NAME

Symbol - manipulate Perl symbols and their names

SYNOPSIS

```
use Symbol;

$sym = gensym;
open($sym, '<', "filename");
$_ = <$sym>;
# etc.

ungensym $sym;      # no effect

# replace *FOO{IO} handle but not $FOO, %FOO, etc.
*FOO = geniosym;

print qualify("x"), "\n";             # "main::x"
print qualify("x", "FOO"), "\n";      # "FOO::x"
print qualify("BAR::x"), "\n";        # "BAR::x"
print qualify("BAR::x", "FOO"), "\n"; # "BAR::x"
print qualify("STDOUT", "FOO"), "\n"; # "main::STDOUT" (global)
print qualify(\*x), "\n";             # returns \*x
print qualify(\*x, "FOO"), "\n";      # returns \*x

use strict refs;
print { qualify_to_ref $fh } "foo!\n";
$ref = qualify_to_ref $name, $pkg;

use Symbol qw(delete_package);
delete_package('Foo::Bar');
print "deleted\n" unless exists $Foo::{'Bar::'};
```

DESCRIPTION

"Symbol::gensym" creates an anonymous glob and returns a reference to it. Such a glob reference can be used as a file or directory handle.

For backward compatibility with older implementations that didn't support anonymous globs, "Symbol::ungensym" is also provided. But it doesn't do anything.

"Symbol::geniosym" creates an anonymous IO handle. This can be assigned into an existing glob

without affecting the non-IO portions of the glob.

"Symbol::qualify" turns unqualified symbol names into qualified variable names (e.g. "myvar" -> "MyPackage::myvar"). If it is given a second parameter, "qualify" uses it as the default package; otherwise, it uses the package of its caller. Regardless, global variable names (e.g. "STDOUT", "ENV", "SIG") are always qualified with "main::".

Qualification applies only to symbol names (strings). References are left unchanged under the assumption that they are glob references, which are qualified by their nature.

"Symbol::qualify_to_ref" is just like "Symbol::qualify" except that it returns a glob ref rather than a symbol name, so you can use the result even if "use strict 'refs'" is in effect.

"Symbol::delete_package" wipes out a whole package namespace. Note this routine is not exported by default--you may want to import it explicitly.

BUGS

"Symbol::delete_package" is a bit too powerful. It undefines every symbol that lives in the specified package. Since perl, for performance reasons, does not perform a symbol table lookup each time a function is called or a global variable is accessed, some code that has already been loaded and that makes use of symbols in package "Foo" may stop working after you delete "Foo", even if you reload the "Foo" module afterwards.