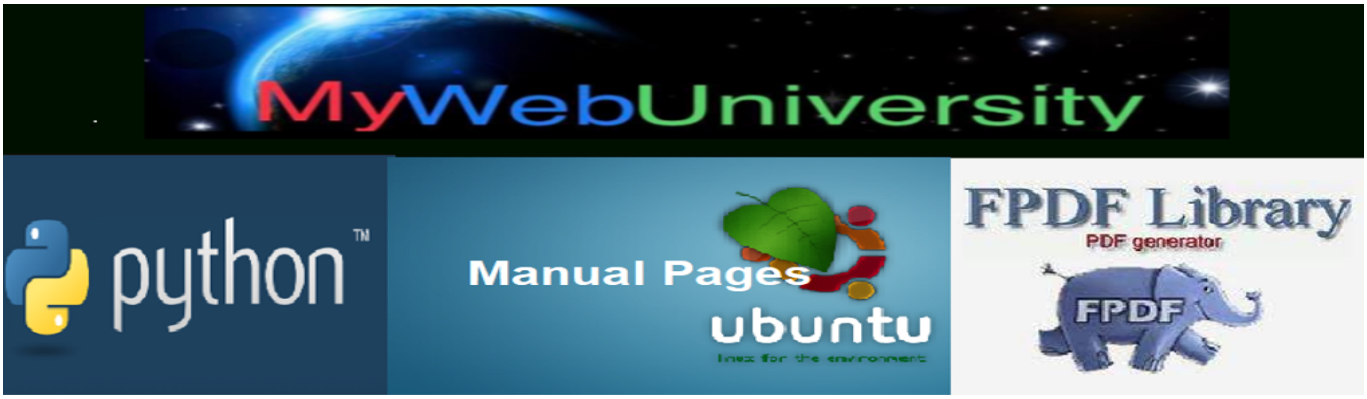


**Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!**

## *Rocky Enterprise Linux 9.2 Manual Pages on command 'Devel::SelfStubber.3perl'*

***$ man Devel::SelfStubber.3perl***

Devel::SelfStubber(3perl) Perl Programmers Reference Guide Devel::SelfStubber(3perl)

NAME

Devel::SelfStubber - generate stubs for a SelfLoading module

SYNOPSIS

To generate just the stubs:

```
use Devel::SelfStubber;
Devel::SelfStubber->stub('MODULENAME','MY_LIB_DIR');
```

or to generate the whole module with stubs inserted correctly

```
use Devel::SelfStubber;
$Devel::SelfStubber::JUST_STUBS=0;
Devel::SelfStubber->stub('MODULENAME','MY_LIB_DIR');
```

MODULENAME is the Perl module name, e.g. Devel::SelfStubber, NOT 'Devel/SelfStubber' or 'Devel/SelfStubber.pm'.

MY_LIB_DIR defaults to '.' if not present.

DESCRIPTION

Devel::SelfStubber prints the stubs you need to put in the module before the __DATA__ token (or you can get it to print the entire module with stubs correctly placed). The stubs ensure that if a method is called, it will get loaded. They are needed specifically for inherited autoloaded methods.

This is best explained using the following example:

Assume four classes, A,B,C & D.

A is the root class, B is a subclass of A, C is a subclass of B, and D is another subclass of A.


If D calls an autoloaded method 'foo' which is defined in class A, then the method is loaded into class A, then executed. If C then calls method 'foo', and that method was reimplemented in class B, but set to be autoloaded, then the lookup mechanism never gets to the AUTOLOAD mechanism in B because it first finds the method already loaded in A, and so erroneously uses that. If the method foo had been stubbed in B, then the lookup mechanism would have found the stub, and correctly loaded and used the sub from B.

So, for classes and subclasses to have inheritance correctly work with autoloading, you need to ensure stubs are loaded.

The SelfLoader can load stubs automatically at module initialization with the statement 'SelfLoader->load_stubs()';, but you may wish to avoid having the stub loading overhead associated with your initialization (though note that the SelfLoader::load_stubs method will be called sooner or later - at latest when the first sub is being autoloaded). In this case, you can put the sub stubs before the __DATA__ token. This can be done manually, but this module allows automatic generation of the stubs.

By default it just prints the stubs, but you can set the global $Devel::SelfStubber::JUST_STUBS to 0 and it will print out the entire module with the stubs positioned correctly.

At the very least, this is useful to see what the SelfLoader thinks are stubs - in order to ensure future versions of the SelfStubber remain in step with the SelfLoader, the SelfStubber actually uses the SelfLoader to determine which stubs are needed.