File System

```
#import "fs"
```

File system module for manipulation with files and directories.

> ***Warning***
>
> This module is experimental and not fully supported across all platforms.

FSFile

Description

File handle type.

FSFileOpenMode

Declaration

```
FSFileOpenMode :: enum {
    Read;
    Write;
};
```

Description

Specify operation with opened file.

Variants

- *Read* Open file for reading.
- *Write* Open file for writing.

---

fs_file_open

Declaration

```
fs_file_open :: fn (filepath: string, mode: ...FSFileOpenMode) (FSFile, Error)
```

Description

Open an existing file specified by *filepath*. Function return file handle and *OK* status code when file was openned, otherwise return *null* and proper error code. File must be closed by :ref:`fs_close` call.

Arguments

- *filepath* File path.
- *mode* Open mode. :ref:`FSFileOpenMode` When no mode is specified, *Read* and *Write* is used.

Result

File handle and status code :ref:`Error`.

Example

```
#import "fs"

main :: fn () s32 {
    file, err :: fs_file_open(#file);
    defer fs_file_close(file);
    if err != OK {
        print_err("Cannot open file!");
        return 1;
    }
    return 0;
}
```

---

fs_file_create

Declaration

```
fs_file_create :: fn (filepath: string, mode: ...FSFileOpenMode) (FSFile, Error)
```

Description

Create new file specified by *filepath*. Return file *handle* and *OK* status code when file was created, otherwise only status code is returned. File must be closed by :ref:`fs_close` call.

Arguments

- *filepath* File path.
- *mode* Open mode. :ref:`FSFileOpenMode` When no mode is specified, *Read* and *Write* is used.

Result

File handle and status code :ref:`Error`.

---

fs_file_close

Declaration

```
fs_file_close :: fn (handle: FSFile) #inline
```

Description

Close opened file.

Arguments

- *handle* File handle.

---

fs_file_delete

Declaration

```
fs_file_delete :: fn (filepath: string) bool #inline
```

Description

Delete file specified by *filepath*.

Arguments

- *filepath* File path.

Result

True when file was deleted, otherwise return false. When *filepath* is invalid or empty string function also return *false* and doesn't produce any file system operation.

---

fs_file_read

Declaration

```
fs_file_read :: fn (handle: FSFile, dest: *u8, size: s64) (s64, Error)
```

Description

Load file content into the *dest* buffer with maximum *size* specified. Fails with *ERR_INVALID_HANDLE* when *dest* is *null*.

Arguments

- *handle* File handle.
- *dest* Destination buffer.
- *size* Maximum size to read.

Result

Count of bytes filled in destination buffer when status is :ref:`OK`.

---

fs_file_read_string

Declaration

```
fs_file_read_string :: fn (handle: FSFile) (string, Error)
```

Description

Load file content into the string.

Arguments

- *handle* File handle.

Result

Return new *string* instance when status is :ref:`OK`. String must be released by :ref:`string_delete` call only in case there is no error reported by function.

Example

```
#import "fs"

main :: fn () s32 {
    // Open this file.
    file, open_err :: fs_file_open(#file, FSFileOpenMode.Read);

    // Always check for errors.
    if open_err != OK {
        panic("Cannot open file with error: '%'!", open_err);
    }
    // Close file at the end of scope.
    defer fs_file_close(file);

    // Read it's content.
    content, read_err :: fs_file_read_string(file);

    // Check for errors.
    if read_err != OK {
        panic("Cannot read file with error: '%'!", read_err);
    }
    // Delete content string at the end of scope.
    defer string_delete(content);

    // Print file content to stdout.
    print("%\n", content);
    return 0;
}
```

fs_file_read_slice

Declaration

```
fs_file_read_slice :: fn (handle: FSFile) ([]u8, Error)
```

Description

Load file content into the slice.

Arguments

- *handle* File handle.

Result

Content of the file and status :ref:`Error`. Returned slice must be released by :ref:`slice_terminate` call in case there is no error reported. When error occured returned slice is zero initialized and should not be released.

---

fs_file_size

Declaration

```
fs_file_size :: fn (handle: FSFile) (usize, Error) #inline
```

Description

Return size of opened file in bytes.

Arguments

- *handle* File handle.

Result

Content size of the file and status :ref:`Error`.

---

fs_file_write

Declaration

```
fs_file_write :: fn (handle: FSFile, src: *u8, size: s64) (s64, Error)
```

Description

Write *size* bytes of *src* buffer content into the file specified by *handle*.

Arguments

- *handle* Valid file handle open for writing.
- *src* Pointer to source buffer.
- *size* Size of bytes to be written from the buffer (maximum is buffer size).

Result

Number of successfuly written bytes when there is no error.

---

fs_file_write_string

Declaration

```
fs_file_write_string :: fn (handle: FSFile, str: string) (s64, Error)
```

Description

Write content of *str* string into file specified by *handle*.

Arguments

- *handle* Valid file handle open for writing.
- *str* String to be written.

Result

Number of successfuly written bytes when there is no error.

---

fs_file_write_slice

Declaration

```
fs_file_write_slice :: fn (handle: FSFile, v: []u8) (s64, Error)
```

Description

Write content of *v* slice into file specified by *handle*.

Arguments

- *handle* Valid file handle open for writing.
- *str* String to be written.

Result

Number of successfuly written bytes when there is no error.

---

fs_exist

Declaration

```
fs_exist :: fn (filepath: string) bool
```

Description

Check whether file or directory exists.

Arguments

- *filepath* File path.

Result

True when file of directory exists.

---

fs_validate_filename

Declaration

```
fs_validate_filename :: fn (name: string) bool
```

Description

Validate file name.

Arguments

- *name* File name (not path).

Result

Return *true* if name is valid file name on target platform.

---

fs_home

Declaration

```
fs_home :: fn () string #inline
```

Description

Get path to *home* directory. Use :ref:`string_delete` to delete result string.

Result

Path to *home* directory or empty string.

---

fs_cwd

Declaration

```
fs_cwd :: fn () string #inline
```

Description

Get current working directory. Use :ref:`string_delete` to delete result string.

Result

Path to current working directory or empty string.

---

fs_tmp

Declaration

```
fs_home :: fn () string #inline
```

Description

Get path to *temp* directory. Use :ref:`string_delete` to delete result string.

Result

Path to *temp* directory or empty string.

---

fs_normalize

Declaration

```
fs_normalize :: fn (filepath: *string) bool
```

Description

Normalize path in *filepath* and check if result path exist; also resolve references *.* and *..*

Result

Return *true* and set *filepath* when path was normalized and points to existing entry.

---

fs_remove_extension

Declaration

```
fs_remove_extension :: fn (filename: string) string #inline
```

Description

Remove file extension (first after dot separator) from file name. In case dot separator is first character in the string we expect it's hidden file.

Arguments

- *filename* File name.

Result

File name without extension (not including dot separator) or empty string.

---

fs_get_extension

Declaration

```
fs_get_extension :: fn (filename: string) string #inline
```

Description

Get file extension from file name. This function just split input *filename* by first occourence of dot character if it's not first one.

Arguments

- *filename* File name.

Result

File extension not including dot separator. In case no extension was found, function return empty string. Returned string is not copy and should not be deleted.