



Data::Dumper and Data::Dump::Streamer

Alberto Manuel Simões
ams@di.uminho.pt

Readers have been asking for more beginner-level material, so Alberto is starting a series on the core modules of Perl, and we're calling it Alberto's Module Beginners' Seminar. If you have a request for a particular module or task, please let us know at editors@thepperlreview.com.

One of the most well known modules in the Perl standard distribution is, without any doubt, `Data::Dumper`. It is the data structure debugging module *par excellence*.

Ninety percent of its usage is for debugging. When building a data structure, I need to print it to the screen or a file so I can see if I've built it as I should. In this example, I start with an anonymous array that has as an element an anonymous hash, then add another anonymous hash as the element with index 3 (so there's nothing at index 2). After I've created this array, I want to make sure it's what I wanted, so I use `Data::Dumper`'s `Dumper` function:

```
use Data::Dumper;

my $structure
  = ['foo', { bar => 'ugh' }];
$structure -> [3]
  = { zbr => 'xpto' };

print Dumper($structure);
```

I pass `Dumper` a reference to a structure and it returns a string that represents the structure for that reference. For the previous code, it dumps:

```
$VAR1 = [
    'foo',
    {
        'bar' => 'ugh'
    },
    undef,
    {
        'zbr' => 'xpto'
    }
];
```

I could just write a simple function on my own to dump structures like this one with little effort, but

it wouldn't be as versatile as `Data::Dumper`, which allows me to twist the output just as I like by changing a few variables. For instance, I can change `$Data::Dumper::Indent` to a value from 0 to 3, to control how many spaces and newlines I get. It goes from none (0) to the one displayed above (2), or with some more comments (3). If I don't like the name `$VAR1`, I can control the variable name used with `$Data::Dumper::Vartname`. Other variables control a lot of other little details so I can get just what I want.

The other 10% of `Data::Dumper` users use it as a simple method for data persistence. It is easy to dump a configuration structure and load it again. Most people forget about a built-in Perl function named `do`. I pass it a filename and it evaluates the Perl code on the file, returning the result of the evaluation.

Now, if I pass `do` a file with a structure dump, it executes it as Perl code and I get it back just as if I'd typed that code right there in my program.

```
$structure = do 'file.dump';
```

`Data::Dumper` can do a lot more for me too. It understands blessed references, code references, circular references, and a lot of other things.

Data::Dump::Streamer

`Data::Dumper` has a brother module named `Data::Dump::Streamer` (DDS), less known, but as useful as the first.

The major difference between DDS and `Data::Dumper` is that `Data::Dumper` creates the full output (the data structure being serialized) to memory before really outputting it, while DDS outputs it as soon as it can, thus consuming less memory.

Another difference is that DDS does a breadth-first instead of a depth-first traversal of the data done by `Data::Dumper`. This makes DDS traverse the