

# 1

## JavaScript File and Folder Classes

### 1.1 JavaScript File and Folder classes

To create a `File` or `Folder` object, use the corresponding `File()` and `Folder()` functions. You can also create them with the `new` operator if you like; both ways of calling `File()` or `Folder()` return a new object. The constructor accepts full or partial path names. In any case, the path stored internally is an absolute, full path name, so a `File` or `Folder` object, once created, always points to a fixed location of the disk. If you do not supply a file or folder name, a temporary name is generated.

The `Folder` object supports file system functionality like walking directories, creating, renaming or removing files, or resolving file aliases. The `File` supports I/O functions to read or write files. Programmers implementing the object can choose to limit the functionality of the object by setting a combination of security flags.

`File` and `Folder` objects can be used at any place where a path name is required; its conversion to a string (the `toString()` method) returns the name as supplied to the constructor. If you need the operating system specific file name, use the `fsName` property.

When you create two `File` objects that refer to the same disk file, they are treated as distinct objects. If you open one of the `File` objects for I/O, the operating system may inhibit access to the opened `File` object from the other `File` object.

There are several methods to distinguish between a `File` and a `Folder` object. Here are some examples:

```
if (f instanceof File) ...
if (typeof f.open == "undefined") ...
```

## 1.2 Common elements for file and folder objects

Both the `File` and the `Folder` objects share a common set of properties and methods. All properties and methods do not resolve file system aliases unless indicated otherwise.

### Class properties

`fs` *String* The name of the file system. Read-only.

### Class methods

`decode (String what);`

The method `File.decode()` or `Folder.decode()` decodes its input string as required by RFC 2396. All characters with a numeric value greater than 127 are encoded in UTF-8; they are stored as escaped characters starting with the percent sign followed by two hex digits. Also the characters `/ - _ . ! ~ * ' ( )` are encoded the same way.

### Returns: *String*

`encode (String what);`

The method `File.encode()`

or `Folder.encode()` encodes its input string as required by RFC 2396. All characters with a numeric value greater than 127 are encoded in UTF-8; they are returned as escaped characters starting with the percent sign followed by two hex digits. Also the characters `/ - _ . ! ~ * ' ( )` are encoded the same way.

### Returns: *String*

### 1.2.1 Object properties

Property	Value	Description
<code>absoluteURI</code>	<i>String</i>	The full path name for the object in URI notation. Read-only.
<code>alias</code>	<i>Boolean</i>	Returns <i>true</i> if the object refers to a file system alias. Read-only.
<code>created</code>	<i>Date</i>	If the object does not refer to a folder or file on disk, the value is <i>null</i> . Read-only.
<code>error</code>	<i>String</i>	Contains a message describing the last file system error. Setting this value clears any error message and resets the error bit for opened files.

Property	Value	Description
exists	Boolean	Returns <i>true</i> if the path name of this object refers to an actually existing file or folder. Read only.
fsName	String	The file-system specific name of the object as a full path name. Read-only.
modified	Date	The date of the object's last modification. If the object does not refer to a folder or file on disk, the value is <i>null</i> . Read-only.
name	String	The name of the object without the path specification. Read-only.
parent	Folder	The folder object containing this object. If this object already is the root folder of a volume, the property value is <i>null</i> . Read-only.
path	String	The path portion of the absolute URI. If the name does not have a path, this property contains the empty string. Read-only.
relativeURI	String	The path name for the object in URI notation, relative to the current folder. Read-only.

### 1.2.2 Object methods

`getRelativeURI (String basePath);`

Calculate and return the relative URI, given a base path, in URI notation. If the base path is omitted, the path of the current folder is assumed.

**Returns:** *String*

`remove();`

Delete the file or folder that this object represents. Folders must be empty before they can be deleted. The return value is `true` if the file or folder has been deleted.

**NOTE:** The `remove()` method deletes the referenced file or folder immediately. It does not move the referenced file or folder to the system trash. The effects of the `remove` method cannot be undone. It is recommended that you prompt the user for permission to delete a file or folder before deleting it. The method does not resolve aliases; it rather deletes the file alias itself.

**Returns:** *Boolean*

`rename (String newName);`

Rename the object to the new name. The new name must not have a path. Returns `true` if the object was renamed. The method does not resolve aliases, but rather renames the alias file.

**Returns:** *Boolean*

`resolve();`

Attempt to resolve the file system alias that this object points to. If successful, a new `File` or `Folder` object is returned that points to the resolved file system element. If the object is not an alias, or if the alias could not be resolved, the return value is `null`.

**Returns:** *File, Folder or null.*

### 1.2.3 Error handling

Each object has an `error` property. If accessing a property or calling a method caused an error, this property contains a message describing the type of the error. On success, the property contains the empty string. The property can be set, but setting it only causes the error message to be cleared. If a file is open, assigning an arbitrary value to the property also resets its error flag.

### 1.2.4 Error texts

The following texts may be returned in the error property.

Text	Explanation
Not allowed	Operation not permitted
Cannot resolve	Alias cannot be resolved
cannot set dire	Cannot change the current folder
Not a directory	Attempt to collect a file list on a file instead of a folder
Cannot create	Cannot create a folder
Cannot rename	Cannot rename a file or folder
Cannot delete	Cannot delete a file or folder
Cannot open	Opening of a file failed
File is open	Cannot set the size of a file while it is open
Cannot set size	Setting the file size failed
Invalid open mode	Open mode was neither read, write, or edit
Cannot close	Closing a file failed
File is not open	Attempt to read from or write to a closed file
Read error	Reading from a file failed
Write error	Writing to a file failed
Bad encoding	Unable to read encoded characters from a file
EOF	Attempt to read behind the end of a file
Cannot seek	Seek failure
Cannot open source	Copy failure
Cannot create target	Copy failure

## 1.3 The Folder object

### 1.3.1 Properties

Property	Value	Description
current	Folder	The current folder is returned as a <code>Folder</code> object. Assigning either a <code>Folder</code> object or a string containing the new path name sets the current folder.
startup	Folder	The folder containing the executable image of the running application. Read-only.
system	Folder	The folder containing the operating system files. Read-only.
temp	Folder	The default folder for temporary files. Read-only.
trash	Folder	The folder containing deleted items. Read-only.

### 1.3.2 Constructors

```
Folder (path);  
new Folder (path);
```

This function constructs a new `Folder` object. If the given path name refers to an already existing disk file, a `File` object is returned instead.

#### Parameters

`path` `String` The full or partial path name of the folder. The folder that the path name refers to does not need to exist. If the argument is omitted, a temporary name is generated.

#### Returns

`Folder` or `File`

### 1.3.3 Methods

```
create();
```

Attempt to create a folder at the location the path name points to. Returns *true* if the folder was created.

**Returns:** *Boolean*

```
getFiles (String mask);
```

Get a list of `File` and `Folder` objects contained in the folder object. The mask is the search mask for the file names. It may contain question marks and asterisks and is preset to `*` to find all files. Alternatively, a function may be supplied. This function is called with a `File` or `Folder` object for every file or folder in the directory search. If the function returns `true`, the object is added to the array.

On Windows, all aliases end with the extension `".lnk"`. This extension is stripped from the file name when found to preserve compatibility with other operating systems. You can, however, search for all aliases by supplying the search mask `"*.lnk"`. This is NOT recommended, however, because it is not portable.

The return value is an array of `File` and/or `Folder` objects that correspond to the files found. The return value is `null` if the folder does not exist.

**Returns:** *Array*

## 1.4 The File object

### 1.4.1 Properties

Property	Value	Description
creator	String	The Macintosh file creator as a four-character string. On Windows, the return value is always "????". Read-only.
encoding	String	Gets or sets the encoding for subsequent read/write operations. The encoding is one of several predefined constants that follow the common Internet encoding names. Valid names are UCS-2, X-SJIS, ISO-8851-9, ASCII or the like. A special encoder, BINARY, is used to read binary files. This encoder stores each byte of the file as one Unicode character regardless of any encoding. When writing, the lower byte of each Unicode character is treated as a single byte to write. See appendix A for a list of encodings. If an unrecognized encoding is used, the encoding reverts to the system default encoding.
eof	Boolean	This property has the value <code>true</code> if a read attempt caused the current position to be behind the end of the file. Read only. If the file is not open, the value is <code>true</code> .
hidden	Boolean	Returns <code>true</code> if the file is invisible. Assigning a Boolean value sets or clears this attribute.
length	Integer	The size of the file in bytes. When setting the file size, the file must not be open.
lineFeed	String	The way line feed characters are written. This can be one of the three values <code>macintosh</code> , <code>unix</code> or <code>windows</code> (actually, only the first character is interpreted).
readonly	Boolean	This attribute, when set, prevents the file from being altered or deleted.
type	String	The Macintosh file type as a four-character string. On the Macintosh, the file type is returned. On Windows, "appl" is returned for .EXE files, "shlb" for .DLLs and "TEXT" for any other file. If the file does not exist, the file type is "????". Read-only.



### 1.4.2 Constructors

```
File (path);
new File (path);
```

This function constructs a new File object. If the given path name refers to an already existing folder, a Folder object is returned instead. The CRLF sequence is preset to the system default, and the encoding is preset to the default system encoding.

#### Parameters

path	String	The full or partial path name of the. The file that the path name refers to does not need to exist. If the argument is omitted, a temporary name is generated
------	--------	---

**Returns:** File or Folder

### 1.4.3 Methods

```
close();
```

Closes the open file. The return value is true if the file was closed, false on I/O errors.

**Returns:** Boolean

```
copy (target);
```

Copies the file to the given location. You can supply an URI path name as well as another File object. If there is a file at the target location, it is overwritten. The method returns true if the copy was successful, false otherwise. The method resolves any aliases to find the source file.

#### Parameters:

target	String/File	The target location.
--------	-------------	----------------------

**Returns:** Boolean

```
execute();
```

Attempt to find the application associated with this file. If found, load the application and cause it to load the file. This method may be used to execute a file much as it had been double-clicked in the Finder or Explorer. It can be used to run scripts, to launch other applications and much more.

Since this method opens a severe security hole, it is disabled by default. To enable the method, use the security flag sSEC\_FILE\_ALLOW\_EXECUTE.

The method returns immediately after launching the application. It does not wait for the application to terminate. It returns true if the launch was successful.

**Returns:** Boolean

```
open (mode, type, creator);
```

Open the file for subsequent read/write operations. The `type` and `creator` arguments optional arguments that are Macintosh specific; they specify the file type and creator as two four-character strings. They are used if the file is newly created. On other platforms, they are ignored.

When `open()` is used to open a file for read access, the method attempts to detect the encoding of the open file. It reads a few bytes at the current location and tries to detect the Byte Order Mark character 0xFFFE. If found, the current position is advanced behind the detected character and the *encoding* property is set to one of the strings UCS-2BE, UCS-2LE, UCS4-BE, UCS-4LE or UTF-8. If the marker character cannot be found, it checks for zero bytes at the current location and makes an assumption about one of the above formats (except for UTF-8). If everything fails, the *encoding* property is set to the system encoding. The method resolves any aliases to find the file.

If you try to open a file for writing or editing more than once the operating system usually permits you to do so, but if you start writing to the file using two different File objects, you may destroy your data!

The return value is `true` if the file has been opened successfully, `false` otherwise.

**Parameters:**

mode	String	<b>r (read)</b> Opens for reading. If the file does not exist or cannot be found the call fails. <b>w (write)</b> Opens an empty file for writing. If the file exists, its contents are destroyed. <b>e (edit)</b> Opens an existing file for reading and writing.
type	String	The Macintosh file type; a four-byte character string; ignored on non-Macintosh operating systems.
creator	String	The Macintosh file creator; a four-byte character string; ignored on non-Macintosh operating systems.

**Returns:** Boolean

```
read (chars);
```

Read the contents of the file from the current position on. Returns a string that contains up to the number of characters that were supposed to be read.

**Parameters:**

chars	Number	The number of characters to read. If the number of characters to read is not supplied, the entire file is read in one big chunk, starting at the current position. If the file is encoded, multiple bytes may be read to create single Unicode characters.
-------	--------	--

**Returns:** String.

```
readch( );
```

Read one single text character. Line feeds are recognized as CR, LF, CRLF or LFCR pairs. If the file is encoded, multiple bytes may be read to create single Unicode characters.

**Returns:** String

```
readln( );
```

Read one line of text. Line feeds are recognized as CR, LF, CRLF or LFCR pairs. If the file is encoded, multiple bytes may be read to create single Unicode characters.

**Returns:** String

```
seek (pos, mode);
```

Seek to a certain position in the file. Returns *true* if the position was changed. This method does not permit seeking to positions less than 0 or greater than the current file size.

**Parameters:**

pos	Number	The new current position inside the file as an offset in bytes, dependent on the seek mode.
mode	Number	The seek mode (0 = seek to absolute position, 1 = seek relative to the current position, 2 = seek backwards from the end of the file).

**Returns:** Boolean

```
tell( );
```

Returns the current position in the file as a an offset in bytes.

**Returns:** Number

```
write (text, ...);
```

Write the given string to the file. The parameters of this function are concatenated to a single string. Returns *true* on success. For encoded files, writing a single Unicode character may result in multiple bytes being written. You should take care not to write to a file that is open in another application or object. This may cause loss of data, since a second write issued from another File object may overwrite your data!

**Parameters:**

text                      String                      All arguments are concatenated to form the string to be written.

**Returns:** Boolean

```
writeln (text, ...);
```

Write the given string to the file. The parameters of this function are concatenated to a single string, and a Line Feed sequence is appended. Returns *true* on success. If the file is encoded, multiple bytes may be read to create single Unicode characters.

**Parameters:**

text                      String                      All arguments are concatenated to form the string to be written.

**Returns:** Boolean

## 1.5 General information

### 1.5.1 Path names

Although you can use operating system dependent file and path names, the `File` and `Folder` objects use an URI-like notation. A full path starts with a slash character; each element is separated by slashes. The first element is supposed to be the volume name on a Macintosh; on Windows, the first element should be the drive name. If the path name starts with two consecutive slashes, the default volume is assumed. The special path elements "." and ".." are recognized on the Macintosh and translated to the current and parent folders, respectively. The usage of the slash character as part of a file name, although a valid character on the Macintosh, is not supported. Optionally, the path can begin with the string "file:///". This prefix is ignored and removed.

Unfortunately, the file system conventions are quite distinct for the root elements of a full path name. It is, therefore, a good idea to use relative path names instead of absolute path names if you want to write portable scripts.

To illustrate the differences how the root element of a full path name is used on different file systems, consider the following examples. The current drive is C: on Windows, and "Macintosh HD" on the Macintosh.

URI	Windows name	Macintosh name
/d/dir/name.ext	D:\dir\name.ext	Macintosh HD:d:dir:name.ext
//dir/name.ext	C:\dir\name.ext	Macintosh HD:dir:name.ext
/Macintosh HD/dir/name.ext	C:\Macintosh HD\dir\name.exe	Macintosh HD:dir:name.ext

If a full path name does not contain a valid drive letter on Windows, the first element is considered to be a directory on the default drive. If the first element does not match a valid volume name on the Macintosh, the first element again is treated as a folder name in the default volume's root folder. You might, therefore, end up with a folder "d" on the Mac, or a directory "Macintosh HD" on Windows. Path names cannot be used to access different Macintosh volumes carrying the same name. If more than one volume with the same name exists, the volume that was mounted first is used.

### 1.5.2 Unicode I/O

Usually, the contents of a file are in some 8-bit encoding; most often, the current system encoding is used. When doing file I/O, the encoding used to convert between 8-bit character sets and Unicode is assumed to be the system encoding. You can, however, set a large number of encodings by setting the *encoding* property of a file to the name of the desired encoding. This name is one of the standard Internet names that are used to describe the encoding of

HTML files. Typical examples are ASCII, X-SJIS, or ISO-8859-1. The `File` objects attempts to find the corresponding encoder in the operating system. If present, this encoder will be used for subsequent I/O. Reading the *encoding* property returns the current encoding.

A special encoder, `BINARY`, is present for binary I/O. This encoder simply extends every 8-bit character it finds to a Unicode character between 0 and 255. When using this encoder to write binary files, the encoder writes the lower 8 bits of the Unicode character. If, for example, the Unicode character 1000 is written, the encoder actually writes the character 232 (1000 is 0x3E8, and 0xE8 gets written, which is 232).

Some of the common file formats (UCS-2, UCS-4, UTF-8) start with a special Byte Order Mark character ("`\uFEFF`"). The `File` method `open()` reads a few bytes of a file and tries to detect this character. If successful, the corresponding encoding is set automatically.

To write 16-bit Unicode files in UCS-2 format, use the encoding UCS-2. This encoding uses whatever endian format the host system supports. Make sure to write the Byte Order Mark character "`\uFEFF`" as the first character of the file. Do this also when using the UTF-8 encoding.

### 1.5.3 Windows aliases

On Windows, all file system aliases end with the extension ".lnk". You should never use this extension directly, however. The `File` and `Folder` classes work fine without these extensions. Imagine a shortcut to the file `some.txt`. The Windows file name would actually be `some.txt.lnk`. Use `some.txt` to create a `File` object matching this link. The `alias` property of this object would return `true`, and the `resolve()` method of the object would return the `File` object of the actual file. This behavior is the same as on the Macintosh.

Aliases on Windows and the Macintosh work much the same. All accesses to the alias file are transparently forwarded to the real file behind the alias file. Only the `rename()` and `remove()` calls affect an alias directly. The ".lnk" extension for Windows aliases is used transparently. This behavior is portable, but please keep in mind that Windows permits a file and its alias to reside in the same folder. If you have a file `test.txt` and its alias (which is `test.txt.lnk`), and you create a `File` object with `test.txt` you will access the original file and not the alias file; there is no way for you to access the alias file on a Windows system in this case.

## 1.6 Supported encoding names

The following list of names is a basic set of encoding names supported by the `File` object. Some of the character encoders are built in, while the operating system is queried for most of the other encoders. Depending on the language packs installed, some of the encodings may not be available. Names that refer to the same encoding are listed in one line. Underlines are replaced with dashes before matching an encoding name.

Note, however, that the `File` object cannot process extended Unicode character with values greater than 65535. These characters are left encoded as specified in the UTF-16 standard in as two characters in the range from 0xD700-0xDFFF.

Built-in encodings are:

US-ASCII,ASCII,ISO646-US,ISO-646.IRV:1991,ISO-IR-6, ANSI-X3.4-1968,CP367,IBM367,US,ISO646.1991-IRV

UCS-2,UCS2, ISO-10646-UCS-2

UCS2LE,UCS-2LE,ISO-10646-UCS-2LE

UCS2BE,UCS-2BE,ISO-10646-UCS-2BE

UCS-4,UCS4, ISO-10646-UCS-4

UCS4LE,UCS-4LE,ISO-10646-UCS-4LE

UCS4BE,UCS-4BE,ISO-10646-UCS-4BE

UTF-8,UTF8,UNICODE-1-1-UTF-8,UNICODE-2-0-UTF-8,X-UNICODE-2-0-UTF-8

UTF16,UTF-16,ISO-10646-UTF-16

UTF16LE,UTF-16LE,ISO-10646-UTF-16LE

UTF16BE,UTF-16BE,ISO-10646-UTF-16BE

CP1252,WINDOWS-1252,MS-ANSI

ISO-8859-1,ISO-8859-1,ISO-8859-1:1987,ISO-IR-100,LATIN1

MACINTOSH,X-MAC-ROMAN

BINARY

The ASCII encoder raises errors for characters greater than 127, and the BINARY encoder simply converts between bytes and Unicode characters by using the lower 8 bits. This encoder is convenient for reading and writing binary data.

### 1.6.1 Additional encodings

In Windows, all encodings use so-called code pages. These code pages are assigned numeric values. The usual Western character set that Windows uses is e.g. the code page 1252.

Windows code pages may be selected by prepending the number of the code page with "CP" or "WINDOWS- like "CP1252" for the code page 1252. The `File` object has a lot of other

encoding names built-in that match predefined code page numbers. If a code page is not present, the encoding cannot be selected.

On the Macintosh, encoders may be selected by name rather than by code page number. The *File* object queries the Macintosh OS directly for an encoder. As far as Macintosh character sets are identical with Windows code pages, the Macintosh also knows the Windows code page numbers.

### Common encoding names

The following encoding names are implemented both on Windows and Macintosh systems:

UTF-7,UTF7,UNICODE-1-1-UTF-7,X-UNICODE-2-0-UTF-7

ISO-8859-2,ISO-8859-2,ISO-8859-2:1987,ISO-IR-101,LATIN2

ISO-8859-3,ISO-8859-3,ISO-8859-3:1988,ISO-IR-109,LATIN3

ISO-8859-4,ISO-8859-4,ISO-8859-4:1988,ISO-IR-110,LATIN4,BALTIC

ISO-8859-5,ISO-8859-5,ISO-8859-5:1988,ISO-IR-144,CYRILLIC

ISO-8859-6,ISO-8859-6,ISO-8859-6:1987,ISO-IR-127,ECMA-114,ASMO-708,ARABIC

ISO-8859-7,ISO-8859-7,ISO-8859-7:1987,ISO-IR-126,ECMA-118,ELOT-928,GREEK8,GREEK

ISO-8859-8,ISO-8859-8,ISO-8859-8:1988,ISO-IR-138,HEBREW

ISO-8859-9,ISO-8859-9,ISO-8859-9:1989,ISO-IR-148,LATIN5,TURKISH

ISO-8859-10,ISO-8859-10,ISO-8859-10:1992,ISO-IR-157,LATIN6

ISO-8859-13,ISO-8859-13,ISO-IR-179,LATIN7

ISO-8859-14,ISO-8859-14,ISO-8859-14,ISO-8859-14:1998,ISO-IR-199,LATIN8

ISO-8859-15,ISO-8859-15,ISO-8859-15:1998,ISO-IR-203

ISO-8859-16,ISO-885,ISO-885,MS-EE

CP850,WINDOWS-850,IBM850

CP866,WINDOWS-866,IBM866

CP932,WINDOWS-932,SJIS,SHIFT-JIS,X-SJIS,X-MS-SJIS,MS-SJIS,MS-KANJI

CP936,WINDOWS-936,GBK,WINDOWS-936,GB2312,GB-2312-80,ISO-IR-58,CHINESE

CP949,WINDOWS-949,UHC,KSC-5601,KS-C-5601-1987,KS-C-5601-1989,ISO-IR-149,KOREAN

CP950,WINDOWS-950,BIG5,BIG-5,BIG-FIVE,BIGFIVE,CN-BIG5,X-X-BIG5

CP1251,WINDOWS-1251,MS-CYRL

CP1252,WINDOWS-1252,MS-ANSI

CP1253,WINDOWS-1253,MS-GREEK



CP1254,WINDOWS-1254,MS-TURK  
CP1255,WINDOWS-1255,MS-HEBR  
CP1256,WINDOWS-1256,MS-ARAB  
CP1257,WINDOWS-1257,WINBALTRIM  
CP1258,WINDOWS-1258  
CP1361,WINDOWS-1361,JOHAB  
EUC-JP,EUCJP,X-EUC-JP  
EUC-KR,EUCKR,X-EUC-KR  
HZ,HZ-GB-2312  
X-MAC-JAPANESE  
X-MAC-GREEK  
X-MAC-CYRILLIC  
X-MAC-LATIN  
X-MAC-ICELANDIC  
X-MAC-TURKISH

**Additional Windows encoding names**

CP437,IBM850,WINDOWS-437  
CP709,WINDOWS-709,ASMO-449,BCONV4  
EBCDIC  
KOI-8R  
KOI-8U  
ISO-2022-JP  
ISO-2022-KR

**Additional Macintosh encoding names**

These names are alias names for encodings that the Macintosh operating system might know.

TIS-620,TIS620,TIS620-0,TIS620.2529-1,TIS620.2533-0,TIS620.2533-1,ISO-IR-166  
CP874,WINDOWS-874  
JP,JIS-C6220-1969-RO,ISO646-JP,ISO-IR-14  
JIS-X0201,JISX0201-1976,X0201  
JIS-X0208,JIS-X0208-1983,JIS-X0208-1990,JIS0208,X0208,ISO-IR-87  
JIS-X0212,JIS-X0212.1990-0,JIS-X0212-1990,X0212,ISO-IR-159

CN,GB-1988-80,ISO646-CN,ISO-IR-57

ISO-IR-16,CN-GB-ISOIR165

KSC-5601,KS-C-5601-1987,KS-C-5601-1989,ISO-IR-149

EUC-CN,EUCCN,GB2312,CN-GB

EUC-TW,EUCTW,X-EUC-TW