Oracle Password Generator

Oracle Password Generator

I got fed up with the in-house Oracle script (which I wrote/adapted!) which generates passwords - you have to be logged into a database, load the script and execute it. You can only do one password at a time, it doesn't generate a script, etc etc. This new utility *generatePassword* is a Windows and/or Linux command line replacement which can generate multiple passwords, write to a script etc. No database connection is required - until you wish to run the generated code.

Passwords generated will begin with a letter, upper or lower case, and will be followed by a number of additional characters made up of:

- Letters UPPER CASE - but without the letter 'O' as it looks like a zero in some fonts;
- Letters lower case - but without the letter 'l' (ell) as it looks like a digit one in some fonts;
- Digits 2 through 9 - to avoid mistaking zero for O and 1 for lower case L in some fonts);
- Special characters - '$', '_' or '#' which are the three permitted by Oracle to be used in passwords. Others *could* be used, but are not advised.

The code is currently configured to return passwords that:

- Are 15 random characters in length;
- Contain at least 2 digits;
- Contain at least 2 punctuation characters;
- Contain at least 2 upper case letters;
- Contain at least 2 lower case letters;

This matches with the *strong* password verification function, `ORA12C_STRONG_VERIFY_FUNCTION`, provided with Oracle 12c, but which can also be installed into 11g. This requires a minimum of 9 characters in a password, but we are going for 15 as this is more secure to a brute force attack.

Generate a Single Password.

In the following examples, I'm on a Linux box. Windows is much the same but doesn't (easily) allow the output from the utility to be assigned to a variable. It can, of course, be easily redirected to a script file. But that's about all.

The code exists on our AWS Jump Off box - 10.128.3.242, the one used for Releases - in the `/home/oracle/bin directory`, which is on the `$PATH`. It can also be installed locally to your own desktop or laptop - it is to be found in the DBA Team Software Repository on the `P:\` drive.

It is executed thus:

```
generatePassword
```

Which results in a single password, in this example, it is:

```
"F6K#5b226r34T49"
```

You can of course, redirect this output to a file, but there's not much point. Much better, at least on Linux, is to redirect it into an environment (or script) variable:

```
export NEW_PW=$(generatePassword)
```

or, alternatively:

```
export NEW_PW=`generatePassword`
```

In a script, the `export` is optional and not required unless sub-scripts are being called and expect the variable to exist.

You can now pass the generated password into scripts etc, as desired, using the variable `$NEW_PW`, for example:

```
NEW_PW=`generatePassword`

sqlplus -s username@database << EOF
create user ${1}
identified by ${NEW_PW}
default tablespace ...
temporary tablespace ...
profile ...;

EOF
```

The above assumes that the script is called with the username passed as the first parameter - `createNewUser dunbarnor` for example. What Oracle will execute is as follows:

```
create user dunbarnor
identified by "F6K#5b226r34T49"
default tablespace ...
temporary tablespace ...
profile ...;
```

You should note that the generated password will *always* be wrapped in double quotes. This is required by Oracle due to the special (punctuation) characters used, so there is no need for your script to add further quotes. The following is *wrong*, for example:

```
NEW_PW=`generatePassword`

sqlplus -s username@database << EOF
create user ${1}
identified by "${NEW_PW}"
...
EOF
```

This would be executed as:

```
create user dunbarnorx
identified by ""F6K#5b226r34T49""
...
```

Oracle will complain!

```
ERROR at line 2:
ORA-01741: illegal zero-length identifier
```

Generate User Passwords.

```
generatePassword username [ username ...]
```

Which results in an `alter user` SQL statement for each user passed as parameters:

```
generatePassword fred barney wilma betty dino
```

```
alter user fred identified by "uD25RP44pn_H57$" account unlock;
alter user barney identified by "x_#7pLEWC$$Vpx4" account unlock;
alter user wilma identified by "j9BG_6Vs9R_kZ_u" account unlock;
alter user betty identified by "Lj_8mzt98Q#DvRI" account unlock;
alter user dino identified by "f_L_$3v55r9aQj3" account unlock;
```

You can redirect this output to a file, where it can then be run against the appropriate database:

```
generatePassword fred barney wilma betty dino > password.sql

sqlplus yourname/yourPassword@yourDatabase @password.sql

User altered.

User altered.

User altered.

User altered.

User altered.
```

Configuring the code

The program is written to comply with our security standards and the password verification function named `ORA12C_STRONG_VERIFY_FUNCTION`:

- 9 characters minimum, but we have chosen 15;
- A minimum of 2 upper case characters;
- A minimum of 2 lower case characters;
- A minimum of 2 numeric characters;
- A minimum of 2 special characters;

The file, `main.cpp`, has the following lines around about line 20:

```
const uint8_t passwordLength = 15;

const uint8_t numberOfUppers = 2;
const uint8_t numberOfLowers = 2;
const uint8_t numberOfDigits = 2;
const uint8_t numberOfSpecials = 2;
```

You may change the values above to configure the password complexity you need. You will, of course, have to recompile the code afterwards.

Compiling the Code

The code is written in such a way as to be able to be compiled on both Windows and Linux using the same source file. If you have a Mac, well, who knows. All compilations have been done using `g++`, both on Linux where it is installed pretty much by default, and on Windows where I'm using the 32/64 bit version of `g++` 5.1.0 available from TDB. Your mileage may vary when compiling with the likes of Visual C++ etc. (Borland aka Embarcadero C++ does work though!)

Command Line Compiling - Windows

For Windows users, a CodeBlocks project file exists which can be used, but you will need to amend the build options to select your own compiler and output folders etc.

If you wish to compile using command line tools instead, then assuming that the TDM version of `g++` is on your path, do this for a release version (no debugging information):

```
mkdir bin\Release
g++ -O3 -o bin\Release\generatePassword.exe -std=c++11 main.cpp
```

or this following for a debug version:

```
mkdir bin\Debug
g++ -O0 -g -o bin\Debug\generatePassword.exe -std=c++11 main.cpp
```

Obviously, you only need to create the directories once.

For Borland/Embarcadero users, and assuming that the compiler `bcc32x` is on the path:

```
mkdir bin\Release
bcc32x -O3 -o bin\Release\generatePassword.exe -std=c++11 main.cpp
```

or this following for a debug version:

```
mkdir bin\Debug
bcc32x -O0 -g -o bin\Debug\generatePassword.exe -std=c++11 main.cpp
```

There's a different version of the Borland compiler, `bcc32c` which takes different command line options to do the same as the above. I prefer the `bcc32x` version as it takes almost all the same options as the `g++` compiler does. Standards!

Command Line Compiling - Linux

The supplied CodeBlocks project file can be used under Linux as well as under Windows. You will still need to amend the build options to select your own compiler and output folders etc.

If you wish to compile using command line tools instead, then it's *almost* the same as for Windows above.

For a release version:

```
mkdir -p bin/release
g++ -O3 -o bin/release/generatePassword -std=c++11 main.cpp
strip generatePassword
```

And for a debug version:

```
mkdir -p bin/debug
g++ -O0 -g -o bin/debug/generatePassword -std=c++11 main.cpp
```

You can then `mv` or `cp` the output file(s) to somewhere on your `$PATH`, or, leave it where it is and add that location to your `$PATH`.