



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 'TAP::Parser::IteratorFactory.3perl'

\$ man TAP::Parser::IteratorFactory.3perl

TAP::Parser::IteratorFactory(3perl) Perl Programmers Reference Guide TAP::Parser::IteratorFactory(3perl)

NAME

TAP::Parser::IteratorFactory - Figures out which SourceHandler objects to use for a given Source

VERSION

Version 3.43

SYNOPSIS

```
use TAP::Parser::IteratorFactory;

my $factory = TAP::Parser::IteratorFactory->new({ %config });

my $iterator = $factory->make_iterator( $filename );
```

DESCRIPTION

This is a factory class that takes a TAP::Parser::Source and runs it through all the registered TAP::Parser::SourceHandlers to see which one should handle the source.

If you're a plugin author, you'll be interested in how to "register_handler"s, how "detect_source" works.

METHODS

Class Methods

"new"

Creates a new factory class:

```
my $sf = TAP::Parser::IteratorFactory->new( $config );
```

\$config is optional. If given, sets "config" and calls "load_handlers".

"register_handler"

Registers a new TAP::Parser::SourceHandler with this factory.

```
__PACKAGE__->register_handler( $handler_class );
```

"handlers"

List of handlers that have been registered.

Instance Methods

"config"

```
my $cfg = $sf->config;  
$sf->config({ Perl => { %config } });
```

Chaining getter/setter for the configuration of the available source handlers. This is a hashref keyed on handler class whose values contain config to be passed onto the handlers during detection & creation. Class names may be fully qualified or abbreviated, eg:

```
# these are equivalent  
$sf->config({ 'TAP::Parser::SourceHandler::Perl' => { %config } });  
$sf->config({ 'Perl' => { %config } });
```

"load_handlers"

```
$sf->load_handlers;
```

Loads the handler classes defined in "config". For example, given a config:

```
$sf->config({  
  MySourceHandler => { some => 'config' },  
});
```

"load_handlers" will attempt to load the "MySourceHandler" class by looking in @INC for it in this order:

```
TAP::Parser::SourceHandler::MySourceHandler  
MySourceHandler
```

"croak"s on error.

"make_iterator"

```
my $iterator = $src_factory->make_iterator( $source );
```

Given a TAP::Parser::Source, finds the most suitable TAP::Parser::SourceHandler to use to create a TAP::Parser::Iterator (see "detect_source"). Dies on error.

"detect_source"

Given a TAP::Parser::Source, detects what kind of source it is and returns one TAP::Parser::SourceHandler (the most confident one). Dies on error.

The detection algorithm works something like this:

```
for (@registered_handlers) {  
  # ask them how confident they are about handling this source
```

```
$confidence{$handler} = $handler->can_handle( $source )
}

# choose the most confident handler
```

Ties are handled by choosing the first handler.

SUBCLASSING

Please see "SUBCLASSING" in TAP::Parser for a subclassing overview.

Example

If we've done things right, you'll probably want to write a new source, rather than subclassing this (see TAP::Parser::SourceHandler for that).

But in case you find the need to...

```
package MyIteratorFactory;

use strict;

use base 'TAP::Parser::IteratorFactory';

# override source detection algorithm
sub detect_source {
    my ($self, $raw_source_ref, $meta) = @_ ;
    # do detective work, using $meta and whatever else...
}

1;
```

AUTHORS

Steve Purkis

ATTRIBUTION

Originally ripped off from Test::Harness.

Moved out of TAP::Parser & converted to a factory class to support extensible TAP source detective work by Steve Purkis.

SEE ALSO

TAP::Object, TAP::Parser, TAP::Parser::SourceHandler, TAP::Parser::SourceHandler::File,
TAP::Parser::SourceHandler::Perl, TAP::Parser::SourceHandler::RawTAP,
TAP::Parser::SourceHandler::Handle, TAP::Parser::SourceHandler::Executable

perl v5.34.0

2023-11-23

TAP::Parser::IteratorFactory(3perl)