This example requires the following typemap entry. Consult the typemap section for more information about adding new typemaps for an extension.

```
TYPEMAP
Netconfig *  T_PTROBJ
```

This example will be used with the following Perl statements.

```
use RPC;
$netconf = getnetconfigent("udp");
```

When Perl destroys the object referenced by $netconf it will send the object to the supplied XSUB DESTROY function. Perl cannot determine, and does not care, that this object is a C struct and not a Perl object. In this sense, there is no difference between the object created by the getnetconfigent() XSUB and an object created by a normal Perl subroutine.

68.1.45 The Typemap

The typemap is a collection of code fragments which are used by the **xsubpp** compiler to map C function parameters and values to Perl values. The typemap file may consist of three sections labelled `TYPEMAP`, `INPUT`, and `OUTPUT`. An unlabelled initial section is assumed to be a `TYPEMAP` section. The INPUT section tells the compiler how to translate Perl values into variables of certain C types. The OUTPUT section tells the compiler how to translate the values from certain C types into values Perl can understand. The TYPEMAP section tells the compiler which of the INPUT and OUTPUT code fragments should be used to map a given C type to a Perl value. The section labels `TYPEMAP`, `INPUT`, or `OUTPUT` must begin in the first column on a line by themselves, and must be in uppercase.

The default typemap in the `lib/ExtUtils` directory of the Perl source contains many useful types which can be used by Perl extensions. Some extensions define additional typemaps which they keep in their own directory. These additional typemaps may reference INPUT and OUTPUT maps in the main typemap. The **xsubpp** compiler will allow the extension's own typemap to override any mappings which are in the default typemap.

Most extensions which require a custom typemap will need only the TYPEMAP section of the typemap file. The custom typemap used in the getnetconfigent() example shown earlier demonstrates what may be the typical use of extension typemaps. That typemap is used to equate a C structure with the T_PTROBJ typemap. The typemap used by getnetconfigent() is shown here. Note that the C type is separated from the XS type with a tab and that the C unary operator * is considered to be a part of the C type name.

```
TYPEMAP
Netconfig *<tab>T_PTROBJ
```

Here's a more complicated example: suppose that you wanted `struct netconfig` to be blessed into the class `Net::Config`. One way to do this is to use underscores (_) to separate package names, as follows:

```
typedef struct netconfig * Net_Config;
```

And then provide a typemap entry `T_PTROBJ_SPECIAL` that maps underscores to double-colons (::), and declare `Net_Config` to be of that type:

```
TYPEMAP
Net_Config      T_PTROBJ_SPECIAL

INPUT
T_PTROBJ_SPECIAL
        if (sv_derived_from($arg, \"${(my $ntt=$ntype)=~s/_/::/g;\$ntt}\")) {
                IV tmp = SvIV((SV*)SvRV($arg));
        $var = ($type) tmp;
        }
        else
                croak(\"$var is not of type ${(my $ntt=$ntype)=~s/_/::/g;\$ntt}\")
```