The default filename translation is roughly `tr|/.|./|;`

Note that `"ADFS::HardDisk.$.File" ne 'ADFS::HardDisk.$.File'` and that the second stage of `$` interpolation in regular expressions will fall foul of the `$.` if scripts are not careful.

Logical paths specified by system variables containing comma-separated search lists are also allowed; hence `System:Modules` is a valid filename, and the filesystem will prefix `Modules` with each section of `System$Path` until a name is made that points to an object on disk. Writing to a new file `System:Modules` would be allowed only if `System$Path` contains a single item list. The filesystem will also expand system variables in filenames if enclosed in angle brackets, so `<System$Dir>.Modules` would look for the file `$ENV{'System$Dir'} . 'Modules'`. The obvious implication of this is that **fully qualified filenames can start with** `<>` and should be protected when `open` is used for input.

Because `.` was in use as a directory separator and filenames could not be assumed to be unique after 10 characters, Acorn implemented the C compiler to strip the trailing `.c` `.h` `.s` and `.o` suffix from filenames specified in source code and store the respective files in subdirectories named after the suffix. Hence files are translated:

```
foo.h           h.foo
C:foo.h         C:h.foo       (logical path variable)
sys/os.h        sys.h.os      (C compiler groks Unix-speak)
10charname.c    c.10charname
10charname.o    o.10charname
11charname_.c   c.11charname  (assuming filesystem truncates at 10)
```

The Unix emulation library's translation of filenames to native assumes that this sort of translation is required, and it allows a user-defined list of known suffixes that it will transpose in this fashion. This may seem transparent, but consider that with these rules `foo/bar/baz.h` and `foo/bar/h/baz` both map to `foo.bar.h.baz`, and that `readdir` and `glob` cannot and do not attempt to emulate the reverse mapping. Other `.`'s in filenames are translated to `/`.

As implied above, the environment accessed through `%ENV` is global, and the convention is that program specific environment variables are of the form `Program$Name`. Each filesystem maintains a current directory, and the current filesystem's current directory is the **global** current directory. Consequently, sociable programs don't change the current directory but rely on full pathnames, and programs (and Makefiles) cannot assume that they can spawn a child process which can change the current directory without affecting its parent (and everyone else for that matter).

Because native operating system filehandles are global and are currently allocated down from 255, with 0 being a reserved value, the Unix emulation library emulates Unix filehandles. Consequently, you can't rely on passing `STDIN`, `STDOUT`, or `STDERR` to your children.

The desire of users to express filenames of the form `<Foo$Dir>.Bar` on the command line unquoted causes problems, too: `` ` `` command output capture has to perform a guessing game. It assumes that a string `<[^<>]+\$[^<>]>` is a reference to an environment variable, whereas anything else involving < or > is redirection, and generally manages to be 99% right. Of course, the problem remains that scripts cannot rely on any Unix tools being available, or that any tools found have Unix-like command line arguments.

Extensions and XS are, in theory, buildable by anyone using free tools. In practice, many don't, as users of the Acorn platform are used to binary distributions. MakeMaker does run, but no available make currently copes with MakeMaker's makefiles; even if and when this should be fixed, the lack of a Unix-like shell will cause problems with makefile rules, especially lines of the form `cd sdbm && make all`, and anything using quoting.

"RISC OS" is the proper name for the operating system, but the value in `$^O` is "riscos" (because we don't like shouting).

52.4.8 Other perls

Perl has been ported to many platforms that do not fit into any of the categories listed above. Some, such as AmigaOS, Atari MiNT, BeOS, HP MPE/iX, QNX, Plan 9, and VOS, have been well-integrated into the standard Perl source code kit. You may need to see the *ports/* directory on CPAN for information, and possibly binaries, for the likes of: aos, Atari ST, lynxos, riscos, Novell Netware, Tandem Guardian, *etc.* (Yes, we know that some of these OSes may fall under the Unix category, but we are not a standards body.)

Some approximate operating system names and their `$^O` values in the "OTHER" category include: