

# Re: Using References to Formats, Examining Scalars With Devel::Peek

---

*Source:* <http://coding.derkeiler.com/Archive/Perl/comp.lang.perl.misc/2006-07/msg00604.html>

---

- *From:* "Veli-Pekka Tättilä" <[vtatila@xxxxxxxxxxxxxxxxxxxxxxxx](mailto:vtatila@xxxxxxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Wed, 12 Jul 2006 00:03:53 +0300
- 

Hi,  
Replying a bit late and slightly out-of-order, too.

attn.steven.kuo@xxxxxxxx wrote:  
I wrote:

To make format references useful at all, I suppose one would have to be able to dereference them somehow. IS that possible, and if so how?

Well, just use another typeglob for dereferencing.

```
sub writeForm
{ # Write out the specified format.
local $~ = shift;
write;
} # sub
```

becomes:

```
use Scalar::Util ('reftype');
```

```
sub writeForm {
if (reftype $_[0] and reftype $_[0] eq 'FORMAT') {
local *FOO;
*FOO = $_[0];
local $~ = 'FOO';
write;
} else {
local $~ = shift;
write;
}
}
```

Hey, thanks for the code. Works fine for me. Somehow aliasing the name of the format to be able to refer to it by another name didn't occur to me. But

## Re: Using References to Formats, Examining Scalars With Devel::Peek

as the built-in functions seem to support using a stringified scalar as a format name, I doubt if using format refs have any benefits at all. It just serves to make the syntax more gory, eh. No wonder they haven't been documented properly.

You can use Devel::Peek instead of Data::Dumper if you want to look at the guts of a format reference.

Ah, nice. Seems most of the fields are the same as for the rest of the Perl data types. The Peek module can be quite handy, I think. I initially overlooked it, when going through all the built-in modules that seemed interesting, because the title talks about XS and I'm not ready to tackle it yet.

Your note about the function got me playing around with other Perl variables and browsing PERLguts to understand at a high level what's going on behind the scenes.

In particular, I've been tracking when exactly scalars change their datatype from integer to double and how frequently the string portions are updated. Again I don't think even PerlGuts covers the rules but I might be wrong, as I didn't read it all through.

I've only experimented a little but it would seem to me that:

- Perl is pretty smart in keeping integer values integers. Even if you add and subtract doubles you might still end up with an integer value, if the result happens to fit exactly to an int.
- Using an operator or a built-in with a double and an int usually converts the int to a double if you assign the result to a variable. This is natural, of course.
- There don't seem to be many instances in which variables which are now doubles would be converted to ints. Even the int function doesn't do that, if I'm interpreting these funny type abbreviations right.
- Perl appears to update all but the most recent types lazily. Sometimes the string or double values can lag behind considerably. They do get updated when current values for the types are needed such as in stringification or calling functions taking doubles.

--

With kind regards Veli-Pekka Tättilä (vtatila@xxxxxxxxxxxxxxxxxxxxxx)  
Accessibility, game music, synthesizers and programming:  
<http://www.student oulu.fi/~vtatila/>

.