127.2.5 Usage Hints for Perl on Win32

Environment Variables

The installation paths that you set during the build get compiled into perl, so you don't have to do anything additional to start using that perl (except add its location to your PATH variable).

If you put extensions in unusual places, you can set PERL5LIB to a list of paths separated by semicolons where you want perl to look for libraries. Look for descriptions of other environment variables you can set in *perlrun*.

You can also control the shell that perl uses to run system() and backtick commands via PERL5SHELL. See *perlrun*.

Perl does not depend on the registry, but it can look up certain default values if you choose to put them there. Perl attempts to read entries from `HKEY_CURRENT_USER\Software\Perl` and `HKEY_LOCAL_MACHINE\Software\Perl`. Entries in the former override entries in the latter. One or more of the following entries (of type REG_SZ or REG_EXPAND_SZ) may be set:

```
lib-$]              version-specific standard library path to add to @INC
lib                 standard library path to add to @INC
sitelib-$]          version-specific site library path to add to @INC
sitelib             site library path to add to @INC
vendorlib-$]        version-specific vendor library path to add to @INC
vendorlib           vendor library path to add to @INC
PERL*               fallback for all %ENV lookups that begin with "PERL"
```

Note the `$]` in the above is not literal. Substitute whatever version of perl you want to honor that entry, e.g. `5.6.0`. Paths must be separated with semicolons, as usual on win32.

File Globbing

By default, perl handles file globbing using the File::Glob extension, which provides portable globbing.

If you want perl to use globbing that emulates the quirks of DOS filename conventions, you might want to consider using File::DosGlob to override the internal glob() implementation. See *File::DosGlob* for details.

Using perl from the command line

If you are accustomed to using perl from various command-line shells found in UNIX environments, you will be less than pleased with what Windows offers by way of a command shell.

The crucial thing to understand about the Windows environment is that the command line you type in is processed twice before Perl sees it. First, your command shell (usually CMD.EXE on Windows NT, and COMMAND.COM on Windows 9x) preprocesses the command line, to handle redirection, environment variable expansion, and location of the executable to run. Then, the perl executable splits the remaining command line into individual arguments, using the C runtime library upon which Perl was built.

It is particularly important to note that neither the shell nor the C runtime do any wildcard expansions of command-line arguments (so wildcards need not be quoted). Also, the quoting behaviours of the shell and the C runtime are rudimentary at best (and may, if you are using a non-standard shell, be inconsistent). The only (useful) quote character is the double quote ("). It can be used to protect spaces and other special characters in arguments.

The Windows NT documentation has almost no description of how the quoting rules are implemented, but here are some general observations based on experiments: The C runtime breaks arguments at spaces and passes them to programs in argc/argv. Double quotes can be used to prevent arguments with spaces in them from being split up. You can put a double quote in an argument by escaping it with a backslash and enclosing the whole argument within double quotes. The backslash and the pair of double quotes surrounding the argument will be stripped by the C runtime.

The file redirection characters "<", ">", and "|" can be quoted by double quotes (although there are suggestions that this may not always be true). Single quotes are not treated as quotes by the shell or the C runtime, they don't get stripped by the shell (just to make this type of quoting completely useless). The caret "^" has also been observed to behave as a quoting character, but this appears to be a shell feature, and the caret is not stripped from the command line, so Perl still sees it (and the C runtime phase does not treat the caret as a quote character).