Into the Arc: System Variables - The Hidden Resource

Mike Williams explains how the RISC OS Desktop remembers all that it needs to know.

The RISC OS Desktop is really quite remarkable. Its operation is so often overlooked that it is easy to forget just how much work is going on beneath the surface. Take something simple like double-clicking on a data file of some kind (like an Fireworkz spreadsheet) - how does your system know where the corresponding application is stored, and what action to perform. Take another example - you run a DTP or word processing application and immediately have access to all the fonts on your system. How did the application know what fonts were available? The answer to this and many other questions about Desktop actions lies with System Variables. In this month's article in this series we'll take a look at what they are and how they are used.

Essentially, system variables provide RISC OS with the means to store all the information it needs to work correctly, and system variables are also used extensively by other applications for the same kind of purpose. A system variable is a name which can be used to reference a piece of information. These names follow a convention laid down by Acorn, and can normally be distinguished by the presence of a $ character within the name, and a certain pattern to its make-up. For example, Font$Path provides information on the fonts currently available to any application in the form of a list of font directories.

Now it is all very well to quote an example like that above, but when you switch off your computer all knowledge of system variables is completely lost. When you first switch on, part of the automatic startup sequence involves defining a standard set of system variables with a set of default values. Sometimes these values will be determined by status settings stored in CMOS RAM. These settings are not lost when your computer is switched off, because there is battery back-up to CMOS RAM to provide sufficient power.

If you want to see what system variables currently exist, you can do this in one of two ways. If you press F12, to exit from the Desktop, just type the command Show and press Return, and you will see all the system variables which your system knows about. Alternatively, and this avoids leaving the Desktop, press the menu button over the Task Manager (the green acorn on the icon bar), and select Task window (a shortcut is to press Ctrl-F12). Then enter the same command and press Return.

If you try this, you may well be somewhat put off by the length and apparent complexity of the resulting list. Don't worry - we aren't going to be looking at this in any detail, but it does give you some indication of just how much information RISC OS has to manage.

BEING SEEN

If you have read through the pages of RISC User, or other magazines, you may well have come across a statement that some application or other must have been 'seen'. The problem which leads to this need is more likely to arise with a floppy only system, but can also occur with hard disc systems as well. Let's look at an obvious example.

You probably know that there is a special application called !System which is normally situated in the root directory of a hard disc. Within this application is a directory called Modules, and inside this you will find a number of files. These are modules which can be loaded and shared by all

applications running on your system. Find your copy of !System, double-click while holding down Shift to open the application directory. Then double-click on the Modules directory. !Scrap is another application used by other applications, including the printer drivers.

In the days of RISC OS 2, modules like CLib and FPE used to live in the Modules directory of !System, but these very commonly used modules were moved into ROM under RISC OS 3. However, other modules including those supplied with third party applications still need to live here - for example, Basic64, Acorn s latest Basic interpreter, FrontEnd supplied as part of Acorn s DDE, plus ABCLib, the Basic compiler run-time library from Oak Solutions. If an application needs to use either !System or !Scrap, then the application needs to know where they are stored. That s where system variables come in. System$Path and Wimp$Scrap are system variables which provide that information, but they have to be set up to start with. Here s how it works.

Whenever a directory is opened on the Desktop, RISC OS checks within that directory each application directory (i.e. a directory beginning with a !) to see if it contains a file called !Boot. Most, but not all applications contain a file of this name. It is usually a short file containing just a few commands which are executed the first time the directory is opened. One of these commands is normally Iconsprites which tells RISC OS about the sprite, or icon, used to represent that application. It is the !Boot file which also defines any system variables. So if you open the !System application directory as described before, and drag the !Boot file to Edit (load Edit so that it is visible on the icon bar first) you will see the commands which check for and set up both System$Path and System$Dir.

This all happens quite automatically when you open the directory containing !System, and all the applications visible in that directory will then have been seen. Most hard disc users use a boot file to set up their system in a more customised way. Such boot files contain special commands (Filer_Boot) which simulate the same process without actually opening the relevant window on screen. This can be useful, as all this takes time, but the boot file can be configured to deal only with certain applications. Opening a window applies the process to all applications.

OTHER USES FOR SYSTEM VARIABLES

Let us suppose that you have a copy of RISC Developments DeskEdit, and that you have installed this ready for use on the icon bar. Double-click on any ASCII text file and it will automatically be loaded into a DeskEdit window. Now load Acorn s Edit, and repeat the double-clicking on the text file. Now it will be loaded into Edit, not DeskEdit. What has changed?

Most files stored on an Archimedes system have an associated filetype. Acorn has allocated filetype numbers for many files, and allocates filetype numbers to third parties to be used with other software. For example, an Ovation document has its own filetype number (&CDD in hexadecimal, but that s another story). At power up RISC OS assigns names to all the common Acorn filetypes, so if you use Edit, for example, you can use the Settype option to change a filetype by reference to a name, not a difficult to remember number. The other thing that happens at start-up, is that system variables are used to specify what action to take when a user double-clicks on a file. When a third party application has been seen its !Boot file defines the action for files belonging to that application. Ovation s !Boot file says that Ovation should be loaded, if not already loaded, and the document loaded into Ovation.

When DeskEdit is seen or run, it specifies in the same way that