Customising the Risc PC Start-Up

David Spencer explains the rather involved auto-boot system used on the Risc PC

If you ever get a chance to look at a number of different serious users computers, there is a good chance that you will find that each has its own weird and wonderful boot sequence to customise it to its owner s requirements. The complexity of the processes involved in such customising has often been criticised, and Acorn have responded by incorporating what they call a structured boot sequence into RISC OS 3.5 in the Risc PC. However, I m sure that many Risc PC users gave up as soon as they saw the innards of the !Boot application. Hopefully, though, this article will make matters somewhat clearer.

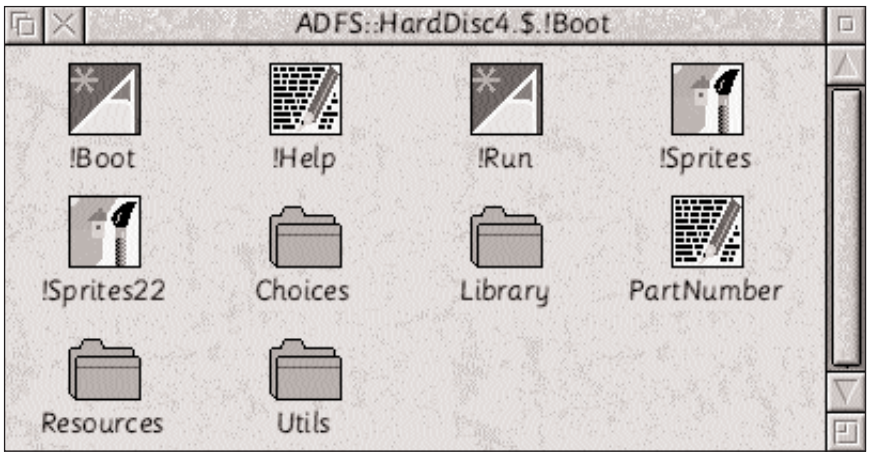
THE !BOOT APPLICATION

The Risc PC is shipped with an application called !Boot on its hard drive, and it is this that is responsible for setting up the user s preferences. By default, the CMOS RAM configuration is set to auto-boot from the default hard drive, which is in turn set to run the !Boot application. Hence, !Boot is always run on reset. Incidentally, double-clicking on !Boot from the Desktop invokes the Configure application for altering CMOS RAM setting. This is described in the RISC OS 3.5 User Guide. Therefore, !Boot is special in that running it automatically brings about a totally different response to running it explicitly.

CUSTOMISING !BOOT

In its simplest form, !Boot can be customised by using the Desktop boot save box provided by the Task Manager to save a Desktop Boot file which holds details of the Desktop s history to date, so for example, directory displays are automatically opened in the same place, and the pinboard is set up. This system is almost unchanged since RISC OS 3.1, though you must click on O K to use the default filename, rather than trying to drag the !Boot file anywhere.

More detailed customisation involves going into the !Boot application directory, by double-clicking on it with Shift held down. The actual internal structure of !Boot is rather complicated, with a large number of files and nested sub-directories. Luckily though, most of these files and directories are fixed resources that do not need altering (and which shouldn t be altered). The main customising procedure concerns the automatic execution or loading of applications and other files during the start-up, and the



The contents of the !Boot application directory.

defining of system variables (such as file types). These fall into two categories: files which are executed before the desktop is started, and files which are executed after it has been started. The former category includes things such as modules needed to drive additional hardware, whilst the latter category includes items such as the !Boot file saved from the Task Manager as mentioned earlier.

The file $.!Boot.Choices.Boot.PreDesktop is an Obey file that is run before the desktop is started. The file is split into five sections by function: setting of aliases for filetypes, the setting of paths (such as the library path), the setting of options, the registering of applications, and finally, the setting of any miscellaneous variables, such as an email name. All of these are fairly self-explanatory, except for the registering of applications. What this does is to use the AddApp command (which is actually a program located elsewhere in !Boot) to add applications to the Apps pseudo filing system. You can see how this is done by looking at the PreDesktop file which is well commented, though note that a much easier way of adding applications is described below. You can add any commands that you like to this file, but to maintain order you should fit them into the appropriate category, and bracket any added lines with the comments |start and |end.

After the PreDesktop file has been run, the contents of the directory $.!Boot.Choices.Boot.PreDesk are run one by one in alphabetical order. This directory can contain applications, modules, sprites or obey files, all of which will be run just before the desktop is started.

Once the pre-desktop procedures have been completed, the desktop is started, and the contents of the directory $.!Boot.Choices.Boot.Tasks are run. By default, this directory contains a configuration file (to set things like the backdrop texture), and also the !Boot file as saved from the Task Manager. You can add anything that you wish to here, but normally it would be applications that are added. These are then run as if the user had opened a directory display with them in and double-clicked on them.

If you want to automatically Filer_Boot an application (equivalent to opening a directory display containing it), but not run it, then this can be done by putting the application in the directory $.!Boot.Resources, alongside the likes of !System and !Fonts.

ADDING RESOURCES

Although both !Fonts and !System are buried within !Boot, neither of them should be altered directly, for fear of destroying data accidentally. Instead, the Configure application invoked by double-clicking on !Boot should be used to merge fonts and system resources, as described in the User Guide.

One thing that can be added directly is extra monitor definition files. These are all contained within the directory $.!Boot.Resources.Configure.Monitors. The definition files (probably supplied with the monitor) can either be put in this directory directly, or, as is the case for the standard Acorn files, grouped into a sub-directory by manufacturer.

ADDING APPLICATIONS AND UTILITIES

Another way that the computer can be customised is by adding additional applications to the Apps pseudo filing system. This can be done simply by copying the desired applications into the $.Apps directory. At start-up these will automatically be added to the Apps filing system and Filer_Booted (equivalent to displaying a directory containing them).

You can also add any utility-type applications to the $.Utilities directory. The boot sequence Filer_Boots all of the applications in this directory at start-up. This means that all the applications !Boot files get run, thus allowing them to perform operations such as setting up filetypes and filetype icons. So for example, if the $.Utilities directory contained an application that processed files of type, say, Template, then it would be invoked automatically when a template file was double-clicked on, even though the application itself hadn't been seen by the user.