NAME

Type::Tiny::Manual – an overview of Type::Tiny

SYNOPSIS

Type::Tiny is a small Perl <http://www.perl.org/> class for writing type constraints, inspired by Moose's type constraint API and MooseX::Types. It has only one non-core dependency (and even that is simply a module that was previously distributed as part of Type::Tiny but has since been spun off), and can be used with Moose, Mouse, or Moo (or none of the above).

Type::Tiny is used by over 800 Perl distributions on the CPAN (Comprehensive Perl Archive Network) and can be considered a stable and mature framework for efficiently and reliably enforcing data types.

Type::Tiny is bundled with Type::Library a framework for organizing type constraints into collections. Also bundled is Types::Standard, a Moose-inspired library of useful type constraints. Type::Params is also provided, to allow very fast checking and coercion of function and method parameters.

The following example gives you an idea of some of the features of these modules. If you don't understand it all, that's fine; that's what the rest of the manual is for. Although the example uses Moo, the `use Moo` could be changed to `use Moose` or `use Mouse` and it would still work.

```
use v5.12;
use strict;
use warnings;

package Horse {
  use Moo;
  use Types::Standard qw( Str Int Enum ArrayRef InstanceOf );
  use Type::Params qw( compile );
  use namespace::autoclean;

  has name => (
    is       => 'ro',
    isa      => Str,
    required => 1,
  );
  has gender => (
    is       => 'ro',
    isa      => Enum[qw( f m )],
  );
  has age => (
    is       => 'rw',
    isa      => Int->where( '$_ >= 0' ),
  );
  has children => (
    is       => 'ro',
    isa      => ArrayRef[ InstanceOf['Horse'] ],
    default  => sub { return [] },
  );

  sub add_child {
    # method signature
    state $check = compile( InstanceOf['Horse'], InstanceOf['Horse'] );

    my ($self, $child) = $check->(@_);   # unpack @_
    push @{ $self->children }, $child;

    return $self;
```

```
  }
}

package main;

my $boldruler = Horse->new(
  name    => "Bold Ruler",
  gender  => 'm',
  age     => 16,
);

my $secretariat = Horse->new(
  name    => "Secretariat",
  gender  => 'm',
  age     => 0,
);

$boldruler->add_child( $secretariat );

use Types::Standard qw( is_Object assert_Object );

# is_Object will return a boolean
#
if ( is_Object($boldruler) ) {
  say $boldruler->name;
}

# assert_Object will return $secretariat or die
#
say assert_Object($secretariat)->name;
```

MANUAL

Even if you are using Type::Tiny with other object-oriented programming toolkits (such as Moose or Mouse), you should start with the Moo sections of the manual. Most of the information is directly transferrable and the Moose and Mouse sections of the manual list the minor differences between using Type::Tiny with Moo and with them.

In general, this manual assumes you use Perl 5.12 or above and may use examples that do not work on older versions of Perl. Type::Tiny does work on earlier versions of Perl, but not all the examples and features in the manual will run without adjustment. (For instance, you may need to replace `state` variables with lexical variables, avoid the `package NAME { BLOCK }` syntax, etc.)

- Type::Tiny::Manual::Installation

  How to install Type::Tiny. If Type::Tiny is already installed, you can skip this.

- Type::Tiny::Manual::UsingWithMoo

  Basic use of Type::Tiny with Moo, including attribute type constraints, parameterized type constraints, coercions, and method parameter checking.

- Type::Tiny::Manual::UsingWithMoo2

  Advanced use of Type::Tiny with Moo, including unions and intersections, `stringifies_to`, `numifies_to`, `with_attribute_values`, and `where`.

- Type::Tiny::Manual::UsingWithMoo3

  There's more than one way to do it! Alternative ways of using Type::Tiny, including type registries, exported functions, and `dwim_type`.

- Type::Tiny::Manual::Libraries

  Defining your own type libraries, including extending existing libraries, defining new types, adding coercions, defining parameterizable types, and the declarative style.

- Type::Tiny::Manual::UsingWithMoose

  How to use Type::Tiny with Moose, including the advantages of Type::Tiny over built-in type constraints, and Moose-specific features.

- Type::Tiny::Manual::UsingWithMouse

  How to use Type::Tiny with Mouse, including the advantages of Type::Tiny over built-in type constraints, and Mouse-specific features.

- Type::Tiny::Manual::UsingWithClassTiny

  Including how to Type::Tiny in your object's `BUILD` method, and third-party shims between Type::Tiny and Class::Tiny.

- Type::Tiny::Manual::UsingWithOther

  Using Type::Tiny with Class::InsideOut, Params::Check, and Object::Accessor.

- Type::Tiny::Manual::UsingWithTestMore

  Type::Tiny for test suites.

- Type::Tiny::Manual::Params

  Advanced information on Type::Params, and using Type::Tiny with other signature modules like Function::Parameters and Kavorka.

- Type::Tiny::Manual::NonOO

  Type::Tiny in non-object-oriented code.

- Type::Tiny::Manual::Optimization

  Squeeze the most out of your CPU.

- Type::Tiny::Manual::Coercions

  Advanced information on coercions.

- Type::Tiny::Manual::AllTypes

  An alphabetical list of all type constraints bundled with Type::Tiny.

- Type::Tiny::Manual::Policies

  Policies related to Type::Tiny development.

- Type::Tiny::Manual::Contributing

  Contributing to Type::Tiny development.

BUGS

Please report any bugs to <http://rt.cpan.org/Dist/Display.html?Queue=Type−Tiny>.

SEE ALSO

The Type::Tiny homepage <http://typetiny.toby.ink/>.

AUTHOR

Toby Inkster <tobyink@cpan.org>.

COPYRIGHT AND LICENCE

DISCLAIMER OF WARRANTIES