NAME

CPANPLUS::Backend

SYNOPSIS

```
my $cb      = CPANPLUS::Backend->new;
my $conf    = $cb->configure_object;

my $author  = $cb->author_tree('KANE');
my $mod     = $cb->module_tree('Some::Module');
my $mod     = $cb->parse_module( module => 'Some::Module' );

my @objs    = $cb->search(  type    => TYPE,
                            allow   => [...] );

$cb->flush('all');
$cb->reload_indices;
$cb->local_mirror;
```

DESCRIPTION

This module provides the programmer's interface to the CPANPLUS libraries.

ENVIRONMENT

When `CPANPLUS::Backend` is loaded, which is necessary for just about every <CPANPLUS> operation, the environment variable `PERL5_CPANPLUS_IS_RUNNING` is set to the current process id.

Additionally, the environment variable `PERL5_CPANPLUS_IS_VERSION` will be set to the version of `CPANPLUS::Backend`.

This information might be useful somehow to spawned processes.

METHODS

$cb = CPANPLUS::Backend->new([CONFIGURE_OBJ])

This method returns a new `CPANPLUS::Backend` object. This also initialises the config corresponding to this object. You have two choices in this:

Provide a valid `CPANPLUS::Configure` object

This will be used verbatim.

No arguments

Your default config will be loaded and used.

New will return a `CPANPLUS::Backend` object on success and die on failure.

$href = $cb->module_tree([@modules_names_list])

Returns a reference to the CPANPLUS module tree.

If you give it any arguments, they will be treated as module names and `module_tree` will try to look up these module names and return the corresponding module objects instead.

See *CPANPLUS::Module* for the operations you can perform on a module object.

$href = $cb->author_tree([@author_names_list])

Returns a reference to the CPANPLUS author tree.

If you give it any arguments, they will be treated as author names and `author_tree` will try to look up these author names and return the corresponding author objects instead.

See *CPANPLUS::Module::Author* for the operations you can perform on an author object.

$conf = $cb->configure_object;

Returns a copy of the `CPANPLUS::Configure` object.

See *CPANPLUS::Configure* for operations you can perform on a configure object.

$su = $cb->selfupdate_object;

Returns a copy of the `CPANPLUS::Selfupdate` object.

See the *CPANPLUS::Selfupdate* manpage for the operations you can perform on the selfupdate object.

@mods = $cb->search(type => TYPE, allow => AREF, [data => AREF, verbose => BOOL])

`search` enables you to search for either module or author objects, based on their data. The `type` you can specify is any of the accessors specified in `CPANPLUS::Module::Author` or `CPANPLUS::Module`. `search` will determine by the `type` you specified whether to search by author object or module object.

You have to specify an array reference of regular expressions or strings to match against. The rules used for this array ref are the same as in `Params::Check`, so read that manpage for details.

The search is an `or` search, meaning that if `any` of the criteria match, the search is considered to be successful.

You can specify the result of a previous search as `data` to limit the new search to these module or author objects, rather than the entire module or author tree. This is how you do `and` searches.

Returns a list of module or author objects on success and false on failure.

See *CPANPLUS::Module* for the operations you can perform on a module object. See *CPANPLUS::Module::Author* for the operations you can perform on an author object.

$backend_rv = $cb->fetch(modules => \@mods)

Fetches a list of modules. `@mods` can be a list of distribution names, module names or module objects--basically anything that *parse_module* can understand.

See the equivalent method in `CPANPLUS::Module` for details on other options you can pass.

Since this is a multi-module method call, the return value is implemented as a `CPANPLUS::Backend::RV` object. Please consult that module's documentation on how to interpret the return value.

$backend_rv = $cb->extract(modules => \@mods)

Extracts a list of modules. `@mods` can be a list of distribution names, module names or module objects--basically anything that *parse_module* can understand.

See the equivalent method in `CPANPLUS::Module` for details on other options you can pass.

Since this is a multi-module method call, the return value is implemented as a `CPANPLUS::Backend::RV` object. Please consult that module's documentation on how to interpret the return value.

$backend_rv = $cb->install(modules => \@mods)

Installs a list of modules. `@mods` can be a list of distribution names, module names or module objects--basically anything that *parse_module* can understand.

See the equivalent method in `CPANPLUS::Module` for details on other options you can pass.

Since this is a multi-module method call, the return value is implemented as a

CPANPLUS::Backend::RV object. Please consult that module's documentation on how to interpret the return value.

$backend_rv = $cb->readme(modules => \@mods)

Fetches the readme for a list of modules. @mods can be a list of distribution names, module names or module objects--basically anything that *parse_module* can understand.

See the equivalent method in CPANPLUS::Module for details on other options you can pass.

Since this is a multi-module method call, the return value is implemented as a CPANPLUS::Backend::RV object. Please consult that module's documentation on how to interpret the return value.

$backend_rv = $cb->files(modules => \@mods)

Returns a list of files used by these modules if they are installed. @mods can be a list of distribution names, module names or module objects--basically anything that *parse_module* can understand.

See the equivalent method in CPANPLUS::Module for details on other options you can pass.

Since this is a multi-module method call, the return value is implemented as a CPANPLUS::Backend::RV object. Please consult that module's documentation on how to interpret the return value.

$backend_rv = $cb->distributions(modules => \@mods)

Returns a list of module objects representing all releases for this module on success. @mods can be a list of distribution names, module names or module objects, basically anything that *parse_module* can understand.

See the equivalent method in CPANPLUS::Module for details on other options you can pass.

Since this is a multi-module method call, the return value is implemented as a CPANPLUS::Backend::RV object. Please consult that module's documentation on how to interpret the return value.

$mod_obj = $cb->parse_module(module => $modname|$distname|$modobj|URI)

parse_module tries to find a CPANPLUS::Module object that matches your query. Here's a list of examples you could give to parse_module;

Text::Bastardize

Text-Bastardize

Text-Bastardize-1.06

AYRNIEU/Text-Bastardize

AYRNIEU/Text-Bastardize-1.06

AYRNIEU/Text-Bastardize-1.06.tar.gz

http://example.com/Text-Bastardize-1.06.tar.gz

file:///tmp/Text-Bastardize-1.06.tar.gz

These items would all come up with a CPANPLUS::Module object for Text::Bastardize. The ones marked explicitly as being version 1.06 would give back a CPANPLUS::Module object of that version. Even if the version on CPAN is currently higher.

If parse_module is unable to actually find the module you are looking for in its module tree, but you supplied it with an author, module and version part in a distribution name or URI, it will create a fake CPANPLUS::Module object for you, that you can use just like the real thing.

See *CPANPLUS::Module* for the operations you can perform on a module object.

If even this fancy guessing doesn't enable `parse_module` to create a fake module object for you to use, it will warn about an error and return false.

$bool = $cb->reload_indices([update_source => BOOL, verbose => BOOL]);

This method reloads the source files.

If `update_source` is set to true, this will fetch new source files from your CPAN mirror. Otherwise, `reload_indices` will do its usual cache checking and only update them if they are out of date.

By default, `update_source` will be false.

The verbose setting defaults to what you have specified in your config file.

Returns true on success and false on failure.

$bool = $cb->flush(CACHE_NAME)

This method allows flushing of caches. There are several things which can be flushed:

* `methods`

  The return status of methods which have been attempted, such as different ways of fetching files. It is recommended that automatic flushing be used instead.

* `hosts`

  The return status of URIs which have been attempted, such as different hosts of fetching files. It is recommended that automatic flushing be used instead.

* `modules`

  Information about modules such as prerequisites and whether installation succeeded, failed, or was not attempted.

* `lib`

  This resets PERL5LIB, which is changed to ensure that while installing modules they are in our @INC.

* `load`

  This resets the cache of modules we've attempted to load, but failed. This enables you to load them again after a failed load, if they somehow have become available.

* `all`

  Flush all of the aforementioned caches.

Returns true on success and false on failure.

@mods = $cb->installed()

Returns a list of module objects of all your installed modules. If an error occurs, it will return false.

See *CPANPLUS::Module* for the operations you can perform on a module object.

$bool = $cb->local_mirror([path => '/dir/to/save/to', index_files => BOOL, force => BOOL, verbose => BOOL])

Creates a local mirror of CPAN, of only the most recent sources in a location you specify. If you set this location equal to a custom host in your `CPANPLUS::Config` you can use your local mirror to install from.

It takes the following arguments:

path

The location where to create the local mirror.

index_files

Enable/disable fetching of index files. You can disable fetching of the index files if you don't plan to use the local mirror as your primary site, or if you'd like up-to-date index files be fetched from elsewhere.

Defaults to true.

force

Forces refetching of packages, even if they are there already.

Defaults to whatever setting you have in your `CPANPLUS::Config`.

verbose

Prints more messages about what its doing.

Defaults to whatever setting you have in your `CPANPLUS::Config`.

Returns true on success and false on error.

$file = $cb->autobundle([path => OUTPUT_PATH, force => BOOL, verbose => BOOL])

Writes out a snapshot of your current installation in `CPAN` bundle style. This can then be used to install the same modules for a different or on a different machine.

It will, by default, write to an 'autobundle' directory under your cpanplus homedirectory, but you can override that by supplying a `path` argument.

It will return the location of the output file on success and false on failure.

CUSTOM MODULE SOURCES

Besides the sources as provided by the general `CPAN` mirrors, it's possible to add your own sources list to your `CPANPLUS` index.

The methodology behind this works much like `Debian's apt-sources`.

The methods below show you how to make use of this functionality. Also note that most of these methods are available through the default shell plugin command `/cs`, making them available as shortcuts through the shell and via the commandline.

%files = $cb->list_custom_sources

Returns a mapping of registered custom sources and their local indices as follows:

```
/full/path/to/local/index => http://remote/source
```

Note that any file starting with an # is being ignored.

$local_index = $cb->add_custom_source(uri => URI, [verbose => BOOL]);

Adds an `URI` to your own sources list and mirrors its index. See the documentation on `$cb->update_custom_source` on how this is done.

Returns the full path to the local index on success, or false on failure.

Note that when adding a new `URI`, the change to the in-memory tree is not saved until you rebuild or save the tree to disk again. You can do this using the `$cb->reload_indices` method.

$local_index = $cb->remove_custom_source(uri => URI, [verbose => BOOL]);

Removes an `URI` from your own sources list and removes its index.

To find out what `URI`s you have as part of your own sources list, use the `$cb->list_custom_sources` method.

Returns the full path to the deleted local index file on success, or false on failure.

$bool = $cb->update_custom_source([remote => URI]);

Updates the indexes for all your custom sources. It does this by fetching a file called `packages.txt` in the root of the custom sources's `URI`. If you provide the `remote` argument, it will only update the index for that specific `URI`.

Here's an example of how custom sources would resolve into index files:

```
file:///path/to/sources       =>  file:///path/to/sources/packages.txt
http://example.com/sources    =>  http://example.com/sources/packages.txt
ftp://example.com/sources     =>  ftp://example.com/sources/packages.txt
```

The file `packages.txt` simply holds a list of packages that can be found under the root of the `URI`. This file can be automatically generated for you when the remote source is a `file:// URI`. For `http://`, `ftp://`, and similar, the administrator of that repository should run the method `$cb->write_custom_source_index` on the repository to allow remote users to index it.

For details, see the `$cb->write_custom_source_index` method below.

All packages that are added via this mechanism will be attributed to the author with `CPANID LOCAL`. You can use this id to search for all added packages.

$file = $cb->write_custom_source_index(path => /path/to/package/root, [to => /path/to/index/file, verbose => BOOL]);

Writes the index for a custom repository root. Most users will not have to worry about this, but administrators of a repository will need to make sure their indexes are up to date.

The index will be written to a file called `packages.txt` in your repository root, which you can specify with the `path` argument. You can override this location by specifying the `to` argument, but in normal operation, that should not be required.

Once the index file is written, users can then add the `URI` pointing to the repository to their custom list of sources and start using it right away. See the `$cb->add_custom_source` method for user details.

BUG REPORTS

Please report bugs or other issues to <bug-cpanplus@rt.cpan.org<gt>.

AUTHOR

This module by Jos Boumans <kane@cpan.org>.

COPYRIGHT

SEE ALSO

CPANPLUS::Configure, *CPANPLUS::Module*, *CPANPLUS::Module::Author*, *CPANPLUS::Selfupdate*