NAME

File::DesktopEntry − Object to handle .desktop files

SYNOPSIS

```
use File::DesktopEntry;

my $entry = File::DesktopEntry->new('firefox');

print "Using ".$entry->Name." to open http://perl.org\n";
$entry->run('http://perl.org');
```

DESCRIPTION

This module is designed to work with *.desktop* files. The format of these files is specified by the freedesktop ‘‘Desktop Entry’’ specification. This module can parse these files but also knows how to run the applications defined by these files.

For this module version 1.0 of the specification was used.

This module was written to support File::MimeInfo::Applications.

Please remember: case is significant for the names of Desktop Entry keys.

VARIABLES

You can set the global variable `$File::DesktopEntry::VERBOSE`. If set the module prints a warning every time a command gets executed.

The global variable `$File::DesktopEntry::LOCALE` tells you what the default locale being used is. However, changing it will not change the default locale.

AUTOLOAD

All methods that start with a capital are autoloaded as `get(KEY)` where key is the autoloaded method name.

METHODS

`new(FILE)`
`new(\$TEXT)`
`new(NAME)`

Constructor. FILE, NAME or TEXT are optional arguments.

When a name is given (a string without ’`/`’, ’`\`’ or ’`.`’) a lookup is done using File::BaseDir. If the file found in this lookup is not writable or if no file was found, the XDG_DATA_HOME path will be used when writing.

`lookup(NAME)`

Returns a filename for a desktop entry with desktop file id NAME.

`wants_uris( )`

Returns true if the Exec string for this desktop entry specifies that the application uses URIs instead of paths. This can be used to determine whether an application uses a VFS library.

`wants_list( )`

Returns true if the Exec string for this desktop entry specifies that the application can handle multiple arguments at once.

`run(@FILES)`

Forks and runs the application specified in this Desktop Entry with arguments FILES as a background process. Returns the pid.

The child process fails when this is not a Desktop Entry of type Application or if the Exec key is missing or invalid.

If the desktop entry specifies that the program needs to be executed in a terminal the `$TERMINAL` environment variable is used. If this variable is not set `x-terminal-emulator -e` is used as default on Debian systems.

(On Windows this method returns a Win32::Process object.)

`system(@FILES)`
: Like `run()` but using the `system()` system call. It only return after the application has ended.

`exec(@FILES)`
: Like `run()` but using the `exec()` system call. This method is expected not to return but to replace the current process with the application you try to run.

  On Windows this method doesn't always work the way you want it to due to the `fork()` emulation on this platform. Try using `run()` or `system()` instead.

`parse_Exec(@FILES)`
: Expands the Exec format in this desktop entry with. Returns a properly quoted string in scalar context or a list of words in list context. Dies when the Exec key is invalid.

  It supports the following fields:

```
%f    single file
%F    multiple files
%u    single url
%U    multiple urls
%i    Icon field prefixed by --icon
%c    Name field, possibly translated
%k    location of this .desktop file
%%    literal '%'
```

  If necessary this method tries to convert between paths and URLs but this is not perfect.

  Fields that are deprecated, but (still) supported by this module:

```
%d    single directory
%D    multiple directories
```

  The fields `%n`, `%N`, `%v` and `%m` are deprecated and will cause a warning if `$VERBOSE` is used. Any other unknown fields will cause an error.

  The fields `%F`, `%U`, `%D` and `%i` can only occur as separate words because they expand to multiple arguments.

  Also see "LIMITATIONS".

`get(KEY)`
`get(GROUP, KEY)`
: Get a value for KEY from GROUP. If GROUP is not specified 'Desktop Entry' is used. All values are treated as string, so e.g. booleans will be returned as the literal strings "true" and "false".

  When KEY does not contain a language code you get the translation in the current locale if available or a sensible default. The request a specific language you can add the language part. E.g. `$entry->get('Name[nl_NL]')` can return either the value of the 'Name[nl_NL]', the 'Name[nl]' or the 'Name' key in the Desktop Entry file. Exact language parsing order can be found in the spec. To force you get the untranslated key use either 'Name[C]' or 'Name[POSIX]'.

`set(KEY => VALUE, ...)`
`set(GROUP, KEY => VALUE, ...)`
: Set values for one or more keys. If GROUP is not given "Desktop Entry" is used. All values are treated as strings, backslashes, newlines and tabs are escaped. To set a boolean key you need to use the literal strings "true" and "false".

  Unlike the `get()` call languages are not handled automatically for `set()`. KEY should include the language part if you want to set a translation. E.g. `$entry->set("Name[nl_NL]" => "Tekst Verwerker")`will set a Dutch translation for the Name k ey. Using either "Name[C]" or "Name[POSIX]" will be equivalent with not giving a language argument.

When setting the Exec key without specifying a group it will be parsed and quoted correctly as required by the spec. You can use quoted arguments to include whitespace in a argument, escaping whitespace does not work. To circumvent this quoting explicitly give the group name 'Desktop Entry'.

`text()`
: Returns the (modified) text of the file.

`read(FILE)`
`read(\$SCALAR)`
: Read Desktop Entry data from file or memory buffer. Without argument defaults to file given at constructor.

If you gave a file, text buffer or name to the constructor this method will be called automatically.

`read_fh(IO)`
: Read Desktop Entry data from filehandle or IO object.

`write(FILE)`
: Write the Desktop Entry data to FILE. Without arguments it writes to the filename given to the constructor if any.

The keys Name and Type are required. Type can be either `Application`, `Link` or `Directory`. For an application set the optional key `Exec`. For a link set the `URL` key.

Backwards Compatibility

Methods supported for backwards compatibility with 0.02.

`new_from_file(FILE)`
: Alias for `new(FILE)`.

`new_from_data(TEXT)`
: Alias for `new(\$TEXT)`.

`get_value(NAME, GROUP, LANG)`
: Identical to `get(GROUP, "NAME[LANG]")`. LANG defaults to 'C', GROUP is optional.

NON-UNIX PLATFORMS

This module has a few bits of code to make it work on Windows. It handles `file://` uri a bit different and it uses Win32::Process. On other platforms your mileage may vary.

Please note that the specification is targeting Unix platforms only and will only have limited relevance on other platforms. Any platform-dependent behavior in this module should be considered an extension of the spec.

LIMITATIONS

If you try to exec a remote file with an application that can only handle files on the local file system we should −according to the spec− download the file to a temp location. This is not supported. Use the `wants_uris()` method to check if an application supports urls.

The values of the various Desktop Entry keys are not parsed (except for the Exec key). This means that booleans will be returned as the strings "true" and "false" and lists will still be ";" separated.

If the icon is given as name and not as path it should be resolved for the `%i` code in the Exec key. We need a separate module for the icon spec to deal with this.

According to the spec comments can contain any encoding. However since this module read files as utf8, invalid UTF−8 characters in a comment will cause an error.

There is no support for Legacy-Mixed Encoding. Everybody is using utf8 now ... right ?

AUTHOR

Jaap Karssenberg (Pardus) <pardus@cpan.org>

Maintained by Michiel Beijen <michielb@cpan.org>

LICENSE

SEE ALSO

<http://standards.freedesktop.org/desktop−entry−spec/desktop−entry−spec−latest.html>

File::BaseDir and File::MimeInfo::Applications

X11::FreeDesktop::DesktopEntry