67.3.7 Input and Output Parameters

You specify the parameters that will be passed into the XSUB on the line(s) after you declare the function's return value and name. Each input parameter line starts with optional white space, and may have an optional terminating semicolon.

The list of output parameters occurs at the very end of the function, just before after the OUTPUT: directive. The use of RETVAL tells Perl that you wish to send this value back as the return value of the XSUB function. In Example 3, we wanted the "return value" placed in the original variable which we passed in, so we listed it (and not RETVAL) in the OUTPUT: section.

67.3.8 The XSUBPP Program

The **xsubpp** program takes the XS code in the .xs file and translates it into C code, placing it in a file whose suffix is .c. The C code created makes heavy use of the C functions within Perl.

67.3.9 The TYPEMAP file

The **xsubpp** program uses rules to convert from Perl's data types (scalar, array, etc.) to C's data types (int, char, etc.). These rules are stored in the typemap file ($PERLLIB/ExtUtils/typemap). This file is split into three parts.

The first section maps various C data types to a name, which corresponds somewhat with the various Perl types. The second section contains C code which **xsubpp** uses to handle input parameters. The third section contains C code which **xsubpp** uses to handle output parameters.

Let's take a look at a portion of the .c file created for our extension. The file name is Mytest.c:

```
XS(XS_Mytest_round)
{
    dXSARGS;
    if (items != 1)
        croak("Usage: Mytest::round(arg)");
    {
        double  arg = (double)SvNV(ST(0));      /* XXXXX */
        if (arg > 0.0) {
                arg = floor(arg + 0.5);
        } else if (arg < 0.0) {
                arg = ceil(arg - 0.5);
        } else {
                arg = 0.0;
        }
        sv_setnv(ST(0), (double)arg);   /* XXXXX */
    }
    XSRETURN(1);
}
```

Notice the two lines commented with "XXXXX". If you check the first section of the typemap file, you'll see that doubles are of type T_DOUBLE. In the INPUT section, an argument that is T_DOUBLE is assigned to the variable arg by calling the routine SvNV on something, then casting it to double, then assigned to the variable arg. Similarly, in the OUTPUT section, once arg has its final value, it is passed to the sv_setnv function to be passed back to the calling subroutine. These two functions are explained in *perlguts*; we'll talk more later about what that "ST(0)" means in the section on the argument stack.

67.3.10 Warning about Output Arguments

In general, it's not a good idea to write extensions that modify their input parameters, as in Example 3. Instead, you should probably return multiple values in an array and let the caller handle them (we'll do this in a later example). However, in order to better accommodate calling pre-existing C routines, which often do modify their input parameters, this behavior is tolerated.