NAME

Pod::Simple::HTMLBatch - convert several Pod files to several HTML files

SYNOPSIS

```
perl -MPod::Simple::HTMLBatch -e 'Pod::Simple::HTMLBatch::go' in out
```

DESCRIPTION

This module is used for running batch-conversions of a lot of HTML documents

This class is NOT a subclass of Pod::Simple::HTML (nor of bad old Pod::Html) -- although it uses Pod::Simple::HTML for doing the conversion of each document.

The normal use of this class is like so:

```
use Pod::Simple::HTMLBatch;
my $batchconv = Pod::Simple::HTMLBatch->new;
$batchconv->some_option( some_value );
$batchconv->some_other_option( some_other_value );
$batchconv->batch_convert( \@search_dirs, $output_dir );
```

FROM THE COMMAND LINE

Note that this class also provides (but does not export) the function Pod::Simple::HTMLBatch::go. This is basically just a shortcut for `Pod::Simple::HTMLBatch->batch_convert(@ARGV)`. It's meant to be handy for calling from the command line.

However, the shortcut requires that you specify exactly two command-line arguments, `indirs` and `outdir`.

Example:

```
  % mkdir out_html
  % perl -MPod::Simple::HTMLBatch -e Pod::Simple::HTMLBatch::go @INC
out_html
      (to convert the pod from Perl's @INC
       files under the directory ./out_html)
```

(Note that the command line there contains a literal atsign-I-N-C. This is handled as a special case by batch_convert, in order to save you having to enter the odd-looking "" as the first command-line parameter when you mean "just use whatever's in @INC".)

Example:

```
  % mkdir ../seekrut
  % chmod og-rx ../seekrut
  % perl -MPod::Simple::HTMLBatch -e Pod::Simple::HTMLBatch::go .
../htmlversion
      (to convert the pod under the current dir into HTML
       files under the directory ./seekrut)
```

Example:

```
  % perl -MPod::Simple::HTMLBatch -e Pod::Simple::HTMLBatch::go happydocs .
      (to convert all pod from happydocs into the current directory)
```

MAIN METHODS

$batchconv = Pod::Simple::HTMLBatch->new;

This TODO

$batchconv->batch_convert(*indirs*, *outdir*);

this TODO

$batchconv->batch_convert(undef , ...);

$batchconv->batch_convert(q{@INC}, ...);

These two values for *indirs* specify that the normal Perl @INC

$batchconv->batch_convert(\@dirs , ...);

This specifies that the input directories are the items in the arrayref `\@dirs`.

$batchconv->batch_convert("somedir" , ...);

This specifies that the director "somedir" is the input. (This can be an absolute or relative path, it doesn't matter.)

A common value you might want would be just "." for the current directory:

```
$batchconv->batch_convert( "." , ...);
```

$batchconv->batch_convert('somedir:someother:also' , ...);

This specifies that you want the dirs "somedir", "someother", and "also" scanned, just as if you'd passed the arrayref `[qw( somedir someother also)]`. Note that a ":"-separator is normal under Unix, but Under MSWin, you'll need `'somedir;someother;also'` instead, since the pathsep on MSWin is ";" instead of ":". (And *that* is because ":" often comes up in paths, like `"c:/perl/lib"`.)

(Exactly what separator character should be used, is gotten from `$Config::Config{'path_sep'}`, via the *Config* module.)

$batchconv->batch_convert(... , undef);

This specifies that you want the HTML output to go into the current directory.

(Note that a missing or undefined value means a different thing in the first slot than in the second. That's so that `batch_convert()` with no arguments (or undef arguments) means "go from @INC, into the current directory.)

$batchconv->batch_convert(... , 'somedir');

This specifies that you want the HTML output to go into the directory 'somedir'. (This can be an absolute or relative path, it doesn't matter.)

Note that you can also call `batch_convert` as a class method, like so:

```
Pod::Simple::HTMLBatch->batch_convert( ... );
```

That is just short for this:

```
Pod::Simple::HTMLBatch-> new-> batch_convert(...);
```

That is, it runs a conversion with default options, for whatever inputdirs and output dir you specify.

ACCESSOR METHODS

The following are all accessor methods -- that is, they don't do anything on their own, but just alter the contents of the conversion object, which comprises the options for this particular batch conversion.

We show the "put" form of the accessors below (i.e., the syntax you use for setting the accessor to a

specific value). But you can also call each method with no parameters to get its current value. For example, `$self->contents_file()` returns the current value of the contents_file attribute.

$batchconv->verbose(*nonnegative_integer*);

This controls how verbose to be during batch conversion, as far as notes to STDOUT (or whatever is `select`'d) about how the conversion is going. If 0, no progress information is printed. If 1 (the default value), some progress information is printed. Higher values print more information.

$batchconv->index(*true-or-false*);

This controls whether or not each HTML page is liable to have a little table of contents at the top (which we call an "index" for historical reasons). This is true by default.

$batchconv->contents_file(*filename*);

If set, should be the name of a file (in the output directory) to write the HTML index to. The default value is "index.html". If you set this to a false value, no contents file will be written.

$batchconv->contents_page_start(*HTML_string*);

This specifies what string should be put at the beginning of the contents page. The default is a string more or less like this:

```
<html>
<head><title>Perl Documentation</title></head>
<body class='contentspage'>
<h1>Perl Documentation</h1>
```

$batchconv->contents_page_end(*HTML_string*);

This specifies what string should be put at the end of the contents page. The default is a string more or less like this:

```
<p class='contentsfooty'>Generated by
Pod::Simple::HTMLBatch v3.01 under Perl v5.008
<br >At Fri May 14 22:26:42 2004 GMT,
which is Fri May 14 14:26:42 2004 local time.</p>
```

$batchconv->add_css($url);

TODO

$batchconv->add_javascript($url);

TODO

$batchconv->css_flurry(*true-or-false*);

If true (the default value), we autogenerate some CSS files in the output directory, and set our HTML files to use those. TODO: continue

$batchconv->javascript_flurry(*true-or-false*);

If true (the default value), we autogenerate a JavaScript in the output directory, and set our HTML files to use it. Currently, the JavaScript is used only to get the browser to remember what stylesheet it prefers. TODO: continue

$batchconv->no_contents_links(*true-or-false*);

TODO

$batchconv->html_render_class(*classname*);

This sets what class is used for rendering the files. The default is "Pod::Simple::HTML". If you set it to something else, it should probably be a subclass of Pod::Simple::HTML, and you

should `require` or `use` that class so that's it's loaded before Pod::Simple::HTMLBatch tries loading it.

$batchconv->search_class(*classname*);

This sets what class is used for searching for the files. The default is "Pod::Simple::Search". If you set it to something else, it should probably be a subclass of Pod::Simple::Search, and you should `require` or `use` that class so that's it's loaded before Pod::Simple::HTMLBatch tries loading it.

NOTES ON CUSTOMIZATION

TODO

```
  call add_css($someurl) to add stylesheet as alternate
  call add_css($someurl,1) to add as primary stylesheet


  call add_javascript


  subclass Pod::Simple::HTML and set $batchconv->html_render_class to
    that classname
  and maybe override
    $page->batch_mode_page_object_init($self, $module, $infile, $outfile,
$depth)
  or maybe override
    $batchconv->batch_mode_page_object_init($page, $module, $infile,
$outfile, $depth)
  subclass Pod::Simple::Search and set $batchconv->search_class to
    that classname
```

ASK ME!

If you want to do some kind of big pod-to-HTML version with some particular kind of option that you don't see how to achieve using this module, email me (`sburke@cpan.org`) and I'll probably have a good idea how to do it. For reasons of concision and energetic laziness, some methods and options in this module (and the dozen modules it depends on) are undocumented; but one of those undocumented bits might be just what you're looking for.

SEE ALSO

Pod::Simple, *Pod::Simple::HTMLBatch*, *perlpod*, *perlpodspec*

SUPPORT

Questions or discussion about POD and Pod::Simple should be sent to the pod-people@perl.org mail list. Send an empty email to pod-people-subscribe@perl.org to subscribe.

This module is managed in an open GitHub repository, *http://github.com/theory/pod-simple/*. Feel free to fork and contribute, or to clone *git://github.com/theory/pod-simple.git* and send patches!

Patches against Pod::Simple are welcome. Please send bug reports to <bug-pod-simple@rt.cpan.org>.

COPYRIGHT AND DISCLAIMERS

AUTHOR

Pod::Simple was created by Sean M. Burke <sburke@cpan.org>. But don't bother him, he's retired.

Pod::Simple is maintained by:

* Allison Randal `allison@perl.org`
* Hans Dieter Pearcey `hdp@cpan.org`
* David E. Wheeler `dwheeler@cpan.org`