

NAME

AptPkg::Config – APT configuration interface

SYNOPSIS

```
use AptPkg::Config;
```

DESCRIPTION

The AptPkg::Config module provides an interface to **APT**'s configuration mechanism.

Provides a configuration file and command line parser for a tree-oriented configuration environment.

AptPkg::Config

The AptPkg::Config package implements the **APT** Configuration class.

A global instance of the libapt-pkg _config instance is provided as `$AptPkg::Config::_config`, and may be imported.

The following methods are implemented:

`get(KEY, [DEFAULT])`

Fetch the value of *KEY* from the configuration object, returning undef if not found (or *DEFAULT* if given).

If the key ends in `::`, an array of values is returned in an array context, or a string containing the values separated by spaces in a scalar context.

A trailing `/f`, `/d`, `/b` or `/i` causes file, directory, boolean or integer interpretation (the underlying XS call is `FindAny`).

`get_file(KEY, [DEFAULT]), get_dir(KEY, [DEFAULT])`

Variants of `get` which prepend the directory value from the parent key. The `get_dir` method additionally appends a `'/'`.

For example, given the configuration file:

```
foo "/some/dir/" { bar "value"; }
```

then:

```
$conf->get("foo::bar")      # "value"
$conf->get_file("foo::bar") # "/some/dir/value"
$conf->get_dir("foo::bar")  # "/some/dir/value/"
```

`get_bool(KEY, [DEFAULT])`

Another `get` variant, which returns true (1) if the value contains any of:

```
1 yes true with on enable
```

otherwise false (").

`set(KEY, VALUE)`

Set configuration entry *KEY* to *VALUE*. Returns *VALUE*. Note that empty parent entries may be created for *KEY*s containing `::`.

`exists(KEY)`

Test if *KEY* exists in the configuration.

`dump`

Principally for debugging, output the contents of the configuration object to `stderr`.

`read_file(FILE, [AS_SECTIONAL, [DEPTH]])`

Load the contents of *FILE* into the object. The format of the file is described in `apt.conf` (5).

If the *AS_SECTIONAL* argument is true, then the file is parsed as a BIND-style config. That is:

```
foo "bar" { baz "quux"; }
```

is interpreted as if it were:

```
foo::bar { baz "quux"; }
```

The *DEPTH* argument may be used to restrict the number of nested include directives processed.

`read_dir(DIR, [AS_SECTIONAL, [DEPTH]])`

Load configuration from all files in *DIR*.

`init` Initialise the configuration object with some default values for the libapt-pkg library and reads the default configuration file `/etc/apt/apt.conf` (or as given by the environment variable `APT_CONFIG`) if it exists.

`system`

Return the `AptPkg::System` object appropriate for this system.

`parse_cmdline(DEFS, [ARG, ...])`

Parse the arguments given by *ARGs* based on the contents of *DEFS* and returns the list of remaining arguments.

Note, the function does not return if there are errors processing the args. Use `eval` to trap such errors.

DEFS is a reference to an array containing a set argument definition arrays. The elements of each definition define: the short argument character, the long argument string, the configuration key and the optional argument type (defaults to Boolean).

Valid argument types are defined by the strings:

<code>HasArg</code>	takes an argument value (<code>-f foo</code>)
<code>IntLevel</code>	defines an integer value (<code>-q -q, -qq, -q2, -q=2</code>)
<code>Boolean</code>	true/false (<code>-d, -d=true, -d=yes, --no-d, -d=false, etc</code>)
<code>InvBoolean</code>	same as Boolean but false with no specified sense (<code>-d</code>)
<code>ConfigFile</code>	load the specified configuration file
<code>ArbItem</code>	arbitrary configuration string of the form <code>key=value</code>

The configuration key in the last two cases is ignored, and for the rest gives the key into which the value is placed.

Single case equivalents also work (`has_arg == HasArg`).

Example:

```
@files = $conf->parse_cmdline([
    [ 'h', 'help', 'help' ],
    [ 'v', 'version', 'version' ],
    [ 'c', 'config-file', '', ConfigFile ],
    [ 'o', 'option', '', ArbItem ],
], @ARGV);
```

The module uses `AptPkg::hash` to provide a hash-like access to the object, so that `$conf->{key}` is equivalent to using the `get/set` methods.

Additionally inherits the constructor (`new`) and keys methods from that module.

Methods of the internal XS object (`AptPkg::_config`) such as `Find` may also be used. See `AptPkg`.

AptPkg::Config::Iter

Iterator object for `AptPkg::Config` which is returned by the `keys` method.

`new(XS_OBJ, [ROOT])`

Constructor, which is invoked by the `keys` method. *ROOT*, if given determines the subset of the tree to walk (may be given as an argument to `keys`).

`next`

Returns the current key and advances to the next.

SEE ALSO

AptPkg::System (3pm), **AptPkg** (3pm), **AptPkg::hash** (3pm).

AUTHOR

Brendan O'Dea <bod@debian.org>