# Squirrel Standard Library 3.0

**Alberto Demichelis**

# Squirrel Standard Library 3.0

Alberto Demichelis

# Table of Contents

# Chapter 1. Introduction

The squirrel standard libraries consist in a set of modules implemented in C++. While are not essential for the language, they provide a set of useful services that are commonly used by a wide range of applications(file I/O, regular expressions, etc...), plus they offer a foundation for developing additional libraries.

All libraries are implemented through the squirrel API and the ANSI C runtime library. The modules are organized in the following way:

- I/O : input and output
- blob : binary buffers manipilation
- math : basic mathematical routines
- system : system access function
- string : string formatting and manipulation

The libraries can be registered independently,except for the IO library that depends from the bloblib.

# Chapter 2. The Input/Output library

the input lib implements basic input/output routines.

# Squirrel API

## Global symbols

```
dofile(path, [raiseerror]);
```

compiles a squirrel script or loads a precompiled one and executes it. returns the value returned by the script or null if no value is returned. if the optional parameter 'raiseerror' is true, the compiler error handler is invoked in case of a syntax error. If raiseerror is omitted or set to false, the compiler error handler is not ivoked. When squirrel is compiled in unicode mode the function can handle different character ecodings, UTF8 with and without prefix and UCS-2 prefixed(both big endian an little endian). If the source stream is not prefixed UTF8 ecoding is used as default.

```
loadfile(path, [raiseerror]);
```

compiles a squirrel script or loads a precompiled one an returns it as as function. if the optional parameter 'raiseerror' is true, the compiler error handler is invoked in case of a syntax error. If raiseerror is omitted or set to false, the compiler error handler is not ivoked. When squirrel is compiled in unicode mode the function can handle different character ecodings, UTF8 with and without prefix and UCS-2 prefixed(both big endian an little endian). If the source stream is not prefixed UTF8 ecoding is used as default.

```
writeclosuretofile(destpath, closure);
```

serializes a closure to a bytecode file (destpath). The serialized file can be loaded using loadfile() and dofile().

`stderr`

File object bound on the os *standard error* stream

`stdin`

File object bound on the os *standard input* stream

`stdout`

File object bound on the os *standard output* stream

## File class

The file object implements a stream on a operating system file. It's contructor imitate the behaviour of the C runtime function fopen for eg.

```
local myfile = file("test.xxx","wb+");
```

creates a file with read/write access in the current directory.

**eos**();

returns a non null value if the read/write pointer is at the end of the stream.

**flush**();

flushes the stream.return a value != null if succeded, otherwise returns null

**len**();

returns the lenght of the stream

**readblob**(size);

read n bytes from the stream and retuns them as blob

**readn**(type);

reads a number from the stream according to the type parameter. *type* can have the following values:

| | | |
|---|---|---|
| 'i' | 32bits number | returns an integer |
| 's' | 16bits signed integer | returns an integer |
| 'w' | 16bits unsigned integer | returns an integer |
| 'c' | 8bits signed integer | returns an integer |
| 'b' | 8bits unsigned integer | returns an integer |
| 'f' | 32bits float | returns an float |
| 'd' | 64bits float | returns an float |

**seek**(seek, [origin]);

Moves the read/write pointer to a specified location. *offset* indicates the number of bytes from *origin*. *origin* can be 'b' beginning of the stream,'c' current location or 'e' end of the stream. If origin is omitted the parameter is defaulted as 'b'(beginning of the stream).

**tell**();

returns read/write pointer absolute position

**writeblob**(blob);

writes a blob in the stream

**writen**(n, type);

writes a number in the stream formatted according to the type parameter. *type* can have the following values:

| | |
|---|---|
| 'l' | processor dependent, 32bits on 32bits processors, 64bits on 64bits prcessors |
| returns an integer | 'i' |
| 32bits number | 's' |
| 16bits signed integer | 'w' |
| 16bits unsigned integer | 'c' |
| 8bits signed integer | 'b' |
| 8bits unsigned integer | 'f' |
| 32bits float | 'd' |
| 64bits float | |

# C API

## Initialization

sqstd_register_iolib

SQRESULT **sqstd_register_iolib**(HSQUIRRELVM v);

initialize and register the io library in the given VM.

parameters: *HSQUIRRELVM v* the target VM

return: an SQRESULT

remarks: The function aspects a table on top of the stack where to register the global library functions.

## File object

sqstd_createfile

SQRESULT **sqstd_createfile**(HSQUIRRELVM v, SQFILE file, SQBool own);

creates a file object bound to the SQFILE passed as parameter and pushes it in the stack

parameters: *HSQUIRRELVM v* the target VM

*SQFILE file* the stream that will be rapresented by the file object

*SQBool own* if different true the stream will be automatically closed when the newly create file object is destroyed.

return: an SQRESULT

sqstd_getfile

SQRESULT **sqstd_getfile**(HSQUIRRELVM v, SQInteger idx, SQFILE * file);

retrieve the pointer of a stream handle from an arbitrary position in the stack.

parameters: *HSQUIRRELVM v* the target VM

*SQInteger idx* and index in the stack

*SQFILE * file* A pointer to a SQFILE handle that will store the result

return: an SQRESULT

# Script loading and serialization

sqstd_loadfile

```
SQRESULT sqstd_loadfile(HSQUIRRELVM v, const SQChar * filename, SQBool printerror);
```

compiles a squirrel script or loads a precompiled one an pushes it as closure in the stack. When squirrel is compiled in unicode mode the function can handle different character ecodings, UTF8 with and without prefix and UCS-2 prefixed(both big endian an little endian). If the source stream is not prefixed UTF8 ecoding is used as default.

parameters: *HSQUIRRELVM v* the target VM

*const SQChar * filename* path of the script that has to be loaded

*SQBool printerror* if true the compiler error handler will be called if a error occurs.

return: an SQRESULT

sqstd_dofile

```
SQRESULT sqstd_dofile(HSQUIRRELVM v, const SQChar * filename, SQBool retval, SQBool printerror);
```

Compiles a squirrel script or loads a precompiled one and executes it. Optionally pushes the return value of the executed script in the stack. When squirrel is compiled in unicode mode the function can handle different character ecodings, UTF8 with and without prefix and UCS-2 prefixed(both big endian an little endian). If the source stream is not prefixed UTF8 ecoding is used as default.

parameters: *HSQUIRRELVM v* the target VM

*const SQChar * filename* path of the script that has to be loaded

*SQBool retval* if true the function will push the return value of the executed script in the stack.

*SQBool printerror* if true the compiler error handler will be called if a error occurs.

return: an SQRESULT

remarks: the function aspects a table on top of the stack that will be used as 'this' for the execution of the script. The 'this' parameter is left untouched in the stack.

eg.

```
sq_pushroottable(v); //push the root table(were the globals of the scr
```

```
sqstd_dofile(v, _SC("test.nut"), SQFalse, SQTrue);// also prints synta
```

### sqstd_writeclosuretofile

```
SQRESULT sqstd_writeclosuretofile(HSQUIRRELVM v, const SQChar * filename);
```

serializes the closure at the top position in the stack as bytecode in the file specified by the paremeter filename. If a file with the same name already exists, it will be overwritten.

| | | |
|---|---|---|
| parameters: | *HSQUIRRELVM v* | the target VM |
| | *const SQChar * filename* | path of the script that has to be loaded |
| return: | an SQRESULT | |

# Chapter 3. The Blob library

The blob library implements binary data manipulations routines. The library is based on `blob objects` that rapresent a buffer of arbitrary binary data.

# Squirrel API

## Global symbols

**blob**(size);

returns a new instance of a blob class of the specified size in bytes

**castf2i**(f);

casts a float to a int

**casti2f**(n);

casts a int to a float

**swap2**(n);

swap the byte order of a number (like it would be a 16bits integer)

**swap4**(n);

swap the byte order of an integer

**swapfloat**(f);

swaps the byteorder of a float

## The blob class

The blob object is a buffer of arbitrary binary data. The object behaves like a file stream, it has a read/write pointer and it automatically grows if data is written out of his boundary.
A blob can also be accessed byte by byte through the `[]` operator.

**eos**();

returns a non null value if the read/write pointer is at the end of the stream.

**flush**();

flushes the stream.return a value != null if succeded, otherwise returns null

**len**();

returns the lenght of the stream

**readblob**(size);

read n bytes from the stream and retuns them as blob

**readn**(type);

reads a number from the stream according to the type pameter. *type* can have the following values:

| | | |
|---|---|---|
| 'l' | processor dependent, 32bits on 32bits processors, 64bits on 64bits prcessors | returns an integer |
| 'i' | 32bits number | returns an integer |
| 's' | 16bits signed integer | returns an integer |
| 'w' | 16bits unsigned integer | returns an integer |
| 'c' | 8bits signed integer | returns an integer |
| 'b' | 8bits unsigned integer | returns an integer |
| 'f' | 32bits float | returns an float |
| 'd' | 64bits float | returns an float |

**resize**(size);

resizes the blob to the specified *size*

**seek**(seek, [origin]);

Moves the read/write pointer to a specified location. *offset* indicates the number of bytes from *origin*. *origin* can be 'b' beginning of the stream,'c' current location or 'e' end of the stream. If origin is omitted the parameter is defaulted as 'b'(beginning of the stream).

**swap2**();

swaps the byte order of the blob content as it would be an array of 16bits integers

**swap4**();

swaps the byte order of the blob content as it would be an array of 32bits integers

**tell**();

returns read/write pointer absolute position

**writeblob**(blob);

writes a blob in the stream

```
writen(n, type);
```

writes a number in the stream formatted according to the type pameter. *type* can have the following values:

| | |
|---|---|
| 'i' | 32bits number |
| 's' | 16bits signed integer |
| 'w' | 16bits unsigned integer |
| 'c' | 8bits signed integer |
| 'b' | 8bits unsigned integer |
| 'f' | 32bits float |
| 'd' | 64bits float |

# C API

## Initialization

```
sqstd_register_bloblib
```

```
SQRESULT sqstd_register_bloblib(HSQUIRRELVM v);
```

initialize and register the blob library in the given VM.

parameters: *HSQUIRRELVM v* the target VM

return: an SQRESULT

remarks: The function aspects a table on top of the stack where to register the global library functions.

## Blob object

```
sqstd_getblob
```

```
SQRESULT sqstd_getblob(HSQUIRRELVM v, SQInteger idx, SQUserPointer * ptr);
```

retrieve the pointer of a blob's payload from an arbitrary position in the stack.

parameters:

| | |
|---|---|
| *HSQUIRRELVM v* | the target VM |
| *SQInteger idx* | and index in the stack |
| *SQUserPointer * ptr* | A pointer to the userpointer that will point to the blob's payload |

return: an SQRESULT

```
sqstd_getblobsize
```

```
SQInteger sqstd_getblobsize(HSQUIRRELVM v, SQInteger idx);
```

retrieve the size of a blob's payload from an arbitrary position in the stack.

parameters: *HSQUIRRELVM v* the target VM

*SQInteger idx* and index in the stack

return: the size of the blob at idx position

`sqstd_createblob`

```
SQUserPointer sqstd_createblob(HSQUIRRELVM v, SQInteger size);
```

creates a blob with the given payload size and pushes it in the stack.

parameters: *HSQUIRRELVM v* the target VM

*SQInteger size* the size of the blob payload that has to be created

return: a pointer to the newly created blob payload

# Chapter 4. The Math library

the math lib provides basic mathematic routines. The library mimics the C runtime library implementation.

# Squirrel API

## Global symbols

**abs**(x);

returns the absolute value of *x* as integer

**acos**(x);

returns the arccosine of *x*

**asin**(x);

returns the arcsine of *x*

**atan**(x);

returns the arctangent of *x*

**atan2**(x, y);

returns the arctangent of *y/x*.

**ceil**(x);

returns a float value representing the smallest integer that is greater than or equal to *x*

**cos**(x);

returns the cosine of *x*

**exp**(x);

returns the exponential value of the float parameter *x*

**fabs**(x);

returns the absolute value of *x* as float

**floor**(x);

returns a float value representing the largest integer that is less than or equal to *x*

**log**(x);

returns the natural logarithm of *x*

**log10**(x);

returns the logarithm base-10 of *x*

**pow**(x, y);

returns *x* raised to the power of *y*.

**rand**();

returns a pseudorandom integer in the range 0 to RAND_MAX

**sin**(x);

returns the sine of *x*

**sqrt**(x);

returns the square root of *x*

**srand**(seed);

sets the starting point for generating a series of pseudorandom integers

**tan**(x);

returns the tangent of *x*

PI

The numeric constant pi (3.141592) is the ratio of the circumference of a circle to its diameter

RAND_MAX

the maximum value that can be returned by the `rand()` function

# C API

## Initialization

```
sqstd_register_mathlib
```

```
SQRESULT sqstd_register_mathlib(HSQUIRRELVM v);
```

initialize and register the math library in the given VM.

parameters: *HSQUIRRELVM v* the target VM

return: an SQRESULT

remarks: The function aspects a table on top of the stack where to register the global library functions.

# Chapter 5. The System library

The system library exposes operating system facilities like environment variables, date time manipulation etc..

# Squirrel API

## Global symbols

**clock**();

returns a float representing the number of seconds elapsed since the start of the process

**date**([time], [format]);

returns a table containing a date/time splitted in the slots:

| | |
|---|---|
| sec | Seconds after minute (0 - 59). |
| min | Minutes after hour (0 - 59). |
| hour | Hours since midnight (0 - 23). |
| day | Day of month (1 - 31). |
| month | Month (0 - 11; January = 0). |
| year | Year (current year). |
| wday | Day of week (0 - 6; Sunday = 0). |
| yday | Day of year (0 - 365; January 1 = 0). |

if *time* is omitted the current time is used.
if *format* can be 'l' local time or 'u' UTC time, if omitted is defaulted as 'l'(local time).

**getenv**(varaname);

Returns a string containing the value of the environment variable *varname*

**remove**(path);

deletes the file specified by *path*

**rename**(oldname, newname);

renames the file or directory specified by *oldname* to the name given by *newname*

**system**(cmd);

executes the string *cmd* through the os command interpreter.

**time**();

returns the number of seconds elapsed since midnight 00:00:00, January 1, 1970.

the result of this function can be formatted through the faunction `date`

# C API

## Initialization

`sqstd_register_systemlib`

```
SQRESULT sqstd_register_systemlib(HSQUIRRELVM v);
```

initialize and register the system library in the given VM.

parameters: *`HSQUIRRELVM v`* the target VM

return: an SQRESULT

remarks: The function aspects a table on top of the stack where to register the global library functions.

# Chapter 6. The String library

the string lib implements string formatting and regular expression matching routines.

# Squirrel API

## Global symbols

**format**(formatstr, ...);

Returns a string formatted according *formatstr* and the optional parameters following it. The format string follows the same rules as the printf family of standard C functions( the "*" is not supported).

```
eg.
sq> print(format("%s %d 0x%02X\n","this is a test :",123,10));
this is a test : 123 0x0A
```

**lstrip**(str);

Strips white-space-only characters that might appear at the beginning of the given string and returns the new stripped string.

**regexp**(pattern);

compiles a regular expression pattern and returns it as a new regexp class instance.

| | |
|---|---|
| \ | Quote the next metacharacter |
| ^ | Match the beginning of the string |
| . | Match any character |
| $ | Match the end of the string |
| \| | Alternation |
| (subexp) | Grouping (creates a capture) |
| (?:subexp) | No Capture Grouping (no capture) |
| [] | Character class |

**GREEDY CLOSURES.**

| | |
|---|---|
| * | Match 0 or more times |
| + | Match 1 or more times |
| ? | Match 1 or 0 times |
| {n} | Match exactly n times |
| {n,} | Match at least n times |
| {n,m} | Match at least n but not more than m times |

**ESCAPE CHARACTERS.**

| | |
|---|---|
| \t | tab (HT, TAB) |
| \n | newline (LF, NL) |

| | |
|---|---|
| `\r` | return (CR) |
| `\f` | form feed (FF) |

**PREDEFINED CLASSES.**

| | |
|---|---|
| `\l` | lowercase next char |
| `\u` | uppercase next char |
| `\a` | letters |
| `\A` | non letters |
| `\w` | alphanumeric [_0-9a-zA-Z] |
| `\W` | non alphanumeric [^_0-9a-zA-Z] |
| `\s` | space |
| `\S` | non space |
| `\d` | digits |
| `\D` | non nondigits |
| `\x` | exadecimal digits |
| `\X` | non exadecimal digits |
| `\c` | control charactrs |
| `\C` | non control charactrs |
| `\p` | punctation |
| `\P` | non punctation |
| `\b` | word boundary |
| `\B` | non word boundary |

**`rstrip`**`(str);`

Strips white-space-only characters that might appear at the end of the given string and returns the new stripped string.

**`split`**`(str, separators);`

returns an array of strings split at each point where a separator character occurs in `str`. The separator is not returned as part of any array element. the parameter `separators` is a string that specifies the characters as to be used for the splitting.

```
eg.
local a = split("1.2-3;4/5",".-/;");
// the result will be  [1,2,3,4,5]
```

**`strip`**`(str);`

Strips white-space-only characters that might appear at the beginning or end of the given string and returns the new stripped string.

# Regexp class

The regexp object rapresent a precompiled regular experssion pattern. The object is created trough the function `regexp()`.

**capture**(str, [start]);

returns an array of tables containing two indexs("begin" and "end")of the first match of the regular expression in the string *str*. An array entry is created for each captured sub expressions. If no match occurs returns null. The search starts from the index *start* of the string, if *start* is omitted the search starts from the beginning of the string.

the first element of the returned array(index 0) always contains the complete match.

```
local ex = regexp(@"(\d+) ([a-zA-Z]+)(\p)");
local string = "stuff 123 Test;";
local res = ex.capture(string);
foreach(i,val in res)
{
print(format("match number[%02d] %s\n",
i,string.slice(val.begin,val.end))); //prints "Test"
}

...
will print
match number[00] 123 Test;
match number[01] 123
match number[02] Test
match number[03] ;
```

**match**(str);

returns a true if the regular expression matches the string *str*, otherwise returns false.

**search**(str, [start]);

returns a table containing two indexs("begin" and "end") of the first match of the regular expression in the string *str*, otherwise if no match occurs returns null. The search starts from the index *start* of the string, if *start* is omitted the search starts from the beginning of the string.

```
local ex = regexp("[a-zA-Z]+");
local string = "123 Test;";
local res = ex.search(string);
print(string.slice(res.begin,res.end)); //prints "Test"
```

# C API

## Initialization

sqstd_register_stringlib

SQRESULT **sqstd_register_stringlib**(HSQUIRRELVM v);

initialize and register the string library in the given VM.

parameters: *HSQUIRRELVM v* the target VM

return: an SQRESULT

remarks: The function aspects a table on top of the stack where to register the global library functions.

# Formatting

sqstd_format

```
SQRESULT sqstd_format(HSQUIRRELVM v, SQInteger nformatstringidx, SQInteger * outlen, SQChar ** output);
```

creates a new string formatted according to the object at positionnformatstringidx and the optional parameters following it. The format string follows the same rules as the printf family of standard C functions( the "*" is not supported).

parameters:

| | |
|---|---|
| *HSQUIRRELVM v* | the target VM |
| *SQInteger nformatstringidx* | index in the stack of the format string |
| *SQInteger * outlen* | a pointer to an integer that will be filled with the length of the newly created string |
| *SQChar ** output* | a pointer to a string pointer that will receive the newly created string |

return: an SQRESULT

remarks: the newly created string is allocated in the scratchpad memory.

# Regular Expessions

sqstd_rex_compile

```
SQRex * sqstd_rex_compile(const SQChar * pattern, const SQChar ** error);
```

compiles an expression and returns a pointer to the compiled version. in case of failure returns NULL.The returned object has to be deleted through the function sqstd_rex_free().

parameters:

| | |
|---|---|
| *const SQChar * pattern* | a pointer to a zero terminated string containing the pattern that has to be compiled. |
| *const SQChar ** error* | a pointer to a string pointer that will be set with an error string in case of failure. |

return: a pointer to the compiled pattern

sqstd_rex_free

```
void sqstd_rex_free(SQRex * exp);
```

deletes a expression structure created with sqstd_rex_compile()

parameters: *SQRex * exp* the expression structure that has to be deleted

`sqstd_rex_match`

```
SQBool sqstd_rex_match(SQRex * exp, const SQChar * text);
```

returns SQTrue if the string specified in the parameter text is an exact match of the expression, otherwise returns SQFalse.

parameters: *SQRex * exp* the compiled expression

*const SQChar * text* the string that has to be tested

return: SQTrue if successful otherwise SQFalse

`sqstd_rex_search`

```
SQBool sqstd_rex_search(SQRex * exp, const SQChar * text, const SQChar
** out_begin, const SQChar ** out_end);
```

searches the first match of the expressin in the string specified in the parameter text. if the match is found returns SQTrue and the sets out_begin to the beginning of the match and out_end at the end of the match; otherwise returns SQFalse.

parameters:

| | |
|---|---|
| *SQRex * exp* | the compiled expression |
| *const SQChar * text* | the string that has to be tested |
| *const SQChar ** out_begin* | a pointer to a string pointer that will be set with the beginning of the match |
| *const SQChar ** out_end* | a pointer to a string pointer that will be set with the end of the match |

return: SQTrue if successful otherwise SQFalse

`sqstd_rex_searchrange`

```
SQBool sqstd_rex_searchrange(SQRex * exp, const SQChar * text_begin,
const SQChar * text_end, const SQChar ** out_begin, const SQChar
** out_end);
```

searches the first match of the expressin in the string delimited by the parameter text_begin and text_end. if the match is found returns SQTrue and the sets out_begin to the beginning of the match and out_end at the end of the match; otherwise returns SQFalse.

parameters:

| | |
|---|---|
| *SQRex * exp* | the compiled expression |
| *const SQChar * text_begin* | a pointer to the beginnning of the string that has to be tested |
| *const SQChar * text_end* | a pointer to the end of the string that has to be tested |
| *const SQChar ** out_begin* | a pointer to a string pointer that will be set with the beginning of the match |

| | | |
|---|---|---|
| | *const SQChar ** out_end* | a pointer to a string pointer that will be set with the end of the match |

return: an SQRESULT

`sqstd_rex_getsubexpcount`

```
SQInteger sqstd_rex_getsubexpcount(SQRex * exp);
```

returns the number of sub expressions matched by the expression

parameters: *SQRex * exp* the compiled expression

return: the number of sub expressions matched by the expression

`sqstd_rex_getsubexp`

```
SQInteger sqstd_rex_getsubexp(SQRex * exp, SQInteger n, SQRexMatch * subexp);
```

retrieve the begin and and pointer to the length of the sub expression indexed by n. The result is passed trhough the struct SQRexMatch.

| | | |
|---|---|---|
| parameters: | *SQRex * exp* | the compiled expression |
| | *SQInteger n* | the index of the submatch(0 is the complete match) |
| | *SQRexMatch * subexp* | a pointer to structure that will store the result |

return: the function returns SQTrue if n is valid index otherwise SQFalse.

# Chapter 7. The Aux library

The aux library implements default handlers for compiler and runtime errors and a stack dumping.

# C API

## Error handling

`sqstd_seterrorhandlers`

```
void sqstd_seterrorhandlers(HSQUIRRELVM v);
```

initialize compiler and runtime error handlers, the handlers use the print function set through(sq_setprintfunc) to output the error.

parameters: *`HSQUIRRELVM v`* the target VM

`sqstd_printcallstack`

```
void sqstd_printcallstack(HSQUIRRELVM v);
```

print the call stack and stack contents.the function uses the print function set through(sq_setprintfunc) to output the stack dump.

parameters: *`HSQUIRRELVM v`* the target VM

# Index

## T



## W