NAME

File::HomeDir – Find your home and other directories on any platform

SYNOPSIS

```
use File::HomeDir;

# Modern Interface (Current User)
$home     = File::HomeDir->my_home;
$desktop  = File::HomeDir->my_desktop;
$docs     = File::HomeDir->my_documents;
$music    = File::HomeDir->my_music;
$pics     = File::HomeDir->my_pictures;
$videos   = File::HomeDir->my_videos;
$data     = File::HomeDir->my_data;
$dist     = File::HomeDir->my_dist_data('File-HomeDir');
$dist     = File::HomeDir->my_dist_config('File-HomeDir');

# Modern Interface (Other Users)
$home    = File::HomeDir->users_home('foo');
$desktop = File::HomeDir->users_desktop('foo');
$docs    = File::HomeDir->users_documents('foo');
$music   = File::HomeDir->users_music('foo');
$pics    = File::HomeDir->users_pictures('foo');
$video   = File::HomeDir->users_videos('foo');
$data    = File::HomeDir->users_data('foo');
```

DESCRIPTION

File::HomeDir is a module for locating the directories that are ''owned'' by a user (typically your user) and to solve the various issues that arise trying to find them consistently across a wide variety of platforms.

The end result is a single API that can find your resources on any platform, making it relatively trivial to create Perl software that works elegantly and correctly no matter where you run it.

Platform Neutrality

In the Unix world, many different types of data can be mixed together in your home directory (although on some Unix platforms this is no longer the case, particularly for ''desktop''−oriented platforms).

On some non-Unix platforms, separate directories are allocated for different types of data and have been for a long time.

When writing applications on top of **File::HomeDir**, you should thus always try to use the most specific method you can. User documents should be saved in `my_documents`, data that supports an application but isn't normally editing by the user directory should go into `my_data`.

On platforms that do not make any distinction, all these different methods will harmlessly degrade to the main home directory, but on platforms that care **File::HomeDir** will always try to Do The Right Thing(tm).

METHODS

Two types of methods are provided. The `my_method` series of methods for finding resources for the current user, and the `users_method` (read as ''user's method'') series for finding resources for arbitrary users.

This split is necessary, as on most platforms it is **much** easier to find information about the current user compared to other users, and indeed on a number you cannot find out information such as `users_desktop` at all, due to security restrictions.

All methods will double check (using a `-d` test) that a directory actually exists before returning it, so you may trust in the values that are returned (subject to the usual caveats of race conditions of directories being deleted at the moment between a directory being returned and you using it).

However, because in some cases platforms may not support the concept of home directories at all, any

method may return `undef` (both in scalar and list context) to indicate that there is no matching directory on the system.

For example, most untrusted 'nobody'−type users do not have a home directory. So any modules that are used in a CGI application that at some level of recursion use your code, will result in calls to File::HomeDir returning undef, even for a basic *home()* call.

my_home

The `my_home` method takes no arguments and returns the main home/profile directory for the current user.

If the distinction is important to you, the term ''current'' refers to the real user, and not the effective user.

This is also the case for all of the other ''my'' methods.

Returns the directory path as a string, `undef` if the current user does not have a home directory, or dies on error.

my_desktop

The `my_desktop` method takes no arguments and returns the ''desktop'' directory for the current user.

Due to the diversity and complexity of implementations required to deal with implementing the required functionality fully and completely, the `my_desktop` method may or may not be implemented on each platform.

That said, I am extremely interested in code to implement `my_desktop` on Unix, as long as it is capable of dealing (as the Windows implementation does) with internationalization. It should also avoid false positive results by making sure it only returns the appropriate directories for the appropriate platforms.

Returns the directory path as a string, `undef` if the current user does not have a desktop directory, or dies on error.

my_documents

The `my_documents` method takes no arguments and returns the directory (for the current user) where the user's documents are stored.

Returns the directory path as a string, `undef` if the current user does not have a documents directory, or dies on error.

my_music

The `my_music` method takes no arguments and returns the directory where the current user's music is stored.

No bias is made to any particular music type or music program, rather the concept of a directory to hold the user's music is made at the level of the underlying operating system or (at least) desktop environment.

Returns the directory path as a string, `undef` if the current user does not have a suitable directory, or dies on error.

my_pictures

The `my_pictures` method takes no arguments and returns the directory where the current user's pictures are stored.

No bias is made to any particular picture type or picture program, rather the concept of a directory to hold the user's pictures is made at the level of the underlying operating system or (at least) desktop environment.

Returns the directory path as a string, `undef` if the current user does not have a suitable directory, or dies on error.

my_videos

The `my_videos` method takes no arguments and returns the directory where the current user's videos are stored.

No bias is made to any particular video type or video program, rather the concept of a directory to hold the user's videos is made at the level of the underlying operating system or (at least) desktop environment.

Returns the directory path as a string, `undef` if the current user does not have a suitable directory, or dies

on error.

my_data

The `my_data` method takes no arguments and returns the directory where local applications should store their internal data for the current user.

Generally an application would create a subdirectory such as `.foo`, beneath this directory, and store its data there. By creating your directory this way, you get an accurate result on the maximum number of platforms. But see the documentation about `my_dist_config()` or `my_dist_data()` below.

For example, on Unix you get `~/.foo` and on Win32 you get `~/Local Settings/Application Data/.foo`

Returns the directory path as a string, `undef` if the current user does not have a data directory, or dies on error.

my_dist_config

```
File::HomeDir->my_dist_config( $dist [, \%params] );

# For example...

File::HomeDir->my_dist_config( 'File-HomeDir' );
File::HomeDir->my_dist_config( 'File-HomeDir', { create => 1 } );
```

The `my_dist_config` method takes a distribution name as argument and returns an application-specific directory where they should store their internal configuration.

The base directory will be either `my_config` if the platform supports it, or `my_documents` otherwise. The subdirectory itself will be `BASE/Perl/Dist-Name`. If the base directory is the user's home directory, `my_dist_config` will be in `~/.perl/Dist-Name` (and thus be hidden on all Unixes).

The optional last argument is a hash reference to tweak the method behaviour. The following hash keys are recognized:

- create

  Passing a true value to this key will force the creation of the directory if it doesn't exist (remember that `File::HomeDir`'s policy is to return `undef` if the directory doesn't exist).

  Defaults to false, meaning no automatic creation of directory.

my_dist_data

```
File::HomeDir->my_dist_data( $dist [, \%params] );

# For example...

File::HomeDir->my_dist_data( 'File-HomeDir' );
File::HomeDir->my_dist_data( 'File-HomeDir', { create => 1 } );
```

The `my_dist_data` method takes a distribution name as argument and returns an application-specific directory where they should store their internal data.

This directory will be of course a subdirectory of `my_data`. Platforms supporting data-specific directories will use `DATA_DIR/perl/dist/Dist-Name` following the common "DATA/vendor/application" pattern. If the `my_data` directory is the user's home directory, `my_dist_data` will be in `~/.perl/dist/Dist-Name` (and thus be hidden on all Unixes).

The optional last argument is a hash reference to tweak the method behaviour. The following hash keys are recognized:

- create

  Passing a true value to this key will force the creation of the directory if it doesn't exist (remember that `File::HomeDir`'s policy is to return `undef` if the directory doesn't exist).

Defaults to false, meaning no automatic creation of directory.

users_home

```
$home = File::HomeDir->users_home('foo');
```

The `users_home` method takes a single parameter and is used to locate the parent home/profile directory for an identified user on the system.

While most of the time this identifier would be some form of user name, it is permitted to vary per-platform to support user ids or UUIDs as applicable for that platform.

Returns the directory path as a string, `undef` if that user does not have a home directory, or dies on error.

users_documents

```
$docs = File::HomeDir->users_documents('foo');
```

Returns the directory path as a string, `undef` if that user does not have a documents directory, or dies on error.

users_data

```
$data = File::HomeDir->users_data('foo');
```

Returns the directory path as a string, `undef` if that user does not have a data directory, or dies on error.

users_desktop

```
$docs = File::HomeDir->users_desktop('foo');
```

Returns the directory path as a string, `undef` if that user does not have a desktop directory, or dies on error.

users_music

```
$docs = File::HomeDir->users_music('foo');
```

Returns the directory path as a string, `undef` if that user does not have a music directory, or dies on error.

users_pictures

```
$docs = File::HomeDir->users_pictures('foo');
```

Returns the directory path as a string, `undef` if that user does not have a pictures directory, or dies on error.

users_videos

```
$docs = File::HomeDir->users_videos('foo');
```

Returns the directory path as a string, `undef` if that user does not have a videos directory, or dies on error.

FUNCTIONS

home

```
use File::HomeDir;
$home = home();
$home = home('foo');
$home = File::HomeDir::home();
$home = File::HomeDir::home('foo');
```

The `home` function is exported by default and is provided for compatibility with legacy applications. In new applications, you should use the newer method-based interface above.

Returns the directory path to a named user's home/profile directory.

If provided no parameter, returns the directory path to the current user's home/profile directory.

TO DO

- Add more granularity to Unix, and add support to VMS and other esoteric platforms, so we can consider going core.
- Add consistent support for users_* methods

SUPPORT

This module is stored in an Open Repository at the following address.

<http://svn.ali.as/cpan/trunk/File-HomeDir>

Write access to the repository is made available automatically to any published CPAN author, and to most other volunteers on request.

If you are able to submit your bug report in the form of new (failing) unit tests, or can apply your fix directly instead of submitting a patch, you are **strongly** encouraged to do so as the author currently maintains over 100 modules and it can take some time to deal with non-Critical bug reports or patches.

This will guarantee that your issue will be addressed in the next release of the module.

If you cannot provide a direct test or fix, or don't have time to do so, then regular bug reports are still accepted and appreciated via the CPAN bug tracker.

<http://rt.cpan.org/NoAuth/ReportBug.html?Queue=File−HomeDir>

For other issues, for commercial enhancement or support, or to have your write access enabled for the repository, contact the author at the email address above.

ACKNOWLEDGEMENTS

The biggest acknowledgement goes to Chris Nandor, who wielded his legendary Mac-fu and turned my initial fairly ordinary Darwin implementation into something that actually worked properly everywhere, and then donated a Mac OS X license to allow it to be maintained properly.

AUTHORS

Adam Kennedy <adamk@cpan.org>

Sean M. Burke <sburke@cpan.org>

Chris Nandor <cnandor@cpan.org>

Stephen Steneker <stennie@cpan.org>

SEE ALSO

File::ShareDir, File::HomeDir::Win32 (legacy)

COPYRIGHT