file

Apr 07, 2024 13:22

Supported Clients

SmartClient WebClient NGClient

Methods Summary

Boolean	appendToTXTFile(file, text)	Appends a string given in parameter to a file, using default platform encoding.
Boolean	appendToTXTFile(file, text, encoding)	Appends a string given in parameter to a file, using the specified encoding.
Boolean	appendToTXTFile(file, text)	Appends a string given in parameter to a file, using default platform encoding.
Boolean	appendToTXTFile(file, text, encoding)	Appends a string given in parameter to a file, using the specified encoding.
JSFile	convertToJSFile(file)	Returns a JSFile instance corresponding to an alternative representation of a file (for example a string).
JSFile	convertToRemoteJSFile(path)	Convenience return to get a JSFile representation of a server file based on its path.
Boolean	copyFile(source, destination)	Copies the source file to the destination file.
Boolean	copyFolder(source, destination)	Copies the sourcefolder to the destination folder, recursively.
JSFile	createFile(targetFile)	Creates a JSFile instance.
Boolean	createFolder(destination)	Creates the folder by the given pathname, including anynecessary but nonexistent parent folders.
JSFile	createTempFile(prefix, suffix)	Creates a temporary file on disk.
Boolean	deleteFile(destination)	Removes a file from disk.
Boolean	deleteFolder(destination, showWarning)	Deletes a folder from disk recursively.
String	getDefaultUploadLocation()	Returns the default upload location path of the server.
JSFile	getDesktopFolder()	Returns a JSFile instance that corresponds to the Desktop folder of the currently logged in user.
Array	getDiskList()	Returns an Array of JSFile instances correponding to the file system root folders.
Number	getFileSize(fileOrPath)	Returns the size of the specified file.
Array	getFolderContents(targetFolder)	Returns an array of JSFile instances corresponding to content of the specified folder.
Array	getFolderContents(targetFolder, fileFilter)	Returns an array of JSFile instances corresponding to content of the specified folder.
Array	getFolderContents(targetFolder, fileFilter, fileOption)	Returns an array of JSFile instances corresponding to content of the specified folder.
Array	getFolderContents(targetFolder, fileFilter, fileOption, visibleOption)	Returns an array of JSFile instances corresponding to content of the specified folder.
Array	getFolderContents(targetFolder, fileFilter, fileOption, visibleOption, lockedOption)	Returns an array of JSFile instances corresponding to content of the specified folder.
Array	getFolderContents(targetFolder)	Returns an array of JSFile instances corresponding to content of the specified folder.
Array	getFolderContents(targetFolder, fileFilter)	Returns an array of JSFile instances corresponding to content of the specified folder.
Array	getFolderContents(targetFolder, fileFilter, fileOption)	Returns an array of JSFile instances corresponding to content of the specified folder.
Array	getFolderContents(targetFolder, fileFilter, fileOption, visibleOption)	Returns an array of JSFile instances corresponding to content of the specified folder.
Array	getFolderContents(targetFolder, fileFilter, fileOption, visibleOption, lockedOption)	Returns an array of JSFile instances corresponding to content of the specified folder.
JSFile	getHomeFolder()	Returns a JSFile instance corresponding to the home folder of the logged in used.
Date	getModificationDate(fileOrPath)	Returns the modification date of a file.
Array	getRemoteFolderContents(targetFolder)	Returns an array of JSFile instances corresponding to content of the specified folder on the server side.
Array	getRemoteFolderContents(targetFolder, fileFilter)	Returns an array of JSFile instances corresponding to content of the specified folder on the server side.
Array	getRemoteFolderContents(targetFolder, fileFilter, fileOption)	Returns an array of JSFile instances corresponding to content of the specified folder on the server side.
Array	getRemoteFolderContents(targetFolder, fileFilter, fileOption, visibleOption)	Returns an array of JSFile instances corresponding to content of the specified folder on the server side.
Array	getRemoteFolderContents(targetFolder, fileFilter, fileOption, visibleOption, lockedOption)	Returns an array of JSFile instances corresponding to content of the specified folder on the server side.
Array	getRemoteFolderContents(targetFolder)	Returns an array of JSFile instances corresponding to content of the specified folder on the server side.
Array	getRemoteFolderContents(targetFolder, fileFilter)	Returns an array of JSFile instances corresponding to content of the specified folder on the server side.
Array	getRemoteFolderContents(targetFolder, fileFilter, fileOption)	Returns an array of JSFile instances corresponding to content of the specified folder on the server side.
Array	getRemoteFolderContents(targetFolder, fileFilter, fileOption, visibleOption)	Returns an array of JSFile instances corresponding to content of the specified folder on the server side.
Array	getRemoteFolderContents(targetFolder, fileFilter, fileOption, visibleOption, lockedOption)	Returns an array of JSFile instances corresponding to content of the specified folder on the server side.
String	getUrlForRemoteFile(file)	Get a url from a remote file that can be used to download the file in a browser.
String	getUrlForRemoteFile(file)	Get a url from a remote file that can be used to download the file in a browser.

Boolean	moveFile(source, destination)	Moves the file from the source to the destination place.
Boolean	openFile(file)	Opens the given local file.
Boolean	openFile(file, webClientTarget, webClientTargetOptions)	Opens the given local file.
Boolean	openFile(fileName, data, mimeType)	Opens the given data as a file.
Boolean	openFile(fileName, data, mimeType, webClientTarget, webClientTargetOptions)	Opens the given data as a file.
Array	readFile()	Reads all or part of the content from a binary file.
Array	readFile(file)	Reads all or part of the content from a binary file.
Array	readFile(file, size)	Reads all or part of the content from a binary file.
Array	readFile(file)	Reads all or part of the content from a binary file.
Array	readFile(file, size)	Reads all or part of the content from a binary file.
String	readTXTFile()	Read all content from a text file.
String	readTXTFile(file)	Read all content from a text file.
String	readTXTFile(file, charsetname)	Read all content from a text file.
String	readTXTFile(file)	Read all content from a text file.
String	readTXTFile(file, charsetname)	Read all content from a text file.
JSFile	showDirectorySelectDialog()	Shows a directory selector dialog.
JSFile	showDirectorySelectDialog(directory)	Shows a directory selector dialog.
JSFile	showDirectorySelectDialog(directory, title)	Shows a directory selector dialog.
JSFile	showDirectorySelectDialog(directory)	Shows a directory selector dialog.
JSFile	showDirectorySelectDialog(directory, title)	Shows a directory selector dialog.
Object	showFileOpenDialog()	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory, multiselect)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory, multiselect, filter)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory, multiselect, filter, callbackfunction)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory, multiselect, filter, callbackfunction, title)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory, multiselect, callbackfunction)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory, callbackfunction)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory, multiselect)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory, multiselect, filter)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory, multiselect, filter, callbackfunction)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory, multiselect, filter, callbackfunction, title)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory, multiselect, callbackfunction)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, startDirectory, callbackfunction)	Shows a file open dialog.
Object	showFileOpenDialog(selectionMode, callbackfunction)	Shows a file open dialog.
Object	showFileOpenDialog(callbackfunction)	Shows a file open dialog.
JSFile	showFileSaveDialog()	Shows a file save dialog.
JSFile	showFileSaveDialog(fileNameDir)	Shows a file save dialog.
JSFile	showFileSaveDialog(fileNameDir, title)	Shows a file save dialog.
JSFile	showFileSaveDialog(fileNameDir)	Shows a file save dialog.
JSFile	showFileSaveDialog(fileNameDir, title)	Shows a file save dialog.
void	streamFile(file)	Stream the given file(path) to the browser with content-disposition:attachment This will not load in the file fully into memory but only stream it right from disk.
void	streamFile(file, contentDisposition)	String the given file(path) to the browser you can provide the content disposition how this should be send (inline or as an attachment) This will not load in the file fully into memory but only stream it right from disk.
void	streamFile(file, contentDisposition, browserTarget)	String the given file(path) to the browser you can provide the content disposition how this should be send (inline or as an attachment) This will not load in the file fully into memory but only stream it right from disk.
JSProgressMonitor	streamFilesFromServer(files, serverFiles)	Stream 1 or more files from the server to the client.

JSProgressMonitor	streamFilesFromServer(files, serverFiles, callback)	Stream 1 or more files from the server to the client, the callback method is invoked after every file, with as argument the filename that was transfered.
JSProgressMonitor	streamFilesToServer(files)	Overloaded method, only defines file(s) to be streamed
JSProgressMonitor	streamFilesToServer(files, serverFiles)	Overloaded method, defines file(s) to be streamed and a callback function
JSProgressMonitor	streamFilesToServer(files, serverFiles, callback)	Overloaded method, defines file(s) to be streamed, a callback function and file name(s) to use on the server
JSProgressMonitor	streamFilesToServer(files, callback)	Overloaded method, defines file(s) to be streamed and a callback function
void	trackFileForDeletion(file)	If the client's solution is closed, the file given to this method wil be deleted.
Boolean	writeFile(file, data)	Writes the given file to disk.
Boolean	writeFile(file, data, mimeType)	Writes the given file to disk.
Boolean	writeFile(file, data)	Writes the given file to disk.
Boolean	writeFile(file, data, mimeType)	Writes the given file to disk.
Boolean	writeTXTFile(file, text_data)	Writes data into a text file.
Boolean	writeTXTFile(file, text_data, charsetname)	Writes data into a text file.
Boolean	writeTXTFile(file, text_data, charsetname, mimeType)	Writes data into a text file.
Boolean	writeTXTFile(file, text_data)	Writes data into a text file.
Boolean	writeTXTFile(file, text_data, charsetname)	Writes data into a text 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.
Boolean	writeXMLFile(file, xml_data, encoding)	Writes data into an XML file.
Boolean	writeXMLFile(file, xml_data)	Writes data into an XML file.
Boolean	writeXMLFile(file, xml_data, encoding)	Writes data into an XML file.

Methods Details

appendToTXTFile(file, text)

Appends a string given in parameter to a file, using default platform encoding.

Parameters

JSFile file a local JSFile
String text the text to append to the file

Returns

Boolean true if appending worked

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

appendToTXTFile(file, text, encoding)

Appends a string given in parameter to a file, using the specified encoding.

Parameters

JSFile file a local JSFile
String text the text to append to the file
String encoding the encoding to use

Returns

Boolean true if appending worked

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

appendToTXTFile(file, text)

Appends a string given in parameter to a file, using default platform encoding.

Parameters

String file the file path as a String
String text the text to append to the file

Returns

Boolean true if appending worked

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

appendToTXTFile(file, text, encoding)

Appends a string given in parameter to a file, using the specified encoding.

Parameters

String file the file path as a String
String text the text to append to the file
String encoding the encoding to use

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

convertToJSFile(file)

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

Parameters

Object file ;

Returns

JSFile JSFile

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

convertToRemoteJSFile(path)

Convenience return to get a JSFile representation of a server file based on its path.

Parameters

String path the path representing a file on the server (should start with "/")

Returns

JSFile the JSFile

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

copyFile(source, destination)

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

Parameters

Object source ;
Object destination ;

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

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(source, destination)

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

Parameters

Object source ;
Object destination ;

Returns

Boolean success boolean

Supported Clients

SmartClient,WebClient,NGClient

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(targetFile)

Creates a JSFile instance. Does not create the file on disk.

Parameters

Object targetFile ;

Returns

JSFile

Supported Clients

SmartClient,WebClient,NGClient

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(destination)

Creates the folder by the given pathname, including anynecessary but nonexistent parent folders.
Note that if this operation fails it may have succeeded in creating some of the necessary parent folders.
Will return true if it could make this folder or if the folder did already exist.

Parameters

Object destination ;

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

createTempFile(prefix, suffix)

Creates a temporary file on disk. A prefix and an extension are specified and they will be part of the file name.

Parameters

String prefix ;
String suffix ;

Returns

JSFile

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

deleteFile(destination)

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

Parameters

Object destination ;

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

//In case the file to delete is a remote file:
var file = plugins.file.convertToRemoteJSFile('/story.txt');
plugins.file.deleteFile(file);
```

deleteFolder(destination, 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

Object destination ;
Boolean showWarning ;

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

//In case the file to delete is a remote folder:
plugins.file.deleteFolder(plugins.file.convertToRemoteJSFile('/stories'), true);
```

getDefaultUploadLocation()

Returns the default upload location path of the server.

Returns

String the location as canonical path

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// get the (server-side) default upload location path:
var serverPath = plugins.file.getDefaultUploadLocation();
```

getDesktopFolder()

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

Returns

JSFile

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getDiskList()

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

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getFileSize(fileOrPath)

Returns the size of the specified file.

Parameters

Object fileOrPath can be a (remote) JSFile or a local file path

Returns

Number

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

//In case the file is remote, located on the server side inside the default upload folder:
var f = plugins.file.convertToRemoteJSFile('/story.txt');
application.output('file size: ' + plugins.file.getFileSize(f));
```

getFolderContents(targetFolder)

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

JSFile targetFolder JSFile object.

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getFolderContents(targetFolder, fileFilter)

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

JSFile targetFolder JSFile object.
Object fileFilter Filter or array of filters for files in folder.

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getFolderContents(targetFolder, fileFilter, fileOption)

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

JSFile targetFolder JSFile object.
Object fileFilter Filter or array of filters for files in folder.
Number fileOption 1=files, 2=dirs

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getFolderContents(targetFolder, fileFilter, fileOption, visibleOption)

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

JSFile	targetFolder	JSFile object.
Object	fileFilter	Filter or array of filters for files in folder.
Number	fileOption	1=files, 2=dirs
Number	visibleOption	1=visible, 2=nonvisible

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getFolderContents(targetFolder, fileFilter, fileOption, visibleOption, lockedOption)

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

JSFile	targetFolder	JSFile object.
Object	fileFilter	Filter or array of filters for files in folder.
Number	fileOption	1=files, 2=dirs
Number	visibleOption	1=visible, 2=nonvisible
Number	lockedOption	1=locked, 2=nonlocked

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getFolderContents(targetFolder)

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

String	targetFolder	File path.

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getFolderContents(targetFolder, fileFilter)

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

String	targetFolder	File path.
Object	fileFilter	Filter or array of filters for files in folder.

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getFolderContents(targetFolder, fileFilter, fileOption)

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

String	targetFolder	File path.
Object	fileFilter	Filter or array of filters for files in folder.
Number	fileOption	1=files, 2=dirs

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getFolderContents(targetFolder, fileFilter, fileOption, visibleOption)

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

String	targetFolder	File path.
Object	fileFilter	Filter or array of filters for files in folder.
Number	fileOption	1=files, 2=dirs
Number	visibleOption	1=visible, 2=nonvisible

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getFolderContents(targetFolder, fileFilter, fileOption, visibleOption, lockedOption)

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

String	targetFolder	File path.
Object	fileFilter	Filter or array of filters for files in folder.
Number	fileOption	1=files, 2=dirs
Number	visibleOption	1=visible, 2=nonvisible
Number	lockedOption	1=locked, 2=nonlocked

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getHomeFolder()

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

Returns

JSFile

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

getModificationDate(fileOrPath)

Returns the modification date of a file.

Parameters

Object fileOrPath can be a (remote) JSFile or a local file path

Returns

Date

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

//In case the file is remote, located on the server side inside the default upload folder:
var f = plugins.file.convertToRemoteJSFile('/story.txt');
application.output('file size: ' + plugins.file.getModificationDate(f));
```

getRemoteFolderContents(targetFolder)

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

Parameters

JSFile targetFolder ;

Returns

Array the array of file names

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// retrieves an array of files located on the server side inside the default upload folder:
var files = plugins.file.getRemoteFolderContents(plugins.file.convertToRemoteJSFile('/'), '.txt');
```

getRemoteFolderContents(targetFolder, fileFilter)

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

Parameters

JSFile targetFolder Folder as JSFile object.
Object fileFilter Filter or array of filters for files in folder.

Returns

Array the array of file names

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// retrieves an array of files located on the server side inside the default upload folder:
var files = plugins.file.getRemoteFolderContents(plugins.file.convertToRemoteJSFile('/'), '.txt');
```

getRemoteFolderContents(targetFolder, fileFilter, fileOption)

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

Parameters

JSFile targetFolder Folder as JSFile object.
Object fileFilter Filter or array of filters for files in folder.
Number fileOption 1=files, 2=dirs

Returns

Array the array of file names

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// retrieves an array of files located on the server side inside the default upload folder:
var files = plugins.file.getRemoteFolderContents(plugins.file.convertToRemoteJSFile('/'), '.txt');
```

getRemoteFolderContents(targetFolder, fileFilter, fileOption, visibleOption)

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

Parameters

JSFile targetFolder Folder as JSFile object.
Object fileFilter Filter or array of filters for files in folder.
Number fileOption 1=files, 2=dirs
Number visibleOption 1=visible, 2=nonvisible

Returns

Array the array of file names

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// retrieves an array of files located on the server side inside the default upload folder:
var files = plugins.file.getRemoteFolderContents(plugins.file.convertToRemoteJSFile('/'), '.txt');
```

getRemoteFolderContents(targetFolder, fileFilter, fileOption, visibleOption, lockedOption)

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

Parameters

JSFile	targetFolder	Folder as JSFile object.
Object	fileFilter	Filter or array of filters for files in folder.
Number	fileOption	1=files, 2=dirs
Number	visibleOption	1=visible, 2=nonvisible
Number	lockedOption	1=locked, 2=nonlocked

Returns

Array the array of file names

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// retrieves an array of files located on the server side inside the default upload folder:
var files = plugins.file.getRemoteFolderContents(plugins.file.convertToRemoteJSFile('/'), '.txt');
```

getRemoteFolderContents(targetFolder)

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

Parameters

String targetFolder ;

Returns

Array the array of file names

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// retrieves an array of files located on the server side inside the default upload folder:
var files = plugins.file.getRemoteFolderContents('/', '.txt');
```

getRemoteFolderContents(targetFolder, fileFilter)

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

Parameters

String	targetFolder	Folder path.
Object	fileFilter	Filter or array of filters for files in folder.

Returns

Array the array of file names

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// retrieves an array of files located on the server side inside the default upload folder:
var files = plugins.file.getRemoteFolderContents('/', '.txt');
```

getRemoteFolderContents(targetFolder, fileFilter, fileOption)

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

Parameters

String	targetFolder	Folder path.
Object	fileFilter	Filter or array of filters for files in folder.
Number	fileOption	1=files, 2=dirs

Returns

Array the array of file names

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// retrieves an array of files located on the server side inside the default upload folder:
var files = plugins.file.getRemoteFolderContents('/', '.txt');
```

getRemoteFolderContents(targetFolder, fileFilter, fileOption, visibleOption)

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

Parameters

String	targetFolder	Folder path.
Object	fileFilter	Filter or array of filters for files in folder.
Number	fileOption	1=files, 2=dirs
Number	visibleOption	1=visible, 2=nonvisible

Returns

Array the array of file names

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// retrieves an array of files located on the server side inside the default upload folder:
var files = plugins.file.getRemoteFolderContents('/', '.txt');
```

getRemoteFolderContents(targetFolder, fileFilter, fileOption, visibleOption, lockedOption)

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

Parameters

String	targetFolder	Folder path.
Object	fileFilter	Filter or array of filters for files in folder.
Number	fileOption	1=files, 2=dirs
Number	visibleOption	1=visible, 2=nonvisible
Number	lockedOption	1=locked, 2=nonlocked

Returns

Array the array of file names

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// retrieves an array of files located on the server side inside the default upload folder:
var files = plugins.file.getRemoteFolderContents('/', '.txt');
```

getUrlForRemoteFile(file)

Get a url from a remote file that can be used to download the file in a browser.
This is a complete url with the server url that is get from application.getServerURL()

Parameters

JSFile file the remote file where the url should be generated from. Must be a remote file

Returns

String the url as a string

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
var file = plugins.file.convertToRemoteJSFile("aremotefile.pdf");
var url = plugins.file.getUrlForRemoteFile(file);
application.showURL(url);
```

getUrlForRemoteFile(file)

Get a url from a remote file that can be used to download the file in a browser.
This is a complete url with the server url that is get from application.getServerURL()

Parameters

String file the remote file where the url should be generated from. Must be a remote file

Returns

String the url as a string

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
var file = plugins.file.convertToRemoteJSFile("aremotefile.pdf");
var url = plugins.file.getUrlForRemoteFile(file);
application.showURL(url);
```

moveFile(source, destination)

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

Parameters

Object source ;
Object destination ;

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

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.');
```

openFile(file)

Opens the given local file.

Smart Client: launches the default OS associated application to open an existing local file.
Web Client: the (server local) file will open inside the browser - if supported (sent using "Content-disposition: inline" HTTP header).

Parameters

JSFile file the local file to open. The file should exist and be accessible.

Returns

Boolean success status of the open operation

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
var myPDF = plugins.file.createFile('my.pdf');
 myPDF.setBytes(data, true)
	plugins.file.openFile(myPDF);
```

openFile(file, webClientTarget, webClientTargetOptions)

Opens the given local file.

Smart Client: launches the default OS associated application to open an existing local file.
Web Client: the (server local) file will open inside the browser - if supported (sent using "Content-disposition: inline" HTTP header).

Parameters

JSFile	file	the local file to open. The file should exist and be accessible.
String	webClientTarget	Target frame or named dialog/window. For example "_self" to open in the same browser window, "_blank" for another browser window. By default "_blank" is used.
String	webClientTargetOptions	window options used when a new browser window is to be shown; see browser JS 'window.open(...)' documentation.

Returns

Boolean success status of the open operation

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
var myPDF = plugins.file.createFile('my.pdf');
 myPDF.setBytes(data, true)
        plugins.file.openFile(myPDF, "_self", null); // show in the same browser window
```

openFile(fileName, data, mimeType)

Opens the given data as a file.

Smart Client: writes the data to a temporary file, then launches the default OS associated application to open it.
Web Client: the data will open as a file inside the browser - if supported (sent using "Content-disposition: inline" HTTP header).

Parameters

String	fileName	the name of the file that should open with the given data. Can be null (but in Smart Client null - so no extension - will probably make open fail).
Array	data	the file's binary content.
String	mimeType	can be left null, and is used for webclient only. Specify one of any valid mime types: https://developer.mozilla.org/en-US/docs/Properly_Configuring_Server_MIME_Types http://www.iana.org/assignments/media-types/media-types.xhtml http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.7

Returns

Boolean success status of the open operation

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// read or generate pdf file bytes
var bytes = plugins.file.readFile("c:/ExportedPdfs/13542.pdf");

// mimeType variable can be left null
var mimeType = 'application/pdf'

if (!plugins.file.openFile("MonthlyStatistics.pdf", bytes, mimeType))
        application.output('Failed to open the file.');
```

openFile(fileName, data, mimeType, webClientTarget, webClientTargetOptions)

Opens the given data as a file.

Smart Client: writes the data to a temporary file, then launches the default OS associated application to open it.
Web Client: the data will open as a file inside the browser - if supported (sent using "Content-disposition: inline" HTTP header).

Parameters

String	fileName	the name of the file that should open with the given data. Can be null (but in Smart Client null - so no extension - will probably make open fail).
Array	data	the file's binary content.
String	mimeType	can be left null, and is used for webclient only. Specify one of any valid mime types: https://developer.mozilla.org/en-US/docs/Properly_Configuring_Server_MIME_Types http://www.iana.org/assignments/media-types/media-types.xhtml http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.7
String	webClientTarget	Target frame or named dialog/window. For example "_self" to open in the same browser window, "_blank" for another browser window. By default "_blank" is used.
String	webClientTargetOptions	window options used when a new browser window is to be shown; see browser JS 'window.open(...)' documentation.

Returns

Boolean success status of the open operation

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// read or generate pdf file bytes
var bytes = plugins.file.readFile("c:/ExportedPdfs/13542.pdf");

// mimeType variable can be left null
var mimeType = 'application/pdf'

if (!plugins.file.openFile("MonthlyStatistics.pdf", bytes, mimeType, "_self", null))
        application.output('Failed to open the file.');
```

readFile()

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 only for a JSFile argument)

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

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);
```

readFile(file)

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 only for a JSFile argument)

Parameters

JSFile file JSFile.

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

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);
```

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 only for a JSFile argument)

Parameters

JSFile file JSFile.
Number size Number of bytes to read.

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

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);
```

readFile(file)

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 only for a JSFile argument)

Parameters

String file the file path.

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

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);
```

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 only for a JSFile argument)

Parameters

String file the file path.
Number size Number of bytes to read.

Returns

Array

Supported Clients

SmartClient,WebClient,NGClient

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()

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 only for a JSFile argument)

Returns

String

Supported Clients

SmartClient,WebClient,NGClient

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);
```

readTXTFile(file)

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 only for a JSFile argument)

Parameters

JSFile file JSFile.

Returns

String

Supported Clients

SmartClient,WebClient,NGClient

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);
```

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 only for a JSFile argument)

Parameters

JSFile file JSFile.
String charsetname Charset name.

Returns

String

Supported Clients

SmartClient,WebClient,NGClient

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);
```

readTXTFile(file)

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 only for a JSFile argument)

Parameters

String file the file path.

Returns

String

Supported Clients

SmartClient,WebClient,NGClient

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);
```

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 only for a JSFile argument)

Parameters

String file the file path.
String charsetname Charset name.

Returns

String

Supported Clients

SmartClient,WebClient,NGClient

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()

Shows a directory selector dialog.

Returns

JSFile

Supported Clients

SmartClient

Sample

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

showDirectorySelectDialog(directory)

Shows a directory selector dialog.

Parameters

JSFile directory Default directory as JSFile.

Returns

JSFile

Supported Clients

SmartClient

Sample

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

showDirectorySelectDialog(directory, title)

Shows a directory selector dialog.

Parameters

JSFile directory Default directory as JSFile.
String title Dialog title.

Returns

JSFile

Supported Clients

SmartClient

Sample

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

showDirectorySelectDialog(directory)

Shows a directory selector dialog.

Parameters

String directory Default directory as file path.

Returns

JSFile

Supported Clients

SmartClient

Sample

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

showDirectorySelectDialog(directory, title)

Shows a directory selector dialog.

Parameters

String directory Default directory as file path.
String title Dialog title.

Returns

JSFile

Supported Clients

SmartClient

Sample

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

showFileOpenDialog()

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Returns

Object

Supported Clients

SmartClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number selectionMode 0=both,1=Files,2=Dirs

Returns

Object

Supported Clients

SmartClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
JSFile	startDirectory	JSFile instance of default folder; null=default/previous

Returns

Object

Supported Clients

SmartClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory, multiselect)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
JSFile	startDirectory	JSFile instance of default folder, null=default/previous
Boolean	multiselect	true/false

Returns

Object

Supported Clients

SmartClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory, multiselect, filter)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
JSFile	startDirectory	JSFile instance of default folder,null=default/previous
Boolean	multiselect	true/false
Object	filter	A filter or array of filters on the folder files.

Returns

Object

Supported Clients

SmartClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory, multiselect, filter, callbackfunction)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
JSFile	startDirectory	JSFile instance of default folder,null=default/previous
Boolean	multiselect	true/false
Object	filter	A filter or array of filters on the folder files.
Function	callbackfunction	A function that takes the (JSFile) array of the selected files as first argument

Returns

Object

Supported Clients

SmartClient,WebClient,NGClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory, multiselect, filter, callbackfunction, title)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
JSFile	startDirectory	JSFile instance of default folder, null=default/previous
Boolean	multiselect	true/false
Object	filter	A filter or array of filters on the folder files.
Function	callbackfunction	A function that takes the (JSFile) array of the selected files as first argument
String	title	The tile of the dialog

Returns

Object

Supported Clients

SmartClient,WebClient,NGClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory, multiselect, callbackfunction)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
JSFile	startDirectory	JSFile instance of default folder,null=default/previous
Boolean	multiselect	true/false
Function	callbackfunction	A function that takes the (JSFile) array of the selected files as first argument

Returns

Object

Supported Clients

SmartClient,WebClient,NGClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory, callbackfunction)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
JSFile	startDirectory	JSFile instance of default folder,null=default/previous
Function	callbackfunction	A function that takes the (JSFile) array of the selected files as first argument

Returns

Object

Supported Clients

SmartClient,WebClient,NGClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
String	startDirectory	Path to default folder; null=default/previous

Returns

Object

Supported Clients

SmartClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory, multiselect)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
String	startDirectory	Path to default folder, null=default/previous
Boolean	multiselect	true/false

Returns

Object

Supported Clients

SmartClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory, multiselect, filter)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
String	startDirectory	Path to default folder,null=default/previous
Boolean	multiselect	true/false
Object	filter	A filter or array of filters on the folder files.

Returns

Object

Supported Clients

SmartClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory, multiselect, filter, callbackfunction)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
String	startDirectory	Path to default folder,null=default/previous
Boolean	multiselect	true/false
Object	filter	A filter or array of filters on the folder files.
Function	callbackfunction	A function that takes the (JSFile) array of the selected files as first argument

Returns

Object

Supported Clients

SmartClient,WebClient,NGClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory, multiselect, filter, callbackfunction, title)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
String	startDirectory	Path to default folder, null=default/previous
Boolean	multiselect	true/false
Object	filter	A filter or array of filters on the folder files.
Function	callbackfunction	A function that takes the (JSFile) array of the selected files as first argument
String	title	The tile of the dialog

Returns

Object

Supported Clients

SmartClient,WebClient,NGClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory, multiselect, callbackfunction)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
String	startDirectory	Path to default folder,null=default/previous
Boolean	multiselect	true/false
Function	callbackfunction	A function that takes the (JSFile) array of the selected files as first argument

Returns

Object

Supported Clients

SmartClient,WebClient,NGClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, startDirectory, callbackfunction)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number	selectionMode	0=both,1=Files,2=Dirs
String	startDirectory	Path to default folder,null=default/previous
Function	callbackfunction	A function that takes the (JSFile) array of the selected files as first argument

Returns

Object

Supported Clients

SmartClient,WebClient,NGClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(selectionMode, callbackfunction)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Number selectionMode 0=both,1=Files,2=Dirs
Function callbackfunction A function that takes the (JSFile) array of the selected files as first argument

Returns

Object

Supported Clients

SmartClient,WebClient,NGClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileOpenDialog(callbackfunction)

Shows a file open dialog. Filters can be applied on what type of files can be selected. (Web Enabled, you must set the callback method for this to work)

Parameters

Function callbackfunction A function that takes the (JSFile) array of the selected files as first argument

Returns

Object

Supported Clients

SmartClient,WebClient,NGClient

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")').
/** @type {JSFile} */
var f = plugins.file.showFileOpenDialog(1, null, false, new Array("JPG and GIF", "jpg", "gif"));
application.output('File: ' + f.getName());
application.output('is dir: ' + f.isDirectory());
application.output('is file: ' + f.isFile());
application.output('path: ' + f.getAbsolutePath());

// This allows mutliple selection of files, using previous dir and the same filter as above. This also casts the
result to the JSFile type using JSDoc.
// if filters are specified, "all file" filter will not show up unless "*" filter is present
/** @type {JSFile[]} */
var files = plugins.file.showFileOpenDialog(1, null, true, new Array("JPG and GIF", "jpg", "gif", "*"));
for (var i = 0; i < files.length; i++)
{
        application.output('File: ' + files[i].getName());
        application.output('content type: ' + files[i].getContentType());
        application.output('last modified: ' + files[i].lastModified());
        application.output('size: ' + files[i].size());
}
//for the web and NG you have to give a callback function that has a JSFile array as its first argument (also
works in smart), only multi select and the title are used in the webclient, others are ignored
plugins.file.showFileOpenDialog(null,null,false,new Array("JPG and GIF", "jpg", "gif"),
mycallbackfunction,'Select some nice files')

When handling big files please look at the admin page properties: "servoy.ng_web_client.tempfile.threshold" and
"servoy.ng_web_client.temp.uploadir", so that big files are mapped to temp files and saved to a good temp dir
so that in the callback method you can try to rename the temp generated file to something on the filesystem with
a specific name. This way there is no need to stream anything again on the server side
(or access the bytes which will load the big file completely in memory)
```

showFileSaveDialog()

Shows a file save dialog. File save is only supported in the SmartClient.

Returns

JSFile

Supported Clients

SmartClient

Sample

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

showFileSaveDialog(fileNameDir)

Shows a file save dialog. File save is only supported in the SmartClient.

Parameters

JSFile fileNameDir JSFile to save.

Returns

JSFile

Supported Clients

SmartClient

Sample

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

showFileSaveDialog(fileNameDir, title)

Shows a file save dialog. File save is only supported in the SmartClient.

Parameters

JSFile fileNameDir JSFile to save
String title Dialog title.

Returns

JSFile

Supported Clients

SmartClient

Sample

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

showFileSaveDialog(fileNameDir)

Shows a file save dialog. File save is only supported in the SmartClient.

Parameters

String fileNameDir File (give as file path) to save.

Returns

JSFile

Supported Clients

SmartClient

Sample

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

showFileSaveDialog(fileNameDir, title)

Shows a file save dialog. File save is only supported in the SmartClient.

Parameters

String fileNameDir File to save (specified as file path)
String title Dialog title.

Returns

JSFile

Supported Clients

SmartClient

Sample

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

streamFile(file)

Stream the given file(path) to the browser with content-disposition:attachment
This will not load in the file fully into memory but only stream it right from disk.
This can be any filepath on the server, but only the simple file name is exposed as the content disposition header filename.
This will throw an exception if anything goes wrong, like the given file is not valid/found.

Parameters

Object file A path string, JSFile or RemoteFile

Supported Clients

NGClient

Sample

```

```

streamFile(file, contentDisposition)

String the given file(path) to the browser you can provide the content disposition how this should be send (inline or as an attachment)
This will not load in the file fully into memory but only stream it right from disk.
This can be any filepath on the server, but only the simple file name is exposed.
This will throw an exception if anything goes wrong, like the given file is not valid/found.

Parameters

Object	file	A path string, JSFile or RemoteFile
String	contentDisposition	can be 'inline' or 'attachment'

Supported Clients

NGClient

Sample

streamFile(file, contentDisposition, browserTarget)

String the given file(path) to the browser you can provide the content disposition how this should be send (inline or as an attachment)
This will not load in the file fully into memory but only stream it right from disk.
This can be any filepath on the server, but only the simple file name is exposed.
This will throw an exception if anything goes wrong, like the given file is not valid/found.
Give the browser target if you want to open the file inside another tab, most usefull in 'inline' content disposition mode.

Parameters

Object	file	A path string, JSFile or RemoteFile
String	contentDisposition	can be 'inline' or 'attachment'
String	browserTarget	_blank or a specific name to open this in a differnt tab (really only usefull in inline mode)

Supported Clients

NGClient

Sample

streamFilesFromServer(files, serverFiles)

Stream 1 or more files from the server to the client.

Parameters

Object	files	file(s) to be streamed into (can be a String path a JSFile) or an Array of these
Object	serverFiles	the files on the server that will be transfered to the client, can be a String or a String[]

Returns

JSProgressMonitor a JSProgressMonitor object to allow client to subscribe to progress notifications

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// transfer all the files of a chosen server folder to a directory on the client
var dir = plugins.file.showDirectorySelectDialog();
if (dir) {
	var list = plugins.file.getRemoteFolderContents('/images/user1/', null, 1);
	if (list) {
		var monitor = plugins.file.streamFilesFromServer(dir, list, callbackFunction);
	}
}

// transfer one file on the client
var monitor = plugins.file.streamFilesFromServer('/path/to/file', 'path/to/serverFile', callbackFunction);

// transfer an array of serverFiles to an array of files on the client
var files = new Array();
files[0] = '/path/to/file1';
files[1] = '/path/to/file2';
var serverFiles = new Array();
serverFiles[0] = '/path/to/serverFile1';
serverFiles[1] = '/path/to/serverFile2';
var monitor = plugins.file.streamFilesFromServer(files, serverFiles, callbackFunction);
```

streamFilesFromServer(files, serverFiles, callback)

Stream 1 or more files from the server to the client, the callback method is invoked after every file, with as argument
the filename that was transfered. An extra second exception parameter can be given if an exception did occur.

Parameters

Object	files	file(s) to be streamed into (can be a String path or a JSFile) or an Array of these
Object	serverFiles	the files on the server that will be transfered to the client, can be a JSFile or JSFile[], a String or String[]
Function	callback	the Function to be called back at the end of the process (for every file); the callback function is invoked with argument the filename that was transfered; an extra second exception parameter can be given if an exception occured

Returns

JSProgressMonitor a JSProgressMonitor object to allow client to subscribe to progress notifications

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
// transfer all the files of a chosen server folder to a directory on the client
var dir = plugins.file.showDirectorySelectDialog();
if (dir) {
	var list = plugins.file.getRemoteFolderContents('/images/user1/', null, 1);
	if (list) {
		var monitor = plugins.file.streamFilesFromServer(dir, list, callbackFunction);
	}
}

// transfer one file on the client
var monitor = plugins.file.streamFilesFromServer('/path/to/file', 'path/to/serverFile', callbackFunction);

// transfer an array of serverFiles to an array of files on the client
var files = new Array();
files[0] = '/path/to/file1';
files[1] = '/path/to/file2';
var serverFiles = new Array();
serverFiles[0] = '/path/to/serverFile1';
serverFiles[1] = '/path/to/serverFile2';
var monitor = plugins.file.streamFilesFromServer(files, serverFiles, callbackFunction);
```

streamFilesToServer(files)

Overloaded method, only defines file(s) to be streamed

Parameters

Object	files	file(s) to be streamed (can be a String path or a JSFile) or an Array of these

Returns

JSProgressMonitor a JSProgressMonitor object to allow client to subscribe to progress notifications

Supported Clients

SmartClient,WebClient,NGClient

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 );
}
//plugins.file.streamFilesToServer( 'servoy.txt', callbackFunction );

// send an array of files:
var folder = plugins.file.showDirectorySelectDialog();
if (folder) {
        var files = plugins.file.getFolderContents(folder);
        if (files) {
                var monitor = plugins.file.streamFilesToServer( files, callbackFunction );
        }
}
// var files = new Array()
// files[0] = 'file1.txt';
// files[1] = 'file2.txt';
// var monitor = plugins.file.streamFilesToServer( files, callbackFunction );
```

streamFilesToServer(files, serverFiles)

`Overloaded method, defines file(s) to be streamed and a callback function`

Parameters

Object files file(s) to be streamed (can be a String path or a JSFile) or an Array of these
Object serverFiles can be a JSFile or JSFile[], a String or String[], representing the file name(s) to use on the server

Returns

JSProgressMonitor a JSProgressMonitor object to allow client to subscribe to progress notifications

Supported Clients

SmartClient,WebClient,NGClient

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 );
}
//plugins.file.streamFilesToServer( 'servoy.txt', callbackFunction );

// send an array of files:
var folder = plugins.file.showDirectorySelectDialog();
if (folder) {
        var files = plugins.file.getFolderContents(folder);
        if (files) {
                var monitor = plugins.file.streamFilesToServer( files, callbackFunction );
        }
}
// var files = new Array()
// files[0] = 'file1.txt';
// files[1] = 'file2.txt';
// var monitor = plugins.file.streamFilesToServer( files, callbackFunction );
```

streamFilesToServer(files, serverFiles, callback)

`Overloaded method, defines file(s) to be streamed, a callback function and file name(s) to use on the server`

Parameters

Object files file(s) to be streamed (can be a String path or a JSFile) or an Array of these)
Object serverFiles can be a JSFile or JSFile[], a String or String[], representing the file name(s) to use on the server
Function callback the Function to be called back at the end of the process (for every file); the callback function is invoked with argument the filename that was transfered; an extra second exception parameter can be given if an exception occured

Returns

JSProgressMonitor a JSProgressMonitor object to allow client to subscribe to progress notifications

Supported Clients

SmartClient,WebClient,NGClient

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 );
}
//plugins.file.streamFilesToServer( 'servoy.txt', callbackFunction );

// send an array of files:
var folder = plugins.file.showDirectorySelectDialog();
if (folder) {
	var files = plugins.file.getFolderContents(folder);
	if (files) {
		var monitor = plugins.file.streamFilesToServer( files, callbackFunction );
	}
}
// var files = new Array()
// files[0] = 'file1.txt';
// files[1] = 'file2.txt';
// var monitor = plugins.file.streamFilesToServer( files, callbackFunction );
```

streamFilesToServer(files, callback)

Overloaded method, defines file(s) to be streamed and a callback function

Parameters

Object	files	file(s) to be streamed (can be a String path or a JSFile) or an Array of these
Function	callback	the Function to be called back at the end of the process (for every file); the callback function is invoked with argument the filename that was transfered; an extra second exception parameter can be given if an exception occured

Returns

JSProgressMonitor a JSProgressMonitor object to allow client to subscribe to progress notifications

Supported Clients

SmartClient,WebClient,NGClient

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 );
}
//plugins.file.streamFilesToServer( 'servoy.txt', callbackFunction );

// send an array of files:
var folder = plugins.file.showDirectorySelectDialog();
if (folder) {
	var files = plugins.file.getFolderContents(folder);
	if (files) {
		var monitor = plugins.file.streamFilesToServer( files, callbackFunction );
	}
}
// var files = new Array()
// files[0] = 'file1.txt';
// files[1] = 'file2.txt';
// var monitor = plugins.file.streamFilesToServer( files, callbackFunction );
```

trackFileForDeletion(file)

If the client's solution is closed, the file given to this method wil be deleted.
This can be a remote or local file.

This can be used to have temp files within a client that will be cleaned up when the solution is closed.
So they live as long as the client has its solution open.

Parameters

JSFile	file	the file to track

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
var file = plugins.file.createFile("newfile.txt");
plugins.file.writeTXTFile(file, "some data");
plugins.file.trackFileForDeletionfile(file);
```

writeFile(file, data)

Writes the given file to disk.

If "file" is a JSFile or you are running in Smart Client, it writes data into a (local) binary file.

If you are running in Web Client and "file" is a String (like 'mypdffile.pdf' to hint the browser what it is) the user will get
prompted by the browser to save the file (sent using "Content-disposition: attachment" HTTP header). If it is a JSFile instance
in this case it will be saved as a file on the server.

Parameters

JSFile file a local JSFile
Array data the data to be written

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
/**@type {Array<byte>}*/
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: https://developer.mozilla.org/en-US/docs/Properly_Configuring_Server_MIME_Types
var mimeType = 'application/vnd.ms-excel'
if (!plugins.file.writeFile(f, bytes, mimeType))
	application.output('Failed to write the file.');
```

writeFile(file, data, mimeType)

Writes the given file to disk.

If "file" is a JSFile or you are running in Smart Client, it writes data into a (local) binary file.

If you are running in Web Client and "file" is a String (like 'mypdffile.pdf' to hint the browser what it is) the user will get
prompted by the browser to save the file (sent using "Content-disposition: attachment" HTTP header). If it is a JSFile instance
in this case it will be saved as a file on the server.

Parameters

JSFile file a local JSFile
Array data the data to be written
String mimeType the mime type (used in Web-Client)

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
/**@type {Array<byte>}*/
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: https://developer.mozilla.org/en-US/docs/Properly_Configuring_Server_MIME_Types
var mimeType = 'application/vnd.ms-excel'
if (!plugins.file.writeFile(f, bytes, mimeType))
        application.output('Failed to write the file.');
```

writeFile(file, data)

Writes the given file to disk.

If "file" is a JSFile or you are running in Smart Client, it writes data into a (local) binary file.

If you are running in Web Client and "file" is a String (like 'mypdffile.pdf' to hint the browser what it is) the user will get
prompted by the browser to save the file (sent using "Content-disposition: attachment" HTTP header). If it is a JSFile instance
in this case it will be saved as a file on the server.

Parameters

String file the file path as a String
Array data the data to be written

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
/**@type {Array<byte>}*/
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: https://developer.mozilla.org/en-US/docs/Properly_Configuring_Server_MIME_Types
var mimeType = 'application/vnd.ms-excel'
if (!plugins.file.writeFile(f, bytes, mimeType))
        application.output('Failed to write the file.');
```

writeFile(file, data, mimeType)

Writes the given file to disk.

If "file" is a JSFile or you are running in Smart Client, it writes data into a (local) binary file.

If you are running in Web Client and "file" is a String (like 'mypdffile.pdf' to hint the browser what it is) the user will get
prompted by the browser to save the file (sent using "Content-disposition: attachment" HTTP header). If it is a JSFile instance
in this case it will be saved as a file on the server.

Parameters

String file the file path as a String
Array data the data to be written
String mimeType the mime type (used in Web-Client)

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

Sample

```
/**@type {Array<byte>}*/
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: https://developer.mozilla.org/en-US/docs/Properly_Configuring_Server_MIME_Types
var mimeType = 'application/vnd.ms-excel'
if (!plugins.file.writeFile(f, bytes, mimeType))
        application.output('Failed to write the file.');
```

writeTXTFile(file, text_data)

Writes data into a text file. (Web Enabled: file parameter can be a string 'mytextfile.txt' to hint the browser what it is, if it is a JSFile instance it will be saved on the server)

Parameters

JSFile	file	JSFile
String	text_data	Text to be written.

Returns

Boolean Success boolean.

Supported Clients

SmartClient,WebClient,NGClient

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://download.oracle.com/javase/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'
```

writeTXTFile(file, text_data, charsetname)

Writes data into a text file. (Web Enabled: file parameter can be a string 'mytextfile.txt' to hint the browser what it is, if it is a JSFile instance it will be saved on the server)

Parameters

JSFile	file	JSFile
String	text_data	Text to be written.
String	charsetname	Charset name.

Returns

Boolean Success boolean.

Supported Clients

SmartClient,WebClient,NGClient

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://download.oracle.com/javase/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'
```

writeTXTFile(file, text_data, charsetname, mimeType)

Writes data into a text file. (Web Enabled: file parameter can be a string 'mytextfile.txt' to hint the browser what it is, if it is a JSFile instance it will be saved on the server)

Parameters

JSFile	file	JSFile
String	text_data	Text to be written.
String	charsetname	Charset name.
String	mimeType	Content type (used only on web).

Returns

Boolean Success boolean.

Supported Clients

SmartClient,WebClient,NGClient

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://download.oracle.com/javase/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'
```

writeTXTFile(file, text_data)

Writes data into a text file. (Web Enabled: file parameter can be a string 'mytextfile.txt' to hint the browser what it is, if it is a JSFile instance it will be saved on the server)

Parameters

String	file	The file path.
String	text_data	Text to be written.

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

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://download.oracle.com/javase/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'
```

writeTXTFile(file, text_data, charsetname)

Writes data into a text file. (Web Enabled: file parameter can be a string 'mytextfile.txt' to hint the browser what it is, if it is a JSFile instance it will be saved on the server)

Parameters

String	file	The file path.
String	text_data	Text to be written.
String	charsetname	Charset name.

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

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://download.oracle.com/javase/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'
```

writeTXTFile(file, text_data, charsetname, mimeType)

Writes data into a text file. (Web Enabled: file parameter can be a string 'mytextfile.txt' to hint the browser what it is, if it is a JSFile instance it will be saved on the server)

Parameters

String	file	The file path.
String	text_data	Text to be written.
String	charsetname	Charset name.
String	mimeType	Content type (used only on web).

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

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://download.oracle.com/javase/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(file, xml_data)

Writes data into an XML file. The file is saved with the encoding specified by the XML itself. (Web Enabled: file parameter can be a string 'myxmlfile.xml' to hint the browser what it is, if it is a JSFile instance it will be saved on the server)

Parameters

JSFile	file	a local JSFile
String	xml_data	the xml data to write

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

writeXMLFile(file, xml_data, encoding)

Writes data into an XML file. The file is saved with the encoding specified by the XML itself. (Web Enabled: file parameter can be a string 'myxmlfile.xml' to hint the browser what it is, if it is a JSFile instance it will be saved on the server)

Parameters

JSFile	file	a local JSFile
String	xml_data	the xml data to write
String	encoding	the specified encoding

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

writeXMLFile(file, xml_data)

Writes data into an XML file. The file is saved with the encoding specified by the XML itself. (Web Enabled: file parameter can be a string 'myxmlfile.xml' to hint the browser what it is, if it is a JSFile instance it will be saved on the server)

Parameters

String file the file path as a String
String xml_data the xml data to write

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

Sample

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

writeXMLFile(file, xml_data, encoding)

Writes data into an XML file. The file is saved with the encoding specified by the XML itself. (Web Enabled: file parameter can be a string 'myxmlfile.xml' to hint the browser what it is, if it is a JSFile instance it will be saved on the server)

Parameters

String file the file path as a String
String xml_data the xml data to write
String encoding the specified encoding

Returns

Boolean

Supported Clients

SmartClient,WebClient,NGClient

Sample

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