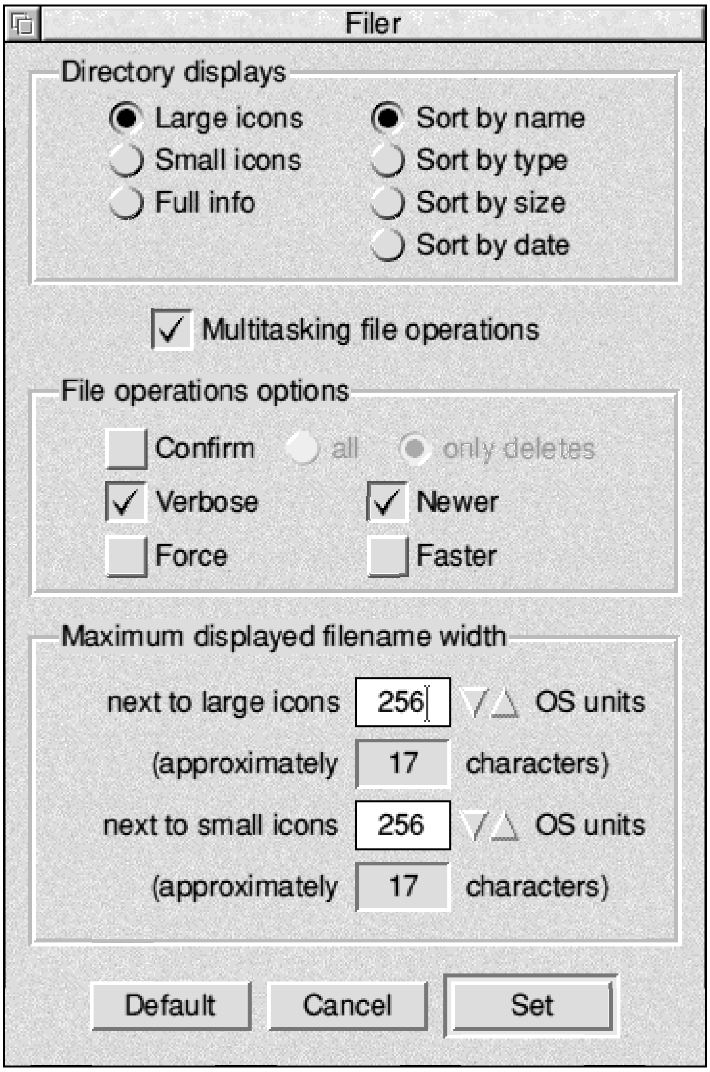
4.2.4. Filer Window

The Filer's various options will be made available for configuration. The options will no longer be held in CMOS, so they will only be altered from the hard-coded defaults when <Choices$Write>.Boot.Tasks.Filer (a file created by this plug-in) is run as part of the boot sequence; options changed from a Filer menu will be forgotten at each reset.

All the configurable enhancements to the Ursula Filer will be catered for, including confirm-only-deletes, remembered "Faster" and filename truncation. The truncation lengths will always be held in the <Choices$Write>.Boot.Tasks.Filer file as OS units, as that is what is required by the * command to set the lengths; lengths from 0 to 4096 OS units will be configurable in units of 4. However, the approximate equivalent size in characters will be displayed alongside for guidance; a character width for this purpose will be taken as the mean of the widths of the "i" and "m" characters in the current desktop font.

The "Interactive file copying" option button will be moved here from the old window manager configuration window, since it is more appropriate here; it will be renamed to "Multi-tasking file operations" to better describe its function.

The Filer window will look like this:



The default values will match the hard-coded Filer defaults, currently 256 OS units in both cases, but this is subject to change.

4.2.5. Fonts Window

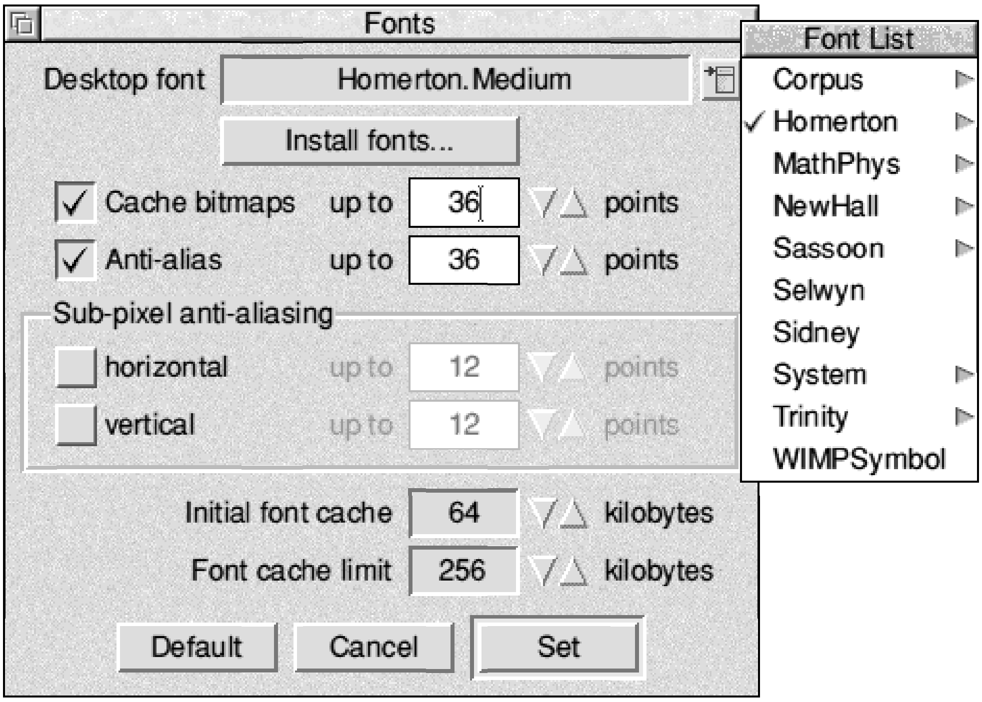
The FontSize and FontMax memory settings will be transferred from the old memory window.

The "Install fonts..." action button will actually activate a second-level plug-in, *FontMerge*, similar in function and user interface to the "Install" boot window and the System window (for more information, see §4.2.13), but replacing the old *FontMerge* utility.

The dropped item must be named and structured the same as !Fonts. The directory will be scanned for fonts in any subdirectories (ie excluding the !Run file etc.), and they will then be merged into objects at the same location within the third-last directory on <Font$Path>. Backups and logs will be kept in <Boot$Dir>.Backup and <Boot$Dir>.Log. A font will be recognised as any collection of files in the same directory, provided one of their names begins "IntMet".

The atom of information for display to the user will be individual styles and weights of a typeface (so we will get options like "Update face Homerton.Medium" or "Add face Homerton.Medium.Oblique"). With the advent of large directories, FontMerge will no longer worry about directing fonts into multiple font directories; one single !Fonts directory is easier to maintain, and faster to access.

The Fonts window will look like this:



Default settings will be unchanged from RISC OS 3.71.

When the !Fonts messages file is updated by the *FontMerge* window (or even the *BootMerge* window), the Installer module will attempt to make a sensible choice about the default face for each new font family - in order of preference: none (ie "Regular"), Medium, Book, Light, Demi, Bold, Heavy, Black, then Oblique and Italic versions of the same.

4.2.6. Keyboard Window

A string set will be added to provide a graphical interface to the *Keyboard command, listing the drivers in alphabetical order; the plug-in will add the appropriate command to <Choices$Write>.Boot.Tasks.Keyboard (to be executed as part of the boot sequence). The default keyboard for the currently loaded territory will be indicated on the pop-up menu of keyboards by an appended "(Default)". This will also provide an interface for the user-requested PC-style delete behaviour, which will be implemented via an alternative UK keyboard driver.

There will also be a change of wording in the window to indicate the meaning of "Shift caps" - it dates back to BBC micro behaviour, and is unlikely to have obvious meaning to the average user.

The Keyboard window will now look like this:

The keyboard driver selected by the "Default" action button will be the default keyboard driver for the currently-configured territory. Other default settings will be unchanged from RISC OS 3.71.

4.2.7. Lock Window

This window will be extended to provide current status information, and to make it clearer which fields need to be filled in for a particular operation. It will now check that the twin "New password" fields match before applying one of them!

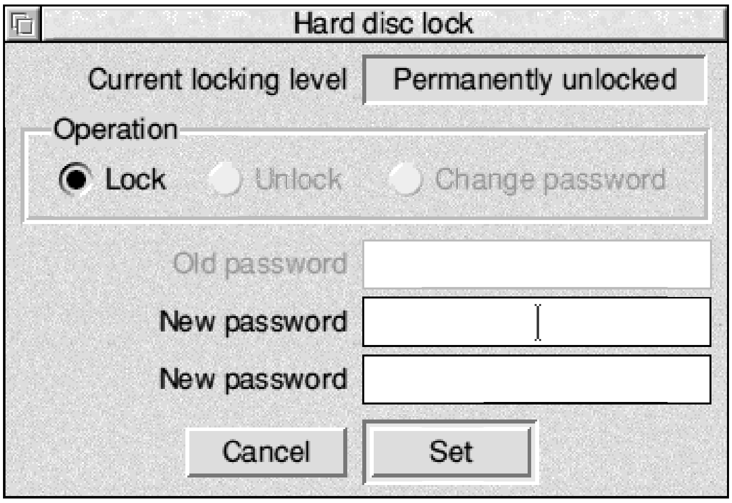
From lock status 0 (displayed as "Permanently unlocked"), only the lock operation will be available. From lock status 1 (displayed as "Temporarily unlocked"), all three operations will be available (unlock causing the unlocking to become permanent). From lock status 2 (displayed as "Locked"), unlock and change-password operations will be available (the former only causing a temporary unlock).

For a lock operation, the new password icons will be unshaded, unless the current lock status is "temporarily unlocked". For an unlock operation, only the old password icons will be unshaded. For a change-password operation, both the old and new password icons will be unshaded.

The passwords will be displayed as minus signs. If an action button is clicked upon with Adjust, the contents of all writable icons will be cleared, to make it more difficult to reach a completely unlocked state from a locked one. This will also be a more intuitive way of doing a complete unlock than changing to a null password.

The filing system to use when initially enabling a lock will default to the filing system the machine was most recently booted from (unless it was remotely booted, in which case it will default to ADFS). A simple way to achieve this is by use of the filing system CMOS byte and the system variable Boot$Remote.

The window will look like this:



4.2.8. Mouse Window

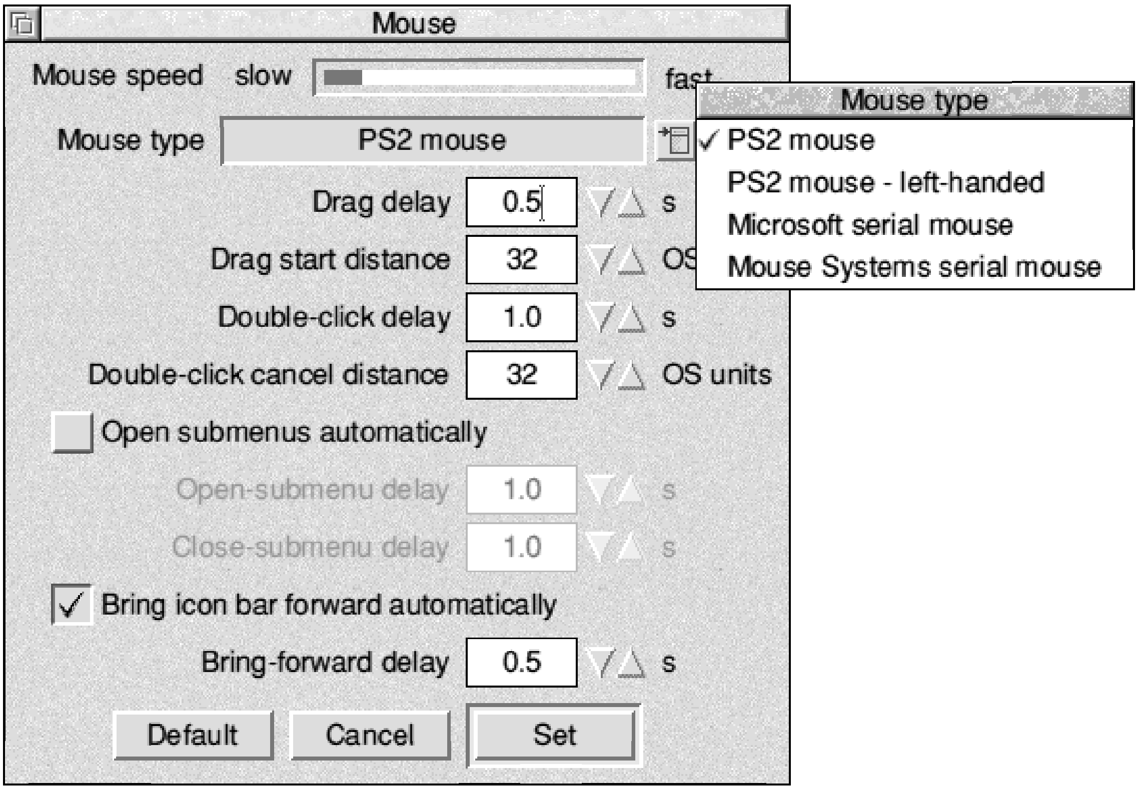
The mouse speed will be adjustable via a slider, and the maximum configurable multiplier will be increased to 9.

The PS/2 mouse driver is to be modified to support left-handed mouse buttons. This will appear as a separate mouse driver, and will as such be automatically listed on the menu. Standard quadrature mice will not be supported by the Phoebe hardware, and so will not have a mouse driver and therefore not appear on the mouse type menu.

The configuration options WimpAutoMenuDelay and WimpMenuDragDelay will be made available; it makes sense to place them with the other mouse delay/distance options in the Mouse window. This will mean that to be sensible, the "open submenus automatically" option has to be moved from the Windows window, and the two new delay options be greyed out as appropriate.

Similarly, the auto-fronting icon bar is to be made configurable. Whether the feature is enabled or disabled will be configurable separately from the delay time, in order to facilitate a delay of zero time.

The Mouse window will look like this:



Default settings will be unchanged from RISC OS 3.71, except that the default mouse driver will be the PS2 driver. The icon bar will be auto-fronting by default, with a default delay of 1/2 second.

4.2.9. Network Windows (*InetSetup*)

InetSetup changes are described in its own separate functional specification [3]. One of those changes will be to make it compliant with the *Configure* plug-in protocols.

4.2.10. Pinboard Window

Pinboard configuration will be separated from monitor and screen saver configuration due to the large number of added configurable features.

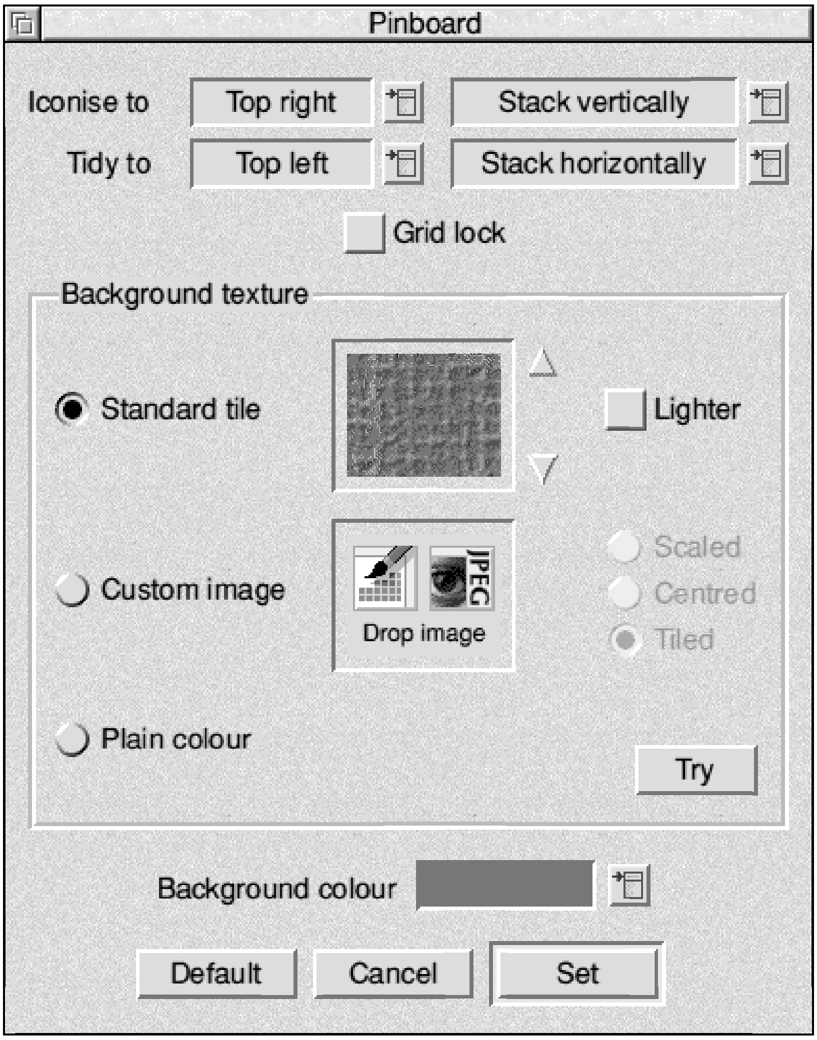
The GUI for selecting user-defined backdrops will be removed from the Pinboard's menus and added here. Setting of the background colour (for use with plain colour backdrops and backdrops made of a centred image that does not fill the screen) is a new Pinboard feature, and will also be configurable.

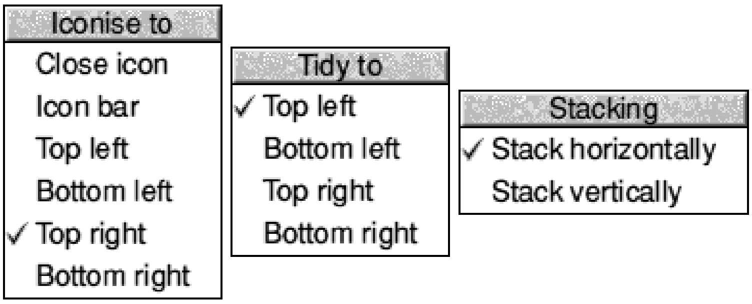
The Pinboard's "Grid Lock" option is joined by options specifying where and in which direction iconised files and window should be placed. Although these can also be changed within the present session from the Pinboard's menus, the settings these default to after a reset will be configurable.

Standard tiles will be selectable using a display icon and adjuster arrows. This allows the user to add further files to <BootResources$Dir>.Configure.Textures (they should use the naming convention Tx[L] to match the provided sprite files).

Backdrop files dragged to the "Custom image" drop zone (including JPEGs, which are now supported by the Pinboard) are used by reference - no copy is taken.

The Pinboard window and menus will look like this:





Default settings will be: iconize to top right, stack vertically; tidy to top left, stack horizontally; grid lock off; standard tile number 3 (Acorn logo, not pictured), not lighter, background colour Wimp grey 4.

See also the Ursula Pinboard Changes functional specification [5].

4.2.11. Screen Windows

4.2.11.1. Main Screen Window

In addition to the conventional monitor configuration facilities, the Screen window will allow the screen saver system (as described in the Miscellaneous GUI functional specification [4]) to be configured.

The monitor definition and default WimpMode to use will be read from and written to the file <PreDesk$Configure> (as in the old *Configure*).

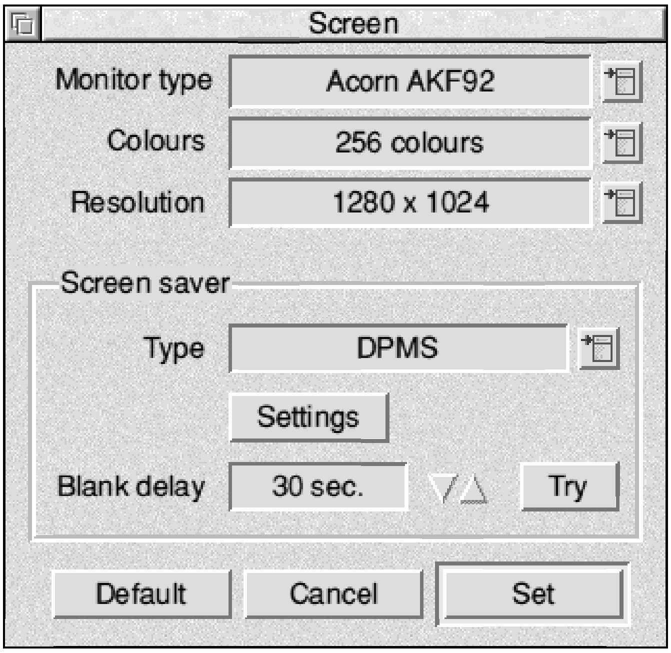
The new screen saver system is based upon the one developed for a licensee, and is extensible. Each screen saver takes the form of an application that is run when the screen needs to be saved; these are held in application directories within <BootResources$Dir>.Configure.ScrSavers (or any subdirectory thereof). Each application may have configurable features of its own, to allow the use of different text, directions, colours, effects etc. as applicable to the screen saver.

Therefore, each screen saver application is known as a *screen saver type*, and the name of each type is the application directory name without the "!" prefix. The screen saver type will be configurable via a string set.

Each screen saver application may contain an application called !Settings. The Screen window provides an action button that will run the !Settings of the currently chosen type (ie not necessarily the configured type). If there is no !Settings in the relevant application directory, the button will be faded.

The blank delay will no longer be instant-effect, in order to match the rest of the window. A "Try" action button will be provided for instant testing of a screen saver.

The Screen window will look like this:



Default settings, where appropriate, will be unchanged from RISC OS 3.71. The default screen saver type will be DPMS.

4.2.11.2. DPMS Screen Saver Window

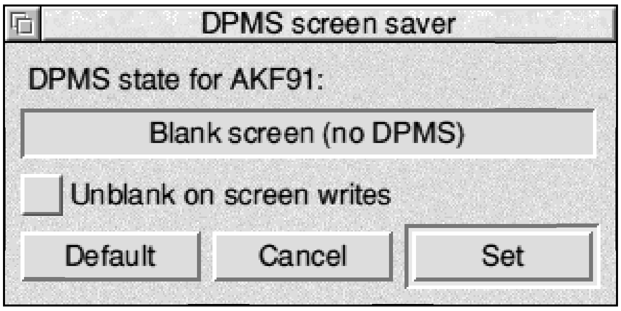
The in-built DPMS screen saver inherited from earlier versions of RISC OS does not, of course, have a screen saver application directory. The settings window for this will therefore be hard-coded into the Screen window plug-in.

The window will give a front-end to the established option to unblank the screen if a call to OS_WriteC is made. (It would be best suited to this location, as the option is meaningless for screen savers that write to the screen themselves.)

The window will also display the currently defined DPMS active state, something only possible in the past by manual inspection of the monitor definition file. The different levels of DPMS support will be described as the following:

- Blank screen (no DPMS)
- Monitor on stand-by (DPMS level 1)
- Monitor on suspend (DPMS level 2)
- Monitor off (DPMS level 3)

The DPMS screen saver configuration window will look like this:



The default setting will remain as not to unblank on screen writes.

4.2.11.3. Other Screen Saver Windows

These are described in the Miscellaneous GUI functional specification [4].

4.2.12. Sound Window

Design is pending decisions on which features of Phoebe's extended sound system are to be configurable and how.

It has been decided that the window will have a "Try" button to allow the user to test the system beep volume and voice when they wish to (rather than every time a parameter is adjusted).

The volume slider will be made draggable. Default settings will match previous machines where applicable.

4.2.13. System Window (*SysMerge*)

The System window, now that it is to be implemented as a plug-in, can also replace the *SysMerge* application, from which it inherits its name. It will correctly handle OS version-dependent subdirectories of !System, and keep logs and backups of operations performed (as implemented for the browser's *Installer*).

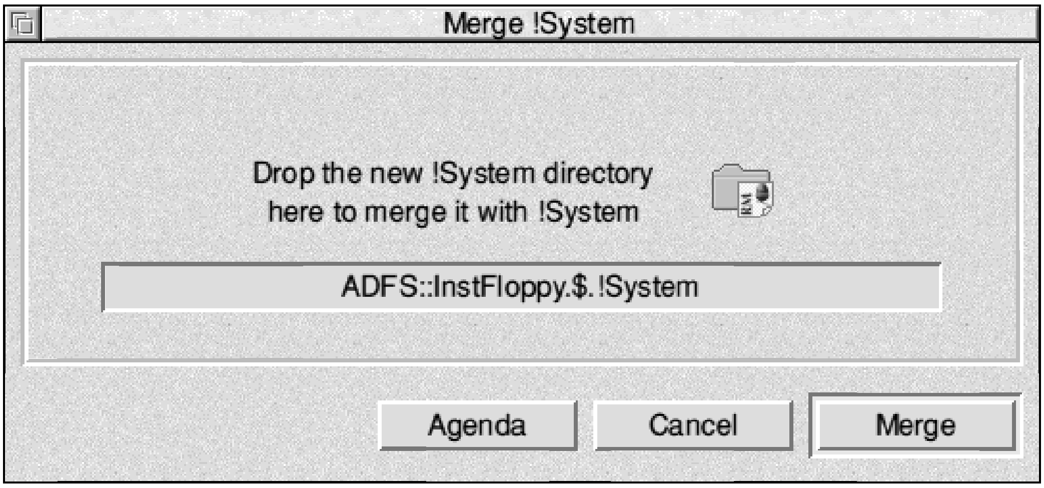
SysMerge, *FontMerge* (now a separate entity from the Fonts window) and the new, but related *BootMerge* (aka the "Install" Boot window) will share both a user interface and a back-end. For ease of reference, they are described together, the user interface and back-end here and in §5.2 respectively.

In each case, a system resource directory is merged with another directory of the same structure, typically (but not exclusively) a skeleton structure containing only the changes required. Such a source directory is dragged from a directory display and dropped on to the *xxxxMerge* window.

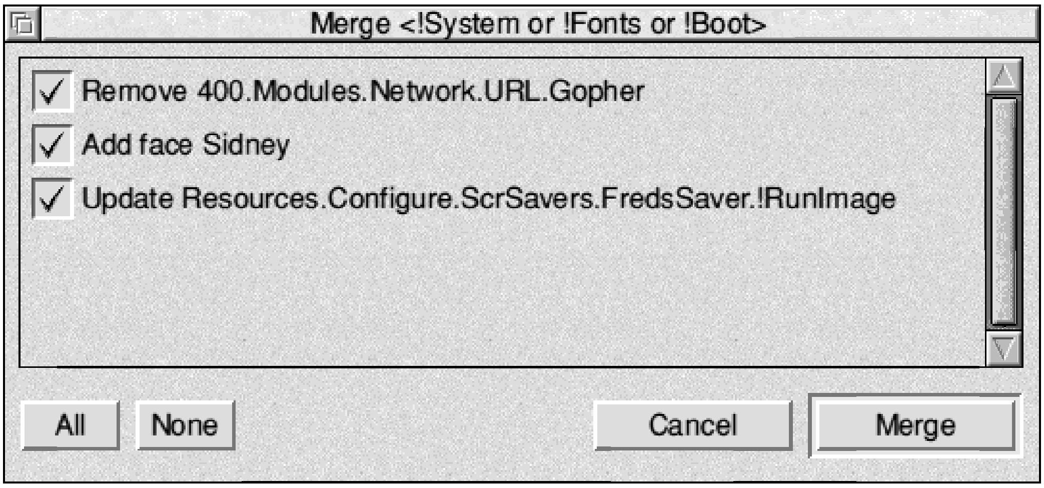
4.2.13.1. Window States

This is the initial state of the window. It will return to the above state whenever Cancel is Adjust-clicked.

Whenever a suitably-named directory is dropped on to the window (even after a preview has been calculated), it changes to the following state:



The user then has the choice of clicking on "Quick merge" to start merging files instantly, or of clicking on "Preview", which examines all the relevant files, determines the necessary operations, and reports back to the user, giving them prior knowledge of and control over the merge process:



The option buttons allow the user to opt out of any operations they prefer not to perform. The All and None action buttons allow quick selection or deselection of all the option buttons. If an error occurs during the merge process, the user will be given the option to Cancel or Continue; in either case, the window will remain in this state, with the option buttons corresponding to successfully-completed operations deselected.

4.2.13.2. Operations

The window directly above demonstrates typical results from each of the three *xxxxMerge* applications; note that a face of a font is considered an atomic component, and that paths are displayed relative to <System$Dir>, the third-last element of <Font$Path>, and <Boot$Dir> respectively. *SysMerge* and *BootMerge* will also have an operation to "Synchronise !System" - generated whenever changes are made within <System$Dir> - referring to an operation that checks that older modules are not masking newer modules of the same name, further along <System$Path>.

Generally speaking, each file at any level within the directory can either be added, updated or removed - this is determined by the state of the destination file before and after the operation. Where a file is updated or removed, the old version is first moved to a backup directory. After any operation is completed, a record of it is appended to a log file.

Special files of load and execution address &DEADDEAD ("corpses") will be considered requests for the object (file or directory) at the equivalent position in the destination structure to be removed. Empty directories will also be detected, and created at the destination using *CDir.

4.2.13.3. Differences

The differences between the three applications are listed below:

	SysMerge	*FontMerge*	*BootMerge*

Destination directory	<System$Dir>	Third-last element of <Font$Path>	<Boot$Dir>
Items looked for within the source directory	Modules, corpses, empty directories and miscellaneous files (eg FrontEnd)	Fonts and corpses	Any file (or font), plus corpses and empty directories
Where to start looking from	Any subdirectory (application or not) except "Backup"	Any subdirectory (excluding applications)	The source directory itself (excluding "Backup" and "Log")
Backup subdirectory and log file location	<System$Dir> .Backup / .Log	<Boot$Dir> .Backup / .Log	Either one, depending upon whether an individual file is going into <System$Dir> or not
Other notes	Does a !System synchronise at the end	Does a `*FontInstall` at the end	Does a !System synchronise, but only if it has been changing the contents of <System$Dir>; <Choices$Write>.Boot .PreDesktop and .Desktop are treated as special cases (using `*Install_Merge`)

4.2.13.4. Interactive Help

Help messages for *SysMerge* (the others will be simple variations on these):

Merge	Click SELECT to close the dialogue box and start the merging process.|MClick ADJUST to do the same, but keep the dialogue box open.
Cancel	Click SELECT to cancel this dialogue box.|MClick ADJUST to forget about the !System directory that has been dropped on to this window.
Agenda (when visible)	Click SELECT to view a list of all the operations that will be performed during the merge.
All (when visible)	Click SELECT to select all the operations.
None (when visible)	Click SELECT to deselect all the operations.
Pane icons	This is one of the operations that will be performed as part of the merge process.|MClick SELECT to disable or re-enable the operation.

4.2.14. Windows Window

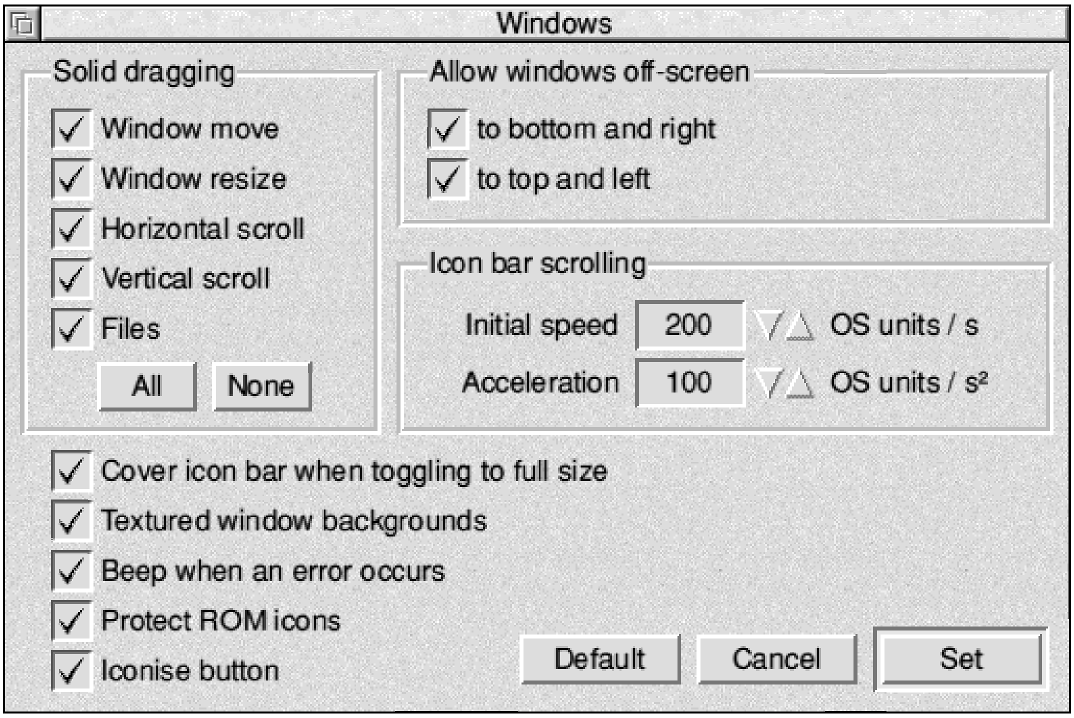
The "Open submenus automatically" option will be removed to the Mouse window, and the "Interactive file copying" option will be removed to the new Filer window. 2D window tools will no longer be available.

The existing configuration bits for non-obscuration of the icon bar when toggling to full size and use of DragASprite / DragAnObject drags rather than rotating-dash dragboxes will be made available.

New Ursula features that will be configurable through this window are:

- disabling of the iconise button (which may be necessary on compatibility grounds);
- write-protection of the new-look icons held in ROM;
- scrolling of the icon bar; acceleration rate and initial scrolling speed will be adjustable in units of 100 OS units / s and 100 OS units / s^2 respectively.

The Windows window will look like this:



Defaults will be as shown above.

5. Programming Interface

5.1. Configure

Configure has not had a programming interface in the past; however, it will become possible to write plug-in extensions to *Configure*, and so third parties will need to be informed of the interface.

It will be very useful to be able to add (or remove) parts of the *Configure* application. Most importantly, it will allow seamless integration of third party hardware configuration - SCSI interfaces, for example. Separate plug-ins can be maintained by different authors, and any necessary upgrades or bugfixes can easily be distributed. System administrators might wish to prevent users from upsetting the network configuration without, for example, preventing them from setting a preferred mouse speed.

In order to allow existing third-party configuration utilities to come under the umbrella of *Configure* plug-ins, the programming interface will be relatively minor, but its use will be required in future releases of applications intended to be used as plug-ins.

5.1.2. Icons for plug-ins

Each icon in the main window has three attributes which the central application has to determine, but which are best known to the plug-in: the (brief) textual name, the sprite to use, and the interactive help string.

In order to support protocol-ignorant applications as plug-ins, by default the text will be the name of the plug-in application (without the !), the sprite will be the name of the plug-in application (with the !), and the help text will be a generalised string. However, these are a fallback position, and must not be relied upon by applications intended for use as plug-ins.

The preferred method is for the plug-in's Messages file to contain three tokens, _ConfigText, _ConfigHelp and _ConfigSprite, to hold this information. _ConfigSprite is the name of a sprite in the Wimp sprite pool; *Configure* will `*IconSprites` a file called CoSprite (or CoSprite22 etc. according to the usual convention) within each plug-in to ensure the sprite is available. To avoid name clashes with other Wimp sprites, the sprite names will be prefixed with the three characters "co_".

Each plug-in will normally also have a !Sprites (or !Sprites22 etc.) file, and optionally a !Boot file as well, to cater for the case where the plug-in is accessed as a stand-alone application. The plug-ins will not normally be Filer_Booted unless and until a directory viewer is opened for <BootResources$Dir>.Configure; this saves time and Wimp sprite pool space. The !Sprites sprite for each plug-in will be different from the CoSprite sprite, because the latter is not constrained by the 68x68 OS units rule. If no better sprite is available, the "!configure" sprite may be re-used.

5.1.3. Information Transfer

In order to support as many existing configuration applications as possible, messaging supported by a plug-in can be limited to Message_Quit, although this will result in inconsistencies in window positioning.

Window positioning is handled via two routes: when the plug-in is started through Wimp_StartTask, the preferred co-ordinates of the top-left corner of the new window (in decimal OS units) are passed as arguments on the command line. A typical command line thus ends:

```
...!InetSetup -openat 120 844
```

(If the plug-in is started without this command line argument - eg if its application directory is double-clicked from a directory display - it must centre itself on screen instead.)

The other communication is necessary to bring the window to the top of the stack when the user clicks on an icon in the main window, and the plug-in window in question is already open. This is handled by Message_OpenConfigWindow; see §6.2.

The other main messaging is involved with allowing plug-ins to object to being quit, for example if complex selections have been made but not yet set. This is handled by the plug-in Quit protocol; see §6.1.

There has been another operation in the past where information was passed between parent and child configuration windows: Adjust-clicking on the close icon of a child would re-open the parent if it had been closed due to an icon in the main window having been clicked using Adjust. This would be difficult to implement across different tasks, and near impossible where the child task predates any release of the new specification of *Configure*. Instead, we will take advantage of the Style Guide convention that windows having a "Cancel" action button don't have a close icon, and apply that convention to all plug-ins; the main window will no longer close when Adjust is used to click on an icon.

5.1.4. Locking

The central application is responsible for detecting whether the condition of the machine is unsuitable for configuration - a comparatively complex decision based upon whether the machine is remotely or locally booted, the <Choices$Write> access permission and the condition of FSLock.

Since the plug-in applications are located deep within !Boot, it is assumed that the average user will only ever access them through *Configure*, which only allows the plug-ins to be loaded when the machine is in a configurable state.

The alternative invocation method involves double-clicking on the plug-in application directly. Rather than requiring each plug-in to duplicate the lock detection code from *Configure* (which might conceivably be subject to change at a later date anyway), the plug-ins must not complain if a write operation to disc or CMOS fails. However, the settings must still be applied for the current session even if such a failure occurs; this will allow more knowledgeable users to adjust the operation of even locked machines where an alternative GUI is not available (eg mouse speed, Filer filename width, screen saver textual message).

5.2. Installer Module

To facilitate *SysMerge*, *FontMerge* and *BootMerge*, the other second-level Boot windows and (if decided) a shared configuration Obey file, a module will be written to provide general purpose installation facilities.

This will ensure, for example, that proper backups and logs are kept of all changes, that modules are always correctly compared, that font messages files are updated, and that !System's version directories are correctly maintained, and it will make it much simpler to update the PreDesktop and Desktop files.

Implementation as * commands allows their integration into customised Obey files to act as installation scripts in more complicated cases. An Installer application could then be written to perform an Obey file one line at a time, with a slider representing the proportion of lines completed; however this will not be done for *Configure*, as a full script will only be produced in one go by the *xxxxMerge* applications, and then only when a preview has already been calculated - the benefits will be marginal.

5.2.1. Definitions

Installer will recognise each of the following objects:

- **Font**: a compound object, consisting of all the files (of any filetype) in the same directory, provided one of which must have a name beginning with "IntMet". The font is referenced by the directory containing the files, for example Resources:$.Fonts.Homerton.Medium.
- **Module**: a file of filetype &FFA.
- **Obeyfile**: a file of filetype &FEB (Obey), &FD7 (TaskObey) or &FEA (Desktop).
- **Datafile**: any other typed or untyped file.

Installer will not install entire directories in a single command, it only deals with one object from the list above at a time.

An **entry** in an obeyfile is defined as follows. Only the first word of the section name will be significant to Installer.

`Start` *`company application version section`*
...assorted * commands...
`End`

5.2.2. *Install_LogDir

Syntax: `*Install_LogDir [`*`directory`*`]`
`directory` the new log directory

*Install_LogDir defines the directory into which Installer will write the log file `Log` and build the backup tree. Backed-up files will have the same path relative to *`directory`*`.Backup` as they used to have relative to *`directory`*. Note that this means that *`directory`* must be higher up the hierarchy than any file to be installed or deinstalled, otherwise it cannot be backed up.

The default is for the log directory to be undefined, so this command will typically be used at the beginning of an installation script.

If no argument is given, the current log directory is printed.

5.2.3. *Install_NoLogDir

Syntax: `*Install_NoLogDir`

*Install_NoLogDir undefines the log directory, and would typically be used at the end of an installation script.

When there is no defined log directory, the log file data is written to the screen instead.

5.2.4. *Install_Update

Syntax: `*Install_Update` *`source_file dest_file`*

*Install_Update copies a file if *`source_file`* is newer than *`dest_file`*. The comparative newness of files is determined by

- if both files are modules, the version number of the modules (or if they match, the datestamps of the files);
- otherwise, a simple datestamp comparison is used; in certain special cases, the datstamp is determined by a special method:
    - untyped files are assigned a datestamp of the beginning of time;
    - fonts are assigned the most recent datestamp of the files in the font directory (non-recursively);
    - directories are assigned the most recent datestamp of the files in the directory (recursively);
    - voids (where there is currently nothing) are checked to see if they are located below a file, rather than a directory; if the former, then the file's datestamp is used.

Any overwritten file is first copied into the backup directory, and details of the operation are appended to the log file.

In the case of modules, the version number multiplied by 100 is appended to the name of the file in the backup directory, and the old and new version numbers are mentioned in the log file.

In the case of fonts, the Messages files in the first application directory above *`dest_file`* are updated to include any new font.

5.2.5. *Install_Remove

Syntax: `*Install_Remove` *`file`*

*Install_Remove moves the specified `file` to its backup directory, and details of the operation are appended to the log file.

Any file of the same name already in the backup directory is first deleted.

In the case of modules, the version number multiplied by 100 is appended to the name of the file in the backup directory, and the version number is mentioned in the log file.

In the case of fonts, the Messages files in the first fonts directory above `file` are updated to reflect the removal of the font.

5.2.6. *Install_Merge

Syntax: `*Install_Merge base_file changes_file dest_file`

`base_file`	the "base" source file
`changes_file`	the "changes" source file
`dest_file`	the destination file

*Install_Merge combines the entries of two obeyfiles (as defined above), placing the result in another file. `dest_file` may be the same as one of the source files.

The `changes_file` may optionally contain a set of rules for the positioning of new entry *section*s, *company*s or *application*s. These consist of any number of special one-line entries, starting `|Section`, `|Company` or `|App`, as appropriate. Each of these lines has the following format:

`|[Section|Company|App] rule [rule [rule [rule ...]]]`

where	`rule`	is either	
		`specifier>specifier`	meaning create after in file
		or	
		`specifier<specifier`	meaning create before in file
		(the lack of spaces is significant)	
and	`specifier`	= case-insensitive single-word `section`, `company` or `application`	
		or `*` to refer to:	any specifier (when used on the left)
			all specifiers (when used on the right)

Rules are used from the first appropriate entry in the file to the last, and left-to-right within each line.

Entries from `changes_file` will be merged one by one into `base_file`. This will be achieved thus:

- Scan `base_file` for the first entry with a section matching that of the new entry. If there are none, scan the command line for the first applicable rule. If no rule applies, `*>*` defaults.
- Scan all consecutive entries in `base_file` with the same section for the first one with a company matching that of the new entry. If there are none, scan the command line for the first applicable rule. If no rule applies, `*>*` defaults.

- Scan all consecutive entries in *base_file* with the same section and company for the one that also matches the application in the new entry. If there is none, scan the command line for the first applicable rule, and if no rule applies, `*>*` defaults. If there already was such an entry, compare the version number; if the new entry is a matching or earlier version, use the entry from *base_file*, otherwise use the entry from *changes_file*.

If *dest_file* is to be overwritten, it is first copied into the backup directory. Details of the operation are appended to the log file; each successful entry update is listed separately (along with new version number and any applicable old version number).

Example 1:

The "Add to Apps" Boot window will use this command:

```
*Install_Merge <Choices$Write>.Boot.PreDesktop tempfile
  <Choices$Write>.Boot.PreDesktop
```

where *tempfile* contains the line:

```
|App Configure>*
```

The section `ResApps` and company `Acorn` can be assumed to already exist (they are part of the default PreDesktop file). However, all other `Acorn ResApps` entries should come first.

Example 2:

The "Run" Boot window will use this command:

```
*Install_Merge <Choices$Write>.Boot.Desktop tempfile
  <Choices$Write>.Boot.Desktop
```

where *tempfile* contains the line:

```
|Section Run>Boot Run>Auto
```

The new section `Run` should be created after `Boot`, or after `Auto` ("Auto tasks") if `Boot` doesn't yet exist. Positioning of the `Acorn Configure` entry within the `Run` section is not expected to be critical.

Example 3:

The nested Wimp installer, were it to be re-written to use this Installer module, would use:

```
*Install_Merge <Choices$Write>.Boot.PreDesktop changesfile
  <Choices$Write>.Boot.PreDesktop
```

where *changesfile* contains the lines:

```
|Section Installation>Comments
```

||Company Acorn<*

`Installation` may be a new section on some machines. `Acorn` should be the first company within the installation section. Since this is, at the time of writing, the first entry in `Acorn Installation`, we'll leave ordering of applications to later installers.

In the event that another entry is written at a later date, which needs to be added to PreDesktop after the Wimp installation entry, a clever approach needs to be taken (because there's a chance the Wimp will be installed at a later time, and because of the default rule `*>*`, it would be added afterwards in PreDesktop).

The new installer should first attempt to install an empty entry called `Acorn Wimp 0.00 Installation`, then install its own entry after the Wimp one. If a genuine Wimp entry already exists, the 0.00 entry is simply ignored. If the Wimp installer were only to be run later, the dummy entry's position would override the (default) `*>*` application rule in the above example.

Because of this, non-empty entries must, in every case, always have version numbers of at least 0.01.

5.2.6. *Install_DeMerge

Syntax: `*Install_DeMerge` *`base_file changes_file dest_file`*

base_file	the "base" source file
changes_file	the "changes" source file
dest_file	the destination file

*Install_DeMerge removes any entries from *base_file* that match the section, company and application of an entry in *changes_file*, unless the version number of the entry in *base_file* is newer than the one in *changes_file*. The result is placed in *dest_file*.

If *dest_file* is to be overwritten or deleted, it is first copied into the backup directory. Details of the operation are appended to the log file.

5.2.7. *Install_CheckSystem

Syntax: `*Install_CheckSystem`

*Install_CheckSystem ensures that versions of modules present in OS-version subdirectories of <System$Dir> are not older than those in previous-version subdirectories, or the "root" Modules subdirectory. Any that are are removed in a manner consistent with *Install_Remove.

6. Data Interchange

With the introduction of modularity into *Configure*, a limited amount of messaging will be introduced between the main application and the plug-ins. This is designed to provide the best possible results when third party configuration applications which are unaware of the protocol are used as plug-ins. There are three separate message protocols involved.

6.1. Quit Protocol

Applicability: all plug-ins where selections would be fiddly or time-consuming to repeat - the boot sequence editors, for example.

The established Message_PreQuit method cannot be used to warn plug-ins that they are about to be killed (which can happen either because of a click on the close icon of the main window, or because of the activation of FSLock). The problem is with the re-initialisation of the procedure if the user confirms that they do indeed want to quit anyway; Message_PreQuit allows only for the quitting of a single task or for the termination of the entire desktop.

When the central application receives a close window request, a Message_PlugInQuitContinue, or a Message_FSLockStateChangeRequest for a lock status of 2, it will broadcast Message_PlugInQuit with event type 18. If no plug-in responds, the Wimp will return the Message_PlugInQuit to the central application, which will then send a Message_Quit to each plug-in individually (except the Lock plug-in, in the case of lock activation).

This protocol is also used between top-level and second-level plug-ins in a similar fashion when necessary. The main difference is that the Message_Quit contains a flag word to indicate whether the quit was generated by a click on "Set" rather than "Cancel", because in the former case the plug-in may assume that the user wants to set the selection (and need not therefore generate a Discard/Cancel/Set dialogue box).

Plug-ins must behave as follows (in addition to handling Message_PreQuit as normal) if they wish to be able to object to being quit:

At initialisation:

- Set "complain" flag.

When Message_PlugInQuit is received:

- If "complain" and "modified" flags are set,
    - acknowledge the message so that it isn't returned to the central application;
    - query the user whether they want to "Discard", "Cancel" or "Set" (this can span Wimp_Polls);
    - if "Discard" or "Set" is selected,
        - clear "complain" flag;
        - send Message_PlugInQuitContinue to the task that broadcast the Message_PlugInQuit.
    - else (ie "Cancel" is selected),
        - broadcast Message_PlugInQuitAborted.

When Message_PlugInQuitAborted is received:

- Set "complain" flag, ready for the next time.

The details of the messages are as follows:

Message_PlugInQuit (&50D80)

This message informs plug-ins that they are about to be quit, unless one of their number aborts the process. The word at R1+20 allows this message to be used in other cases than *Configure* plug-ins; if a task receives this message, but does not recognise the word, it must ignore the message. Similarly, if the plug-in was invoked without the `-openat` switch, it must also ignore this message, as it will not be receiving a subsequent Message_Quit.

R1+0 length: 24 (or more to indicate additional data is present)

R1+12 your_ref: 0

R1+20 plug-in system descriptor:
&50DBF for the main *Configure* plug-in system
&50DBE for the Boot window plug-in system
&50DBD for the Fonts window / *FontMerge* dialogue
other values may be used, but should be kept within the relevant chunk

R1+24 The Boot and Fonts plug-in systems have bit 1 of this word set to indicate that the user clicked on "Set" rather than "Cancel"

Message_PlugInQuitContinue (&50D81)

This message is used by plug-ins to tell the central application to resume the quitting process if it has been halted while the plug-in requests confirmation from the user. The word at R1+20 allows this message to be used in other cases than *Configure* plug-ins; the plug-in must fill it in correctly, or the central application will ignore the message (or worse, take the wrong subsequent action).

R1+0 length: 24

R1+12 your_ref: 0

R1+20 plug-in system descriptor

Message_PlugInQuitAbort (&50D82)

This message is used between plug-ins so that a plug-in can be aware that the quit process has been aborted by another, and therefore that it must reset its internal "complain" flag so that subsequent quit requests can also be complained at. The word at R1+20 allows this message to be used in other cases than *Configure* plug-ins; if a task receives this message, but does not recognise the word, it must ignore the message.

R1+0 length: 24

R1+12 your_ref: 0

R1+20 plug-in system descriptor

6.2. Re-open Window Protocol

Applicability: all plug-ins.

Message_OpenConfigWindow (&50D83)

This message is generated by the central application when the user clicks on an icon in the main window and the relevant plug-in is still active. The plug-in that receives the message must re-open its window at the top of the window stack in the same x/y position, but *not* make any other changes (equivalent to an Adjust-click on "Cancel" or otherwise).

R1+0 length: 20

R1+12 your_ref: 0

6.3. Lock / Unlock Protocol

Applicability: exclusive to the central application and the Lock plug-in.

Message_FSLockStateChangeRequest (&50D84)

This message is broadcast by the Lock plug-in, for the attention of the central application. Other tasks must ignore it. If R1+20 is 2, the central application will quit all other plug-ins and shade their icons in the main window; the plug-ins will have a chance to object using the Quit protocol in §6.1.

R1+0 length: 24

R1+12 your_ref: 0

R1+20 new lock status (0, 1 or 2)

Message_FSLockStateChangeConfirm (&50D85)

This message is broadcast by the main application, and received by the Lock plug-in to confirm the lock state change. It is never sent if one of the other plug-ins objected to being quit.

R1+0 length: 20

R1+12 your_ref: 0

Message_FSLockStateChanged (&50D86)

This message is broadcast by the Lock plug-in to announce that the FSLock state has been changed. It allows, for example, multiple instantiations of the Lock plug-in to update their windows synchronously.

R1+0 length: 20

R1+12 your_ref: 0