PASSWORDTECH

OPEN-SOURCE PASSWORD GENERATOR & MANAGER

SCRIPTING MANUAL

Version 3.5.1

Contents

Introduction

As of version 3.0.0, Password Tech (PwTech) allows running custom *Lua scripts* to generate passwords, passphrases, and generally any kind of random sequences. Lua is a high-level, dynamically typed programming language, providing basic mathematical functions, string operations, and list/array functionalities. For an introduction to Lua programming see, for example, the online version of the book *Programming in Lua* by R. Ierusalimschy.

For creating and editing Lua scripts, you can use any text editor such as *Notepad++* or a code editor such as *Lua Development Tools*.

Input of user parameters (PwTech → script), output of passwords (script → PwTech), and generating random data (including passwords, passphrases, …) within a script is accomplished through a dedicated application programming interface (API). This API as well as the general structure of Lua scripts for PwTech are explained in the following.

Lua Notes

Data Types

The PwTech Lua API uses the following data types for transferring input and output arguments between the main application and the scripting level:

Integer

Internally, Lua uses a signed 64-bit integer value, ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (Int64). However, most input/output arguments of type integer that are passed to and received from Lua functions are converted into signed 32-bit integers, ranging from -2,147,483,648 to 2,147,483,647 (Int32). The supported type (Int32 or Int64), which is relevant for the value range of an integer argument, is indicated for every integer argument throughout this document.

Float

64-bit floating point number (double precision, corresponding to `double` type in C/C++), ranging from 10^{-308} to 10^{308} with a precision of 15-17 decimal digits.

String

String of 8-bit integers (each ranging from 0 to 255), also called *octets*, with UTF-8 encoding. In UTF-8, a single Unicode character is encoded as up to four octets. Thus, a string of length `len` (number of octets) may actually consist of less than `len` Unicode characters. If the password generated by the main application contains Unicode characters outside the 7-bit ASCII character set, you have to treat the UTF-8 octet representations of these characters as units that must not be separated.

To determine the UTF-8 octet length of a Unicode character in a Lua string, you can use the following rule: If the first octet *o* of the UTF-8 unit is <192, 192 ≤ *o* < 224, 224 ≤ *o* < 240, or ≥240, the octet length of the Unicode character is 1, 2, 3, or 4, respectively.

Function Syntax

A Lua function may look like this:

```
function fun(param1, param2, param3)
      --do something...
      return ret1, ret2
end
```

In this manual, the input arguments `param1`, `param2`, `param3` are called *parameters*, and the output arguments `ret1`, `ret2` are called *return values*. The following syntax is used to illustrate the usage of a function:

`fun(`*`param1`* `[, `*`param2`* `[, `*`param3`*`]]) -> `*`ret1`*`, `*`ret2`*

Input and output arguments are set in *italic*. Parameters without brackets are mandatory, those enclosed with square brackets `[]` are optional. In the above example, only *`param1`* is mandatory, whereas both *`param2`* and *`param3`* are optional. Thus, the function can be called as `fun(1)`, `fun(1, 2)`, and `fun(1, 2, 3)`. Return values are specified after a stylized arrow "->" for better visibility. Note that the caller is not obliged to catch all return values but may decide to catch only the first *n* values or none of them, e.g., via `r = fun(42)`.

Bit fields

PwTech uses bit fields to encode binary options, i.e., options that can be either enabled or disabled, in 32-bit integer numbers. Each option or *bit flag* occupies one bit position in the number and sets the bit to 0 (*false*) or 1 (*true*) if the option is disabled or enabled, respectively. To set a bit to 1 in the bit field integer and thus enable the corresponding option, the integer is combined by bitwise "or" with the number corresponding to the bit position. The *i*-th bit corresponds to the number 2^{i-1} (first bit = 1, second bit = 2, third bit = 4, etc.). For example, to set the 2nd, 3rd, and 5th bit, the values 2, 4, and 16 are combined by bitwise "or" to yield the bit field integer 22 (in binary notation: 10110).

Conversely, to check whether a bit flag at a specified position in the integer is set, the bit field is combined by bitwise "and" with the number corresponding to the bit position. If the result of this operation is nonzero or zero, the bit flag is set (enabled) or not set (disabled), respectively. For example, to check whether the 3rd bit is set in the bit field 22 (10110), calculate 22 *and* 4 = 4 (nonzero), hence the flag at this position is set.

You can use the Lua operators `|` and `&` for the bitwise "or" and "and" operations, respectively. For example, to set the 4th and 6th bit (values of 8 and 32) in a bit field integer, you can use the statement:

```
flags = flags | 8 | 32
```

To check whether the 4th bit (value of 8) is set, you can use the following *if* statement:

```
if flags & 8 ~= 0 then
      --do something...
end
```

Structure of a Lua Script

When generating passwords from a script, PwTech calls two functions, `init()` and `generate()`. `init()` is *optional* and only called once, namely when generating the first password. It can be used to transfer the user parameters related to password generation to the scripting level. `generate()` is *mandatory*—scripts without this function will not be loaded—and called whenever a new password is requested. It passes the password that has been generated by PwTech before (using the other *Include …* options, if applicable) to the script via input arguments, which may then be manipulated by the script. However, the script may also generate a completely new password. The generated password is then returned to PwTech via output arguments (return values).

The script may also expose specific properties to PwTech, which can be used to optimize the password generation. The specification of properties is optional, but they must be defined as global variables in the form `script_property=`*`setting`* (all variables have global scope unless explicitly declared local via the `local` keyword) in order to be recognized by PwTech.

Script Properties

The following properties are currently supported:

script_flags	Int32	Bit field (bit flags combined by bitwise "or") that encodes binary (on/off) properties of the script. Default is 0 (if variable is not specified). 1 – The script generates passwords completely on its own (stand-alone) and does not use previously generated passwords (via *Include …* options in PwTech). If specified, PwTech will not generate any passwords, passphrases, or formatted passwords before executing the script, and always pass an empty password to the `generate()` function.

Function *init()*

The specification of this function is optional. It is used to pass user parameters related to password generation to the script. These parameters can then be evaluated in the script to further customize the password generation.

Function definition

```
init(num_passw, dest, gen_flags, advanced_flags, num_chars, num_words,
      format_str)
```

Parameters

num_passw	Int64	Number of passwords to be generated (up to 1 trillion (10^{12}))
dest	Int32	Destination of the generated password(s): 0 – Single password in password box (main window) 1 – Multiple passwords in password list window 2 – Write password(s) to file 3 – Copy password to clipboard 4 – Show password in message box 5 – Display password(s) on the console 6 – Autotype password
gen_flags	Int32	Bit field (bit flags combined by bitwise "or") that encodes the types of password to be generated: 1 – Include characters 2 – Include words 4 – Format password Example: 5 = Include characters + Format password.
advanced_flags	Int32	Bit field that encodes the "Advanced password options" flags (list of checkboxes). The options in the list correspond to the flags 1, 2, 4, …, 2^{N-1} (from 1st to *N*-th list entry) with *N* being the number of options.
num_chars	Int32	Number of characters for option *Include characters.*
num_words	Int32	Number of words for option *Include words.*
format_str	String	Format sequence for option *Format password* – may be `nil` (empty).

Return values

None.

Function *generate()*

This function *must* be specified in the script. PwTech calls it whenever a new password is requested and passes the password that has been generated so far as an input parameter. This password can be manipulated on the scripting level. Also, a new random password can be generated by calling dedicated API functions. The function should return the new password and the corresponding entropy value.

Function definition

`generate(`*`passw_num, in_passw, in_entropy`*`) -> `*`out_passw, out_entropy`*

Parameters

passw_num	Int64	Password number: Always 1 if a single password is generated, up to 1 trillion (10^{12}) if multiple passwords are generated (1 for

		1st, 2 for 2nd password, …). Note that *passw_num* is not necessarily incremented with every call: If the "Exclude duplicates" option is enabled and the previously generated password was a duplicate, the password number does not change.
in_passw	String	Password that has been generated before executing the script. This may be a password, passphrase, formatted password, or a combination thereof. The password may be `nil` (empty).
in_entropy	Float	Entropy value associated with *in_passw*. 0 if password is empty or does not contain any random characters.

Return values

out_passw	String	Password to be transferred to the main application. Make sure that the password is properly UTF-8 encoded!
out_entropy	Float	Entropy value associated with *out_passw*.

PwTech Lua API Functions

The PwTech Lua API provides functions for generating random numbers, passwords, passphrases, formatted passwords, and more. All functions can be called from the `init()` and `generate()` function within the script.

Function *pwtech_random()*

Returns a random floating point number between 0 and 1, or an integer number in a specified range. It is functionally equivalent to `math.random()` from the Lua standard library. There are three versions of this function.

Function definition

```
(1) pwtech_random() -> rnd_f

(2) pwtech_random(n) -> rnd_n

(3) pwtech_random(n1, n2) -> rnd_n
```

Parameters

(1) –	–	Generates floating point number between 0 and 1 (0 inclusive, 1 exclusive).
(2) *n*	Int32	Generates random integer in the range 1 to *n* (both inclusive).
(3) *n1, n2*	Int32	Generates random integer in the range *n1* to *n2* (both inclusive).

Return values

(1) *rnd_f*	Float	Floating point number between 0 and 1.
(2) *rnd_n* (3) *rnd_n*	Int32	Integer in the specified range.

Function *pwtech_password()*

Generates a random password of a specified length, using the character set that is currently loaded in PwTech.

Function definition

`pwtech_password(`*`len`* `[, `*`flags`*`]) -> `*`passw, entropy`*

Parameters

len	Int32	Desired password length (number of Unicode characters). Note that the returned string may be longer than *len* due to the UTF-8 encoding.
flags	Int32	*(optional)* Bit field that encodes various options to customize the password generation: 1 – First character must not be a lower-case letter 2 – Exclude repeating consecutive characters 4 – Only include additional characters if custom character set contains appropriate subset 8 – Include at least one upper-case letter 16 – Include at least one lower-case letter 32 – Include at least one digit 64 – Include at least one special symbol 512 – Each character must occur only once

Return values

passw	String	Resulting password, UTF-8 encoded.
entropy	Float	Entropy of the password.

Function *pwtech_phonetic()*

Generates a random phonetic password of a specified length, using the phonetic trigrams that are currently loaded in PwTech. Letters are lower-case by default.

Function definition

`pwtech_phonetic(`*`len`* `[, `*`flags`*`]) -> `*`passw, entropy`*

Parameters

len	Int32	Desired password length (number of Unicode characters, up to 16,000). Note that the returned string may be longer than *len* due to the UTF-8 encoding.
flags	Int32	*(optional)* Bit field that encodes various options to customize the password generation: 8 – Include at least one upper-case letter 16 – Include at least one lower-case letter 32 – Include at least one digit 64 – Include at least one special symbol 128 – Upper-case letters 256 – Lower-case and upper-case (mixed-case) letters, selected randomly for each letter; increases entropy by 1 bit per letter

Return values

passw	String	Resulting password, UTF-8 encoded.
entropy	Float	Entropy of the password.

Function *pwtech_passphrase()*

Generates a passphrase composed of random words from the currently loaded word list.

Function definition

`pwtech_passphrase(`*num* `[, `*chars* `[, `*flags*`]]) -> `*passphr, entropy*

Parameters

num	Int32	Desired number of words (up to 100).
chars	String	*(optional)* Characters to be combined with the words (equivalent to the option "Combine words with characters" in PwTech's main window) if corresponding flag is set (see next parameter). May be `nil` if the combination with characters is not desired.
flags	Integer	*(optional)* Bit field that encodes various options to customize the passphrase generation: 1 – Combine words with characters (if *chars* is a valid string) 2 – Do not separate words by a "-" character 4 – Do not separate words and characters by a "-" character 8 – Reverse order of words and characters (i.e., characters first)

Return values

passphr	String	Resulting passphrase, UTF-8 encoded.
entropy	Float	Entropy of the passphrase (without considering characters given in *chars* parameter).

Function *pwtech_word()*

Returns a word from the currently loaded word list.

Function definition

`(1) pwtech_word() -> word`

`(2) pwtech_word(idx) -> word`

Parameters

(1) –	–	Returns a *random* word from the word list.
(2) *idx*	Int32	Returns the word at position *idx* in the word list. Consistent with Lua conventions, the first word has index 1 and the last word has index *N*. To get the number of words *N* in the list, call `pwtech_numwords()`.

Return values

word	String	Word from the word list, UTF-8 encoded.

Function *pwtech_numwords()*

Returns the size (number of words) of the currently loaded word list.

Function definition

`pwtech_numwords() -> num`

Parameters

None.

Return values

num	Int32	Size of the word list.

Function *pwtech_format()*

Generates a formatted password using the specified format string.

Function definition

`pwtech_format(format [, flags]) -> passw`

Parameters

(1) *format*	String	Format sequence for generating the password. Refer to the PwTech User Manual for more details on the syntax of format se-

		quences.
(2) *flags*	Int32	*(optional)* Bit field that encodes various options to customize the password generation: 1 – Exclude repeating consecutive characters

Return values

passw	String	Resulting password, UTF-8 encoded.

Example Scripts

Example Lua scripts are contained in the folder *script_examples* within the folder where Password Tech resides.

Happy scripting!