4 Using File and Folder Objects

Overview

Because path name syntax is very different in Windows®, Mac OS® and UNIX®, Adobe ExendScript defines the `File` and `Folder` objects to provide platform-independent access to the underlying file system. A File Object represents a disk file, a Folder Object represents a directory or folder.

- The `Folder` object supports file system functionality such as traversing the hierarchy; creating, renaming or removing files; or resolving file aliases.
- The `File` object supports input/output functions to read or write files.

There are several ways to distinguish between a `File` and a `Folder` object. For example:

```
if (f instanceof File) ...
if (typeof f.open == "undefined") ...// Folders do not open
```

`File` and `Folder` objects can be used anywhere that a path name is required, such as in properties and arguments for files and folders. For details about the objects and their properties and methods, see Chapter 7, "File and Folder Object Reference".

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

Specifying Paths

When creating a `File` or `Folder` object, you can specify a platform-specific path name, or an absolute or relative path in a platform-independent format known as *universal resource identifier* (*URI*) notation. The path stored in the object is always an absolute, full path name that points to a fixed location on the disk.

- Use the `toString` method to obtain the name of the file or folder as string containing an absolute path name in URI notation.
- Use the fsName property to obtain the platform-specific file name.

Absolute and relative path names

An absolute path name in URI notation describes the full path from a root directory down to a specific file or folder. It starts with one or two slashes (`/`), and a slash separates path elements. For example, the following describes an absolute location for the file `myFile.jsx`:

```
/dir1/dir2/mydir/myFile.jsx
```

A relative path name in URI notation is appended to the path of the current directory, as stored in the globally-available current property of the `Folder` class. It starts with a folder or file name, or with one of the special names dot (`.`) for the current directory, or dot dot (`..`) for the parent of the current directory. A

slash (/) separates path elements. For example, the following paths describe various relative locations for the file myFile.jsx:

myFile.jsx ./myFile.jsx	In the current directory.
../myFile.jsx	In the parent of the current directory.
../../myFile.jsx	In the grandparent of the current directory.
../dir1/myFile.jsx	In dir1, which is parallel to the current directory.

Relative path names are independent of different volume names on different machines and operating systems, and therefore make your code considerably more portable. You can, for example, use an absolute path for a single operation, to set the current directory in the Folder.current property, and use relative paths for all other operations. You would then need only a single code change to update to a new platform or file location.

Character interpretation in paths

There are some platform differences in how pathnames are interpreted:

- In Windows and Mac OS, path names are not case sensitive. In UNIX, paths are case sensitive.
- In Windows, both the slash (/) and the backslash (\) are valid path element separators.
- In Mac OS, both the slash (/) and the colon (:) are valid path element separators.

If a path name starts with two slashes (or backslashes in Windows), the first element refers to a remote server. For example, //myhost/mydir/myfile refers to the path /mydir/myfile on the server myhost.

URI notation allows special characters in pathnames, but they must specified with an escape character (%) followed by a hexadecimal character code. Special characters are those which are not alphanumeric and not one of the characters:

```
/ - — . ! ~ * ' ( )
```

A space, for example, is encoded as %20, so the file name "my file" is specified as "my%20file". Similarly, the character ä is encoded as %E4, so the file name "Bräun" is specified as "Br%E4un".

This encoding scheme is compatible with the global JavaScript functions encodeURI and decodeURI.

The home directory

A path name can start with a tilde (~) to indicate the user's home directory. It corresponds to the platform's HOME environment variable.

UNIX and Mac OS assign the HOME environment variable according to the user login. In Mac OS, the default home directory is /Users/*username*. In UNIX, it is typically /home/*username* or /users/*username*. Extend Script assigns the home directory value directly from the platform value.

In Windows, the HOME environment variable is optional. If it is assigned, its value must be a Windows path name or a path name referring to a remote server (such as \\myhost\mydir). If the HOME environment variable is undefined, the Extend Script default is the user's home directory, usually the C:\Documents and Settings*username* folder.

Note: A script can access many of the folders that are specified with platform-specific variables through static, globally-available Folder class properties; for instance, appData contains the folder that stores application data for all users.

Volume and drive names

A volume or drive name can be the first part of an absolute path in URI notation. The values are interpreted according to the platform.

Mac OS volumes

When Mac OS X starts, the startup volume is the root directory of the file system. All other volumes, including remote volumes, are part of the `/Volumes` directory. The `File` and `Folder` objects use these rules to interpret the first element of a path name:

- If the name is the name of the startup volume, discard it.
- If the name is a volume name, prepend `/Volumes`.
- Otherwise, leave the path as is.

Mac OS 9 is not supported as an operating system, but the use of the colon as a path separator is still supported and corresponds to URI and to Mac OS X paths as shown in the following table. These examples assume that the startup volume is `MacOSX`, and that there is a mounted volume `Remote`.

URI path name	Mac OS 9 path name	Mac OS X path name
`/MacOSX/dir/file`	`MacOSX:dir:file`	`/dir/file`
`/Remote/dir/file`	`Remote:dir:file`	`/Volumes/Remote/dir/file`
`/root/dir/file`	`Root:dir:file`	`/root/dir/file`
`~/dir/file`		`/Users/jdoe/dir/file`

Windows drives

In Windows, volume names correspond to drive letters. The URI path `/c/temp/file` normally translates to the Windows path `C:\temp\file`.

If a drive exists with a name matching the first part of the path, that part is always interpreted as that drive. It is possible for there to be a folder in the root that has the same name as the drive; imagine, for example, a folder `C:\C` in Windows. A path starting with `/c` always addresses the drive `C:`, so in this case, to access the folder by name, you must use both the drive name and the folder name, for example `/c/c` for `C:\C`.

If the current drive contains a root folder with the same name as another drive letter, that name is considered to be a folder. That is, if there is a folder `D:\C`, and if the current drive is `D:`, the URI path `/c/temp/file` translates to the Windows path `D:\c\temp\file`. In this case, to access drive C, you would have to use the Windows path name conventions.

To access a remote volume, use a uniform naming convention (UNC) path name of the form `//servername/sharename`. These path names are portable, because both Max OS X and UNIX ignore multiple slash characters. Note that in Windows, UNC names do *not* work for local volumes.

These examples assume that the current drive is `D:`

URI path name	Windows path name
/c/dir/file	c:\dir\file
/remote/dir/file	D:\remote\dir\file
/root/dir/file	D:\root\dir\file
~/dir/file	C:\Documents and Settings\jdoe\dir\file

Aliases

When you access an alias, the operation is transparently forwarded to the real file. The only operations that affect the alias are calls to `rename` and `remove`, and setting properties `readonly` and `hidden`. When a `File` object represents an alias, the `alias` property of the object returns `true`, and the `resolve` method returns the `File` or `Folder` object for the target of the alias.

In Windows, all file system aliases (called *shortcuts*) are actual files whose names end with the extension `.lnk`. Never use this extension directly; the `File` and `Folder` objects work without it.

For example, suppose there is a shortcut to the file `/folder1/some.txt` in the folder `/folder2`. The full Windows file name of the shortcut file is `\folder2\some.txt.lnk`.

To access the shortcut from a `File` object, specify the path `/folder2/some.txt`. Calling that `File` object's `open` method opens the linked file (in `/folder1`). Calling the `File` object's `rename` method renames the shortcut file itself (leaving the `.lnk` extension intact).

However, Windows permits a file and its shortcut to reside in the same folder. In this case, the `File` object always accesses the original file. You cannot create a `File` object to access the shortcut when it is in the same folder as its linked file.

A script can create a file alias by creating a `File` object for a file that does not yet exist on disk, and using its createAlias method to specify the target of the alias.

Portability issues

If your application will run on multiple platforms, use relative path names, or try to originate path names from the home directory. If that is not possible, work with Mac OS X and UNIX aliases, and store your files on a machine that is remote to your Windows machine so that you can use UNC names.

As an example, suppose you use the UNIX machine `myServer` for data storage. If you set up an alias share in the root directory of `myServer`, and if you set up a Windows-accessible share at `share` pointing to the same data location, the path name `//myServer/share/file` would work for all three platforms.

Unicode I/O

When doing file I/O, Adobe applications convert 8-bit character encoding to Unicode. By default, this conversion process assumes that the system encoding is used (code page 1252 in Windows or Mac Roman in Mac OS). The `encoding` property of a `File` object returns the current encoding. You can set the `encoding` property to the name of the desired encoding. The `File` object looks for the corresponding encoder in the operating system to use for subsequent I/O. The name is one of the standard Internet names that are used to describe the encoding of HTML files, such as `ASCII`, `X-SJIS`, or `ISO-8859-1`. For a complete list, see File and Folder Supported Encoding Names.

A special encoder, `BINARY`, is provided 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. For example, to write the Unicode character `1000`, which is 0x3E8, the encoder actually writes the character 232 (0xE8).

The data of some of the common file formats (UCS-2, UCS-4, UTF-8, UTF-16) starts with a special byte order mark (BOM) character (`\uFEFF`). The `File.open` method reads a few bytes of a file looking for this character. If it is found, the corresponding encoding is set automatically and the character is skipped. If there is no BOM character at the beginning of the file, `open()` reads the first 2 KB of the file and checks whether the data might be valid UTF-8 encoded data, and if so, sets the encoding to UTF-8.

To write 16-bit Unicode files in UTF-16 format, use the encoding UCS-2. This encoding uses whatever byte-order format the host platform supports.

When using UTF-8 encoding or 16-bit Unicode, always write the BOM character "`\uFEFF`" as the first character of the file.

File Error Handling

Each object has an `error` property. If accessing a property or calling a method causes an error, this property contains a message describing the type of the error. On success, the property contains the empty string. You can set the property, 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.

For a complete list of supported error messages, see File and Folder Error Messages.

7 File and Folder Object Reference

Overview

Because path name syntax is very different in Windows, Mac OS and UNIX, the `File` and `Folder` objects are defined to provide platform-independent access to the underlying file system. A `File` object is associated with a disk file, a `Folder` object with a directory or folder.

- The `Folder` object supports file-system functionality such as traversing the hierarchy, creating, renaming or removing files, or resolving file aliases.
- The `File` object supports I/O functions to read or write files.

`File` and `Folder` objects can be used anywhere a path name is required, such as in properties and arguments for files and folders.

For a description of the pathname syntax and object usage, see Chapter 4, "Using File and Folder Objects." This chapter provides detail about the classes and objects, their properties and methods, and the supported encoding names:

File Object

Represents a file in the local file system in an platform-independent manner. All properties and methods resolve file system aliases automatically and act on the original file unless otherwise noted.

File object constructors

To create a `File` object, use the `File` function or the `new` operator. The constructor accepts full or partial path names, and returns the new object. The CRLF sequence for the file is preset to the system default, and the encoding is preset to the default system encoding.

```
File ([path]); //can return a Folder object
new File ([path]); //always returns a File object
```

path	Optional. The absolute or relative path to the file associated with this object, specified in platform-specific or URI format; see Specifying Paths. The value stored in the object is the absolute path. The path need not refer to an existing file. If not supplied, a temporary name is generated. If the path refers to an existing folder: • The `File` function returns a `Folder` object instead of a `File` object. • The `new` operator returns a `File` object for a nonexisting file with the same name.

File class properties

This property is available as a static property of the `File` class. It is not necessary to create an instance to access it.

fs	String	The name of the file system. Read-only. One of `Windows`, `Macintosh`, or `Unix`.

File class functions

These functions are available as static methods of the `File` class. It is not necessary to create an instance to call them.

decode `File.decode (what)`	Decodes the specified string as required by RFC 2396 and returns the decoded string.
what	String. The encoded string to decode. All special characters must be encoded in UTF-8 and stored as escaped characters starting with the percent sign followed by two hexadecimal digits. For example, the string "`my%20file`" is decoded as "`my file`". Special characters are those with a numeric value greater than 127, except the following: `/ - _ . ! ~ * ' ( )`
encode `File.encode (what)`	Encodes the specified string as required by RFC 2396 and returns the encoded string. All special characters are encoded in UTF-8 and stored as escaped characters starting with the percent sign followed by two hexadecimal digits. For example, the string "`my file`" is encoded as "`my%20file`". Special characters are those with a numeric value greater than 127, except the following: `/ - _ . ! ~ * ' ( )`
what	String. The string to encode.
isEncodingAvailable `File.isEncodingAvailable (name)`	Returns `true` if your system supports the specified encoding, `false` otherwise.
name	String. The encoding name.

openDialog

```
File.openDialog
  ([prompt][,select])
```

Opens the built-in platform-specific file-browsing dialog in which a user can select an existing file to open.

If the user clicks **OK**, returns a `File` object for the selected file. If the user cancels, returns `null`.

prompt Optional. A string containing the prompt text, if the dialog allows a prompt.

select Optional. A file or files to be preselected when the dialog opens:

- In Windows, a string containing a comma-separated list of file types with descriptive text, to be displayed in the bottom of the dialog as a drop-down list from which the user can select which types of files to display.

  Each element starts with the descriptive text, followed by a colon and the file search masks for this text, separated by semicolons. For example, to display two choices, one labeled **Text Files** that allows selection of text files with extensions `.TXT` and `.DOC`, and the other labeled **All files** that allows selection of all files:

  ```
  Text Files:*.TXT;*.DOC,All files:*
  ```

- In Mac OS, a string containing the name of a function defined in the current JavaScript scope that takes a `File` object argument. The function is called for each file about to be displayed in the dialog, and the file is displayed only when the function returns `true`.

saveDialog

```
File.saveDialog
  ([prompt][,select])
```

Opens the built-in platform-specific file-browsing dialog in which a user can select an existing file location to which to save this file.

If the user clicks **OK**, returns a `File` object for the selected file, and overwrites the existing file. If the user cancels, returns `null`.

prompt Optional. A string containing the prompt text, if the dialog allows a prompt.

select Optional. A file or files to be preselected when the dialog opens:

- In Windows, a string containing a comma-separated list of file types with descriptive text, to be displayed in the bottom of the dialog as a drop-down list from which the user can select which types of files to display.

  Each element starts with the descriptive text, followed by a colon and the file search masks for this text, separated by semicolons. For example, to display two choices, one labeled **Text Files** that allows selection of text files with extensions `.TXT` and `.DOC`, and the other labeled **All files** that allows selection of all files:

  ```
  Text Files:*.TXT;*.DOC,All files:*
  ```

- In Mac OS, a string containing the name of a function defined in the current JavaScript scope that takes a `File` object argument. The function is called for each file about to be displayed in the dialog, and the file is displayed only when the function returns `true`.

File object properties

These properties are available for `File` objects.

absoluteURI	String	The full path name for the referenced file in URI notation. Read-only.
alias	Boolean	When `true`, the object refers to a file system alias or shortcut. Read-only.
created	Date	The creation date of the referenced file, or `null` if the object does not refer to a file on disk. Read-only.
creator	String	The Mac OS file creator as a four-character string. In Windows or UNIX, value is `"????"`. Read-only.
encoding	String	Gets or sets the encoding for subsequent read/write operations. One of the encoding constants listed in File and Folder Supported Encoding Names. If the value is not recognized, uses the system default encoding. A special encoder, `BINARY`, is used to read binary files. It 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.
eof	Boolean	When `true`, a read attempt caused the current position to be beyond the end of the file, or the file is not open. Read only.
error	String	A message describing the last file system error; see File and Folder Error Messages. Setting this value clears any error message and resets the error bit for opened files.
exists	Boolean	When `true`, the path name of this object refers to an existing file. Read only.
fsName	String	The platform-specific name of the referenced file as a full path name. Read-only.
hidden	Boolean	When `true`, the file is not shown in the platform-specific file browser. Read/write. If the object references a file-system alias or shortcut, the flag is altered on the alias, not on the original file.
length	Number	The size of the file in bytes. Can be set only for a file that is not open, in which case it truncates or pads the file with 0-bytes to the new length.
lineFeed	String	How line feed characters are written. One of: `windows`: Windows style `mac`: Mac OS style `unix`: UNIX style
modified	Date	The date of the referenced file's last modification, or `null` if the object does not refer to a file on disk. Read-only.
name	String	The name of the referenced file without the path specification. Read-only.
parent	Folder	The `Folder` object for the folder that contains this file. Read-only.

path	String	The path portion of the absolute URI, or the empty string If the name does not have a path. Read-only.
readonly	Boolean	When `true`, prevents the file from being altered or deleted. If the referenced file is a file-system alias or shortcut, the flag is altered on the alias, not on the original file.
relativeURI	String	The path name for the referenced file in URI notation, relative to the current folder. Read-only.
type	String	The Mac OS file type as a four-character string. In Windows and UNIX, the value is `"????"`. Read-only.

File object functions

These functions are available for `File` objects.

close *fileObj*.close ()	Closes this open file. Returns `true` on success, `false` if there are I/O errors.
copy *fileObj*.copy (*target*)	Copies this object's referenced file to the specified target location. Resolves any aliases to find the source file. If a file exists at the target location, it is overwritten. Returns `true` if the copy was successful, `false` otherwise.
target	A string with the URI path to the target location, or a `File` object that references the target location.
createAlias *fileObj*.createAlias (*toFile*, [*isFinderAlias*])	Makes this file into a file-system alias or shortcut to the specified file. The referenced file for this object must not yet exist on disk. Returns `true` if the operation was successful, `false` otherwise.
toFile	The `File` object for the target of the new alias.
isFinderAlias	Optional, Mac OS only. When `true`, the alias is created as a legacy Finder alias. When `false` (the default), the alias is created as a Unix symlink.
execute *fileObj*.execute ()	Opens this file using the appropriate application (as if it had been double-clicked in a file browser). You can use this method to run scripts, launch applications, and so on. Returns `true` immediately if the application launch was successful.
getRelativeURI *fileObj*.getRelativeURI ([*basePath*])	Returns a string containing the URI for this file or folder relative to the specified base path, in URI notation. If no base path is supplied, returns the URI relative to the path of the current folder.
basePath	Optional. A string containing the base path for the relative URI. Default is the current folder.

open

fileObj.open
(*mode*[,*type*][,*creator*])

Open the file for subsequent read/write operations. The method resolves any aliases to find the file. Returns `true` if the file has been opened successfully, `false` otherwise.

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 is not found, it checks for zero bytes at the current location and makes an assumption about one of the above formats (except UTF-8). If everything fails, the `encoding` property is set to the system encoding.

Note: Be careful about opening a file 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 can destroy your data.

mode	A string indicating the read/write mode. One of: `r`: (read) Opens for reading. If the file does not exist or cannot be found, the call fails. `w`: (write) Opens a file for writing. If the file exists, its contents are destroyed. If the file does not exist, creates a new, empty file. `e`: (edit) Opens an existing file for reading and writing.
type	Optional. In Mac OS, the type of a newly created file, a 4-character string. Ignored in Windows and UNIX.
creator	Optional. In Mac OS, the creator of a newly created file, a 4-character string. Ignored in Windows and UNIX.

openDlg
fileObj.OpenDlg ([*prompt*][,*select*])

Opens the built-in platform-specific file-browsing dialog, in which the user can select an existing file to open. If the user clicks **OK**, returns a `File` or `Folder` object for the selected file or folder. If the user cancels, returns `null`.

Differs from the class method `openDialog()` in that it presets the current folder to this `File` object's parent folder and the current file to this object's associated file.

prompt Optional. A string containing the prompt text, if the dialog allows a prompt.

select Optional. A file or files to be preselected when the dialog opens:

- In Windows, a string containing a comma-separated list of file types with descriptive text, to be displayed in the bottom of the dialog as a drop-down list from which the user can select which types of files to display.

  Each element starts with the descriptive text, followed by a colon and the file search masks for this text, separated by semicolons. For example, to display two choices, one labeled **Text Files** that allows selection of text files with extensions `.TXT` and `.DOC`, and the other labeled **All files** that allows selection of all files:

  ```
  Text Files:*.TXT;*.DOC,All files:*
  ```

- In Mac OS, a string containing the name of a function defined in the current JavaScript scope that takes a `File` object argument. The function is called for each file about to be displayed in the dialog, and the file is displayed only when the function returns `true`.

read
fileObj.read ([*chars*])

Reads the contents of the file starting at the current position, and returns a string that contains up to the specified number of characters.

chars Optional. An integer specifying the number of characters to read. By default, reads from the current position to the end of the file. If the file is encoded, multiple bytes might be read to create single Unicode characters.

readch
fileObj.readch ()

Reads a single text character from the file at the current position, and returns it in a string. Line feeds are recognized as `CR`, `LF`, `CRLF`, or `LFCR` pairs. If the file is encoded, multiple bytes might be read to create single Unicode characters.

readln
fileObj.readln ()

Reads a single line of text from the file at the current position, and returns it in a string. Line feeds are recognized as `CR`, `LF`, `CRLF`, or `LFCR` pairs. If the file is encoded, multiple bytes might be read to create single Unicode characters.

remove
fileObj.remove ()

Deletes the file associated with this object from disk, immediately, without moving it to the system trash. Returns `true` if the file is deleted successfully.

Does not resolve aliases; instead, deletes the referenced alias or shortcut file itself.

Note: Cannot be undone. It is recommended that you prompt the user for permission before deleting.

rename
fileObj.rename (*newName*)

Renames the associated file. Returns `true` on success.

Does not resolve aliases, but renames the referenced alias or shortcut file itself.

newName The new file or folder name, with no path.

resolve
fileObj.resolve ()

If this object references an alias or shortcut, this method resolves that alias and returns a new `File` object that references the file-system element to which the alias resolves.

Returns `null` if this object does not reference an alias, or if the alias cannot be resolved.

saveDlg
fileObj.saveDlg ([*prompt*][,*preset*])

Opens the built-in platform-specific file-browsing dialog, in which the user can select an existing file location at which to save this file. If the user clicks **OK**, returns a `File` or `Folder` object for the selected file or folder. If the user cancels, returns `null`.

Differs from the class method `saveDialog()` in that it presets the current folder to this `File` object's parent folder and the file to this object's associated file, and prompts the user to confirm before overwriting an existing file.

prompt Optional. A string containing the prompt text, if the dialog allows a prompt.

preset Optional. A file or files to be preselected when the dialog opens:

- In Windows, a string containing a comma-separated list of file types with descriptive text, to be displayed in the bottom of the dialog as a drop-down list from which the user can select which types of files to display.

  Each element starts with the descriptive text, followed by a colon and the file search masks for this text, separated by semicolons. For example, to display two choices, one labeled **Text Files** that allows selection of text files with extensions `.TXT` and `.DOC`, and the other labeled **All files** that allows selection of all files:

  ```
  Text Files:*.TXT;*.DOC,All files:*
  ```

- In Mac OS, a string containing the name of a function defined in the current JavaScript scope that takes a `File` object argument. The function is called for each file about to be displayed in the dialog, and the file is displayed only when the function returns `true`.

seek
fileObj.seek (*pos*, *mode*)

Seeks to the specified position in the file, and returns `true` if the position was changed. The new position cannot be less than 0 or greater than the current file size.

pos The new current position in the file as an offset in bytes from the start, current position, or end, depending on the `mode`.

mode The seek mode, one of:

0: Seek to absolute position, where `pos=0` is the first byte of the file.
1: Seek relative to the current position.
2. Seek backward from the end of the file.

tell
fileObj.tell ()

Returns the current position as a byte offset from the start of the file.

write *fileObj*.write (*text*[, *text*...]...)	Writes the specified text to the file at the current position. Returns `true` on success. For encoded files, writing a single Unicode character may write multiple bytes. **Note:** Be careful not to write to a file that is open in another application or object, as this can overwrite existing data.
text	One or more strings to write, which are concatenated to form a single string.
writeln *fileObj*.writeln (*text*[, *text*...]...)	Writes the specified text to the file at the current position, and appends a Line Feed sequence in the style specified by the `linefeed` property. Returns `true` on success. For encoded files, writing a single Unicode character may write multiple bytes. **Note:** Be careful not to write to a file that is open in another application or object, as this can overwrite existing data.
text	One or more strings to write, which are concatenated to form a single string.

Folder Object

Represents a file-system folder or directory in a platform-independent manner. All properties and methods resolve file system aliases automatically and act on the original file unless otherwise noted.

Folder object constructors

To create a `Folder` object, use the `Folder` function or the `new` operator. The constructor accepts full or partial path names, and returns the new object.

```
Folder ([path]); //can return a File object
new Folder ([path]); //always returns a Folder object
```

path	Optional. The absolute or relative path to the folder associated with this object, specified in URI format; see Specifying Paths. The value stored in the object is the absolute path. The path need not refer to an existing folder. If not supplied, a temporary name is generated. If the path refers to an existing file: • The `Folder` function returns a `File` object instead of a `Folder` object. • The `new` operator returns a `Folder` object for a nonexisting folder with the same name.

Folder class properties

These properties are available as static properties of the `Folder` class. It is not necessary to create an instance to access them.

appData	Folder	A `Folder` object for the folder that contains application data for all users. Read-only. • In Windows, the value of `%APPDATA%` (by default, `C:\Documents and Settings\All Users\Application Data`) • In Mac OS, `/Library/Application Support`
commonFiles	Folder	A `Folder` object for the folder that contains files common to all programs. Read-only. • In Windows, the value of `%CommonProgramFiles%` (by default, `C:\Program Files\Common Files`) • In Mac OS,`/Library/Application Support`
current	Folder	A `Folder` object for the current folder. Assign either a `Folder` object or a string containing the new path name to set the current folder.
fs	String	The name of the file system. Read-only. One of `Windows`, `Macintosh`, or `Unix`.
myDocuments	Folder	A `Folder` object for the default document folder. Read-only. • In Windows, `C:\Documents and Settings\`*username*`\My Documents` • In Mac OS, `~/Documents`
startup	Folder	A `Folder` object for the folder containing the executable image of the running application. Read-only.

system	Folder	A `Folder` object for the folder containing the operating system files. Read-only. • In Windows, the value of `%windir%` (by default, `C:\Windows`) • In Mac OS, `/System`
temp	Folder	A `Folder` object for the default folder for temporary files. Read-only.
trash	Folder	A `Folder` object for the folder containing deleted items. Read-only.
userData	Folder	A `Folder` object for the folder that contains application data for the current user. Read-only. • In Windows, the value of `%APPDATA%` (by default, `C:\Documents and Settings\`*`username`*`\Application Data`) • In Mac OS, `~/Library/Application Support`

Folder class functions

These functions are available as a static methods of the `Folder` class. It is not necessary to create an instance in order to call them.

decode
`Folder.decode (`*what*`)`

Decodes the specified string as required by RFC 2396 and returns the decoded string.

what — String. The encoded string to decode.

All special characters must be encoded in UTF-8 and stored as escaped characters starting with the percent sign followed by two hexadecimal digits. For example, the string "`my%20file`" is decoded as "`my file`".

Special characters are those with a numeric value greater than 127, except the following:

`/ - _ . ! ~ * ' ( )`

encode
`Folder.encode (`*what*`)`

Encodes the specified string as required by RFC 2396 and returns the encoded string.

All special characters are encoded in UTF-8 and stored as escaped characters starting with the percent sign followed by two hexadecimal digits. For example, the string "`my file`" is encoded as "`my%20file`".

Special characters are those with a numeric value greater than 127, except the following:

`/ - _ . ! ~ * ' ( )`

what — String. The string to encode.

isEncodingAvailable
`File.isEncodingAvailable (`*name*`)`

Returns `true` if your system supports the specified encoding, `false` otherwise.

name — String. The encoding name.

selectDialog `Folder.selectDialog ([prompt][,preset])`	Opens the built-in platform-specific file-browsing dialog. If the user clicks **OK**, returns a `Folder` object for the selected folder. If the user cancels, returns `null`. Differs from the object method `selectDlg()` in that it does not preselect a folder.
prompt	Optional. A string containing the prompt text, if the dialog allows a prompt.
preset	Optional. A `Folder` object for a folder to be preselected when the dialog opens.

Folder object properties

These properties are available for `Folder` objects.

absoluteURI	String	The full path name for the referenced folder in URI notation. Read-only.
alias	Boolean	When `true`, the object refers to a file system alias or shortcut. Read-only.
created	Date	The creation date of the referenced folder, or `null` if the object does not refer to a folder on disk. Read-only.
error	String	A message describing the last file system error; see File and Folder Error Messages. Setting this value clears any error message and resets the error bit for opened folders.
exists	Boolean	When `true`, the path name of this object refers to an existing folder. Read only.
fsName	String	The platform-specific name of the referenced folder as a full path name. Read-only.
modified	Date	The date of the referenced folder's last modification, or `null` if the object does not refer to a folder on disk. Read-only.
name	String	The name of the referenced folder without the path specification. Read-only.
parent	Folder	The `Folder` object for the folder that contains this folder, or `null` if this object refers to the root folder of a volume. Read-only.
path	String	The path portion of the absolute URI, or the empty string If the name does not have a path. Read-only.
relativeURI	String	The path name for the referenced folder in URI notation, relative to the current folder. Read-only.

Folder object functions

These functions are available for `Folder` objects.

create *folderObj*.create ()	Creates a folder at the location to which the path name points. Returns `true` if the folder was created successfully.
execute *folderObj*.execute ()	Opens this folder in the file browser (as if it had been double-clicked in a file browser). Returns `true` immediately if the folder was opened successfully.
getFiles *folderObj*.getFiles ([*mask*])	Returns an array of `File` and `Folder` objects for the contents of this folder, filtered by the supplied `mask`, or `null` if this object's referenced folder does not exist.
mask	Optional. A search mask for file names. A string that can contain question mark (`?`) and asterisk (`*`) wild cards. Default is `"*"`, which matches all file names. Can also be the name of a function that takes a `File` or `Folder` object as its argument. It is called for each file or folder found in the search; if it returns `true`, the object is added to the return array. **Note:** In Windows, all aliases end with the extension `.lnk`, which is stripped from the file name when found to preserve compatibility with other operating systems. You can search for all aliases by supplying the search mask `"*.lnk"`, but note that such code is not portable.
getRelativeURI *folderObj*.getRelativeURI ([*basePath*])	Returns a string containing the URI for this folder relative to the specified base path, in URI notation. If no base path is supplied, returns the URI relative to the path of the current folder.
basePath	Optional. A string containing the base path for the relative URI. Default is the current folder.
remove *folderObj*.remove ()	Deletes the empty folder associated with this object from disk, immediately, without moving it to the system trash. Returns `true` if the folder is deleted successfully. • Folders must be empty before they can be deleted. • Does not resolve aliases; instead, deletes the referenced alias or shortcut file itself. **Note:** Cannot be undone. It is recommended that you prompt the user for permission before deleting.
rename *folderObj*.rename (*newName*)	Renames the associated folder. Returns `true` on success. • Does not resolve aliases; instead, renames the referenced alias or shortcut file itself.
newName	The new folder name, with no path.
resolve *folderObj*.resolve ()	If this object references an alias or shortcut, this method resolves that alias and returns a new `Folder` object that references the file-system element to which the alias resolves. Returns `null` if this object does not reference an alias, or if the alias cannot be resolved.

selectDlg *folderObj*.selectDlg ([*prompt*][,*preset*])	Opens the built-in platform-specific file-browsing dialog. If the user clicks **OK**, returns a `File` or `Folder` object for the selected file or folder. If the user cancels, returns `null`. Differs from the class method `selectDialog()` in that it preselects this folder.
prompt	Optional. A string containing the prompt text, if the dialog allows a prompt.
preset	Optional. A `Folder` object for a folder to be preselected when the dialog opens.

File and Folder Error Messages

The following messages can be returned in the `error` property.

`File or folder does not exist`	The file or folder does not exist, but the parent folder exists.
`File or folder already exists`	The file or folder already exists.
`I/O device is not open`	An I/O operation was attempted on a file that was closed.
`Read past EOF`	Attempt to read beyond the end of a file.
`Conversion error`	The content of the file cannot be converted to Unicode.
`Partial multibyte character found`	The character encoding of the file data has errors.
`Permission denied`	The OS did not allow the attempted operation.
`Cannot change directory`	Cannot change the current folder.
`Cannot create`	Cannot create a folder.
`Cannot rename`	Cannot rename a file or folder.
`Cannot delete`	Cannot delete a file or folder.
`I/O error`	Unspecified I/O error.
`Cannot set size`	Setting the file size failed.
`Cannot open`	Opening of a file failed.
`Cannot close`	Closing a file failed.
`Read error`	Reading from a file failed.
`Write error`	Writing to a file failed.
`Cannot seek`	Seek failure.
`Cannot execute`	Unable to execute the specified file.

File and Folder 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.

The `File` object processes an extended Unicode character with a value greater that 65535 as a Unicode surrogate pair (two characters in the range between 0xD700-0xDFFF).

Built-in encodings are:

```
US-ASCII, ASCII,ISO646-US,I SO-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. The latter encoder is convenient for reading and writing binary data.

Additional encodings

In Windows, all encodings use code pages, which are assigned numeric values. The usual Western character set that Windows uses, for example, is the code page 1252. You can select Windows code pages by prepending the number of the code page with "CP" or "WINDOWS": for example, "CP1252" for the code page 1252. The `File` object has many other built-in encoding names that match predefined code page numbers. If a code page is not present, the encoding cannot be selected.

In Mac OS, you can select encoders by name rather than by code page number. The `File` object queries Mac OS directly for an encoder. As far as Mac OS character sets are identical with Windows code pages, Mac OS also knows the Windows code page numbers.

In UNIX, the number of available encoders depends on the installation of the `iconv` library.

Common encoding names

The following encoding names are implemented both in Windows and in Mac OS:

```
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 Mac OS encoding names

These names are alias names for encodings that Mac OS 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
```

UNIX encodings

In UNIX, the `File` object looks for the presence of the `iconv` library, and uses whatever encoding it finds there. If you need a special encoding in UNIX, make sure that there is an `iconv` encoding module installed that converts between UTF-16 (the internal format that the `File` object uses) and the desired encoding.