Squirrel Standard Library

Release 3.1 stable

Alberto Demichelis

April 03, 2016

CONTENTS

CHAPTER

ONE

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
- aux : auxiliary functions

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

CHAPTER

TWO

THE INPUT/OUTPUT LIBRARY

the i/o library 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

The file class

The file object implements a stream on a operating system file.

class file (*path*, *patten*)
It's contructor imitates 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.

`file.`**`close`**`()`
closes the file.

`file.`**`eos`**`()`
returns a non null value if the read/write pointer is at the end of the stream.

`file.`**`flush`**`()`
flushes the stream.return a value != null if succeded, otherwise returns null

`file.`**`len`**`()`
returns the lenght of the stream

`file.`**`readblob`**`(`*size*`)`

Arguments

- **`size`** (*int*) – number of bytes to read

read n bytes from the stream and retuns them as blob

`file.`**`readn`**`(`*type*`)`

Arguments

- **`type`** (*int*) – type of the number to read

reads a number from the stream according to the type pameter.

type can have the following values:

parameter	return description	return type
'l'	processor dependent, 32bits on 32bits processors, 64bits on 64bits prcessors	integer
'i'	32bits number	integer
's'	16bits signed integer	integer
'w'	16bits unsigned integer	integer
'c'	8bits signed integer	integer
'b'	8bits unsigned integer	integer
'f'	32bits float	float
'd'	64bits float	float

`file.`**`resize`**`(`*size*`)`

Arguments

- **`size`** (*int*) – the new size of the blobl in bytes

resizes the blob to the specified *size*

`file.`**`seek`**`(`*offset*[, *origin*]`)`

Arguments

- **`offset`** (*int*) – indicates the number of bytes from *origin*.
- **`origin`** (*int*) – origin of the seek

'b'	beginning of the stream
'c'	current location
'e'	end of the stream

Moves the read/write pointer to a specified location.

Note: If origin is omitted the parameter is defaulted as 'b'(beginning of the stream).

`file.`**`tell`**`()`
returns the read/write pointer absolute position

`file.`**`writeblob`**`(`*src*`)`

Arguments

- **`src`** (`blob`) – the source blob containing the data to be written

writes a blob in the stream

`file.`**`writen`**`(`*n*, *type*`)`

Arguments

- **`n`** (*`number`*) – the value to be written
- **`type`** (*`int`*) – type of the number to write

writes a number in the stream formatted according to the *type* pameter

type can have the following values:

parameter	return description
'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

SQRESULT **`sqstd_register_iolib`**(HSQUIRRELVM *v*)

Parameters

- **`v`** (*`HSQUIRRELVM`*) – the target VM

Returns an SQRESULT

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

initialize and register the io library in the given VM.

File Object

SQRESULT **`sqstd_createfile`**(HSQUIRRELVM *v*, SQFILE *file*, SQBool *owns*)

Parameters

- **`v`** (*`HSQUIRRELVM`*) – the target VM
- **`file`** (*`SQFILE`*) – the stream that will be rapresented by the file object

- **owns** (*SQBool*) – if different true the stream will be automatically closed when the newly create file object is destroyed.

Returns an SQRESULT

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

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

Parameters

- **v** (*HSQUIRRELVM*) – the target VM
- **idx** (*SQInteger*) – and index in the stack
- **file** (*SQFILE**) – A pointer to a SQFILE handle that will store the result

Returns an SQRESULT

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

Script loading and serialization

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

Parameters

- **v** (*HSQUIRRELVM*) – the target VM
- **filename** (*SQChar**) – path of the script that has to be loaded
- **printerror** (*SQBool*) – if true the compiler error handler will be called if a error occurs

Returns an SQRESULT

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.

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

Parameters

- **v** (*HSQUIRRELVM*) – the target VM
- **filename** (*SQChar**) – path of the script that has to be loaded
- **retval** (*SQBool*) – if true the function will push the return value of the executed script in the stack.
- **printerror** (*SQBool*) – if true the compiler error handler will be called if a error occurs

Returns 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.

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.

```
    sq_pushroottable(v); //push the root table(were the globals of the script will are stored)
    sqstd_dofile(v, _SC("test.nut"), SQFalse, SQTrue);// also prints syntax errors if any
```

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

Parameters

- **v** (*HSQUIRRELVM*) – the target VM
- **filename** (*SQChar**) – destination path of serialized closure

Returns an SQRESULT

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.

CHAPTER

THREE

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

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(*n*)
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.

class blob(*size*)

Arguments

- **size** (*int*) – initial size of the blob

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

blob.**eos**()
returns a non null value if the read/write pointer is at the end of the stream.

blob.**flush**()
flushes the stream.return a value != null if succeded, otherwise returns null

blob.**len**()
returns the lenght of the stream

blob.**readblob**(*size*)

Arguments

- **size** (*int*) – number of bytes to read

read n bytes from the stream and retuns them as blob

blob.**readn**(*type*)

Arguments

- **type** (*int*) – type of the number to read

reads a number from the stream according to the type pameter.

type can have the following values:

parameter	return description	return type
'l'	processor dependent, 32bits on 32bits processors, 64bits on 64bits prcessors	integer
'i'	32bits number	integer
's'	16bits signed integer	integer
'w'	16bits unsigned integer	integer
'c'	8bits signed integer	integer
'b'	8bits unsigned integer	integer
'f'	32bits float	float
'd'	64bits float	float

blob.**resize**(*size*)

Arguments

- **size** (*int*) – the new size of the blobl in bytes

resizes the blob to the specified *size*

blob.**seek**(*offset*[, *origin*])

Arguments

- **offset** (*int*) – indicates the number of bytes from *origin*.
- **origin** (*int*) – origin of the seek

'b'	beginning of the stream
'c'	current location
'e'	end of the stream

Moves the read/write pointer to a specified location.

Note: If origin is omitted the parameter is defaulted as 'b'(beginning of the stream).

---

blob.**swap2**()
swaps the byte order of the blob content as it would be an array of *16bits integers*

blob.**swap4**()
swaps the byte order of the blob content as it would be an array of *32bits integers*

blob.**tell**()
returns the read/write pointer absolute position

blob.**writeblob**(*src*)

Arguments

- **src** (blob) – the source blob containing the data to be written

writes a blob in the stream

blob.**writen**(*n*, *type*)

Arguments

- **n** (*number*) – the value to be written
- **type** (*int*) – type of the number to write

writes a number in the stream formatted according to the *type* pameter

type can have the following values:

parameter	return description
'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

SQRESULT **sqstd_register_bloblib**(HSQUIRRELVM *v*)

Parameters

- **v** (*HSQUIRRELVM*) – the target VM

Returns an SQRESULT

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

initializes and registers the blob library in the given VM.

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

Parameters

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

Returns an SQRESULT

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

SQInteger **sqstd_getblobsize**(HSQUIRRELVM *v*, SQInteger *idx*)

Parameters

- **v** (*HSQUIRRELVM*) – the target VM
- **idx** (*SQInteger*) – and index in the stack

Returns the size of the blob at *idx* position

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

SQUserPointer **sqstd_createblob**(HSQUIRRELVM *v*, SQInteger *size*)

Parameters

- **v** (*HSQUIRRELVM*) – the target VM
- **size** (*SQInteger*) – the size of the blob payload that has to be created

Returns a pointer to the newly created blob payload

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

CHAPTER

FOUR

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 an 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 *x*/*y*

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 a 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*)
: rreturns 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*

RAND_MAX
: the maximum value that can be returned by the *rand()* function

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

C API

SQRESULT **sqstd_register_mathlib**(HSQUIRRELVM *v*)

Parameters

- **v** (*HSQUIRRELVM*) – the target VM

Returns an SQRESULT

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

initializes and register the math library in the given VM.

CHAPTER

FIVE

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*)
: xecutes 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 function *date()*

C API

SQRESULT **sqstd_register_systemlib**(HSQUIRRELVM *v*)

Parameters

- **v** (*HSQUIRRELVM*) – the target VM

Returns an SQRESULT

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

initialize and register the system library in the given VM.

CHAPTER

SIX

THE STRING LIBRARY

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

Squirrel API

Global Symbols

endswith (*str*, *cmp*)
: returns *true* if the end of the string *str* matches a the string *cmp* otherwise returns *false*

ecape (*str*)
: Returns a string with backslashes before characters that need to be escaped(*",a,b,t,n,v,f,r,\,",',0,xnn*).

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.

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*, *separtators*)
: 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]
```

startswith (*str*, *cmp*)
: returns *true* if the beginning of the string *str* matches a the string *cmp* otherwise returns *false*

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.

The regexp class

class regexp (*pattern*)

The regexp object rapresent a precompiled regular experssion pattern. The object is created trough *regexp(patern)*.

\	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 characters
\C	non control characters
\p	punctation
\P	non punctation
\b	word boundary
\B	non word boundary

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] ;
```

regexp.**match**(*str*)

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

regexp.**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

SQRESULT **sqstd_register_stringlib**(HSQUIRRELVM *v*)

Parameters

- **v** (*HSQUIRRELVM*) – the target VM

Returns an SQRESULT

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

initialize and register the string library in the given VM.

Formatting

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

Parameters

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

- **output** (*SQChar***) – a pointer to a string pointer that will receive the newly created string

Returns an SQRESULT

Remarks the newly created string is allocated in the scratchpad memory.

creates a new string formatted according to the object at position *nformatstringidx* 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).

Regular Expessions

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

Parameters

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

Returns a pointer to the compiled pattern

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().

void **sqstd_rex_free**(SQRex * *exp*)

Parameters

- **exp** (*SQRex**) – the expression structure that has to be deleted.

deletes a expression structure created with sqstd_rex_compile()

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

Parameters

- **exp** (*SQRex**) – a compiled expression
- **text** (*SQChar**) – the string that has to be tested

Returns SQTrue if successful otherwise SQFalse

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

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

Parameters

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

Returns SQTrue if successful otherwise SQFalse

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.

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

Parameters

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

Returns SQTrue if successful otherwise SQFalse

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.

SQInteger **sqstd_rex_getsubexpcount**(SQRex * *exp*)

Parameters

- **exp** (*SQRex**) – a compiled expression

Returns the number of sub expressions matched by the expression

returns the number of sub expressions matched by the expression

SQBool **sqstd_rex_getsubexp**(SQRex * *exp*, SQInteger *n*, SQRexMatch **subexp*)

Parameters

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

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

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

CHAPTER

SEVEN

THE AUX LIBRARY

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

C API

void **sqstd_seterrorhandlers** (HSQUIRRELVM *v*)

Parameters

- **v** (*HSQUIRRELVM*) – the target VM

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

void **sqstd_printcallstack** (HSQUIRRELVM *v*)

Parameters

- **v** (*HSQUIRRELVM*) – the target VM

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

T

W