Squirrel Shell 1.2.3

User Manual

Constantin “Dinosaur” Makshin

Squirrel Shell

Table of Contents

Overview

Squirrel Shell (shorter name – **squirrelsh**) is made as a cross-platform alternative to system shells like *bash* in *nix and *command.com* (*cmd.exe*) in Microsoft Windows. It is based on Squirrel scripting language.

Cross-platform nature of Squirrel Shell lets users write one script and use it everywhere instead of writing several scripts for doing the same thing, but in different OSes.

As Squirrel is a generic scripting language, this gives users more power as they aren't limited to functionality, specific to some dedicated purpose.

Starting from version 1.0rc1, this shell supports interactive mode. Squirrel Shell 1.0rc2 and newer support input, output and error streams redirection for child processes.

This version uses slightly enhanced of Squirrel (with support of octal integer numbers).

Language Features

- object-oriented programming;
- C++-like syntax;
- dynamic typing;
- delegation;
- generators;
- exception handling;
- weak references;
- more on http://www.squirrel-lang.org.

Examples

Simple script that outputs “Hello, world!” text:

squirrelsh	`#!/usr/bin/squirrelsh`
	`// Output simple text message`
	`printl("Hello, world!");`

bash	`#!/bin/bash`
	`# Output simple text message`
	`echo "Hello, world!"`

command.com	`@echo off`
	`rem Output simple text message`
	`echo Hello, world!`

Another simple script that runs external program:

squirrelsh	`#!/usr/bin/squirrelsh`
	`// Run external program`
	`run("foo");`

bash	`#!/bin/bash`
	`# Run external program`
	`foo`

command.com	`@echo off`
	`rem Run external program`
	`foo`

More complex script that multiplies two 2x2 matrices (this is an example only, you can use any size you want):

squirrelsh

```
#!/usr/bin/squirrelsh
// Multiply two 2x2 matrices
function matrix (w, h)
{
   local m = array(h);
   for (local i = 0; i < h; i++)
      m[i] = array(w, 0);
   return m;
}
local m1 = matrix(2, 2),
      m2 = matrix(2, 2),
      r  = matrix(2, 2);
r[0][0] = m1[0][0] * m2[0][0] + m1[0][1] * m2[1][0];
r[0][1] = m1[0][0] * m2[0][1] + m1[0][1] * m2[1][1];
r[1][0] = m1[1][0] * m2[0][0] + m1[1][1] * m2[1][0];
r[1][1] = m1[1][0] * m2[0][1] + m1[1][1] * m2[1][1];
```

bash

```
#!/bin/bash
# Multiply two 2x2 matrices
# It's possible to use matrices of any size
# Based on code by Charles Duffy
# matrix (w, h)
function matrix()
{
   local i=0
   local current_array=( )
   while [ “$i” -lt “$(($1 * $2))” ]; do
      current_array[$i]=0
      i=$((i + 1))
   done
   echo “${current_array[@]}”
}
# m_idx (w, y, x)
function m_idx()
{
   echo “$(($1 * $2 + $3))”
}
m1=(`matrix 2 2`)
m2=(`matrix 2 2`)
r=(`matrix 2 2`)
r[$(m_idx 2 0 0)]=$((${m1[$(m_idx 2 0 0)]} * ${m2[$(m_idx 2 0 \
0)]} + ${m1[$(m_idx 2 0 1)]} * ${m2[$(m_idx 2 1 0)]}))
r[$(m_idx 2 0 1)]=$((${m1[$(m_idx 2 0 0)]} * ${m2[$(m_idx 2 0 \
1)]} + ${m1[$(m_idx 2 0 1)]} * ${m2[$(m_idx 2 1 1)]}))
r[$(m_idx 2 1 0)]=$((${m1[$(m_idx 2 1 0)]} * ${m2[$(m_idx 2 0 \
0)]} + ${m1[$(m_idx 2 1 1)]} * ${m2[$(m_idx 2 1 0)]}))
r[$(m_idx 2 1 1)]=$((${m1[$(m_idx 2 1 0)]} * ${m2[$(m_idx 2 0 \
1)]} + ${m1[$(m_idx 2 1 1)]} * ${m2[$(m_idx 2 1 1)]}))
```

command.com

```
@rem This is quiet impossible
```

First line (*`#!/usr/bin/squirrelsh`*) is necessary for ability to run scripts directly from your *nix shell in manner like `./foo.nut` instead of `squirrelsh foo.nut` and is ignored by **Squirrel Shell**.

License Agreement

Squirrel Shell is distributed under terms of GNU GPL license. Look file “COPYING” for details.

PCRE is distributed under terms of BSD license. See file "COPYING-pcre" for details.

Squirrel is distributed under terms of zlib license. See file "COPYING-squirrel" for details.

Parts of zlib library are distributed under terms of zlib license. See file "COPYING-zlib" for details.

Installation

Linux

If you have downloaded self-executable installer (i.e. squirrelsh-1.2.2.run), then just do “sh ./squirrelsh-1.2.2.run”. This operation may need root privileges.

For source archive unpack its contents to some temporary directory and do “./configure && make” (run “./configure –help” to see available compilation options). After compilation process completes without any errors, do “make install”. Root privileges may be necessary, too.

Microsoft Windows

Run installer and follow its instructions.

If you have downloaded source archive and want to compile shell by yourself, there are Visual Studio 2005 solution “squirrelsh.sln” and Visual Studio .NET 2003 solution “squirrelsh-msvs7.1.sln”. Makefiles “Makefile.mingw” for MinGW and “Makefile.msvc” for NMake can be used as well.

Scripting Reference

Notes

Squirrel is language with dynamic typing, but in this document these types are used:

- `array` – array of valued of specified type;
- `bool` – boolean value;
- `handle` – used only in file I/O functions and contains handle of opened file;
- `integer` – integer number;
- `float` – floating point number;
- `string` – text string;
- `string_array` – array of `string` values;
- `variant` – value whose type depends on some circumstances.

Detailed description of types mentioned above (except `string_array`) can be found in **Squirrel Reference Manual**.

All functions that expect paths as their parameters accept them both in *nix format (with slashes as delimiters) and in DOS format (with backslashes as delimiters). But paths that are returned by functions are in *nix format unless format is specified explicitly.

Long filenames are supported, but full paths cannot be longer than 260 characters. This number is the limit for most of file-related functions in Windows API.

In *nix systems, if you're running script directly from your shell, line feeds must be in *nix format (LF). This is a limitation of OS and I can't change it. Else your system will complain that it cannot find script interpreter.

Use of functions and values marked as deprecated is not recommended since they may be removed at any time. Consider updating your scripts.

Functions

acos

Synopsis

```
float acos (float x);
```

Description

Calculate arccosine of `x`.

Return Values

The function returns arccosine of `x`. The returned angle is in radians.

adler32

Synopsis

```
string adler32 (string path[, integer from = 0[, integer to = -1]]);
```

Description

Calculates Adler32 checksum for block of file `path`. This block begins at `from` and ends at `to`.

Parameters

`path`

Path to file whose block's checksum should be calculated.

`from`

First byte (offset) of block for checksum calculation. If this parameter is less than or equal to zero, data is read from the file beginning.

`to`

Last byte (offset) of block for checksum calculation. If this parameter is less than zero or greater than size of the file, data is read until end of file is reached. If this parameter is greater than zero, but less than or equal to `from`, the function fails.

Return Values

If the function succeeds, it returns string representation of calculated checksum value in hexadecimal. Else *"00000000"* (string of zeroes) is returned.

asin

Synopsis

```
float asin (float x);
```

Description

Calculates arcsine of `x`.

Return Values

The return value is the arcsine of `x`. The returned angle is in radians.

atan

Synopsis

```
float atan (float x);
```

Description

Calculates arctangent of x.

Return Values

The return value is arctangent of x measured in radians.

ceil

Synopsis

```
float ceil (float x);
```

Description

Finds the smallest integer greater than or equal to x.

Return Values

The function returns the smallest integer greater than or equal to x.

chdir

Synopsis

```
bool chdir (string path);
```

Description

Sets current working directory. This function is an equivalent to *cd* command in other shells.

Parameters

`path`

Path to directory that should be made current.

Return Values

If the function succeeds, it returns *true*. Else the return value is *false*.

chgrp

Synopsis

```
chgrp (string path, string group);
```

Description

Changes group of file or directory. This function is an equivalent to *chgrp* command in *nix.

Parameters

`path`

Path to file or directory whose group should be changed.

`group`

Name of new group that owns `path`.

chmod

Synopsis

```
chmod (string path, integer mode);
```

```
chmod (string path, string mode); (DEPRECATED)
```

Description

Changes access permissions of file or directory. This function is an equivalent to *chmod* command in *nix.

Parameters

`path`

Path to file or directory whose permissions should be changed.

`mode`

New access permissions that should be set to `path`. If the second form of the function is used, this parameter must contain text representation of octal mode.

chown

Synopsis

```
chown (string path, string owner);
```

Description

Changes owner of file or directory. This function is an equivalent to *chown* command in *nix.

Parameters

`path`

Path to file or directory whose owner should be changed.

`owner`

New owner of `path`.

clear

Synopsis

```
clear ();
```

Description

Clears console. This is an equivalent to *clear* command in *nix and *cls* in Microsoft Windows.

convpath

Synopsis

```
string convpath (string path, bool toNative);
```

Description

Converts path to OS's native or *nix format.

Parameters

`path`

Path that should be converted.

`toNative`

If this parameter is *true*, path should be converted to OS's native format. Else path should be converted to *nix format.

Return Values

If the function succeeds, the return value contains converted path.

If the function fails, the return value is *null*.

copy

Synopsis

```
copy (string source, string destination[, bool overwrite = false]);
```

Description

Copies one file to another. This function is an equivalent to *cp* command in *nix and *copy* in Microsoft Windows.

Parameters

`source`

Path to source file. Must not point to a directory.

`destination`

Path to destination file. Must not point to a directory, but name must be in existing one.

`overwrite`

If this parameter is *false* and destination file already exists, the function will return without copying any data. Else existing files will be overwritten.

cos

Synopsis

```
float cos (float x);
```

Description

Calculates cosine of `x`.

Parameters

`x`

Angle in radians whose cosine should be calculated.

Return Values

The return value is the cosine of `x`.

cosh

Synopsis

```
float cosh (float x);
```

Description

Calculates hyperbolic cosine of x.

Parameters

x

Angle in radians whose hyperbolic cosine should be calculated.

Return Values

The function returns the hyperbolic cosine of x.

cpuarch

Synopsis

```
string cpuarch ();
```

Description

Get CPU architecture shell was compiled for.

Return Values

The function returns one of these values:

- “alpha” – DEC Alpha;
- “arm” – ARM;
- “hppa” – HP/PA RISC;
- “ia64” – Intel Itanium;
- “mips” – MIPS;
- “ppc” – PowerPC;
- “sparc” – SPARC;
- “x86” – 16-bit or 32-bit Intel or compatible;
- “x64” – 64-bit AMD or compatible;
- “unknown” – none of the above.

This function never fails.

crc32

Synopsis

```
string crc32 (string path[, integer from = 0[, integer to = -1]]);
```

Description

Calculates CRC32 checksum for block of file `path`. This block begins at `from` and ends at `to`.

Parameters

Parameters are the same as for `adler32()` function.

Return Values

Return values are the same as for `adler32()` function.

deg2rad

Synopsis

```
float deg2rad (float x);
```

Description

Converts angle from degrees to radians.

Parameters

`x`

Angle in degrees that should be converted.

Return Values

The function returns converted angle.

delenv

Synopsis

```
delenv (string name);
```

Description

Deletes specified environment variable.

Parameters

`name`

Name of environment variable that should be deleted.

Notes

Staring with version 1.2.1, changes made by this function are not permanent, i.e. system environment variables are not modified and all changes are lost when **Squirrel Shell** is closed.

exist

Synopsis

```
bool exist (string path);
```

Description

Checks whether the file or directory at `path` exists or not.

Parameters

`path`

Path that should be checked for existence.

Return Values

If the file or directory at `path` exists, the function returns *true*. Else the return value is *false*.

exit

Synopsis

```
exit ([integer code = 0]);
```

Description

Closes the shell with specified exit code.

exp

Synopsis

```
float exp (float x);
```

Description

Calculates exponential value of `x`.

Return Values

The function returns the exponential value of `x`.

fclose

Synopsis

```
fclose (handle file);
```

Description

Closes opened file.

Parameters

`file`

Handle of file that should be closed. This must be handle returned by `fopen()` function.

fcloseall

Synopsis

```
fcloseall ();
```

Description

Closes all opened files.

Notes

All files that remain opened after the script exits are closed automatically. Though, leaving files opened while they aren't needed anymore isn't recommended, as this leads to waste of memory and is a bad form in programming.

fileext

Synopsis

```
string fileext (string path);
```

Description

Extracts file's extension from path to that file.

Parameters

`path`

Path to file whose extension should be extracted.

Return Values

The function returns extracted extension without period that separates file's name and extension. For example, call `fileext("./i/am/the.file")` will return "file".

filename

Synopsis

```
string filename (string path[, bool ext = false]);
```

Description

Extracts file's name from path to that file. Returned string may or may not contain file's extension, depending on value of ext parameter.

Parameters

`path`

Path to file whose name should be extracted.

`ext`

If this parameter is *true*, the return value will contain file's extension (`filename("./i/am/the.file", true)` will return "the.file"). Else only name will be returned (call `filename("./i/am/the.file")` will result in "the").

Return Values

The function returns name with or without extension of file specified by `path`.

filepath

Synopsis

```
string filepath (string path);
```

Description

Extracts path to directory with file (everything before last path delimiter).

Parameters

`path`

Path to file whose directory's path should be extracted.

Return Values

The function returns path to directory which contains file `path` without trailing delimiter. For example, call `filepath("./i/am/the.file")` will return "./i/am".

filetime

Synopsis

```
integer filetime (string path[, integer which = MODIFICATION]);
```

Description

Retrieves date and time of creation, last access, last modification or last change of directory entry `path`.

Parameters

`path`

Path to directory entry whose date and time should be retrieved.

`which`

Specifies which date and time should be retrieved. Can be one of the following values:

`CREATION` – date and time when the entry was created;

`ACCESS` – date and time when the entry was last accessed;

`MODIFICATION` – date and time when the entry contents were modified;

`CHANGE` – date and time when the entry properties (like owner, group, etc.) was changed.

Return Values

If the function succeeds, it returns date and time when specified action was performed on directory entry `path`. The returned time is in UTC. To convert it to local time, pass this value to `utctolocal()` function.

If the function fails, the return value is *`0`* (zero).

filetype

Synopsis

```
integer filetype (string path);
```

Description

Retrieves type of directory entry `path`.

Parameters

`path`

Path to directory entry whose type should be retrieved.

Return Values

The function returns one of these values:

`0` – the function failed;

`DIR` – path points to a directory;

`FILE` – path points to a file (or symbolic link in *nix).

floor

Synopsis

```
float floor (float x);
```

Description

Find largest integer less than or equal to `x`.

Return Values

The function returns the largest integer less than or equal to `x`.

fopen

Synopsis

```
handle fopen (string path, string mode);
```

Description

Opens file.

Parameters

`path`

Path to file that should be opened.

`mode`

Access mode. These modes are supported:

`READ_ONLY` – open for reading only (the file must exist);

`WRITE_ONLY` – open for writing only;

`READ_WRITE` – open for both reading and writing (the file must exist);

`APPEND` – open for writing data to the end of file.

Modes from the *fopen()* function in *libc* (“`r`”, “`w`”, “`a`”, “`r+`”, “`w+`”, “`a+`”) are also supported, but using values from the list above is encouraged.

Return Values

If the function succeeds, the return value will contain handle of opened file. Else *null* is returned.

fprint

Synopsis

```
fprint (handle file, string text);
```

Description

Outputs text to file. File must be opened with write-enabled access mode.

Parameters

`file`

Handle of file text should be written to.

`text`

Text that should be written to file.

fprintl

Synopsis

```
fprintl (handle file[, string text]);
```

Description

Outputs text with trailing linefeed to file. File must be opened with write-enabled access mode.

Parameters

`file`

Handle of file text should be written to.

`text`

Text that should be written to file. If this parameter is omitted, empty line is printed to file.

fscan

Synopsis

```
variant fscan (handle file[, integer type = TEXT]);
```

Description

Reads data from file. File must be opened with read-enabled access mode.

Parameters

`file`

Handle of file data should be read from.

`type`

Type of data that should be read from file. Following values are supported:

`TEXT` – single line of text (default);

`CHAR` – single character;

`INT` – signed integer number;

`UINT` – unsigned integer number;

`FLOAT` – floating point number.

Values `'t'`, `'c'`, `'i'`, `'u'` and `'f'` are deprecated.

Return Values

If valid type was specified and data was read successfully, the function returns value of requested type. Else *null* is returned.

getcwd

Synopsis

```
string getcwd ();
```

Description

Gets current working directory.

Return Values

The function returns path to current working directory.

getenv

Synopsis

```
string getenv (string name);
```

Description

Gets value of specified environment variable.

Parameters

`name`

Name of environment variable whose value should be retrieved.

Returned Values

If the function succeeds, the return value is the value of specified environment variable. Else *null* is returned.

localtime

Synopsis

```
integer localtime ();
```

Description

Retrieves current local time.

Return Values

This function returns current local time.

localtoutc

Synopsis

```
integer localtoutc (integer time);
```

Description

Converts local date and time to UTC.

Parameters

`time`

Local date and time that should be converted to UTC.

Return Values

The function returns local date and time converted to UTC.

log

Synopsis

```
float log (float x);
```

Description

Calculates the natural logarithm of `x`.

Return Values

The function returns the value of *ln(x)*.

log10

Synopsis

```
float log10 (float x);
```

Description

Calculates base-10 logarithm of `x`.

Return Values

The function returns the value of *lg(x)*.

md5

Synopsis

```
string md5 (string path[, integer from = 0[, integer to = -1]]);
```

Description

Calculates MD5 hash for block of file `path`. This block begins at `from` and ends at `to`.

Parameters

Parameters are the same as for `adler32()` function.

Return Values

Return values are the same as for `adler32()` function, except the string returned by this function has length of 32 characters instead of 8.

mkdir

Synopsis

```
bool mkdir (string path[, integer mode = 0755]);
```

```
bool mkdir (string path[, string mode = "755"]);
```
(DEPRECATED)

Description

Creates a directory. This function is an equivalent to *mkdir* command in other shells.

Parameters

`path`

Path to new directory that should be created.

`mode`

Access permissions that should be set to created directory. If the second form of the function is used, this parameter must contain text representation of octal mode.

Return Values

If the function succeeds, the return value is *true*. Else the function returns *false*.

mktime

Synopsis

```
integer mktime (integer year, integer month, integer day, integer hour, integer minute, integer second);
```

Description

Converts date and time to integer number that can be used in other time-related functions of this shell.

Parameters

`year, month, day`

Date that should be converted. It must be between January 1, 1980 and February 1, 2068 inclusive. Values outside this range will be clamped.

`hour, minute, second`

Time that should be converted. It must be between 00:00:00 and 23:59:59 inclusive. Values outside this range will be clamped.

Return Values

This function returns date and time converted to integer number.

move

Synopsis

```
move (string source, string destination);
```

Description

Copies data from source to destination and deletes source file. This function is an equivalent to *mv* command in *nix and *move* one in Microsoft Windows.

Parameters

`source`

Path to file data should be copied from. After successful data transfer this file will be deleted. Must not point to a directory.

`destination`

Path to file data should be written to. Name in existing directory must be specified. Must not point to a directory.

pathenv

Synopsis

```
string pathenv (string path1[, string path2[, ...]]);
```

Description

Build value for “PATH” environment variable. Using this function is highly recommended as it converts all paths into OS's native format and takes different entry delimiters (colon in *nix and semicolon in Microsoft Windows) into account.

Parameters

`path*`

Directory entries to be included in “PATH” environment variable.

Return Values

The function returns text string that can be set as “PATH” environment variable with `setenv()` function (see below).

platform

Synopsis

```
string platform ();
```

Description

Get platform shell was compiled for.

Return Values

The function returns one of these values:

- “bsd” – “generic” variant of BSD;
- “cygwin32” – 32-bit Cygwin;
- “freebsd” – FreeBSD;
- “hpux” – HP-UX;
- “hurd” – GNU Hurd;
- “linux” – 32-bit Linux;
- “linux64” – 64-bit Linux;
- “macintosh” – old versions of Mac OS;
- “macosx” – Mac OS X;
- “mingw32” – 32-bit MinGW;
- “msdos” – MS-DOS;
- “netbsd” – NetBSD;
- “next” – NeXT (NeXTSTEP);

- “openbsd” – OpenBSD;
- “os2” – OS/2;
- “qnx” – QNX;
- “solaris” – Solaris;
- “sunos” – SunOS;
- “symbian” – Symbian;
- “vms” – VMS;
- “win32” – 32-bit Microsoft Windows;
- “win64” – 64-bit Microsoft Windows;
- “unknown” – none of the above.

This function never fails.

pow

Synopsis

```
float pow (float x, float y);
```

Description

Calculates `x` raised to power `y`.

Return Values

The function returns x^y.

print

Synopsis

```
print (string text);
```

Description

Outputs `text` to console.

Parameters

`text`

Text that should be output to console.

printl

Synopsis

```
printl ([string text]);
```

Description

Outputs `text` with terminating linefeed to console. This function is equivalent to `print(text + “\n”)`.

Parameters

`text`

Text that should be output to console. If this parameter is omitted, empty line is printed to console.

rad2deg

Synopsis

```
float rad2deg (float x);
```

Description

Converts angle from radians to degrees.

Parameters

`x`

Angle in radians that should be converted to degrees.

Return Values

The return value is angle `x` in degrees.

readdir

Synopsis

```
string_array readdir (string path);
```

Description

Lists entries in specified directory.

Parameters

`path`

Path to directory whose contents should be retrieved.

Return Values

The function returns array of directory's entries names.

regcompile

Synopsis

```
handle regcompile (string pattern[, bool caseSensitive = true[, bool multiLine = false]]);
```

Description

Compiles regular expression for further matching. Pattern must be specified using Perl regular expressions syntax.

Parameters

`pattern`

Regular expression in Perl-compatible format.

`caseSensitive`

If this is *true*, letters in the pattern match both upper and lower case

letters. It is equivalent to Perl's /i option, and it can be changed within a pattern by a (?i) option setting.

`multiLine`

If this is *true*, the "start of line" and "end of line" constructs match immediately following or immediately before internal newlines in the subject string, respectively, as well as at the very start and end. This is equivalent to Perl's /m option, and it can be changed within a pattern by a (?m) option setting. If there are no newlines in a subject string, or no occurrences of ^ or $ in a pattern, this flag has no effect.

Return Values

On success, the function returns handle of the compiled regular expression. Else *null* is returned.

regerror

Synopsis

```
string regerror ();
```

Description

Returns description of last occurred regular expression-related error.

regfree

Synopsis

```
regfree (handle regExp);
```

Description

Frees resources used by regular expression.

Parameters

`regExp`

Handle of regular expression returned by `regcompile()` function.

regfreeall

Synopsis

```
regfree ();
```

Description

Frees resources used by all compiled regular expressions.

Notes

All regular expressions are freed automatically when the shell closes. However, leaving resources not freed when they are no longer needed should be avoided.

regmatch

Synopsis

```
array regmatch (handle regExp, string text[, integer startOffset = 0[, bool partial = false]]);

array regmatch (string pattern, string text[, integer startOffset = 0[, bool partial = false]]);
```

Description

Matches regular expression against text.

Parameters

`regExp`

: Handle of regular expression returned by `regcompile()` function.

`pattern`

: Regular expression in Perl-compatible format.

`text`

: Text that should be matched against regular expression.

`startOffset`

: Offset of the first character of `text` from which matching should be started.

`partial`

: If this is *true*, partial matching is performed.

Return Values

On success, the function returns array of two-element integer arrays containing matched sub-strings. The first element of each sub-array specifies start of matched sub-string, while the second is the end of respective match.

If the function failed, *null* is returned.

remove

Synopsis

```
remove (string path);
```

Description

Deletes file. This function is an equivalent to *rm* command in *nix and *del* in Microsoft Windows.

Parameters

`path`

: Path of file that should be deleted.

rmdir

Synopsis

```
rmdir (string path[, bool recursive = false]);
```

Description

Removes empty directory. This function is an equivalent to *rmdir* command in other shells.

Parameters

`path`

Path to directory that should be removed.

`recursive`

Remove directory with all its contents. If this parameter is set to *false*, directory must be empty.

rsqrt

Synopsis

```
float rsqrt (float x);
```

Description

Calculates reciprocal square root of `x`.

Return Values

The function returns the value of $\frac{1}{\sqrt{x}}$.

run

Synopsis

```
integer run (string path[, string_array args[, string redirIn = null[, bool redirOut = false[, bool redirErr = false]]]]);
```

Description

Runs another program or script.

Parameters

`path`

Path to executable file or script that should be run. May contain only filename if path to desired file is present in “PATH” environment variable. In Microsoft Windows, if no extension present, “exe” is assumed.

`args`

Array of command line arguments that should be passed to child process.

`redirIn`

If set to non-empty string, this will be passed to child process's standard input.

`redirOut`

If set to true, standard output for child process will be passed to `OUTPUT` variable.

`redirErr`

If set to true, standard error stream for child process will be passed to `ERROR` variable.

Return Value

If the function succeeds, child process's exit status is returned. Else the return value is *-1*.

scan

Synopsis

```
variant scan ([integer type = TEXT]);
```

Description

Reads data from console.

Parameters

`type`

Type of data that should be read. For list of supported data types look `fscan()` function specification.

Return Values

If valid type was specified and data was read successfully, the function returns value of requested type. Else *null* is returned.

setenv

Synopsis

```
bool setenv(string name, string value);
```

Description

Sets value of environment variable `name`.

Parameters

`name`

Name of environment variable whose value should be set.

`value`

New value of environment variable.

Return Values

This function returns *true* on success and *false* on failure.

Notes

Staring with version 1.2.1, changes made by this function are not permanent, i.e. system environment variables are not modified and all changes are lost when **Squirrel Shell** is closed.

setfiletime

Synopsis

```
setfiletime (string path, integer which, integer time);
```

Description

Changes timestamp of specified directory entry.

Parameters

`path`

Path to directory entry whose date and time should be retrieved.

`which`

Specifies which date and time should be retrieved. See `filetime()` function for list of possible values.

`time`

New timestamp.

sin

Synopsis

```
float sin (float x);
```

Description

Calculates sine of `x`.

Parameters

`x`

Angle in radians whose cosine should be calculated.

Return Values

The function returns the sine of `x`.

sinh

Synopsis

```
float sinh (float x);
```

Description

Calculates hyperbolic sine of `x`.

Parameters

`x`

Angle in radians whose hyperbolic sine should be calculated.

Return Values

The return value is the hyperbolic sine of `x`.

sqrt

Synopsis

```
float sqrt (float x);
```

Description

Calculates square root of `x`.

Return Values

The function returns the value of .

strchar

Synopsis

```
integer strchar (string str, integer i);
```

Description

Get `i`'th character of string.

Parameters

`str`

String character should be extracted from.

`i`

Zero-based index (offset) of character that should be extracted from `str`.

Return Values

If `i` lies in range from `0` to `str.len()` exclusive, the function returns requested character. Else zero is returned.

strtime

Synopsis

```
string strtime (integer time[, string format = "%b %d %T %Y"]);
```

Description

The function represents date and time as text in specified format.

Parameters

`time`

Date and time which should be represented as text.

`format`

Format for resulting text. When a percent sign (%) appears, it and the character after it are not output, but rather identify part of the date or time to be output in a particular way:

`%b` – abbreviated month name (“Jan”, “Feb”, etc.);

`%B` – full month name (“January”, “February”, etc.);

`%C` – century;

`%d` – day of month (always two digits);

`%D` – month/day/year (for example, “12/22/06”);

`%e` – day of month (without leading zero);

`%H` – 24-hour-clock hour (two digits);

`%h` – 12-hour-clock hour (two digits);

`%k` – 12-hour-clock hour (without leading zero);

`%l` – 24-hour-clock hour (without leading zero);

`%m` – month number (two digits);

`%M` – minute (two digits);

`%p` – AM/PM designation;

`%r` – hour:minute:second AM/PM designation (for example, “05:37:25 PM”);

`%R` – hour:minute (for example, “17:37”);

`%S` – second (two digits);

`%T` – hour:minute:second (e.g., "17:37:25");

`%y` – last two digits of year;

`%Y` – year in full.

Return Values

The function returns text representation of specified date and time. Maximum length of resulting string is 1023 characters. Fields that don't not fit in this limit are entirely ignored (so the result does not contain incomplete components).

substr

Synopsis

```
string substr (string str, integer start, integer end);
```

Description

Get sub-string beginning at `start` and ending at `end` (inclusive) characters from `str`. This function is an equivalent to `str.slice(start, end + 1)` delegate in Squirrel except this function does not support negative offsets.

Parameters

`str`

String characters should be extracted from.

`start`

Zero-based index (offset) of the first character that should be copied from `str`.

`end`

Zero-based index (offset) of the last character that should be extracted from `str`. Must not be less than `start`.

Return Values

If `start` lies in range from `0` to `str.len()` exclusive and `end` is in range from `start` to `str.len()` exclusive, then the function returns requested sub-string. Else *null* is returned.

systime

Synopsis

```
integer systime ();
```

Description

Retrieves current system time (in UTC).

Return Values

This function returns current system time (in UTC).

tan

Synopsis

```
float tan (float x);
```

Description

Calculates tangent of x.

Parameters

x

Angle in radians whose tangent should be calculated.

Return Values

The function returns tangent of x.

tanh

Synopsis

```
float tanh (float x);
```

Description

Calculates hyperbolic tangent of x.

Parameters

x

Angle in radians whose hyperbolic tangent should be calculated.

Return Values

The function returns hyperbolic tangent of x.

utctolocal

Synopsis

```
integer utctolocal (integer time);
```

Description

This function converts UTC time to local time.

Parameters

time

UTC time that should be converted. This may be value returned by `filetime()` or `mktime()` function.

Return Values

This function returns specified UTC time to local date and time.

Constants

__argc

Synopsis

```
integer __argc
```

Description

Number of command line arguments passed to script.

Note

Minimal value is *1*, as first argument always contains path to script file.

__argv

Synopsis

```
string_array __argv
```

Description

Array of command line arguments passed to script. Contains `__argc` elements (`__argv[0] ... __argv[__argc - 1]`).

Note

`__argv[0]` always contains path to script file.

CPU_ARCH

Synopsis

```
string CPU_ARCH
```

Description

CPU architecture shell was compiled for. Values are the same as for `cpuarch()` function.

DEGREES_IN_RADIAN

Synopsis

```
float DEGREES_IN_RADIAN
```

Description

Number of degrees in one radian (57.295779513082320876798154814105).

E

Synopsis

```
float E
```

Description

Mathematical constant e (2.71828182845904523536028747135 27).

PCRE_VERSION

Synopsis

```
string PCRE_VERSION
```

Description

Version number of PCRE library used for working with regular expressions.

PI

Synopsis

```
float PI
```

Description

Mathematical constant π (3.14159265358979323846264338327 95).

PLATFORM

Synopsis

```
string PLATFORM
```

Description

Name of OS the shell was compiled for. Values are the same as for `platform()` function.

RADIANS_IN_DEGREE

Synopsis

```
float RADIANS_IN_DEGREE
```

Description

Number of radians in one degree (0.01745329251994329576923690768 4886).

SHELL_VERSION

Synopsis

```
string SHELL_VERSION
```

Description

Version number of shell script is running in. For this version of **Squirrel Shell** `SHELL_VERSION` = “`1.2.2`”.

SQUIRREL_VERSION

Synopsis

```
string SQUIRREL_VERSION
```

Description

Version number of Squirrel script is running in. For Squirrel provided with this version of shell `SQUIRREL_VERSION` = “`2.2.2″`.

Variables

ERROR

Synopsis

```
string ERROR
```

Description

Text redirected from child process's standard error stream.

OUTPUT

Synopsis

```
string OUTPUT
```

Description

Text redirected from child process's standard output stream.

Links

http://squirrelsh.sourceforge.net – **Squirrel Shell**'s web site.

http://www.squirrel-lang.org – Web site of Squirrel scripting language.

http://sourceforge.net – Home of open source software.