The Magic of tie

brian d foy
brian@stonehenge.com
Stonehenge Consulting Services
May 26, 2006
Dallas-Ft. Worth Perl Mongers

Fun with variables

* Tied variables are really hidden objects
* The objects have a hidden interface to make them act like normal variables
* Behind that interface, we do we what

Normal variables

```
$value = 5;
$other_value = $value;

@array = qw( a b c );
$array[3] = 'd';
$last = $#array
shift @array;

%hash = map { $_, 1 } @array;
$hash{a} = 2;
delete $hash{b};
my @keys = keys %hash;
```

Do what we want

* The tie interface let's us decide what to do in each case
* We can store values any way we like

```
tie my $scalar, 'Tie::Foo', @args;
tie my @array, 'Tie::Bar', @args;
tie my %hash, 'Tie::Quux', @args;
tie my $fh, 'Tie::Baz', @args;
```

Do what the user knows

* Scalars, arrays, hashes
* Common, "Learning Perl" level syntax
* Hide all of the complexity

DBM::Deep

Persistent hashes

```
use DBM::Deep;
my $db = DBM::Deep->new( "foo.db" );

$db->{key} = 'value';
print $db->{key};

# true multi-level support
$db->{my_complex} = [
        'hello', { perl => 'rules' },
        42, 99 ];
```

Tie:: modules

Tie::Scalar Tie::Array Tie::Hash Tie::Handle
Tie::File
Tie::Cycle Tie::Toggle Tie::FlipFlop
Tie::Handle::CSV
Tie::SortHash
IO::Scalar IO::String

Two variables

* The normal looking variable
* and it's secret life as an object

```
my $object =
   tie my $scalar,
      'Tie::ClassName',
      @args;
```

We do this...	Perl does this...
`$scalar`	`$object->FETCH`
`$scalar = 5`	`$object->STORE(5)`

Scalars

`tie my $scalar, 'Tie::Foo', @args;`	`my $object = Tie::Foo->TIESCALAR( @args )`
`$n = $scalar;`	`$n = $object->FETCH;`
`$scalar = 5;`	`$object->STORE(5);`
`tied($scalar)->foo(5);`	`$object->foo(5);`

and a few others: DESTROY, UNTIE

Tie::Cycle

```
use Tie::Cycle;

tie my $cycle, 'Tie::Cycle',
     [ qw( FFFFFF 000000 FFFF00 ) ];

print $cycle; # FFFFFF
print $cycle; # 000000
print $cycle; # FFFF00
print $cycle; # FFFFFF  back to the beginning

(tied $cycle)->reset;  # back to the beginning
```

IO::Scalar

```
use 5.005;
use IO::Scalar;
$data = "My message:\n";

### Open a handle on a string
$SH = IO::Scalar->new( \$data );
$SH->print("Hello");
$SH->print(", world!\nBye now!\n");
print "The string is now: ", $data, "\n";
```

== OR ==

```
open my $fh, ">", \$variable;
```

Tie::Scalar

* Base class for tied scalars
* Override for the parts you need

Arrays

`tie my @array, 'Tie::Foo', @args;`	`my $object = Tie::Foo->TIEARRAY( @args )`
`$n = $array[$i];`	`$n = $object->FETCH($i);`
`$array[$i] = 5;`	`$object->STORE($i, 5);`
`$n = @array`	`$object->FETCHSIZE;`
`$#array = $n;`	`$object->STORESIZE($n+1);`
`$#array += $n;`	`$object->EXTEND($n);`
`@array = ();`	`$object->CLEAR;`

and a few others: DELETE, EXISTS, PUSH, POP, SHIFT, UNSHIFT, SPLICE, DESTROY, UNTIE

Tie::Array

* Base class for tied arrays
* Override the parts you need

Hashes

tie my %hash, 'Tie::Foo', @args;	my $obj = Tie::Foo->TIEHASH(@args)
$n = $hash{ $key };	$n = $obj->FETCH($key);
$hash{ $key } = 5;	$obj->STORE($key, 5);
delete $hash{ $key };	$obj->DELETE($key)
exists $hash{ $key };	$obj->EXISTS($key)
%hash = ();	$obj->CLEAR
my @keys = keys %hash;	@keys = ($obj->FIRSTKEY, $obj->NEXTKEY ...)

and a few others: SCALAR, UNTIE, DESTROY

Tie::Hash

* Base class for tied hashes
* Override the parts you need

Filehandles

`tie my $fh, 'Tie::Foo', @args;`	`my $object = Tie::Foo->TIEHANDLE( @args )`
`print $fh @stuff;`	`$object->PRINT(@stuff);`
`my $line = <$fh>`	`$object->READLINE;`
`close $fh;`	`$object->CLOSE;`

and a few others: WRITE, PRINTF, READ, DESTROY, UNTIE

Tie::Handle

* Base class for tied handles
* Override the parts you need