file

The File plugin provides functionality to work with files.

Main features are:

- Creating temporary files
- Reading from and writing to the filesystem
- Retrieving information on files and directories on the file system
- Streaming files between the Smart Client and the Server (and vise versa)

When using the File plugin, it's important to take into account the differences and compatibility between different clients:

- Interacting with the file system through the plugin in a Smart Client happens client-side, so on the machine where the Smart Client is launched. On all other clients (Web, Headless & Batchprocessor), the operations are performed on the Server.
- All showXxxxDialog(...) functions interact with the user through a UI. These function can only be used in Clients that provide a UI, like the Smart and Web Client.
- The showXxxxDialog(...) functions, when used in the Web Client, have certain limitations due to being operated in a browser. Browser security (currently) limits interaction with the local file system, except for single file select operations initiated by the user clicking a button.

Return Types

JSFile

Server Property Summery

#servoy.FileServerService.defaultFolder

Method Summery

Boolean	#appendToTXTFile(file/fileName, text, [encoding]) Appends data into a text file.
JSFile	#convertToJSFile(file) Returns a JSFile instance corresponding to an alternative representation of a file (for example a string).
JSFile	#convertToRemoteJSFile(serverPath) Returns the JSFile object of a server file, given its path (relative the default server location)
Boolean	#copyFile(sourceFile, destinationFile) Copies the sourcefile to the destination file.
Boolean	#copyFolder(sourceFolder, destinationFolder) Copies the sourcefolder to the destination folder, recursively.
JSFile	#createFile(targetFile) Creates a JSFile instance.
Boolean	#createFolder(targetFolder) Creates a folder on disk.
JSFile	#createTempFile(filePrefix, fileSuffix) Creates a temporary file on disk.
Boolean	#deleteFile(targetFile) Removes a file from disk.
Boolean	#deleteFolder(targetFolder, showWarning) Deletes a folder from disk recursively.
JSFile	#getDesktopFolder() Returns a JSFile instance that corresponds to the Desktop folder of the currently logged in user.
JSFile[]	#getDiskList() Returns an Array of JSFile instances correponding to the file system root folders.
Number	#getFileSize(targetFile) Returns the size of the specified file.
JSFile[]	#getFolderContents(targetFolder, [fileFilter], [fileOption(1=files,2=dirs)], [visibleOption(1=visible,2=nonvisible)], [lockedOption(1=locked,2=nonlocked)]) Returns an array of JSFile instances corresponding to content of the specified folder.
JSFile	#getHomeDirectory() Returns a JSFile instance corresponding to the home folder of the logged in used.
Date	#getModificationDate(targetFile) Returns the modification date of a file.
JSFile[]	#getRemoteList(serverFolder/serverFolderPath, [filesOnly]) Retrieves a list of files existing in a folder on the server side (in the path provided, relative to the default server location)
Boolean	#moveFile(sourceFile, destinationFile) Moves the file from the source to the destination place.
byte[]	#readFile([file], [size]) Reads all or part of the content from a binary file.
String	#readTXTFile([file], [charsetname]) Read all content from a text file.
JSFile	#showDirectorySelectDialog([directory suggestion], [dialog title text]) Shows a directory selector dialog.
Object	#showFileOpenDialog([selectionMode(0=both,1=Files,2=Dirs)], [startDirectory(null=default/previous)], [multiselect(true/false)], [filterarray], [callbackmethod], [dialog title text]) Shows a file open dialog.
JSFile	#showFileSaveDialog([fileName/dir suggestion], [dialog title text]) Shows a file save dialog.
void	#streamFilesFromServer(file/fileName|fileArray/fileNameArray, serverFile/serverFileName|serverFileArray/serverFileNameArray, [callbackFunction]) Streams a file or an array of files from the server in a background task to a file (or files) on the client.
void	#streamFilesToServer(file/fileName|fileArray/fileNameArray, [serverFile/serverFileName|serverFileArray/serverFileNameArray], [callbackFunction]) Streams a file or an array of files to the server in a background task - with optional relative path(s)/(new) name(s).
Boolean	#writeFile(file, binary_data, [mimeType]) Writes data into a binary file.
Boolean	#writeTXTFile(file, text_data, [charsetname], [mimeType]) Writes data into a text file.
Boolean	#writeXMLFile(file, xml_data) Writes data into an XML file.

Server Property Details

servoy.FileServerService.defaultFolder

Set the default folder path to save files sent by client (will default to /uploads/)

Method Details

appendToTXTFile

Boolean **appendToTXTFile**(file/fileName, text, [encoding])

Appends data into a text file.

Parameters

file/fileName
text
[encoding]

Returns

Boolean

Sample

```
// append some text to a text file:
        var ok = plugins.file.appendToTXTFile('myTextFile.txt', '\nMy fantastic new line of text\n');
```

convertToJSFile

JSFile **convertToJSFile**(file)

Returns a JSFile instance corresponding to an alternative representation of a file (for example a string).

Parameters

file

Returns

JSFile

Sample

```
var f = plugins.file.convertToJSFile("story.txt");
if (f.canRead())
        application.output("File can be read.");
```

convertToRemoteJSFile

JSFile **convertToRemoteJSFile**(serverPath)

Returns the JSFile object of a server file, given its path (relative the default server location)

Parameters

serverPath

Returns

JSFile

Sample

```
var f = plugins.file.convertToRemoteJSFile('/story.txt');
if (f && f.canRead())
        application.output('File can be read.');
```

copyFile

Boolean **copyFile**(sourceFile, destinationFile)

Copies the sourcefile to the destination file. Returns true if the copy succeeds, false if any error occurs.

Parameters

sourceFile
destinationFile

Returns

Boolean

Sample

```
// Copy based on file names.
if (!plugins.file.copyFile("story.txt", "story.txt.copy"))
        application.output("Copy failed.");
// Copy based on JSFile instances.
var f = plugins.file.createFile("story.txt");
var fcopy = plugins.file.createFile("story.txt.copy2");
if (!plugins.file.copyFile(f, fcopy))
        application.output("Copy failed.");
```

copyFolder

Boolean **copyFolder**(sourceFolder, destinationFolder)

Copies the sourcefolder to the destination folder, recursively. Returns true if the copy succeeds, false if any error occurs.

Parameters
sourceFolder
destinationFolder
Returns
Boolean
Sample

```
// Copy folder based on names.
if (!plugins.file.copyFolder("stories", "stories_copy"))
        application.output("Folder copy failed.");
// Copy folder based on JSFile instances.
var d = plugins.file.createFile("stories");
var dcopy = plugins.file.createFile("stories_copy_2");
if (!plugins.file.copyFolder(d, dcopy))
        application.output("Folder copy failed.");
```

createFile

JSFile **createFile**(targetFile)

Creates a JSFile instance. Does not create the file on disk.
Parameters
targetFile
Returns
JSFile
Sample

```
// Create the JSFile instance based on the file name.
var f = plugins.file.createFile("newfile.txt");
// Create the file on disk.
if (!f.createNewFile())
        application.output("The file could not be created.");
```

createFolder

Boolean **createFolder**(targetFolder)

Creates a folder on disk. Returns true if the folder is successfully created, false if any error occurs.
Parameters
targetFolder
Returns
Boolean
Sample

```
var d = plugins.file.convertToJSFile("newfolder");
if (!plugins.file.createFolder(d))
        application.output("Folder could not be created.");
```

createTempFile

JSFile **createTempFile**(filePrefix, fileSuffix)

Creates a temporary file on disk. A prefix and an extension are specified and they will be part of the file name.
Parameters
filePrefix
fileSuffix
Returns
JSFile
Sample

```
var tempFile = plugins.file.createTempFile('myfile','.txt');
application.output('Temporary file created as: ' + tempFile.getAbsolutePath());
plugins.file.writeTXTFile(tempFile, 'abcdefg');
```

deleteFile

Boolean **deleteFile**(targetFile)

Removes a file from disk. Returns true on success, false otherwise.
Parameters
targetFile

Returns
Boolean
Sample

```
if (plugins.file.deleteFile('story.txt'))
        application.output('File deleted.');
```

deleteFolder

Boolean **deleteFolder**(targetFolder, showWarning)

Deletes a folder from disk recursively. Returns true on success, false otherwise. If the second parameter is set to true, then a warning will be issued to the user before actually removing the folder.

Parameters
targetFolder
showWarning
Returns
Boolean
Sample

```
if (plugins.file.deleteFolder('stories', true))
        application.output('Folder deleted.');
```

getDesktopFolder

JSFile **getDesktopFolder**()

Returns a JSFile instance that corresponds to the Desktop folder of the currently logged in user.

Returns
JSFile
Sample

```
var d = plugins.file.getDesktopFolder();
application.output('desktop folder is: ' + d.getAbsolutePath());
```

getDiskList

JSFile[] **getDiskList**()

Returns an Array of JSFile instances correponding to the file system root folders.

Returns
JSFile[]
Sample

```
var roots = plugins.file.getDiskList();
for (var i = 0; i < roots.length; i++)
        application.output(roots[i].getAbsolutePath());
```

getFileSize

Number **getFileSize**(targetFile)

Returns the size of the specified file.

Parameters
targetFile
Returns
Number
Sample

```
var f = plugins.file.convertToJSFile('story.txt');
application.output('file size: ' + plugins.file.getFileSize(f));
```

getFolderContents

JSFile[] **getFolderContents**
(targetFolder, [fileFilter], [fileOption(1=files,2=dirs)], [visibleOption(1=visible,2=nonvisible)], [lockedOption(1=locked,2=nonlocked)])

Returns an array of JSFile instances corresponding to content of the specified folder. The content can be filtered by optional name filter(s), by type, by visibility and by lock status.

Parameters
targetFolder
[fileFilter]
[fileOption(1=files,2=dirs)]
[visibleOption(1=visible,2=nonvisible)]
[lockedOption(1=locked,2=nonlocked)]
Returns
JSFile[]
Sample

```
var files = plugins.file.getFolderContents('stories', '.txt');
for (var i=0; i<files.length; i++)
        application.output(files[i].getAbsolutePath());
```

getHomeDirectory

JSFile **getHomeDirectory**()

Returns a JSFile instance corresponding to the home folder of the logged in used.

Returns
JSFile
Sample

```
var d = plugins.file.getHomeDirectory();
application.output('home folder: ' + d.getAbsolutePath());
```

getModificationDate

Date **getModificationDate**(targetFile)

Returns the modification date of a file.

Parameters
targetFile
Returns
Date
Sample

```
var f = plugins.file.convertToJSFile('story.txt');
application.output('last changed: ' + plugins.file.getModificationDate(f));
```

getRemoteList

JSFile[] **getRemoteList**(serverFolder/serverFolderPath, [filesOnly])

Retrieves a list of files existing in a folder on the server side (in the path provided, relative to the default server location)

Parameters
serverFolder/serverFolderPath
[filesOnly]
Returns
JSFile[]
Sample

```

```

moveFile

Boolean **moveFile**(sourceFile, destinationFile)

Moves the file from the source to the destination place. Returns true on success, false otherwise.

Parameters
sourceFile
destinationFile
Returns
Boolean

Sample

```
// Move file based on names.
if (!plugins.file.moveFile('story.txt','story.txt.new'))
	application.output('File move failed.');
// Move file based on JSFile instances.
var f = plugins.file.convertToJSFile('story.txt.new');
var fmoved = plugins.file.convertToJSFile('story.txt');
if (!plugins.file.moveFile(f, fmoved))
	application.output('File move back failed.');
```

readFile

byte[] **readFile**([file], [size])

Reads all or part of the content from a binary file. If a file name is not specified, then a file selection dialog pops up for selecting a file. (Web Enabled)

Parameters
[file]
[size]

Returns
byte[]

Sample

```
// Read all content from the file.
var bytes = plugins.file.readFile('big.jpg');
application.output('file size: ' + bytes.length);
// Read only the first 1KB from the file.
var bytesPartial = plugins.file.readFile('big.jpg', 1024);
application.output('partial file size: ' + bytesPartial.length);
// Read all content from a file selected from the file open dialog.
var bytesUnknownFile = plugins.file.readFile();
application.output('unknown file size: ' + bytesUnknownFile.length);
```

readTXTFile

String **readTXTFile**([file], [charsetname])

Read all content from a text file. If a file name is not specified, then a file selection dialog pops up for selecting a file. The encoding can be also specified. (Web Enabled)

Parameters
[file]
[charsetname]

Returns
String

Sample

```
// Read content from a known text file.
var txt = plugins.file.readTXTFile('story.txt');
application.output(txt);
// Read content from a text file selected from the file open dialog.
var txtUnknown = plugins.file.readTXTFile();
application.output(txtUnknown);
```

showDirectorySelectDialog

JSFile **showDirectorySelectDialog**([directory suggestion], [dialog title text])

Shows a directory selector dialog.

Parameters
[directory suggestion]
[dialog title text]

Returns
JSFile

Sample

```
var dir = plugins.file.showDirectorySelectDialog();
application.output("you've selected folder: " + dir.getAbsolutePath());
```

showFileOpenDialog

Object **showFileOpenDialog**
([selectionMode(0=both,1=Files,2=Dirs)], [startDirectory(null=default/previous)], [multiselect(true/false)], [filterarray], [callbackmethod], [dialog title text])
Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled)

Parameters
[selectionMode(0=both,1=Files,2=Dirs)]
[startDirectory(null=default/previous)]
[multiselect(true/false)]
[filterarray]
[callbackmethod]
[dialog title text]

Returns
Object

Sample

```
// This selects only files ('1'), previous dir must be used ('null'), no multiselect ('false') and
// the filter "JPG and GIF" should be used: ('new Array("JPG and GIF","jpg","gif")').
var file = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF","jpg","gif"));
application.output("you've selected file: " + file.getAbsolutePath());
//for the web you have to give a callback function that has a JSFile array as its first argument (also works in
smart), other options can be set but are not used in the webclient (yet)
var file = plugins.file.showFileOpenDialog(myCallbackMethod)
```

showFileSaveDialog

JSFile **showFileSaveDialog**([fileName/dir suggestion], [dialog title text])
Shows a file save dialog.

Parameters
[fileName/dir suggestion]
[dialog title text]

Returns
JSFile

Sample

```
var file = plugins.file.showFileSaveDialog();
application.output("you've selected file: " + file.getAbsolutePath());
```

streamFilesFromServer

void **streamFilesFromServer**(file/fileName|fileArray/fileNameArray, serverFile/serverFileName|serverFileArray/serverFileNameArray, [callbackFunction])
Streams a file or an array of files from the server in a background task to a file (or files) on the client. If provided, calls back a Servoy function when done

Parameters
file/fileName|fileArray/fileNameArray
serverFile/serverFileName|serverFileArray/serverFileNameArray
[callbackFunction]

Returns
void

Sample

```
// transfer the first file of the default server folder to a chosen file on the client
	var dir = plugins.file.getDesktopFolder();
	var file = plugins.file.showFileSaveDialog(dir,'Save file to');
	if (file) {
		var list = plugins.file.getRemoteList('/', true);
		if (list && list.length > 0) {
			plugins.file.streamFilesFromServer(file, list[0], callbackFunction);
		}
	}
```

streamFilesToServer

void **streamFilesToServer**(file/fileName|fileArray/fileNameArray, [serverFile/serverFileName|serverFileArray/serverFileNameArray], [callbackFunction])
Streams a file or an array of files to the server in a background task - with optional relative path(s)/(new) name(s). If provided, calls back a Servoy function when done

Parameters
file/fileName|fileArray/fileNameArray
[serverFile/serverFileName|serverFileArray/serverFileNameArray]
[callbackFunction]

Returns
void

Sample

```
// send one file:
        var file = plugins.file.showFileOpenDialog( 1, null, false, null, null, 'Choose a file to transfer' );
        if (file) {
                plugins.file.streamFilesToServer( file, callbackFunction );
        }
        // send an array of files:
        var folder = plugins.file.showDirectorySelectDialog();
        if (folder) {
                var files = plugins.file.getFolderContents(folder);
                if (files) {
                        plugins.file.streamFilesToServer( files, callbackFunction );
                }
        }
```

writeFile

Boolean **writeFile**(file, binary_data, [mimeType])

Writes data into a binary file. (Web Enabled)

Parameters

file
binary_data
[mimeType]

Returns

Boolean

Sample

```
var bytes = new Array();
        for (var i=0; i<1024; i++)
                bytes[i] = i % 100;
        var f = plugins.file.convertToJSFile('bin.dat');
        if (!plugins.file.writeFile(f, bytes))
                application.output('Failed to write the file.');
        // mimeType variable can be left null, and is used for webclient only. Specify one of any valid mime
types as referenced here: http://www.w3schools.com/media/media_mimeref.asp'
        var mimeType = 'application/vnd.ms-excel'
        if (!plugins.file.writeFile(f, bytes, mimeType))
                application.output('Failed to write the file.');
```

writeTXTFile

Boolean **writeTXTFile**(file, text_data, [charsetname], [mimeType])

Writes data into a text file. (Web Enabled)

Parameters

file
text_data
[charsetname]
[mimeType]

Returns

Boolean

Sample

```
var fileNameSuggestion = 'myspecialexport.tab'
        var textData = 'load of data...'
        var success = plugins.file.writeTXTFile(fileNameSuggestion, textData);
        if (!success) application.output('Could not write file.');
        // For file-encoding parameter options (default OS encoding is used), http://java.sun.com/j2se/1.4.2
/docs/guide/intl/encoding.doc.html
        // mimeType variable can be left null, and is used for webclient only. Specify one of any valid mime
types as referenced here: http://www.w3schools.com/media/media_mimeref.asp'
```

writeXMLFile

Boolean **writeXMLFile**(file, xml_data)

Writes data into an XML file. The file is saved with the encoding specified by the XML itself. (Web Enabled)

Parameters

file
xml_data

Returns
Boolean
Sample

```
var fileName = 'form.xml'
var xml = controller.printXML()
var success = plugins.file.writeXMLFile(fileName, xml);
if (!success) application.output('Could not write file.');
```