Fs 1.0

September 9, 2024

Contents

Fs

The `Fs` module is used to interact with the filesystem of the local machine.

::: warning Requires Manatee v1.29 or greater

This version of the `Fs` module cannot be used with Manatee v1.28 or earlier.

:::

System folders

Provides access to the following system folders:

- `tmpfolder`: A directory for temporarily storing files
- `desktop`: The user's windows desktop
- `appdata`: The user app data folder. Applications can write user specific data here without requiring administrator privilege
- `startup`: The folder which contains shortcuts to applications that should start when the user logs in
- `personal`: The root user folder - eg `C:\Users\<user name>`

Example

```
var folder = Fs.tmpfolder;
```

List (ls)

Returns a list of files and directories found in the directory given by the `path` argument. The path may contain wildcards `*` in its last segment.

A second option argument can be passed with the following properties:

- `deepMatch` boolean indicating if the listing should include contents of subdirectories. Defaults to false. When this property is set to true, files matching the filename given in the `path` argument in any sub-folder will be returned.
- `includeDirectories` boolean indicating if the listing should include directories. Defaults to false. So by default only files are included.

Default behavior is to do a shallow file listing of only the files in the given folder.

::: tip Weird behaviour with 3-letter extensions

When you use the asterisk wildcard character in a searchPattern such as `*.txt`, the number of characters in the specified extension affects the search as follows:

If the specified extension is exactly three characters long, the method returns files with extensions that begin with the specified extension. For example, `*.xls` returns both “book.xls” and “book.xlsx”. In all other cases, the method returns files that exactly match the specified extension. For example, `*.ai` returns ” file.ai” but not “file.aif”. When you use the question mark wildcard character, this method returns only files that match the specified file extension. For example, given two files, “file1.txt” and “file1.txtother”, in a directory, a search pattern of `file?.txt` returns just the first file, whereas a search pattern of `file*.txt` returns both files.

:::

Because this method checks against file names with both the 8.3 file name format and the long file name format, a search pattern similar to `*1*.txt` may return unexpected file names. For example, using a search pattern of `*1*.txt` returns “longfilename.txt” because the equivalent 8.3 file name format is “LONGFI~1.TXT”.

Return value

The resulting array can be used as a string array of the paths to the files. It can also be used as an array of objects with detailed information about the files. Each such object has the following properties:

- `folder` is the folder part of the path. `C:\folder\file.txt` has the folder path `C:\folder`.
- `path` is the full path of the item. Corresponds to the string value of the object.
- `extension` is the extension of the item. `C:\folder\file.txt` has the extension `.txt`.
- `name` is the name of the item. `C:\folder\file.txt` has the name `file.txt`. `C:\folder` has the name `folder`.
- `readonly` boolean value indicating if the file is read only.
- `size` is the size of the file in bytes.
- `created` is the time of creation.
- `modified` is the time of the last modification.
- `accessed` is the time of the last file access.

The objects further have the following methods:

- `mv` moves the file or directory. Pass the new path as an argument.
- `cp` copies the file or directory. Pass the new path as an argument.
- `rm` deletes the file.
- `encrypt` encrypts the file.
- `decrypt` decrypts the file.

Example

```
// Get all .txt files prefixed with somefile in somedir
var files = Fs.ls('c:\\somedir\\somefile*.txt');

// Get all .txt files in any sub directory under C:\somedir - at any
   depth
var files = Fs.ls('c:\\somedir\\*.txt', {deepMatch: true});

// Copy readonly files to a backup sub directory
var readonlyFiles = files.filter(function (file) {
    return file.readonly;
});
_.each(readonlyFiles, function (file) {
    file.cp(file.folder + '\\backup\\' + file.name)
});
```

Make a new directory

Create a new directory if it does not already exist.

```
Fs.mkdir("C:\\some\\path");
```

Move file/folder

Move a file or folder to a different path.

```
Fs.mv('C:\\some\\path\\file.txt', 'C:\\some\\other\\path\\file.txt');
// or a folder
Fs.mv('C:\\some\\path', 'C:\\some\\other\\path')
```

If you want to allow the target file to be overwritten (if it exists):

```
Fs.mv('C:\\some\\path\\file.txt', 'C:\\some\\other\\path\\file.txt', {
    overwrite: true});
```

Copy file/folder

Copy a file or folder to a different path

```
Fs.cp('C:\\some\\path\\file.txt', 'C:\\some\\other\\path\\file.txt');
// and to allow overwrite of target file
Fs.cp('C:\\some\\path\\file.txt', 'C:\\some\\other\\path\\file.txt', {
    overwrite: true});
```

Delete file/folder

Delete a file or folder

```
Fs.rm('C:\\some\\path\\file.txt');
```

Check file/folder presence

Determines if a file exists at a given path

Example

```
if (!Fs.exists('C:\\some\\path\\file.txt')) {
    // Create the file
}
```

Build a path

Convenience for building a valid file system path to a file or a directory.

Example

```
var p = Fs.buildPath('C:\\root', 'foo', 'bar.txt')
// p represents the path C:\root\foo\bar.txt
```

Encrypt file

Activates windows file encryption for the file at the given path. Only the currently logged in user will be able to read the file.

Example

```
Fs.encrypt('C:\\some\\path\\file.txt');
```

Decrypt file

Deactivates windows file encryption for the file at the given path. Any user will be able to read the file.

Example

```
Fs.decrypt('C:\\some\\path\\file.txt');
```

Read

Read the contents of a file with the `read` function.

Example

```
var html = Fs.read('c:\\somedir\\somefile.html');
```

Both `Fs.read` and `Fs.write` methods can take an `encoding` option, like:

```
Fs.write("C:/somewhere/test.txt", "String to write", {encoding: "UTF-16
    "});
// or for short
Fs.write("C:/somewhere/test.txt", "String to write", {encoding: Fs.
    UTF16});
// and
Fs.read("C:/somewhere/test.txt", {encoding: "UTF-16"});
// default if no `encoding` arg is given is UTF-8 no bom
```

The list of encoding (names) which can be used is found at https://www.iana.org/assignments/character-sets/character-sets.xhtml. *Note* that not all of these may be available on your machine, to see those, run:

```
Debug.ger(Fs.encodings);
```

The following are encodings are defined on `Fs`;

- `Fs.UTF8`
- `Fs.UTF16`
- `Fs.ASCII`

If you think your file is ANSI or ASCII encoded, but none if these seem to work then you might be looking for the `ISO-8859-1` encoding which sometimes does the trick.

Write

Writes arbitrary text to an arbitrary text file. If the file exists, it will be overwritten. If the file doesn't exist, it will be created with the given contents. The contents are written using UTF-8 encoding without a byte order mark (BOM).

Throws appropriate exceptions if the write fails.

Parameters

- `path` the file system path to write to
- `data` a string with the data to write
- `options` an optional options object. Supported options are;
  - `base64` a boolean. If true, interprets the data argument as a base64 string and writes the data to disk as binary data. Defaults to false

- `writeBom` a boolean. If true, a utf-8 byte-order-mark sequence is prepended to the file. This helps other applications detect the encoding of the file. Defaults to false. Is ignored if the `base64` option is true.
- `encoding` The encoding with with to write the file (default is `"UTF-8"`).
- `append` a boolean. If true, text is appended to the file in the path in question. Is ignored if the `base64` option is true.

Example

```
Fs.write('c:\\somedir\\somefile.html', '<html><body><h1>Generated html
    !</h1></body></html>');
```

Synchronise two directories

If you need two synchronise the files in two directories, i.e. make sure all files in the *source* directory are copied to the *destination* directory you can use the `Fs.sync(...)` method.

Examples

```
// Make sure the two directories are completely synchronised, delete
    superfluous files from destination
Fs.sync("C:\\MySourceDirectory", "C:\\MyDestinationDirectory");
// uhe same but don't delete those files in the destination directory
    which are not present in the source
Fs.sync("C:\\MySourceDirectory", "C:\\MyDestinationDirectory", {
    deleteSuperfluous: false});
```

Temp file

The `tmpfile` function will generate a random, non-conflicting filename in the temp folder.

Example

```
var tmpFilePath = Fs.tmpfile();
```