NAME

perlclib - Internal replacements for standard C library functions

DESCRIPTION

One thing Perl porters should note is that *perl* doesn't tend to use that much of the C standard library internally; you'll see very little use of, for example, the *ctype.h* functions in there. This is because Perl tends to reimplement or abstract standard library functions, so that we know exactly how they're going to operate.

This is a reference card for people who are familiar with the C library and who want to do things the Perl way; to tell them which functions they ought to use instead of the more normal C functions.

Conventions

In the following tables:

`t`

is a type.

`p`

is a pointer.

`n`

is a number.

`s`

is a string.

`sv`, `av`, `hv`, etc. represent variables of their respective types.

File Operations

Instead of the *stdio.h* functions, you should use the Perl abstraction layer. Instead of `FILE*` types, you need to be handling `PerlIO*` types. Don't forget that with the new PerlIO layered I/O abstraction `FILE*` types may not even be available. See also the `perlapio` documentation for more information about the following functions:

```
    Instead Of:                 Use:

    stdin                       PerlIO_stdin()
    stdout                      PerlIO_stdout()
    stderr                      PerlIO_stderr()

    fopen(fn, mode)             PerlIO_open(fn, mode)
    freopen(fn, mode, stream)   PerlIO_reopen(fn, mode, perlio)
(Deprecated)
    fflush(stream)              PerlIO_flush(perlio)
    fclose(stream)              PerlIO_close(perlio)
```

File Input and Output

```
    Instead Of:                 Use:

    fprintf(stream, fmt, ...)   PerlIO_printf(perlio, fmt, ...)

    [f]getc(stream)             PerlIO_getc(perlio)
    [f]putc(stream, n)          PerlIO_putc(perlio, n)
    ungetc(n, stream)           PerlIO_ungetc(perlio, n)
```

Note that the PerlIO equivalents of `fread` and `fwrite` are slightly different from their C library counterparts:

```
    fread(p, size, n, stream)   PerlIO_read(perlio, buf, numbytes)
    fwrite(p, size, n, stream)  PerlIO_write(perlio, buf, numbytes)

    fputs(s, stream)            PerlIO_puts(perlio, s)
```

There is no equivalent to `fgets`; one should use `sv_gets` instead:

```
    fgets(s, n, stream)         sv_gets(sv, perlio, append)
```

File Positioning

```
    Instead Of:                 Use:

    feof(stream)                PerlIO_eof(perlio)
    fseek(stream, n, whence)    PerlIO_seek(perlio, n, whence)
    rewind(stream)              PerlIO_rewind(perlio)

    fgetpos(stream, p)          PerlIO_getpos(perlio, sv)
    fsetpos(stream, p)          PerlIO_setpos(perlio, sv)

    ferror(stream)              PerlIO_error(perlio)
    clearerr(stream)            PerlIO_clearerr(perlio)
```

Memory Management and String Handling

```
    Instead Of:                  Use:

    t* p = malloc(n)             Newx(id, p, n, t)
    t* p = calloc(n, s)          Newxz(id, p, n, t)
    p = realloc(p, n)            Renew(p, n, t)
    memcpy(dst, src, n)          Copy(src, dst, n, t)
    memmove(dst, src, n)         Move(src, dst, n, t)
    memcpy/*(struct foo *)       StructCopy(src, dst, t)
    memset(dst, 0, n * sizeof(t)) Zero(dst, n, t)
    memzero(dst, 0)   Zero(dst, n, char)
    free(p)                      Safefree(p)

    strdup(p)                    savepv(p)
    strndup(p, n)                savepvn(p, n) (Hey, strndup doesn't exist!)

    strstr(big, little)          instr(big, little)
    strcmp(s1, s2)               strLE(s1, s2) / strEQ(s1, s2) /
strGT(s1,s2)
    strncmp(s1, s2, n)           strnNE(s1, s2, n) / strnEQ(s1, s2, n)
```

Notice the different order of arguments to `Copy` and `Move` than used in `memcpy` and `memmove`.

Most of the time, though, you'll want to be dealing with SVs internally instead of raw `char *` strings:

```
    strlen(s)                    sv_len(sv)
    strcpy(dt, src)              sv_setpv(sv, s)
    strncpy(dt, src, n)          sv_setpvn(sv, s, n)
```

```
strcat(dt, src)             sv_catpv(sv, s)
strncat(dt, src)            sv_catpvn(sv, s)
sprintf(s, fmt, ...)        sv_setpvf(sv, fmt, ...)
```

Note also the existence of `sv_catpvf` and `sv_vcatpvfn`, combining concatenation with formatting.

Sometimes instead of zeroing the allocated heap by using Newxz() you should consider "poisoning" the data. This means writing a bit pattern into it that should be illegal as pointers (and floating point numbers), and also hopefully surprising enough as integers, so that any code attempting to use the data without forethought will break sooner rather than later. Poisoning can be done using the Poison() macros, which have similar arguments as Zero():

```
PoisonWith(dst, n, t, b)    scribble memory with byte b
PoisonNew(dst, n, t)        equal to PoisonWith(dst, n, t, 0xAB)
PoisonFree(dst, n, t)       equal to PoisonWith(dst, n, t, 0xEF)
Poison(dst, n, t)           equal to PoisonFree(dst, n, t)
```

Character Class Tests

There are two types of character class tests that Perl implements: one type deals in `chars` and are thus **not** Unicode aware (and hence deprecated unless you **know** you should use them) and the other type deal in UVs and know about Unicode properties. In the following table, c is a `char`, and u is a Unicode codepoint.

```
Instead Of:                 Use:            But better use:

isalnum(c)                  isALNUM(c)      isALNUM_uni(u)
isalpha(c)                  isALPHA(c)      isALPHA_uni(u)
iscntrl(c)                  isCNTRL(c)      isCNTRL_uni(u)
isdigit(c)                  isDIGIT(c)      isDIGIT_uni(u)
isgraph(c)                  isGRAPH(c)      isGRAPH_uni(u)
islower(c)                  isLOWER(c)      isLOWER_uni(u)
isprint(c)                  isPRINT(c)      isPRINT_uni(u)
ispunct(c)                  isPUNCT(c)      isPUNCT_uni(u)
isspace(c)                  isSPACE(c)      isSPACE_uni(u)
isupper(c)                  isUPPER(c)      isUPPER_uni(u)
isxdigit(c)                 isXDIGIT(c)     isXDIGIT_uni(u)

tolower(c)                  toLOWER(c)      toLOWER_uni(u)
toupper(c)                  toUPPER(c)      toUPPER_uni(u)
```

stdlib.h functions

```
Instead Of:                 Use:

atof(s)                     Atof(s)
atol(s)                     Atol(s)
strtod(s, *p)               Nothing.  Just don't use it.
strtol(s, *p, n)            Strtol(s, *p, n)
strtoul(s, *p, n)           Strtoul(s, *p, n)
```

Notice also the `grok_bin`, `grok_hex`, and `grok_oct` functions in *numeric.c* for converting strings representing numbers in the respective bases into NVs.

In theory `Strtol` and `Strtoul` may not be defined if the machine perl is built on doesn't actually have strtol and strtoul. But as those 2 functions are part of the 1989 ANSI C spec we suspect you'll

find them everywhere by now.

```
int rand()                  double Drand01()
srand(n)                    { seedDrand01((Rand_seed_t)n);
                              PL_srand_called = TRUE; }

exit(n)                     my_exit(n)
system(s)                   Don't. Look at pp_system or use my_popen

getenv(s)                   PerlEnv_getenv(s)
setenv(s, val)              my_putenv(s, val)
```

Miscellaneous functions

You should not even **want** to use *setjmp.h* functions, but if you think you do, use the `JMPENV` stack in *scope.h* instead.

For `signal`/`sigaction`, use `rsignal(signo, handler)`.

SEE ALSO

`perlapi`, `perlapio`, `perlguts`