Table of Contents

Stdlib

Stdlib is not really a class. It is used, instead, as a namespace. This means that instead of adding 200+ functions into the global namespace, they are collected under Stdlib. In other words, instead of calling `getColorAt(doc, 100, 100)` you call `Stdlib.getColorAt(doc, 100, 100).` This helps keep things nice and neat at the expense of a little more typing. The most important benefit of doing this is that functions in Stdlib, such as *getColorAt*, will not conflict with functions of the same name written somewhere else. For small scripts this is not so much a problem. For larger scripts, this is a huge problem.

Stdlib does provide extensions to a few built-in classes, namely Date, Folder, and String. They are described in separate sections.

Stdlib does have a few functions in the global namespace. They are described in the section called Functions.

The main body of functions are grouped by their similarity. For instance, all Layer and Mask functions are grouped together. Each group has sample code that illustrates how the functions should be called. Please note that the code in these samples may or may not execute correctly as written. Many of them reference images files that that you likely do not have locally.

Date Methods

For my work, I've standardized on the ISO 8601 for my date formats. There is no ambiguity in the format and files and folders that use the format will automatically sort in chronological order. Very handy.

The **strftime** function is new addition to xtools. It provides a very powerful mechanism for formatting date strings.

Function	Parameter Type	Returns	What it does
`strftime` `(fmtspec)`	`string`	`string`	Returns a string representation of the data formatted according to *fmtspec*. This method is identical to the C function of the same name. A detailed description can be found in Appendix **strftime** Specification,
`toISODateString` `([timeDesignator, [dateOnly]])`	`string, boolean`	`string`	Return the Date as a string in ISO 8601: YYYY-MM-DDTHH:MM:SS. *timeDesignator* is character separator between the date and time portions of the string (default 'T'). *dateOnly* controls whether or not the time is excluded from the string (default: *false*).
`toISO` `([timeDesignator, [dateOnly]])`	`string, boolean`	`string`	**Date.toISODateString** implemented using **Date.strftime**.

Function	Parameter Type	Returns	What it does
`toISODateString` `([timeDesignator, [dateOnly]])`	`string, boolean`	`string`	Return the Date as a string in ISO 8601: YYYY-MM-DDTHH:MM:SS. *timeDesignator* is character separator between the date and time portions of the string (default 'T'). *dateOnly* controls whether or not the time is excluded from the string (default: *false*).
`toISOString` `([timeDesignator, [dateOnly]])`	`string, boolean`	`string`	

Date Sample Script

DateSample.jsx

```
// Open an alert in with the current date in ISO 8601 format

alert(new Date().toISODateString(undefined, true));
alert(new Date().strftime(“Todays date is %m/%d/%y”));
```

File and Folder Methods

Some time ago a bug was found in *Folder.getFiles*. If you had enough files in a folder (>75) and you specified a masking function that only returned a subset of those files, memory would get corrupted in Photoshop causing some rather severe problems requiring a restart of Photoshop. This bug ocurred in PS7 and CS but was fixed for CS2. A work-around was developed (*Stdlib.getFiles*). Once this was complete, I realized that RegExp support would be a wonderful addition, so I added that and a recursive version called *findFiles*. S*tdlib* adds *findFiles* to the Folder class and replaces *getFiles* with the new version.

Folder Methods

Function	Parameter Type	Returns	What it does
`findFiles` `(mask)`	`string, function, or RegExp`	`array of File`	The recursive version of *Folder.getFiles*.
`getFiles` `(mask)`	`string, function, or RegExp`	`array of File`	Returns the set of files in this folder that match the given mask. See *Stdlib.getFiles* for details.
`toUIString` `()`		`string`	Returns the full path of the folder suitably formatted for a UI.

File Methods

Function	Parameter Type	Returns	What it does
`strf` `(fmtspec, fs)`	`string, boolean`	`string`	Returns a string formatted according to *fmtspec* and *fs*.
`toUIString` `()`		`string`	Returns the full path of the folder suitably formatted for a UI.

File.strf Details

This is based of the file name formatting facility in *exiftool*. Part of the description is copied directly from there. You can find *exiftool* at:
http://www.sno.phy.queensu.ca/~phil/exiftool/

Description:
Format a file string using a printf-like format string.

fmtspec is a string where the following substitutions occur
%d - the directory name (no trailing /)
%f - the file name without the extension
%e - the file extension without the leading '.'
%% - the '%' character

Any other '%' characters are simply passed through to the final string.
If fs is true the folder is in local file system format
(e.g. C:\images instead of /c/images)

Examples:

Reformat the file name:

```
var f = new File("/c/work/test.jpg");
f.strf("%d/%f_%e.txt") == "/c/work/test_jpg.txt"
```

Change the file extension

```
f.strf("%d/%f.psd") == "/c/work/test.psd"
```

Convert to a file name in a subdirectory named after the extension

```
f.strf("%d/%e/%f.%e") == "/c/work/jpg/test.jpg"
```

Change the file extension and convert to a file name in a subdirectory named after the new extension

```
f.strf("%d/psd/%f.psd") == "/c/work/psd/test.psd"
```

Advanced Substitution

A substring of the original file name, directory or extension may be taken by specifying a string length immediately following the % character. If the length is negative, the substring is taken from the end. The substring position (characters to ignore at the start or end of the string) may be given by a second optional value after a decimal point.

For example:

```
var f = new File("Picture-123.jpg");

f.strf("%7f.psd") == "Picture.psd"
f.strf("%-.4f.psd") == "Picture.psd"
f.strf("%7f.%-3f") == "Picture.123"
f.strf("Meta%-3.1f.xmp") == "Meta123.xmp"
```

Folder Sample Script

FolderSample.jsx

```
// Exercise the getFiles and findFiles methods

var folder = new Folder(app.path + "/Samples");  // Use the PS Samples
folder
var files = folder.getFiles();          // get everything in folder
files = folder.getFiles("*.jpg");       // get all .jpg files in folder
files = folder.findFiles("*.jpg");      // get all .jpg files in folder and
in any
                                        // subfolders beneath it
files = folder.getFiles(/\.dng$/);      // get all .dng files in folder

// get the entire folder hierarchy beneath folder
function matchFolder(f) { return f instanceof Folder; }
var folders = folder.findFiles(matchFolder);
```

File and Folder Functions

This assortment of File and Folder functions simplifies read and writing text files, including CSV and INI files, as well as provides a much richer search capability than that provided by the native *Folder.getFiles* interface.

Function	Parameter Type	Returns	What it does
compareFiles (file1, file2)	File, File	string or null	Returns null if the files are identical. Returns a descriptive string if they aren't.
convertFptr (fptr)	string|File|Folder	File or Folder	If *fptr* is a *File* or *Folder*, it is simply returned. If it is a string, *File(fptr)* is returned. An exception is thrown in all other cases.

Function	**Parameter Type**	**Returns**	**What it does**
`createFolder` `(arg1, arg2)`	`string|File|Folder`	`boolean`	If *fptr* is a *File,* any parent folders that do not exist are created. If *fptr* is a *Folder*, it and any parent folders that do not exist are created. If it is a string, *Stdlib.createFolder(Folder(fptr))* is returned. The function returns true if the operation was successful, false if some problem occured.
`findFiles` `(folder, mask)`	`Folder, string|RegExp|function`	`array`	This is the recursive version of *getFiles*.
`fromCSVString` `(csvstr, [ary, [headers]])`	`string, array, boolean`	`array`	Parses *csvstr* as a CSV string and returns an array of the results. If *headers* is *true* (default), an array of objects is returned. If not, the array elements are arrays of strings. If *ary* is specified, the results are added to that array. If not, a new array is returned.
`fromIniString` `(inistr obj)`	`string, object`	`object`	Parses *inistr* as an ini string (e.g. the contents of a .ini file). The *key:value* pairs found are used to populate *obj*. If *obj* is not specified, a new object is created an populated.
`getFiles` `(folder, mask)`	`Folder, string|RegExp|function`	`array`	Returns an array of *File* objects for which *mask* is true. If *mask* is a string or function, then this behaves identically to the builtin *File.getFiles* method except that it gets around a bug found in PS7 and CS. If *mask* is a RegExp then the names of the files are matched against the RegExp.
`getFolders` `(folder)`	`Folder`	`array`	Returns an array of *Folders* that are subfolders of *folder*.
`getImageFiles` `(folder, [recursive, [complete]])`	`Folder, boolean, boolean`	`array`	Returns an arrary of image files found in *folder*. If *recursive* is *true* (default), all subfolders are searched as well. If *complete* is *true* (default:*false*) all known Photoshop file types are returned rather than the set of common file types.
`isImageFile` `(file)`	`File or string`	`boolean`	Returns true if *file* is a common image file type, false if not.
`log` `(msg)`	`string`		Logs *msg* to the file named in *Stdlib.log.filename* (default "~/logfile.txt").

Function	Parameter Type	Returns	What it does
`readCSVFile` `(csvfptr, [ary, [headers]])`	`File or string, array, string`	`array`	Reads the contents of the file into a string and returns the result of *Stdlib.*`fromCSVString`.
`readFromFile` `(fptr, [encoding])`	`File or string, string`	`string`	Reads the contents of *fptr* and returns the result as a string. The file is read using *encoding* if it is specified.
`readIniFile` `(fptr, obj)`	`File or string, object`	`object`	Reads the contents of the file into a string and returns the result of *Stdlib.*`fromINIString`.
`selectFile` `(prompt, mask, [start])`	`string, string, File|Folder| string`	`File`	Opens a file selection dialog with *prompt* and *mask*, using the *start* as the default (if specified). The selected file is returned.
`selectFolder` `(prompt, [start])`	`string, File|Folder| string`	`Folder`	Opens a folder selection dialog with *prompt*, using the *start* as the default (if specified). The selected folder is returned.
`toIniString` `(obj)`	`object`	`string`	Returns a string in INI format containing the *key:value* pairs for the string, boolean, and number properties in *obj*.
`writeIniFile` `(fptr, obj, [header])`	`File or string object, string`		Converts the *obj* to a string (using *toIniString*) and writes it to *fptr*. If *header* is specified, it is written to the file before the ini string.
`writeToFile` `(fptr, str, [encoding])`	`File or string string, string`		Writes the string *str* to the file *fptr*. The file is written using *encoding* if it is specified.

File and Folder Sample Script

FileFolderSample.jsx

```
var fname = "/c/temp/kitty.psd";
var file1 = new File(fname);
var file2 = new File("/c/temp/stdlib.js");

Stdlib.compareFiles(file1, file2);       // false

Stdlib.createFolder("~/images/kitty/2006-08-27");  // true

var f = Stdlib.convertFptr(fname);
f instanceof File;                       // true
f.toString() == file1.toString();        // true

// Specify a log file and write a couple of entries into it
Stdlib.log.filename = "/c/temp/log.txt";
Stdlib.log("This is a log message);
Stdlib.log("This is the second message in the log");

var file = Stdlib.selectFile("Choose a file",
                             "JPEG Files: *.jpg",
                             "/c/tmp/red-flower.jpg");

var folder = Stdlib.selectFolder("Choose a folder", app.path);

var files = Stdib.getFiles(folder);
var files = Stdib.getFiles(folder, "*.jpg");
var files = Stdib.getFiles(folder, /\.jpg$/);

function folderChk(f) {
   return f instanceof Folder;
}
var files = Stdib.getFiles(folder, folderChk);

var folders = Stdlib.getFolders("~/images");

var files = Stdlib.findFiles("~/images", "*.jpg");
var files = Stdlib.getImageFiles(new Folder("~/images"));

Stdlib.isImageFile(doc.fullName);    // true

var str = Stdlib.readFromFile("/c/tmp/test.txt");
var str = Stdlib.readFromFile("/c/tmp/test.atn", 'BINARY');

Stdlib.writeToFile(file, "Some string....");

var rows = [];
Stdlib.readCSVFile("~/colors.csv", rows, true);
var rows = Stdlib.fromCSVString(csvstr, undefined, true);

var obj = Stdlib.readIniFile("~/defaultSettings.ini");
Stdlib.writeIniFile("~/defaultSettings.ini", obj, "# Default Settings
File");

var iniObj = { name : 'Bob', age: 24 };
var iniStr = Stdlib.toIniString(iniObj);
Stdlib.fromIniString(iniStr, obj);
```

String Methods

This section contains a set of new String methods that are particulary handy when working with text.

Function	Parameter Type	Returns	What it does
contains (str)	string	boolean	Returns *true* if this string contains the substring *str, false* if not.
containsWord (str)	string	boolean	Returns *true* if this string contains the word *str, false* if not.

Function	Parameter Type	Returns	What it does
`endsWith` `(str)`	`string`	`boolean`	Returns *true* if this string ends with the substring *str, false* if not.
`reverse`		`string`	Returns a string that has the characters of this string in reverse order.
`sprintf` `(args)`	`arguments`	`string`	This is a JavaScript implementation of the C function fsprintf. See this page for details: http://www.opengroup.org/onlinepubs/007908799/xsh/fprintf.html
`startsWith` `(str)`	`string`	`boolean`	Returns *true* if this string starts with the substring *str, false* if not.
`trim`		`string`	Returns a string that has the whitespace trimmed off the beginning and end of this string.

String Sample Script

StringSample.jsx

```
// Exercise the new String methods

var str1 = "JavaScript uses prototypes, not classes.";
var str2 = " What? ";
var str3 = "12345";

str1.contains("proto");          // true
str1.containsWord("proto");      // false
str1.containsWord("prototype"); // true
str3.endsWith("45");             // true
str3.reverse() == "54321";       // true
str1.startsWith("JavaScript");   // true
str2.trim() == "What?";          // true
var str = String.sprintf("%d droplets generated in %.3f seconds",
                   cnt, elapsedTime);
```

Global Functions

These are normal functions. They are not a part of the Stdlib name space so you can call them 'as-is'. e.g.

```
var id = cTID('Rvrt');
```

IDs are 32 bit integers that can be mapped to and from 4 character strings or aribitrarily long strings. They are used heavily in Action Manager/Scripting Listener code.

Function	Parameter Type	Returns	What it does
`cTID` `(s)`	`string`	`ID`	Converts a four character string into an ID. See Application.charIDToTypeID.
`getUnitValue` `(unit)`	`Unit or number`	`number`	Returns the value portion of a Unit object. Under PS7, *unit* is returned.

Function	Parameter Type	Returns	What it does
`id2char` `(id)`	`ID`	`string`	Converts an ID into a human readable string. If no mapping is found, the unsigned representation of the ID is returned.
`isCS`		`boolean`	Returns true if the Photoshop version is CS.
`isCS2`		`boolean`	Returns true if the Photoshop version is CS2.
`isCS3`		`boolean`	Returns true if the Photoshop version is CS3.
`isPS7`		`boolean`	Returns true if the Photoshop version is PS7.
`listGlobals`		`array`	Returns an array of *key: value* pairs of all current global symbols.
`listProps` `(obj)`	`object`	`array`	Returns an array of *key: value* pairs of all the properties in *object*.
`sTID` `(s)`	`string`	`ID`	Converts a string into an ID. See Application.stringIDToTypeID.
`throwError` `(e)`	`string or Error`		Throws an exception in an expression context. e.g. `isCS2() || throwError("Must be CS2");`

`throwFileError` `(file, msg)`	`File,`
	`string`
			Throws an File exception in an expression context. e.g.
			`file.open("r") || throwFileError(file, "Unable to open file");`

`xTID` `(s)`	`number or string`	`ID`	Returns *s* as an ID.

Functions Sample Script

FunctionsSample.jsx

```
// Invoke the Free Transform menu item, illustrates use of cTID and sTID
var ref = new ActionReference();
ref.putEnumerated(cTID("Mn  "), cTID("MnIt"), cTID("FrTr"));
var desc = new ActionDescriptor();
desc.putReference(cTID("null"), ref);
executeAction(sTID("select"), desc, DialogModes.ALL);

// getUnitValue
var width = getUnitValue(app.activeDocument.width);

// id2char
var c = cTID("Rvrt");
id2char(c) == "Revert";   // true
```

```
// listGlobals
alert(listGlobals());

// listProps
alert(listProps(app.activeDocument));


// throw a exception as part of an expression
 isCS2() || throwError("Must be CS2.");

 // throw a file error exception as part of an expression
file.open("r") || throwFileError(file, "Unable to open file");

// xTID
xTID("Mn  ") == cTID("Mn  ");       // true
xTID("select") == sTID("select");   // true
```

Action Functions

The Action functions are primarily concerned with the managment of actions in the Actions Palette. More detailed APIs for manipulating action files and interpreting actions from action files exists elsewhere in this toolkit.

Function	Parameter Type	Returns	What it does
`backupActionsPalette` `([file])`	`File`		Backup the Actions Palette. If *file* is set, use that as the backup. If *file* is not set, open a dialog to select a backup file.
`createDroplet` `(atn, atnSet, fptr)`	`string, string, File`		Creates a droplet *file* from the Action *atn* rom the Action Set *atnSet*.
`deleteAction` `(atn, atnSet)`	`string, string`		Delete the Action *atn* from the Action Set *atnSet*.
`deleteActionSet` `(atnSet)`	`string`		Delete the Action Set *atnSet.*
`deleteActionStep` `(idx, atn, atnSet)`	`number, string, string`		Delete the step *idx* from the Action *atn* in the Action Set *atnSet*.
`deleteAllActionSets` `([confirmDelete])`	`boolean`		Deletes all Action Sets from the Actions Palette. If *confirmDelete* is *true* (default), a *confirm* dialog will be opened to confirm the deletions.
`getActions` `(atnSet)`	`string`	`array`	Returns an array of the names of the actions in the Action Set.
`getActionSets`		`array`	Returns an array of the names of the Action Sets in the Actions Palette.

Function	**Parameter Type**	**Returns**	**What it does**
`hasAction` `(atn, atnSet)`	`string, string`	`boolean`	Returns *true* if the Action *atn* is found in the Action Set *atnSet*.
`loadActionFile` `(file)`	`File`		Loads the actions in *file*. Actions will not be availble until the calling script has finished.
`loadActionFiles` `(files)`	`array of File`		Loads the actions from each file in *files*. Actions will not be availble until the calling script has finished.
`setActionPlaybackOption` `(opt, [arg])`	`string, value`		Sets a playback option. If arg is specified, it is used.
`setPlaybackAcclerated`			Sets the playback mode to Accelerated.
`setPlaybackPaused` `([secs])`	`number`		Sets the playback mode to Paused with *secs* seconds delay (if specified).
`setPlaybackStepByStep`			Sets the playback mode to step-by-step.

Actions Sample Script

ActionsSample.jsx

```
// backup the ActionsPalette
Stdlib.backupActionsPalette(new File("~/bak/Actions Palette.psp"));

Stdlib.deleteAction("Blizzard", "Image Effects");

Stdlib.deleteActionSet("Image Effects");
Stdlib.deleteActionStep(1, "Blizzard", "Image Effects");

Stdlib.deleteAllActionSets(true);

var ary = Stdlib.getActions("Image Effects");

var ary = Stdlib.getActionSets();

if (!Stdlib.hasAction("Blizzard", "Image Effects")) {
   throw "Images Effects Action Set is not loaded.";
}

Stdlib.loadActionFile("~/atns/Image Effects.atn");
var file = new Folder("~/atns").getFiles("*.atn");
Stdlib.loadActionFiles(files);

Stdlib.setActionPlaybackOption("accelerated");
Stdlib.setPlaybackAcclerated();                // same as previous line
Stdlib.setPlaybackPaused(2);
Stdlib.setPlaybackStepByStep();
```

Color Functions

These are some utility functions for dealing with colors in Photoshop.

Function	Parameter Type	Returns	What it does
`createRGBColor` `(arg1, [green, blue])`	`number or array, number, number`	`SolidColor`	Creates an color with the RGB values specified. e.g. `Stdlib.createRGBColor([0,25,0]);` `Stdlib.createRGBColor(0,25,0);`
`createSwatch` `(name, red, green, blue)`	`string, number, number, number`		Adds an RGBcolor called name to the current swatch palette. `Stdlib.createSwatch("linen", 250, 240, 230);`
`getColorAt` `(doc, x, y)`	`Document, number, number`	`SolidColor`	Returns the color in *doc* at the coordinates *x, y* (in pixels). Note that this may only work in RGB mode.
`rgbToArray` `(color)`	`SolidColor`	`array`	Returns the color as an array of numbers.
`rgbToString` `(color)`	`SolidColor`	`string`	Returns the color as a string of numbers.
`selectColorRange` `(doc, color, [range, [inverse]])`	`Document, SolidColor, number, boolean`		Selects a color range in *doc*. *range* is the fuzziness of the selection (default 40). *inverse* indicates whether or not to invert the selection (default *false*);
`selectColorRangeRGB` `(doc, color, [range, [inverse]])`	`Document, RGBColor or array, number, boolean`		Selects a color range in *doc*. The color is an RGB color or an array of RGB values. *range* is the fuzziness of the selection (default 40). *inverse* indicates whether or not to invert the selection (default *false*);

Color Sample Script

ColorsSample.jsx

```
app.foregroundColor = Stdlib.createRGBColor(128, 128, 128);
app.foregroundColor = Stdlib.createRGBColor([128, 128, 128]);

Stdlib.createSwatch("Middle Grey", 128, 128, 128);

app.foregroundColor = Stdlib.getColorAt(app.activeDocument, 10, 10);

var ary = Stdlib.rgbToArray(app.foregroundColor);
alert(Stdlib.rgbToString(app.foregroundColor));

Stdlib.selectColorRange(app.activeDocument, app.foregroundColor, 0, false);
Stdlib.selectColorRangeRGB(app.activeDocument, [128, 128, 128]);
```

Container Functions

Photoshop has a number of *container* classes that all behave similarly. *Documents* and *ArtLayers* are good examples of these containers. The containers are composed of a set of objects. This set is indexable by name or by number.

Note that some containers (e.g. Layers) may nested into multiple levels in a hierarchy. This set of functions does **not** operate on nested containers. Nesting is handled by different APIs.

Function	Parameter Type	Returns	What it does
`getAllByName` `(container, name)`	`container object, string or RegExp`	`array`	Returns all elements of *container* with names that match *name.*
`getByName` `(container, name, [all])`	`container object, string or RegExp, boolean`	`object or array`	Returns elements of *container* with names that match *name* If *all* is *false* (default), only the first match is returned. If *all* is *true*, all matches are returned.
`getByFunction` `(container, ftn, [all])`	`container object, function, boolean`	`object or array`	Returns elements of *container* for which *ftn* evaluates true. If *all* is *false* (default), only the first match is returned. If *all* is *true*, all matches are returned.
`getByProperty` `(container, prop, value, [all])`	`container object, string, {string|number|boolean}, boolean`	`object or array`	Returns elements of *container* for the property *prop* matches *value*. If *all* is *false* (default), only the first match is returned. If *all* is *true*, all matches are returned.

Container Sample Script

ContainerSample.jsx

```
var doc = app.activeDocument;

// get all layers with names that start with 'Layer'
var layers = Stdlib.getAllByName(doc.layers, /^Layer/);

// get a layer called "Layer 1"
var layer = Stdlib.getByName(doc.layers, "Layer 1");

// get all layers that are locked and visible
function chkLayer(layer) {
  return layer.visible && layer.allLocked;
};
var layers = Stdlib.getByFunction(doc.layers, chkLayer, true);

// get all text layers
var layers = Stdlib.getByProperty(doc.layers, "kind", LayerKind.TEXT,
true);
```

Document Functions

This section contains a handful of functions that deal strictly with Documents.

Function	Parameter Type	Returns	What it does
`duplicateDocument` `(doc, [name, [merged]])`	`Document, string, boolean`	`Document`	Returns a duplicate of *doc* with the name *name*. If *merged* is true, the layers in the new document are merged. *name* defaults to system behavior, *merged* defaults to *false*.
`fitImage` `(doc, width, height)`	`Document, number, number`		Resizes *doc* to fit inside a rectangle with dimensions of *width* and *height* (in pixels).
`getDocumentName` `(doc)`	`Document`	`string`	Returns the name of the document. This does not require that the document had been previously saved.
`hasBackground` `(doc)`	`Document`	`boolean`	Returns *true* if *doc* has a background layer, *false* if not.
`isDocumentNew` `(doc)`	`Document`	`boolean`	Returns *true* if *doc* is new and has not yet been saved, *false* if not.
`isDocumentOpen` `(file)`	`File`	`boolean`	Returns *true* if the document referred to by *file* is already open, *false* if not.
`isLandscapeMode` `(doc)`	`Document`	`boolean`	Returns *true* if *doc* is in landscape mode, *false* if not.
`isPortraitMode` `(doc)`	`Document`	`boolean`	Returns *true* if *doc* is in portrait mode, *false* if not.
`isSquareMode` `(doc)`	`Document`	`boolean`	Returns *true* if *doc* is in square mode, *false* if not.
`newDocument` `(name, mode, width, height, resolution, depth)`	`string, string, number, number, number, number`	`Document`	Create a new document using the specified parameters. *mode* is one of "BtmM", "CMYK", "Grys", "IndC", "LbCM", "MltC", "RGBM". *depth* is one of 1, 8, or 16. ex: `Stdlib.newDocument("bbb.psd", "RGBM", 250, 500, 72, 16)`
`openDialogPS7`		`Document`	Open a UI to load a document. This is for PS7 which lacks the *File.openDialog* function.
`revertDocument` `(doc)`	`Document`		Return *doc* to its original state.

Document Sample Script

DocumentSample.jsx

```
var doc = app.activeDocument;

var dupe = Stdlib.duplicateDocument(doc, "My Dupe", false);

Stdlib.getDocumentName(dupe) == "My Dupe";    // true

var bg = Stdlib.hasBackground(doc);

var isOpen = Stdlib.isDocumentOpen(new File("~/images/Some Doc.psd"));

var newDoc = Stdlib.newDocument("New Document", "CMYK", 1000, 1000, 16);

Stdlib.isDocumentNew(newDoc);      // true
Stdlib.isSquareMode(newDoc);       // true
Stdlib.isPortraitMode(newDoc);     // false
Stdlib.isLandscapeMode(newDoc);    // false

var doc = Stdlib.openDialogPS7("/c/tmp/kitty.psd");

Stdlib.revertDocument(doc);
```

History Functions

There are a number of non-Photoshop-related functions in Stdlib that are useful to have handy.

Function	Parameter Type	Returns	What it does
`deleteSnapshot` `(doc, name)`	`Document, string`	`boolean`	Deletes the snapshot called *name* from the history of *doc*. Returns false if that history state does not exist.
`hist` `(dir)`	`string`		Moves the history in the direction indicated by *dir*. Must be *"Prvs"* or *"Nxt "*.
`redo`			Move forward to the next history state.
`revertToLastSnapshot` `(doc)`	`Document`	`boolean`	Revert to the last auto-named history state (one that begins with 'Snapshot'). Returns *true* if successful.
`revertToSnapshot` `(doc, [sname])`	`Document, sname`	`boolean`	Revert to the history state *sname*. If *sname* is not specified, *revertToLastSnapshot* is called. Returns *true* if successful.
`takeSnapshot` `(doc, [sname])`	`Document, string`		Take a snapshot of the current history state. If *sname* is specified it is used as the name of the snapshot. If not, the snapshot is autonamed.

Function	**Parameter Type**	**Returns**	**What it does**
`undo`			Undo the last operation.

History Sample Script

HistorySample.jsx

```
var doc = app.activeDocument;

Stdlib.undo();
Stdlib.redo();
Stdlib.hist("Prvs");

Stdlib.takeSnapshot(doc);
Stdlib.revertToLastSnapshot(doc);
Stdlib.takeSnapshot(doc, "Before filter state");
Stdlib.revertToSnapshot(doc, "Before filter state");
Stdlib.deleteSnapshot(doc, "Before filter state");
```

Layer and Mask Functions

The Photoshop JavaScript is missing some fairly important Layers methods and has none at all that deal with layer masks. The functions in this section address these shortcomings.

Layer Styles/Effects Functions

Function	**Parameter Type**	**Returns**	**What it does**
`clearEffects` `(doc, layer)`	`Document, Layer`		Clears the layer styles from *layer*.
`copyEffects` `(doc)`	`Document`		See *copyStyles*.
`copyStyles` `(doc)`	`Document`		Copies styles from the current layer to the styles clipboard.
`hasEffects` `(doc, layer)`	`Document, Layer`	`boolean`	Returns *true* if *layer* has layer styles, *false* if not.
`hideLayerEffects` `(doc, layer)`	`Document, Layer,`		Hides layer effects for *layer*.
`pasteEffects` `(doc)`	`Document`		See *pasteStyles*.
`pasteStyles` `(doc)`	`Document`		Pastes the contents of the styles clipboard into the current layer.
`showLayerEffects` `(doc, layer)`	`Document, Layer`		Shows the layer effects for *layer*.

Layer Mask Functions

Function	Parameter Type	Returns	What it does
`appendMaskToSelection` `(doc, layer)`	`Document, Layer`		Appends the mask of *layer* to the current selection.
`createLayerMask` `(doc, layer, [fromSelection)`	`Document, Layer, boolean`		Create a layer mask for *layer*. If *fromSelection* is *true*, use the current selection as the layer mask.
`disableLayerMask` `(doc, layer)`	`Document, Layer`		Disables the layer mask for *layer*.
`enableLayerMask` `(doc, layer)`	`Document, Layer`		Enables the layer mask for *layer*.
`getMaskBounds` `(doc, layer)`	`Document, Layer`	`bounds array`	Returns the bounds of the mask on *layer*.
`hasLayerMask` `(doc, layer)`	`Document, Layer`	`boolean`	Returns *true* if *layer* has a mask, *false* if not.
`insertImageIntoMask` `(doc, layer, fptr)`	`Document, Layer, File or string`		Inserts the image in file *fptr* into the mask on *layer*.
`linkLayerMask` `(doc, layer)`	`Document, Layer`		Link the mask for *layer*.
`removeLayerMask` `(doc, layer, [apply])`	`Document, Layer, boolean`		Removes the mask from *layer*. If *apply* is *true* (default:*false*) the mask is applied on removal.
`selectLayerMask` `(doc, layer)`	`Document, Layer`		Selects the layer mask of *layer*.
`selectMaskChannel` `(doc, layer)`	`Document, Layer`		Selects the mask channel of *layer*.
`setLayerMaskEnabledState` `(doc, layer, state)`	`Document, Layer, boolean`		Enables or disables the layer mask based on the value of *state*.
`unlinkLayerMask` `(doc, layer)`	`Document, Layer`		Unlink the mask on *layer*.

Multi-Layer / Document Functions

Function	Parameter Type	Returns	What it does
`copyLayerToDocument` `(doc, layer, otherDoc)`	`Document, Layer, Document`		Copies a layer from one document to another.
`deleteAllHiddenLayers` `(doc)`	`Document`		Deletes all hidden layers from *doc*.
`deleteSelectedLayers` `(doc)`	`Document`		Deletes all selected layers from *doc*.
`findLayer` `(doc, name)`	`Document, string`		Finds a layer called *name* in the document.
`getLayersList` `(doc)`	`Document`	`array`	Returns an array of all layers in *doc*.
`getSelectedLayers` `(doc)`	`Document`	`array`	Returns an array of all currently selected layers in *doc*.
`getVisibleLayers` `(doc)`	`Document`	`array`	Returns an array of all visible layers in *doc*.
`hideAllLayers` `(doc)`	`Document`		Hides all top-level layers in *doc*.
`linkSelectedLayers` `(doc)`	`Document`		Link the currently selected layers in *doc*.
`makeDocFromLayer` `(doc, layer, docName)`	`Document, Layer, string`	`Document`	Returns a new document called *docName* using the contents of *layer*.
`mergeVisible` `(doc)`	`Document`		Merges the visible layers in *doc*.
`newGroupFromLayers` `(doc)`	`Document`		Creates a new group from the currently selected layers. *Note: this needs to be extended so that an array of layers can be passed in.*
`selectLayers` `(doc, layers, append)`	`Document, arrary of Layer, boolean`		Selects *layers*. If *append* is *true*, *layers* is added to the current selection.
`selectLinkedLayers` `(doc)`	`Document`		Selects layers linked to the current layer in *doc*.
`showAllLayers` `(doc)`	`Document`		Shows all top-level layers in *doc*.
`traverseLayers` `(doc, ftn, [reverse])`	`Document, function, boolean`		Executes the function *ftn* on all layers in *doc*. If *reverse* is *true* (default:*false*) the layers are traversed in reverse order. ftn takes the form of: `function myFtn(doc, layer) {}` It should return *true* to continue the traversal or *false* to stop.

Function	**Parameter Type**	**Returns**	**What it does**
`unlinkSelectedLayers` `(doc)`	`Document`		Unlink the layers linked to the current *layer*.

Layer Transformation

Function	**Parameter Type**	**Returns**	**What it does**
`freeTransform` `(doc, layer)`	`Document, Layer`		Initiates an interactve free transform on *layer*.
`rotateLayer` `(doc, layer, angle)`	`Document, Layer, number`		Rotates *layer* by *angle* number of degreees.
`transformLayer` `(doc, layer)`	`Document, Layer`		Peforms an interactive free transform on *layer*.
`transformScale` `(doc, layer)`	`Document, Layer`		Peforms an interactive scale transform on *layer*.

Layer Index Functions

Note that the indexes in this section are an internal Photoshop numbers that have nothing to do with with a layer's index in *Document.layers*.

Function	**Parameter Type**	**Returns**	**What it does**
`deselectLayerByIndex` `(doc, index)`	`Document, number`		Deselects the layer using the internal Photoshop index *index*.
`deselectLayersByIndex` `(doc, indexes)`	`Document, numbers`		Deselects the layers in *indexes*.
`getActiveLayerIndex` `(doc)`	`Document`	`number`	Returns the internal Photoshop index of the active layer.
`getLayerBoundsByIndex` `(doc, index)`	`Document, number`	`bounds`	Returns the bounds for the layer at *index*.
`getLayerDescriptorByIndex` `(doc, index)`	`Document, number`	`ActionDescriptor`	Returns the ActionDescriptor for the layer at *index*.
`getLayerIndex` `(doc, layer)`	`Document, Layer`	`number`	Returns the internal Photoshop index for *layer*.
`getLayerNameByIndex` `(doc, idx)`	`Document, number`	`String`	Returns the name of the layer *idx*.

Function	Parameter Type	Returns	What it does
`getLayerOpacityByIndex` `(doc, idx)`	`Document, number`	`number`	Returns the opacity of the layer *idx*.
`getLayerTypeByIndex` `(doc, idx)`	`Document, number`	`number`	Returns the type of the layer *idx* as an ID.
`moveLayerContentByIndex` `(doc, idx, dx, dy)`	`Document, number, number, number`		Moves the contents of the layer *idx* by *dx* and *dy*.
`selectLayerByIndex` `(doc, index)`	`Document, number`		Selects the layer using the internal Photoshop index *index*.
`selectLayersByIndex` `(doc, indexes)`	`Document, numbers`		Selects the layers in *indexes*.

Miscellaneous Layer Functions

Function	Parameter Type	Returns	What it does
`getLayerDescriptor` `(doc, layer)`	`Document, Layer`	`ActionDescriptor`	Returns the ActionDescriptor for *layer*.
`getLayerID` `(doc, layer)`	`Document, Layer`	`number`	Returns the internal Photoshop ID for *layer*.
`isLayerEmpty` `(doc, layer)`	`Document, Layer`		Returns *true* if *layer* is empty, *false* if not.
`isLayerSelected` `(doc, layer)`	`Document, Layer`		Not yet implemented.
`loadSelection` `(doc, layer, kind, [invert])`	`Document, Layer, string, boolean`		Creates a selection based on the channel *kind* of *layer*. *kind* is one of *"Trsp"* or *"Msk "*. If *invert* is *true* (default:*false*), the selection is inverted.
`rasterizeLayer` `(doc, layer)`	`Document, Layer`		Rasterizes *layer*.
`selectLayer` `(doc, layer, append)`	`Document, Layer, boolean`		Selects *layer*. If *append* is *true*, *layer* is added to the current selection.
`selectTransparencyChannel` `(doc, layer)`	`Document, Layer`		Selects the transpareny channel of *layer*.

Layer and Mask Sample Script

LayerMaskSample.jsx

```
var doc = app.activeDocument;

var layers = Stdlib.getLayersList(doc);
Stdlib.deleteAllHiddenLayers(doc);
Stdlib.hideAllLayers(doc);
Stdlib.showAllLayers(doc);


var layer = doc.activeLayer;

Stdlib.makeDocFromLayer(doc, layer, "New Document");
Stdlib.insertImageIntoMask(doc, layer, "~/images/kitty.psd");
Stdlib.rotateLayer(doc, layer, 45.0);
Stdlib.selectLayer(doc, layer);
Stdlib.rasterizeLayer(doc, layer);
Stdlib.transformLayer(doc, layer);
Stdlib.transformScale(doc, layer);


Stdlib.copyEffects(doc);              // Stdlib.copyStyles(doc);
Stdlib.hideLayerEffects(doc, layer);
Stdlib.pasteEffects(doc);             // Stdlib.pasteStyles
Stdlib.showLayerEffects(doc, layer);


Stdlib.createLayerMask(doc, layer, true);
Stdlib.appendMaskToSelection(doc, layer);
Stdlib.disableLayerMask(doc, layer);
Stdlib.enableLayerMask(doc, layer);
Stdlib.setLayerMaskEnabledState(doc, layer, true);
var hasMask = Stdlib.hasLayerMask(doc, layer);
Stdlib.removeLayerMask(doc, layer, false);
Stdlib.selectLayerMask(doc, layer);
Stdlib.linkLayerMask(doc, layer);
Stdlib.unlinkLayerMask(doc, layer);

Stdlib.loadSelection(doc, layer, "Trsp");
Stdlib.selectMaskChannel(doc, layer);
Stdlib.selectTransparencyChannel(doc, layer);


Stdlib.linkSelectedLayers(doc);
Stdlib.selectLinkedLayers(doc, layer);
Stdlib.unlinkSelectedLayers(doc);

var desc = Stdlib.getLayerDescriptor(doc, layer);
var desc = Stdlib.getLayerDescriptorByIndex(doc, 2);
var idx = Stdlib.getLayerIndex(doc, layer);
Stdlib.selectLayerByIndex(doc, idx);

// get all layers that are locked and visible
```

```
function chkLayer(layer) {
  return layer.visible && layer.allLocked;
};
var layers = Stdlib.traverseLayers(doc, chkLayers);
```

Selection and Path Functions

This section contains a set of function for operating on Selections and Paths. Several deal with get the bounds of the selection. There are here because the *Selection.bounds* property is broke in CS2 and non-existent in CS and PS7. The preferred way to get the selection bounds is *Stdlib.getSelectionBounds*.

Function	Parameter Type	Returns	What it does
computeSelectionBounds (doc)	Document	array	Returns the bounds of the current selection computed using the *LayerSet* techique.
computeSelectionBoundsLS (doc)	Document	array	Returns the bounds of the current selection computed using the *WorkPath* techique.
computeSelectionBoundsPS7 (doc)	Document	array	Returns the bounds of the current selection computed using a PS7 compatible version of the *WorkPath* techique.
computeSelectionRegion (doc)	Document	array	Returns the precise region of the current selection using the *WorkPath* techique.
crop (doc)	Document		Crops *doc* based on the current selection.
deselectActivePath (doc)	Document		Deselects the active path in *doc*.
getPathItems (doc)	Document	ActionList	A PS7 compatible way of getting the path items in the current document.
getSelectionBounds (doc)	Document	array	Returns the bounds of the current selection using the *Layer Dupe* techique. This is the preferred way of getting the bounds of a selection.
interactiveCrop (doc, bnds)	Document, array		Performs an interactive crop on *doc* using *bnds* as the initial area of the crop.
hasSelection (doc)	Document	boolean	Returns *true* if *doc* has an active selection, *false* if not.
magicWand (doc, x, y, [tol, [aa]])	Document, number, number, number, boolean		Peforms a magic wand operation on *doc* at the coordinates *x, y,* using *tol* as the tolerance, if specified. If *aa* is *true* (default), anti-alias is turned on for the selection.
makeWorkPath (doc)	Document		A PS7 compatible way of converting the current selection to a path on *doc*.

Function	Parameter Type	Returns	What it does

`selectBounds`
`(doc,`
`bnds,`
`[type,`
`[feather,`
`[alias]]])`
	`Document,`
	`array,`
	`SelectionType,`
	`number,`
	`boolean`
			This is a wrapper for *Selection.select*. It differs in that it selects via a bounds array. This is, the array is [x1, y1, x2,, y2] instead of an array of coordinates.

`selectSimilar`
`(doc,`
`[tol,`
`[aa]])`
	`Document,`
	`number,`
	`boolean`
			Peforms a *select similar* operation on *doc* using *tol* as the tolerance, if specified. If *aa* is *true* (default), anti-alias is turned on for the selection.

`transformSelection`
`(doc)`
	`Document`		Performs an interactive transform operation on the current selection in *doc*.

Selection and Path Script

SelectionPathSample.jsx

```
var doc = app.activeDocument;

Stdlib.hasSelection(doc);
Stdlib.transformSelection(doc);

Stdlib.crop(doc);
Stdlib.interactiveCrop(doc);

Stdlib.computeSelectionBounds(doc);
Stdlib.computeSelectionBoundsLS(doc);
Stdlib.computeSelectionBoundsPS7(doc);
Stdlib.computeSelectionRegion(doc);
Stdlib.getSelectionBounds(doc);

Stdlib.selectBounds(doc);

Stdlib.magicWand(doc);
Stdlib.selectSimilar(doc);

Stdlib.deselectActivePath(doc);
Stdlib.getPathItems(doc);
Stdlib.makeWorkPath(doc);
```

Miscellaneous Photoshop Functions

There are a number of Photoshop-related functions in Stdlib that don't quite fit in the previous sections. They are collected here.

Function	Parameter Type	Returns	What it does

`addTextLayer`
`(doc,`
`contents,`
	`Document,`
	`string,`
	`string,`
		`ArtLayer`	Returns a text layer populated with *contents*. name is optional. If the

Function	Parameter Type	Returns	What it does
`[name, [size]])`	`number`		font size *size* is not specified, 24 is used as a default. The layer is positioned 5% from the upper-left corner. The color and font are hardwired.
`applyDataSet` `(setName)`	`string`		Applys the dataset *setName* to the current document.
`batch` `(src, atn, atnSet, opts, mask, recurse)`	`Files and/or Folders, string, string, BatchOptions, string, boolean`		This is a wrapper around the Batch command. It does not use the *Application.batch* interface. *source* can be a *File*, *Folder*, or an array of the two. *atn* and *atnSet* specify the action to be executed. *opts* is describe in the Photoshop documentation for *BatchOptions*. *mask* is a Stdlib.getFiles mask. *recurse* indicates whether or not to go into subfolders (default:*false*). One siginificant note is that the *fileNaming* propery of *opts* may also contain arbitrary strings as components.
`btRunScript` `(script, [app])`	`string, string`		Executes *script* by send a command over BridgeTalk to the application *app*. If *app* is not specified, it is sent to the current application.
`clearGuides` `(doc)`	`Document`		Removes all guides from *doc*.
`createGuide` `(doc, orientation, location)`	`Document, string, number`		Creates a guide at *location* (in pixels). Orientation is one of "Vrtc" or "Hrzn"
`createHorizontalGuide` `(doc)`	`Document, number`		Creates a horizontal guide at *location* (in pixels).
`createVerticalGuide` `(arg1, arg2)`	`Document, number`		Creates a vertical guide at *location* (in pixels).
`doEvent` `([doc], eid, [interactive, [noDesc]])`	`Document, string | number, boolean, boolean`		Executes an event indicated by *eid*. If *interactive* is *true* (default:*false*), the event is run interactively. If *noDesc* is *true* (default:*false*), no ActionDescriptor is used for the invocation.
`doMenuItem` `(menuItemId, [interactive])`	`string | number, boolean,`		Executes a menu item indicated by *menuItemId*. If *interactive* is *true* (default:*false*), the menu item is run interactively.

Function	Parameter Type	Returns	What it does

`drawLine`
`(doc,`
`start,`
`stop)`
	`doc,`
	`array,`
	`array`
			Draws a line on *doc* using the coordinates specified by *start* and *stop*. e.g.
			`Stdlib.drawLine(doc,`
			`[20,20], [20, 40]);`

`dumpRTI`
`(obj)`
	`class`		Returns a string containing runtime information about *obj* gathered using the *Reflect* API.

`fileImportDataSets`
`(filename)`
	`string`		Imports the dataset information in the file *filename* into the current document.

`findFont`
`(fstr)`
	`string`	`TextFont`	Returns a TextFont with a PostScript name of *fstr*.

`findPSFont`
`(fstr)`
	`string`	`string`	Returns a PostScript name of a font with the name *fstr*.

`getLastJSLogEntry`
`([fptr])`
	`File or string`	`string`	Returns the last entry in the ScriptingListener log indicated by *fptr*. If *fptr* is not specified, *"/c/ScriptingListenerJS.log"* is used.

`getPSFontList`		`array`	Returns an array containing the PostScript names of all available fonts.

`makeActive`
`(obj)`
	`object`	`object`	Make *obj* the active object of its type. The previous active object is returned. Currently on Document and Layer objects are supported.

`renameChannel`
`(doc,`
`oldName,`
`newName)`
	`Document,`
	`string,`
	`string`
			Changes the name of the channel with the name *oldName* to.

`runScript`
`(fstr)`
	`string`		Runs the script in the file *fstr*.

`saveAllPatterns`
`(file)`
	`File`		Saves all patterns to *file*.

`savePattern`
`(file,`
`index)`
	`File,`
	`number`
			Saves a pattern at *index* in the specified file.

`savePatterns`
`(file,`
`indexes)`
	`File,`
	`array`
			Saves the patterns with the indexes in *ary* to the specified file.

`selectChannel`
`(chnl)`
	`string`		Selects the channel with the id *chnl*.

`selectTool`
`(tool)`
	`ID or ToolType`		Selects the tool specified by *tool*.

`stop`
`([msg,`
`[cont]])`
	`string,`
	`boolean`
			Issues a *Stop* command with the message *msg*.
			If msg is not specified "Operation

Function	**Parameter Type**	**Returns**	**What it does**
			Canceled" is used. If continue is *true* (default:*false*), the dialog will offer a "Continue" button. In its current form, the function serves little purpose.
`waitForRedraw`			Waits for a Redraw command to occur. This is helpful if you need to force a refresh of the current document.

Miscellaneous Photoshop Sample Script

MiscPSSample.jsx

```
var doc = app.activeDocument;

Stdlib.addTextLayer(doc, doc.fullName.fsName, "Filename", 24);
Stdlib.batch(doc);
Stdlib.drawLine(doc);

Stdlib.fileImportDataSets(doc);
Stdlib.applyDataSet(doc);

Stdlib.clearGuides(doc);
Stdlib.createGuide(doc);
Stdlib.createHorizontalGuide(doc);
Stdlib.createVerticalGuide(doc);

Stdlib.findFont(doc);
Stdlib.findPSFont(doc);
Stdlib.getPSFontList(doc);

Stdlib.saveAllPatterns(doc);
Stdlib.savePattern(doc);
Stdlib.savePatterns(doc);

Stdlib.renameChannel(doc);
Stdlib.selectChannel(doc);
Stdlib.selectTool(doc);
Stdlib.waitForRedraw(doc);

Stdlib.doEvent(doc);
Stdlib.doMenuItem(doc);
Stdlib.makeActive(doc);
Stdlib.wrapLC(doc);
Stdlib.wrapLCLayer(doc);

Stdlib.btRunScript(doc);
Stdlib.runScript(doc);

Stdlib.dumpRTI(obj);
Stdlib.getLastJSLogEntry(doc);
```

Miscellaneous Other Functions

There are a number of non-Photoshop-related functions in Stdlib that are useful to have handy. They are collected here.

Function	Parameter Type	Returns	What it does
`binToHex` `(str, [whitespace])`	`string, boolean`	`string`	Converts a binary string *str* to a hex string, which is then returned. If *whitespace* is *true* (default:*false*) a '\n' character is insert into the string every 16 characters.
`clearObject` `(obj)`	`object`	`object`	Deletes all properties from *obj*. *obj* is returned.
`copyFromTo` `(from, to)`	`object, object`		Copies all properties from object *from* to object *to*. No functions are copied.
`createObject` `(cls, attrs)`	`class, object`	`object`	Create as new object of class *cls*. The properties in *attrs* are copied to the new object, which is returned.
`dumpGlobals` `([fname])`	`string`		The results of *listGlobals* are written to the file *fname*. If fname is not specified, *"/c/temp/globals.log"* is used.
`fullStop`			Stops the script by launching the ExtendScript debugger.
`getScriptFile`		`File`	Returns the File of the running script.
`getScriptFileName`		`String`	Returns the filename of the running script.
`getScriptFolder`		`Folder`	Returns the Folder containing the File of the running script.
`hexToBin` `(str)`	`string`		Returns the hex string *str* converted to a binary string.
`hexToJS` `(str)`	`string`	`string`	Returns a JavaScript string contant of the hex string *str*. When used with *binToHex*, this makes it possible to embed binary data in a script.
`hexToLong` `(str)`	`string`	`number`	Returns the conversion of a hex string to a 32 bit unsigned integer.
`longToHex` `(num)`	`number`	`string`	Returns the conversion of a 32 bit unsigned integer to a hex string.
`numberToAscii` `(num)`	`number`	`string`	The 32 bit number num is converted to a string by convert the numeric value of each byte to an ascii character. e.g. `var num = 1383494260;` `Stdlib.numberToAscii(num) == "Rvrt"`
`parseISODateString` `(str)`	`string`	`Date`	Parses the string as an ISO 8601 string and returns that date. See **Stdlib.toISODateString**.
`popRandomElement` `(ary)`	`array`	`object`	Returns a random object from the array *ary* and removes it from the array.

Function	**Parameter Type**	**Returns**	**What it does**
`randomElement` `(ary)`	`array`	`object`	Returns a random object from the array *ary*.
`shortToHex` `(num)`	`number`	`string`	Returns the conversion of a 16 bit unsigned integer to a hex string.
`toISODateString` `(date, [timeDesignator, [dateOnly]])`	`Date string, boolean`	`string`	Return *date* as a string in ISO 8601: YYYY-MM-DDTHH:MM:SS. *timeDesignator* is character separator between the date and time portions of the string (default 'T'). *dateOnly* controls whether or not the time is excluded from the string (default: *false*).

Miscellaneous Other Sample Script

MiscOtherSample.jsx

```
var str = Stdlib.readFromFile("/c/tmp/xxx.asl", 'BINARY');
var hex = Stdlib.binToHex(s);

var jsStr = Stdlib.hexToJS(hex);
var num = Stdlib.hexToLong("DEADBEEF");

var binstr = Stdlib.hexToBin(hex);

var hex = Stdlib.longToHex(num);
var num = Stdlib.hexToLong("61626364");
Stdlib.numberToAscii(num) == 'abcd';     // true

Stdlib.clearObject(obj);
Stdlib.copyFromTo(fromObj, toObj);

function MyClass() {}
var obj = Stdlib.createObject(MyClass, app.activeDocument);

Stdlib.dumpGlobals();
Stdlib.fullStop();

Stdlib.getScriptFileName();
Stdlib.getScriptFolder();

var ary = [5, 7, 1, 4, 12, 67, 99];
var element = Stdlib.popRandomElement(ary);
var element = Stdlib.randomElement(ary);
Stdlib.toISODateString(new Date());
```

Appendix A: Date Format Specification

One style of Date format specifications is based on the Unix *strftime(3)* implementation. The format specification is text that includes characters that will be substituted according to the following rules. Characters in a format specification that are not matched are left unchanged. A date/time of October 31, 2006 20:43:02 will be used for the example below.

Specifier	**Substitution**
%a	A three-letter abbreviation for the day of the week, one of 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', or 'Sat'. *Tue*
%A	The full name for the day of the week, one of 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', or 'Saturday'. *Tuesday*
%b	A three-letter abbreviation for the name of the month, one of 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', or 'Dec'. *Oct*
%B	The full name of the month. One of 'January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', or 'December'. *October*
%c	The complete date and time in this format: "Tue Oct 31 20:43:02 2006".
%C	The century portion of the year, zero padded. *20*
%d	The day of the month, zero padded. *31*
%D	The date in the format *"%m/%d/%y"*. *10/31/06*
%e	The day of the month, space padded. *31*
%F	The date in the format *"%Y-%m-%d"*. *2006-10-31*
%h	A three-letter abbreviation for the name of the month, same as *%b*. *Oct*
%H	The hour on a 24 hour clock, zero padded. *20*
%I	The hour on a 12 hour clock, zero padded. *08*
%j	The Julian date. *304*
%k	The hour on a 24 hour clock, space padded. *20*
%l	The hour on a 12 hour clock, space padded. *8*
%m	The month number, zero padded. *10*

%M	The minutes, zero padded. *43*
%n	A newline character.
%p	Either AM or PM as appropriate. *PM*
%r	12 hour time to the second with this format: "%I:%M:%S %p". *08:43:02 PM*
%S	The seconds, zero padded. *02*
%t	A tab character.
%T	24 hour time to the seconds with this format:*"%H:%M:%S"*. *20:43:02*
%u	The weekday as a number where Monday is 1 and Sunday is 7. *2*
%w	The weekday as a number where Sunday is 0 and Saturday is 6. *2*
%x	The date in the format *"%m/%d/%y"*. Same as *%D*.
%X	24 hour time to the seconds with this format: *"%H:%M:%S"*. Same as *%T.*
%y	The last two digits of the year. *06*
%Y	The full four digits of the year. *2006*
%%	The '%' character.

Using these formats, you could create a text layer with the name *@Date*, set its contents to *"Created on %Y/%m/%d"*, and end up with a layer that looks like "Created on 2006/10/31".