peterodding.com / code / vim / misc

vim-misc Miscellaneous auto-load Vim scripts

GitHub: Watched by 124 people, most recent update was 125 months ago

Vim Online: Rated 19, downloaded 12104 times

The vim-misc plug-in contains Vim scripts that are used by most of the Vim plug-ins I've written yet don't really belong with any single one of the plug-ins. Basically it's an extended standard library of Vim script functions that I wrote during the development of my Vim profile and plug-ins.

In the past these scripts were bundled with each plug-in, however that turned out to be a maintenance nightmare for me. That's why the miscellaneous scripts are now a proper plug-in with their own page on Vim Online.

Because the miscellaneous scripts are no longer bundled with my Vim plug-ins, users are now required to install the miscellaneous scripts separately. This is unfortunate for users who are upgrading from a previous release that did bundle the miscellaneous scripts, but I don't see any way around this. Sorry!

Installation

Please refer to the installation instructions on GitHub.

Function documentation

Below is the documentation for the functions included in the miscellaneous scripts. Anyone is free to use these functions in their own Vim profile and/or plug-ins. I care about backwards compatibility so won't break it without a good reason to do so.

For those who are curious: The function descriptions given below were extracted from the source code of the miscellaneous scripts using the Python module `vimdoctool.py` included in vim-tools.

The documentation of the 95 functions below was extracted from 19 Vim scripts on April 1, 2015 at 23:39.

Asynchronous Vim script evaluation

The `xolox#misc#async#call()` function builds on top of `xolox#misc#os#exec()` to support asynchronous evaluation of Vim scripts. The first (and for now only) use case is my vim-easytags plug-in which has a bunch of conflicting requirements:

1. I want the vim-easytags plug-in to be as portable as possible. Ideally everything is implemented in Vim script because that's the only thing I can rely on to be available for all potential users of the plug-in!

2. Because of point one I've been forced to implement tags file reading, parsing, (fold case) sorting and writing in Vim script. This is fine for small tags files but once they grow to a couple of megabytes it becomes annoying because Vim is unresponsive during tags file updates (key presses are fortunately buffered due to Vim's input model but that doesn't make it a nice user experience :-).

3. I could (and did in the past) come up with all sorts of hacks to speed things up without switching away from Vim script, but none of them are going to solve the fundamental problem that Vim's unresponsive hiccups become longer as tags files grow larger.

By now it should be clear where this is heading: *Why not handle tags file updates in a Vim process that runs in the background without blocking the Vim process that the user is interacting with?* It turns out that there are quite a few details to take care of, but with those out of the way, it might just work! I'm actually hoping to make asynchronous updates the default mode in vim-easytags. This means I need this functionality to be as portable and robust as possible.

Status: This code has seen little testing so I wouldn't trust it too much just yet. On the other hand, as I said, my intention is to make this functionality as portable and robust as possible. You be the judge :-).

The `xolox#misc#async#call()` function

Call a Vim script function asynchronously by starting a hidden Vim process in the background. Once the function returns the hidden Vim process terminates itself. This function takes a single

argument which is a dictionary with the following key/value pairs:

- **function** (required): The name of the Vim function to call inside the child process (a string). I suggest using an autoload function for this, see below.
- **arguments** (optional): A list of arguments to pass to the function. This list is serialized to a string using string() and deserialized using eval().
- **callback** (optional): The name of a Vim function to call in the parent process when the child process has completed (a string).
- **clientserver** (optional): If this is true (1) the child process will notify the parent process when it has finished (the default is true). This works using Vim's client/server support which is not always available. As a fall back Vim's CursorHold automatic command is also supported (although the effect is not quite as instantaneous :-).

This functionality is experimental and non trivial to use, so consider yourself warned :-).

Limitations

I'm making this functionality available in vim-misc because I think it can be useful to other plug-ins, however if you are going to use it you should be aware of the following limitations:

- Because of the use of multiple processes this functionality is only suitable for 'heavy' tasks.
- The function arguments are serialized to a string which is passed to the hidden Vim process as a command line argument, so the amount of data you can pass will be limited by your operating environment.
- The hidden Vim process is explicitly isolated from the user in several ways (see below for more details). This is to make sure that the hidden Vim processes are fast and don't clobber the user's editing sessions in any way.

Changes to how Vim normally works

You have to be aware that the hidden Vim process is initialized in a specific way that is very different from your regular Vim editing sessions:

- Your vimrc file is ignored using the `-u NONE` command line option.

- Your gvimrc file (if you even knew it existed ;-) is ignored using the `-U NONE` command line option.
- Plug-in loading is skipped using the `--noplugin` command line option.
- Swap files (see swap-file) are disabled using the `-n` command line option. This makes sure asynchronous Vim processes don't disturb the user's editing session.
- Your viminfo file is ignored using the `-i NONE` command line option. Just like with swap files this makes sure asynchronous Vim processes don't disturb the user's editing session.
- No-compatible mode is enabled using the `-N` command line option (usually the existence of your vimrc script would have achieved the same effect but since we disable loading of your vimrc we need to spell things out for Vim).

Use an auto-load function

The function you want to call is identified by its name which has to be defined, but I just explained above that all regular initialization is disabled for asynchronous Vim processes, so what gives? The answer is to use an autoload function. This should work fine because the asynchronous Vim process 'inherits' the value of the 'runtimepath' option from your editing session.

The `xolox#misc#async#inside_child()` function

Entry point inside the hidden Vim process that runs in the background. Invoked indirectly by `xolox#misc#async#call()` because it runs a command similar to the following:

```
vim --cmd 'call xolox#misc#async#inside_child(...)'
```

This function is responsible for calling the user defined function, capturing exceptions and reporting the results back to the parent Vim process using Vim's client/server support or a temporary file.

The `xolox#misc#async#callback_to_parent()` function

When Vim was compiled with client/server support this function (in the parent process) will be called by `xolox#misc#async#inside_child()` (in the child process) after the user

defined function has returned. This enables more or less instant callbacks after running an asynchronous function.

The `xolox#misc#async#periodic_callback()` function

When client/server support is not being used the vim-misc plug-in improvises: It uses Vim's CursorHold event to periodically check if an asynchronous process has written its results to one of the expected temporary files. If a response is found the temporary file is read and deleted and then `xolox#misc#async#callback_to_parent()` is called to process the response.

Handling of special buffers

The functions defined here make it easier to deal with special Vim buffers that contain text generated by a Vim plug-in. For example my vim-notes plug-in generates several such buffers:

- :RecentNotes lists recently modified notes
- :ShowTaggedNotes lists notes grouped by tags
- etc.

Because the text in these buffers is generated, Vim shouldn't bother with swap files and it should never prompt the user whether to save changes to the generated text.

The `xolox#misc#buffer#is_empty()` function

Checks if the current buffer is an empty, unchanged buffer which can be reused. Returns 1 if an empty buffer is found, 0 otherwise.

The `xolox#misc#buffer#prepare()` function

Open a special buffer, i.e. a buffer that will hold generated contents, not directly edited by the user. The buffer can be customized by passing a dictionary with the following key/value pairs as the first argument:

- **name** (required): The base name of the buffer (i.e. the base name of the file loaded in the buffer, even though it isn't really a file and nothing is really 'loaded' :-)
- **path** (required): The pathname of the buffer. May be relevant if :lcd or 'autochdir' is being used.

The `xolox#misc#buffer#lock()` function

Lock a special buffer so that its contents can no longer be edited.

The `xolox#misc#buffer#unlock()` function

Unlock a special buffer so that its content can be updated.

Tab completion for user defined commands

The `xolox#misc#complete#keywords()` function

This function can be used to perform keyword completion for user defined Vim commands based on the contents of the current buffer. Here's an example of how you would use it:

```
:command -nargs=* -complete=customlist,xolox#misc#complete#keywords MyCmd ca
```

Rate limiting for Vim's CursorHold event

Several of my Vim plug-ins (e.g. vim-easytags, vim-notes and vim-session) use Vim's CursorHold and CursorHoldI events to perform periodic tasks when the user doesn't press any keys for a couple of seconds. These events by default fire after four seconds, this is configurable using Vim's 'updatetime' option. The problem that this script solves is that there are Vim plug-ins which set the 'updatetime' option to unreasonably low values, thereby breaking my Vim plug-ins and probably a lot of other Vim plug-ins out there. When users complain about this I can tell them that another Vim plug-in is to blame, but users don't care for the difference, their Vim is broken! So I implemented a workaround. This script enables registration of CursorHold event handlers with a configurable interval (expressed in seconds). The event handlers will be called no more than once every interval.

The `xolox#misc#cursorhold#register()` function

Register a CursorHold event handler with a custom interval. This function takes a single argument which is a dictionary with the following fields:

- **function** (required): The name of the event handler function (a string).
- **arguments** (optional): A list of arguments to pass to the event handler function (defaults to an empty list).

- **interval** (optional): The number of seconds between calls to the event handler (defaults to 4).

The `xolox#misc#cursorhold#autocmd()` function

The 'top level event handler' that's called by Vim whenever the CursorHold or CursorHoldI event fires. It iterates through the event handlers registered using `xolox#misc#cursorhold#register()` and calls each event handler at the appropriate interval, keeping track of the time when each event handler was last run.

String escaping functions

The `xolox#misc#escape#pattern()` function

Takes a single string argument and converts it into a :substitute / substitute() pattern string that matches the given string literally.

The `xolox#misc#escape#substitute()` function

Takes a single string argument and converts it into a :substitute / substitute() replacement string that inserts the given string literally.

The `xolox#misc#escape#shell()` function

Takes a single string argument and converts it into a quoted command line argument.

I was going to add a long rant here about Vim's 'shellslash' option, but really, it won't make any difference. Let's just suffice to say that I have yet to encounter a single person out there who uses this option for its intended purpose (running a UNIX style shell on Microsoft Windows).

Human friendly string formatting for Vim

The `xolox#misc#format#pluralize()` function

Concatenate a counter (the first argument, expected to be an integer) with a singular or plural label (the second and third arguments, both expected to be strings).

The `xolox#misc#format#timestamp()` function

Format a time stamp (a string containing a formatted floating point number) into a human friendly format, for example 70 seconds is phrased as “1 minute and 10 seconds”.

List handling functions

The `xolox#misc#list#unique()` function

Remove duplicate values from the given list in-place (preserves order).

The `xolox#misc#list#binsert()` function

Performs in-place binary insertion, which depending on your use case can be more efficient than calling Vim’s sort() function after each insertion (in cases where a single, final sort is not an option). Expects three arguments:

1. A list
2. A value to insert
3. 1 (true) when case should be ignored, 0 (false) otherwise

Functions to interact with the user

The `xolox#misc#msg#info()` function

Show a formatted informational message to the user.

This function has the same argument handling as Vim’s printf() function with one notable difference: Any arguments which are not numbers or strings are coerced to strings using Vim’s string() function.

In the case of `xolox#misc#msg#info()`, automatic string coercion simply makes the function a bit easier to use.

The messages emitted by this function have no highlighting. Previously these messages were highlighted using the Title group, but it was pointed out in pull request 16 that this group shouldn’t be used for informational messages because it is meant for titles and because of this some color schemes use colors that stand out quite a bit, causing the informational messages to look like errors.

The `xolox#misc#msg#warn()` function

Show a formatted warning message to the user.

This function has the same argument handling as the `xolox#misc#msg#info()` function.

The `xolox#misc#msg#debug()` function

Show a formatted debugging message to the user, *if the user has enabled increased verbosity by setting Vim's 'verbose' option to one (1) or higher*.

This function has the same argument handling as the `xolox#misc#msg#info()` function.

In the case of `xolox#misc#msg#debug()`, automatic string coercion provides lazy evaluation in the sense that complex data structures are only converted to strings when the user has enabled increased verbosity.

Integration between Vim and its environment

The `xolox#misc#open#file()` function

Given a pathname or URL as the first argument, this opens the file with the program associated with the file type. So for example a text file might open in Vim, an `*.html` file would probably open in your web browser and a media file would open in a media player.

This should work on Windows, Mac OS X and most Linux distributions. If this fails to find a file association, you can pass one or more external commands to try as additional arguments. For example:

```
:call xolox#misc#open#file('/path/to/my/file', 'firefox', 'google-chrome')
```

This generally shouldn't be necessary but it might come in handy now and then.

The `xolox#misc#open#url()` function

Given a URL as the first argument, this opens the URL in your preferred or best available web browser:

- In GUI environments a graphical web browser will open (or a new tab will be created in an existing window)
- In console Vim without a GUI environment, when you have any of `lynx`, `links` or `w3m` installed it will launch a command line web browser in front of Vim (temporarily

suspending Vim)

Vim and plug-in option handling

The xolox#misc#option#get() function

Expects one or two arguments: 1. The name of a variable and 2. the default value if the variable does not exist.

Returns the value of the variable from a buffer local variable, global variable or the default value, depending on which is defined.

This is used by some of my Vim plug-ins for option handling, so that users can customize options for specific buffers.

The xolox#misc#option#split() function

Given a multi-value Vim option like 'runtimepath' this returns a list of strings. For example:

```
:echo xolox#misc#option#split(&runtimepath)
['/home/peter/Projects/Vim/misc',
 '/home/peter/Projects/Vim/colorscheme-switcher',
 '/home/peter/Projects/Vim/easytags',
 ...]
```

The xolox#misc#option#join() function

Given a list of strings like the ones returned by `xolox#misc#option#split()`, this joins the strings together into a single value that can be used to set a Vim option.

The xolox#misc#option#split_tags() function

Customized version of `xolox#misc#option#split()` with specialized handling for Vim's 'tags' option.

The xolox#misc#option#join_tags() function

Customized version of `xolox#misc#option#join()` with specialized handling for Vim's 'tags' option.

The xolox#misc#option#eval_tags() function

Evaluate Vim's 'tags' option without looking at the file system, i.e. this will report tags files that don't exist yet. Expects the value of the 'tags' option as the first argument. If the optional second argument is 1 (true) only the first match is returned, otherwise (so by default) a list with all matches is returned.

Operating system interfaces

The `xolox#misc#os#is_mac()` function

Returns 1 (true) when on Mac OS X, 0 (false) otherwise. You would expect this to simply check the Vim feature list, but for some obscure reason the `/usr/bin/vim` included in Mac OS X (verified on version 10.7.5) returns 0 (false) in response to `has('mac')`, so we check the output of `uname` to avoid false negatives.

The `xolox#misc#os#is_win()` function

Returns 1 (true) when on Microsoft Windows, 0 (false) otherwise.

The `xolox#misc#os#find_vim()` function

Returns the program name of Vim as a string. On Windows and UNIX this just v:progname as an absolute pathname while on Mac OS X there is some special magic to find MacVim's executable even though it's usually not on the executable search path. If you want, you can override the value returned from this function by setting the global variable `g:xolox#misc#os#vim_progname`.

By default the choice of console Vim vs graphical Vim is made based on the value of v:progname, but if you have a preference you can pass the string `vim` or `gvim` as the first and only argument.

The `xolox#misc#os#exec()` function

Execute an external command (hiding the console on Microsoft Windows when my vim-shell plug-in is installed).

Expects a dictionary with the following key/value pairs as the first argument:

- **command** (required): The command line to execute

- **async** (optional): set this to 1 (true) to execute the command in the background (asynchronously)
- **stdin** (optional): a string or list of strings with the input for the external command
- **check** (optional): set this to 0 (false) to disable checking of the exit code of the external command (by default an exception will be raised when the command fails)

Returns a dictionary with one or more of the following key/value pairs:

- **command** (always available): the generated command line that was used to run the external command
- **exit_code** (only in synchronous mode): the exit status of the external command (an integer, zero on success)
- **stdout** (only in synchronous mode): the output of the command on the standard output stream (a list of strings, one for each line)
- **stderr** (only in synchronous mode): the output of the command on the standard error stream (as a list of strings, one for each line)

The `xolox#misc#os#can_use_dll()` function

If a) we're on Microsoft Windows, b) the vim-shell plug-in is installed and c) the compiled DLL included in vim-shell works, we can use the vim-shell plug-in to execute external commands! Returns 1 (true) if we can use the DLL, 0 (false) otherwise.

Pathname manipulation functions

The `xolox#misc#path#which()` function

Scan the executable search path (`$PATH`) for one or more external programs. Expects one or more string arguments with program names. Returns a list with the absolute pathnames of all found programs. Here's an example:

```
:echo xolox#misc#path#which('gvim', 'vim')
['/usr/local/bin/gvim',
 '/usr/bin/gvim',
 '/usr/local/bin/vim',
 '/usr/bin/vim']
```

The `xolox#misc#path#split()` function

Split a pathname (the first and only argument) into a list of pathname components.

On Windows, pathnames starting with two slashes or backslashes are UNC paths where the leading slashes are significant… In this case we split like this:

- Input: `'//server/share/directory'`
- Result: `['//server', 'share', 'directory']`

Everything except Windows is treated like UNIX until someone has a better suggestion :-). In this case we split like this:

- Input: `'/foo/bar/baz'`
- Result: `['/', 'foo', 'bar', 'baz']`

To join a list of pathname components back into a single pathname string, use the `xolox#misc#path#join()` function.

The `xolox#misc#path#join()` function

Join a list of pathname components (the first and only argument) into a single pathname string. This is the counterpart to the `xolox#misc#path#split()` function and it expects a list of pathname components as returned by `xolox#misc#path#split()`.

The `xolox#misc#path#directory_separator()` function

Find the preferred directory separator for the platform and settings.

The `xolox#misc#path#absolute()` function

Canonicalize and resolve a pathname, *regardless of whether it exists*. This is intended to support string comparison to determine whether two pathnames point to the same directory or file.

The `xolox#misc#path#relative()` function

Make an absolute pathname (the first argument) relative to a directory (the second argument).

The `xolox#misc#path#merge()` function

Join a directory pathname and filename into a single pathname.

The `xolox#misc#path#commonprefix()` function

Find the common prefix of path components in a list of pathnames.

The `xolox#misc#path#starts_with()` function

Check whether the first pathname starts with the second pathname (expected to be a directory). This does not perform a regular string comparison; first it normalizes both pathnames, then it splits them into their pathname segments and then it compares the segments.

The `xolox#misc#path#encode()` function

Encode a pathname so it can be used as a filename. This uses URL encoding to encode special characters.

The `xolox#misc#path#decode()` function

Decode a pathname previously encoded with `xolox#misc#path#encode()`.

The `xolox#misc#path#is_relative()` function

Returns true (1) when the pathname given as the first argument is relative, false (0) otherwise.

The `xolox#misc#path#tempdir()` function

Create a temporary directory and return the pathname of the directory.

Manipulation of UNIX file permissions

Vim's writefile() function cannot set file permissions for newly created files and although Vim script has a function to get file permissions (see getfperm()) there is no equivalent for changing a file's permissions.

This omission breaks the otherwise very useful idiom of updating a file by writing its new contents to a temporary file and then renaming the temporary file into place (which is as close as you're going to get to atomically updating a file's contents on UNIX) because the file's permissions will not be preserved!

Here's a practical example: My vim-easytags plug-in writes tags file updates to a temporary file and renames the temporary file into place. When I use `sudo -s` on Ubuntu Linux it

preserves my environment variables so my `~/.vimrc` and the vim-easytags plug-in are still loaded. Now when a tags file is written the file becomes owned by root (my effective user id in the `sudo` session). Once I leave the `sudo` session I can no longer update my tags file because it's now owned by root … ಠ_ಠ

The `xolox#misc#perm#update()` function

Atomically update a file's contents while preserving the owner, group and mode. The first argument is the pathname of the file to update (a string). The second argument is the list of lines to be written to the file. Writes the new contents to a temporary file and renames the temporary file into place, thereby preventing readers from reading a partially written file. Returns 1 if the file is successfully updated, 0 otherwise.

Note that if `xolox#misc#perm#get()` and `xolox#misc#perm#set()` cannot be used to preserve the file owner/group/mode the file is still updated using a rename (for compatibility with non-UNIX systems and incompatible `/usr/bin/stat` implementations) so in that case you can still lose the file's owner/group/mode.

The `xolox#misc#perm#get()` function

Get the owner, group and permissions of the pathname given as the first argument. Returns an opaque value which you can later pass to `xolox#misc#perm#set()`.

The `xolox#misc#perm#set()` function

Set the permissions (the second argument) of the pathname given as the first argument. Expects a permissions value created by `xolox#misc#perm#get()`.

Persist/recall Vim values from/to files

Vim's string() function can be used to serialize Vim script values like numbers, strings, lists, dictionaries and composites of them to a string which can later be evaluated using the eval() function to turn it back into the original value. This Vim script provides functions to use these functions to persist and recall Vim values from/to files. This is very useful for communication between (possibly concurrent) Vim processes.

The `xolox#misc#persist#load()` function

Read a Vim value like a number, string, list or dictionary from a file using readfile() and eval(). The first argument is the filename of the file to read (a string). The optional second argument specifies the default value which is returned when the file can't be loaded. This function returns the loaded value or the default value (which itself defaults to the integer 0).

The `xolox#misc#persist#save()` function

Write a Vim value like a number, string, list or dictionary to a file using string() and writefile(). The first argument is the filename of the file to write (a string) and the second argument is the value to write (any value).

This function writes the serialized value to an intermediate file which is then renamed into place atomically. This avoids issues with concurrent processes where for example a producer has written a partial file which is read by a consumer before the file is complete. In this case the consumer would read a corrupt value. The rename trick avoids this problem.

String handling

The `xolox#misc#str#slug()` function

Convert a string to a "slug" - something that can be safely used in filenames and URLs without worrying about quoting/escaping of special characters.

The `xolox#misc#str#ucfirst()` function

Uppercase the first character in a string (the first argument).

The `xolox#misc#str#unescape()` function

Remove back slash escapes from a string (the first argument).

The `xolox#misc#str#compact()` function

Compact whitespace in a string (the first argument).

The `xolox#misc#str#trim()` function

Trim all whitespace from the start and end of a string (the first argument).

The `xolox#misc#str#indent()` function

Indent all lines in a multi-line string (the first argument) with a specific number of *space characters* (the second argument, an integer).

The `xolox#misc#str#dedent()` function

Remove common whitespace from a multi line string.

Test runner & infrastructure for Vim plug-ins

The Vim auto-load script `autoload/xolox/misc/test.vim` contains infrastructure that can be used to run an automated Vim plug-in test suite. It provides a framework for running test functions, keeping track of the test status, making assertions and reporting test results to the user.

The `xolox#misc#test#reset()` function

Reset counters for executed tests and passed/failed assertions.

The `xolox#misc#test#summarize()` function

Print a summary of test results, to be interpreted interactively.

The `xolox#misc#test#wrap()` function

Call a function in a try/catch block and prevent exceptions from bubbling. The name of the function should be passed as the first and only argument; it should be a string containing the name of a Vim auto-load function.

The `xolox#misc#test#passed()` function

Record a test which succeeded.

The `xolox#misc#test#failed()` function

Record a test which failed.

The `xolox#misc#test#assert_true()` function

Check whether an expression is true.

The `xolox#misc#test#assert_equals()` function

Check whether two values are the same.

The `xolox#misc#test#assert_same_type()` function

Check whether two values are of the same type.

Tests for the miscellaneous Vim scripts

The Vim auto-load script `autoload/xolox/misc/tests.vim` contains the automated test suite of the miscellaneous Vim scripts. Right now the coverage is not very high yet, but this will improve over time.

The `xolox#misc#tests#run()` function

Run the automated test suite of the miscellaneous Vim scripts. To be used interactively. Intended to be safe to execute irrespective of context.

The `xolox#misc#tests#pattern_escaping()` function

Test escaping of regular expression patterns with `xolox#misc#escape#pattern()`.

The `xolox#misc#tests#substitute_escaping()` function

Test escaping of substitution strings with `xolox#misc#escape#substitute()`.

The `xolox#misc#tests#shell_escaping()` function

Test escaping of shell arguments with `xolox#misc#escape#shell()`.

The `xolox#misc#tests#making_a_list_unique()` function

Test removing of duplicate values from lists with `xolox#misc#list#unique()`.

The `xolox#misc#tests#binary_insertion()` function

Test the binary insertion algorithm implemented in `xolox#misc#list#binsert()`.

The `xolox#misc#tests#getting_configuration_options()` function

Test getting of scoped plug-in configuration “options” with `xolox#misc#option#get()`.

The xolox#misc#tests#splitting_of_multi_valued_options() function

Test splitting of multi-valued Vim options with `xolox#misc#option#split()`.

The xolox#misc#tests#joining_of_multi_valued_options() function

Test joining of multi-valued Vim options with `xolox#misc#option#join()`.

The xolox#misc#tests#finding_vim_on_the_search_path() function

Test looking up Vim's executable on the search path using v:progname with `xolox#misc#os#find_vim()`.

The xolox#misc#tests#synchronous_command_execution() function

Test basic functionality of synchronous command execution with `xolox#misc#os#exec()`.

The xolox#misc#tests#synchronous_command_execution_with_stderr() function

Test basic functionality of synchronous command execution with `xolox#misc#os#exec()` including the standard error stream (not available on Windows when vim-shell is not installed).

The xolox#misc#tests#synchronous_command_execution_with_raising_of_errors() function

Test raising of errors during synchronous command execution with `xolox#misc#os#exec()`.

The xolox#misc#tests#synchronous_command_execution_without_raising_errors() function

Test synchronous command execution without raising of errors with `xolox#misc#os#exec()`.

The xolox#misc#tests#asynchronous_command_execution() function

Test the basic functionality of asynchronous command execution with `xolox#misc#os#exec()`. This runs the external command `mkdir` and tests that the side effect of creating the directory takes place. This might seem like a peculiar choice, but it's one

of the few 100% portable commands (Windows + UNIX) that doesn't involve input/output streams.

The `xolox#misc#tests#string_case_transformation()` function

Test string case transformation with `xolox#misc#str#ucfirst()`.

The `xolox#misc#tests#string_whitespace_compaction()` function

Test compaction of whitespace in strings with `xolox#misc#str#compact()`.

The `xolox#misc#tests#string_whitespace_trimming()` function

Test trimming of whitespace in strings with `xolox#misc#str#trim()`.

The `xolox#misc#tests#multiline_string_dedent()` function

Test dedenting of multi-line strings with `xolox#misc#str#dedent()`.

The `xolox#misc#tests#version_string_parsing()` function

Test parsing of version strings with `xolox#misc#version#parse()`.

The `xolox#misc#tests#version_string_comparison()` function

Test comparison of version strings with `xolox#misc#version#at_least()`.

Timing of long during operations

The `xolox#misc#timer#resumable()` function

Create a resumable timer object. This returns an object (a dictionary with functions) with the following "methods":

- `start()` instructs the timer object to start counting elapsed time (when a timer object is created it is not automatically started).
- `stop()` instructs the timer object to stop counting elapsed time. This adds the time elapsed since `start()` was last called to the total elapsed time. This method will raise an error if called out of sequence.

- `format()` takes the total elapsed time and reports it as a string containing a formatted floating point number.

Timer objects are meant to accurately time short running operations so they're dependent on Vim's reltime() and reltimestr() functions. In order to make it possible to use timer objects in my Vim plug-ins unconditionally there's a fall back to localtime() when reltime() is not available. In this mode the timer objects are not very useful but at least they shouldn't raise errors.

The `xolox#misc#timer#start()` function

Start a timer. This returns a list which can later be passed to `xolox#misc#timer#stop()`.

The `xolox#misc#timer#stop()` function

Show a formatted debugging message to the user, if the user has enabled increased verbosity by setting Vim's 'verbose' option to one (1) or higher.

This function has the same argument handling as Vim's printf() function with one difference: At the point where you want the elapsed time to be embedded, you write `%s` and you pass the list returned by `xolox#misc#timer#start()` as an argument.

The `xolox#misc#timer#force()` function

Show a formatted message to the user. This function has the same argument handling as Vim's printf() function with one difference: At the point where you want the elapsed time to be embedded, you write `%s` and you pass the list returned by `xolox#misc#timer#start()` as an argument.

The `xolox#misc#timer#convert()` function

Convert the value returned by `xolox#misc#timer#start()` to a string representation of the elapsed time since `xolox#misc#timer#start()` was called. Other values are returned unmodified (this allows using it with Vim's map() function).

Version string handling

The `xolox#misc#version#parse()` function

Convert a version string to a list of integers.

The `xolox#misc#version#at_least()` function

Check whether the second version string is equal to or greater than the first version string. Returns 1 (true) when it is, 0 (false) otherwise.

Contact

If you have questions, bug reports, suggestions, etc. please open an issue or pull request on GitHub. Download links and documentation can be found on the plug-in's homepage. If you like the script please vote for it on Vim Online.

License

Last updated 125 months ago.