57.1.5 Perl Modules

A module is just a set of related functions in a library file, i.e., a Perl package with the same name as the file. It is specifically designed to be reusable by other modules or programs. It may do this by providing a mechanism for exporting some of its symbols into the symbol table of any package using it, or it may function as a class definition and make its semantics available implicitly through method calls on the class and its objects, without explicitly exporting anything. Or it can do a little of both.

For example, to start a traditional, non-OO module called Some::Module, create a file called *Some/Module.pm* and start with this template:

```
package Some::Module;  # assumes Some/Module.pm

use strict;
use warnings;

BEGIN {
    use Exporter   ();
    our ($VERSION, @ISA, @EXPORT, @EXPORT_OK, %EXPORT_TAGS);

    # set the version for version checking
    $VERSION     = 1.00;
    # if using RCS/CVS, this may be preferred
    $VERSION = sprintf "%d.%03d", q$Revision: 1.1 $ =~ /(\d+)/g;

    @ISA         = qw(Exporter);
    @EXPORT      = qw(&func1 &func2 &func4);
    %EXPORT_TAGS = ( );     # eg: TAG => [ qw!name1 name2! ],

    # your exported package globals go here,
    # as well as any optionally exported functions
    @EXPORT_OK   = qw($Var1 %Hashit &func3);
}
our @EXPORT_OK;

# exported package globals go here
our $Var1;
our %Hashit;

# non-exported package globals go here
our @more;
our $stuff;

# initialize package globals, first exported ones
$Var1   = '';
%Hashit = ();

# then the others (which are still accessible as $Some::Module::stuff)
$stuff  = '';
@more   = ();

# all file-scoped lexicals must be created before
# the functions below that use them.

# file-private lexicals go here
my $priv_var    = '';
my %secret_hash = ();
```