

PASSWORD GENERATION & EVALUATION

DOCUMENTATION

MAY 21, 2010

AUTHOR:

THORSTEINN SÆVAR HJARTARSON
CODECANYON.NET EXCLUSIVE AUTHOR

Table of contents

1	Manual	5
1.1	Main Form	5
1.2	Settings	6
1.2.1	Word Jumble	6
1.2.2	Text Jargon	7
1.2.3	l33t Speak	7
1.2.4	Strength Checker	8
1.3	Mass Generate	8
2	Ratio Strings	9
2.1	What are they?	9
2.2	What are the allowed symbols?	9
2.3	Why not just use Regular Expressions?	10
2.4	Does the order not matter?	10
2.5	Can you show me some more examples?	10
2.6	What about some really difficult ones?	10
3	Code examples	12
3.1	Generate Word Jumble	12
3.1.1	Default Jumble gives you two words between lengths 2-9	12
3.1.2	Use the overloads to specify better what you want	12
3.1.3	Use l33t speak to obscure the words (password)	13
3.1.4	Use TextJargon to obscure the passwords	13
3.1.5	Wordlists and Bannlist - support your native language	14
3.2	Generate Text Jargon	16
3.2.1	Without the Ratio string configured, the "random" is not so random	16
3.2.2	Set Ratio string and enable shuffling to get better random passwords	16
3.2.3	It is very easy to construct your own ratio string	17

3.3	Evaluate Ratio Strings	18
3.3.1	Evaluate password strength using internal checker	18
3.3.2	Ratio objects can "accept" strings based on pattern	18
3.3.3	Ratio can tell you what the string needs too be accepted	19
3.3.4	Use OR-brackets to allow characters to be of more than one type	19
3.3.5	Let Ratio only care about length and symbols	20
3.4	Evaluate Password Policies	22
3.4.1	Evaluate password strength using Password Policy	22
3.4.2	Policies are built on Ratio objects with assigned strengths	22
3.4.3	Customize the Password Policy to create your own Strength Checker	23
3.4.4	Let the Evaluator tell you what you need to improve strength	23
4	Password Policies in general	25
4.1	Password guidelines	25
4.2	Human algorithms	26
4.3	Password strength evaluation	27

Requirements

Using the software

- Windows operating system (2000, XP, Vista, 7)
- Microsoft's .NET 3.5 installed
- Run the setup or compile binaries from source code
- The binaries might work on Mono project for Linux

Implementation

- Visual Studio 2008 is preferred editor
- Knowledge of C#
- Can be downgraded to .NET 2.0

Documentation

- MiKTeX 2.7 or later installed
- TeXnicCenter RC1 or later is preferred editor

Part 1

Manual

1.1 Main Form

The main form is pretty straight forward, you generate or evaluate passwords. The features are in the configuration options which you can open up with right-clicking onto the form.

If you *right-click* on the *Generation Method* you get the settings for that method, if you *right-click* on the *Strength Bar* you get the settings for customizing the strength checker.

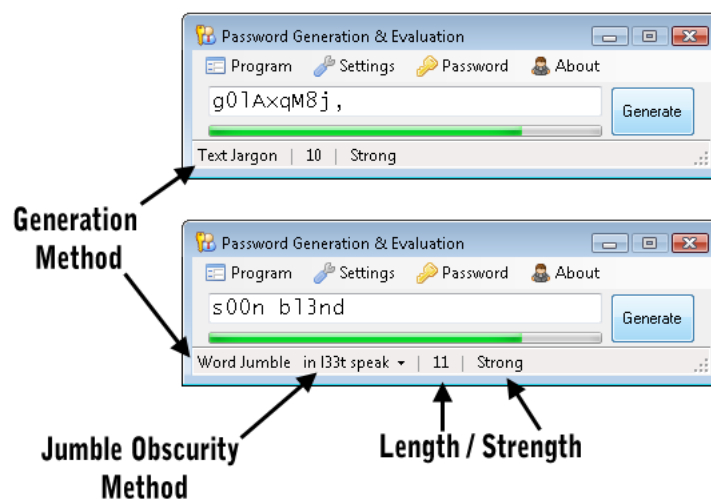


Image 1.1: What the main form looks like

You switch between *Generation Methods* by *left-clicking* on it. Same goes for the obscurity, you *left-click* on the *Obscurity Method* to change it (or disable it).

1.2 Settings

Open up the settings with *Settings* -> *Advanced Settings* or right-click on something on the form to open up the settings.

1.2.1 Word Jumble

Word Jumble randomly selects words from a word list. You customize it with how many words you want and upper & lower length bounds. See the left side on image 1.2.

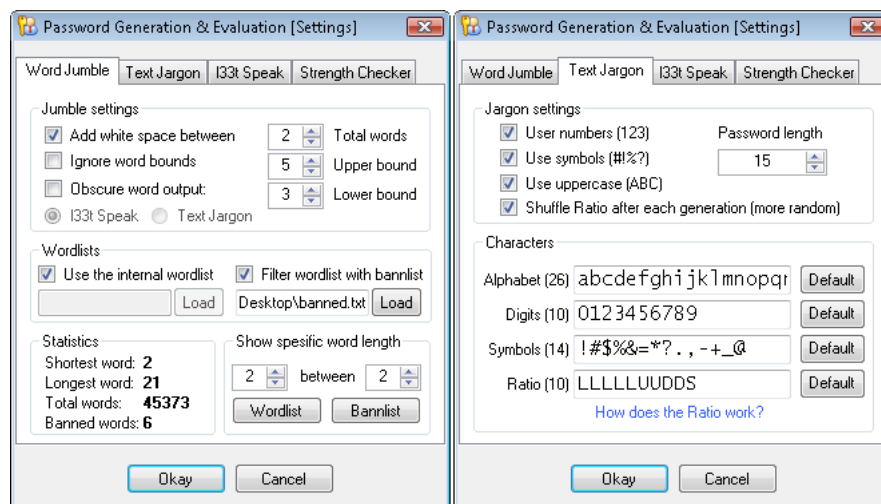


Image 1.2: Showing settings for Word Jumble and Text Jargon

- Word Jumble can obscure the selected words using I33t settings or Text Jargon settings.
- You can either use the internal wordlist of 45.373 words or make your own and load it in.
- You can prevent profanity by providing a "bannlist", which is a list of banned words that will not be used in generation.
- Get word list statistics and the option to view words ranging from specific lengths

1.2.2 Text Jargon

Text Jargon generates complete jibberish! In order for the Text Jargon to function you need to configure the **Ratio String**, see Part. 2 on page 9 for more information about Ratio strings. See the right side on image 1.2 on the facing page.

- Specify the length of the password
- Decide what character types to use in the generation
- Specify characters and symbols in the generation

1.2.3 133t Speak

133t¹ Speak only affects Word Jumble, and it substitutes strings with other strings. See the left side on image 1.3.

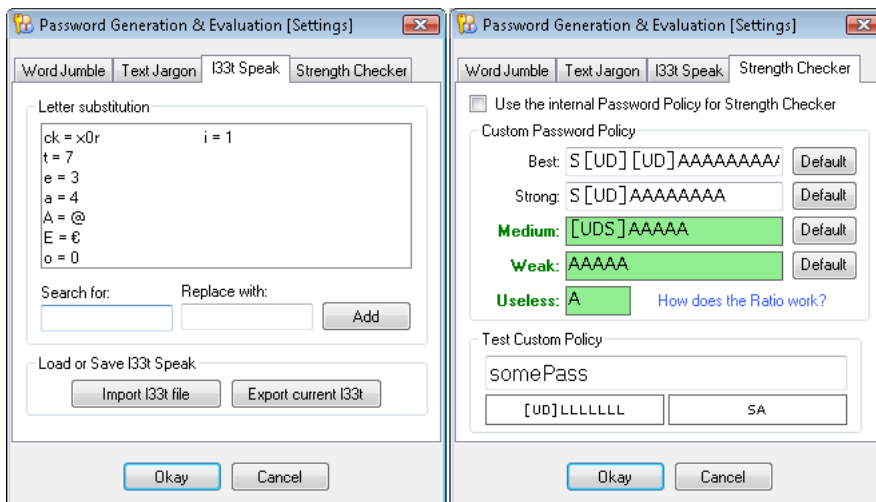


Image 1.3: Showing settings for 133t Speak and Strength Checker

- Specify the *search string* or *character* and *replacement string* or *character* that will be substituted in all generated words
- You can Import and Export 133t Speak in case you need to switch between substitutions faster
- The same search string cannot be entered twice, but the list is not case-sensitive: a is not the same as A

¹Leet Speak: <http://en.wikipedia.org/wiki/Leet>

1.2.4 Strength Checker

Strenght Checker evaluates the password and tells you if it is strong or not. See the right side on image 1.3 on the preceding page.

Based on either internal rules or your own custom rules, the Strength Checker will tell you what has been *accepted* and what is *needed* to reach the next level of strength.

- The rules are specified with **Ratio strings** (see Part. 2 on the next page) and they define what the password must contain
- When you try out the custom policy you can see the accepted strengths as they are colored green
- The boxes below the test-field are: *Accepted parts* and *Needed for next strength*
- If you specify any Ratio string incorrectly the form will color the box red to notify you
- You can use the internal strength checker if you are not sure how to configure the Ratio strings

1.3 Mass Generate

From the `Program` selection in the main menu you can select *Mass Generate Passwords* to get as many as you like.

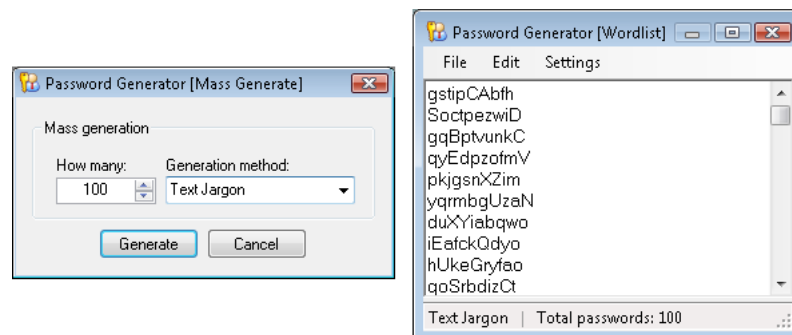


Image 1.4: Mass generate passwords

Part 2

Ratio Strings

2.1 What are they?

Ratio strings are sort of *rules* you specify to tell the Library how strings are to be generated or evaluated.

2.2 What are the allowed symbols?

There are 7 control characters, they are they only ones allowed:

L : **lowercase** character, example: a b c d e

U : **UPPERCASE** character, example: A B C D E

D : **Digits** or numbers, example: 1 2 3 4 0

S : **Symbols** special characters, example: ! # & ? \$

A : **Anything**, can be any of the four control characters. This one is used to specify length, example: AAA means *any string of length 3*, but LLL would mean *any string containing 3 lowercase characters*.

OR brackets : [and] are used to specify that a character can have multiple type, example: AA[UD] means *any string of length 3, that contains at least 1 uppercase OR 1 digit*

[LUDS] means anything, which could also be the A : *A character that can be of the type lower, upper, digit or symbol*

2.3 Why not just use Regular Expressions?

They are very powerful and could easily replace this mechanism, but it is not very user friendly. The ratio strings are targeted at people that may not be technically minded, or those who do not know Regular Expressions.

You have to be experienced in Regular Expressions to be able to specify the rule: "any string of length 3, that contains at least 1 uppercase OR 1 digit", which in Ratio strings you simply type: AA[UD] or A[UD]A or [UD]AA.

2.4 Does the order not matter?

No, as the word implies it is a *ratio*, example: DLU is the same as UDL.

When generating Text Jargon however, the password will be generated as the Ratio string is constructed, but you can enable *Ratio shuffling* such that the ratio is shuffled after each generation.

2.5 Can you show me some more examples?

Sure, here are the default Ratio strings used in the custom Strength Checker:

A : Any string of length 1

AAAAA : Any string of length 5

AAAAA[UDS] : Any string of length 6 that contains at least [1 Uppercase or 1 Digit or 1 Symbol]

AAAAAAAA[UD]S : Any string of length 10 that contains at least 1 Symbol and [1 Uppercase or 1 Digit]

AAAAAAAAAAAA[UD][UD]S : Any string of length 14 that contains at least 1 Symbol, and 2 characters are either [Uppercase or Digit]

2.6 What about some really difficult ones?

Okay, let's look at this ratio string: **DLSU[DS][LU][LUDS]AAA**

- The string must be 10 characters or longer
- It **MUST** contain 1 digit, 1 lowercase, 1 symbol and 1 uppercase
- 1 character must be of type digit or symbol
- 1 character must be of type lowercase or uppercase
- [LUDS] means A so it can be anything

Take a look at table 2.1 on the facing page for strings that will be accepted by this Ratio string.

DLSU	[DS]	[LU]	[LUDS]	AAA	Results
2g=H	&	e	@	!fu	2g=H&e@!fu
7m?H	9	I	-	A6\$	7m?H9I-A6\$
6u_A	5	O	i	yqT	6u_A5OiyqT
4b#U	0	p	K	M5_	4b#U0pKM5_
8q%E	5	v	_	wXL	8q%E5v_wXL
4a,K	@	m	7	86-	4a,K@m786-
9y-P	5	Z	6	qS0	9y-P5Z6qS0
0z-Q	=	h	\$	5@j	0z-Q=h\$5@j
1b#K	,	A	f	HX7	1b#K,AfHX7
8t\$X	?	p	C	%3z	8t\$X?pC%3z

Table 2.1: Generated Text Jargon strings from a Ratio string

Part 3

Code examples

3.1 Generate Word Jumble

```
1 Generator.Wordlist = Generator.GetInternalWordlist();
```

Code 3.1: Load the default wordlist before running the examples

3.1.1 Default Jumble gives you two words between lengths 2-9

```
1 for (int i = 0; i < 10; i++)  
2     Console.WriteLine(Generator.GetJumble());
```

Code 3.2: Example.1

```
threadersreversals  
recursionsensed  
decreedexamined  
instantabandon  
snortinggable  
salvagesmallish  
teamingcruising  
stenchfruitless  
briarperishes  
encodermilder
```

3.1.2 Use the overloads to specify better what you want

```
1 int wordCount = 2;  
2 int upperBound = 5;  
3 int lowerBound = 3;  
4 bool whiteSpace = true;  
5
```

```
6 for (int i = 0; i < 10; i++)
7     Console.WriteLine(Generator.GetJumble(wordCount, upperBound, lowerBound, whiteSpace));
```

Code 3.3: Example.2

```
work job
twist pause
inane dyed
sheik mount
ways cocks
danes deer
goods purrs
idaho ball
ditto tamed
memos savor
```

3.1.3 Use l33t speak to obscure the words (password)

```
1 // Create obscurity object and enable it
2 ObscureJumble obscure = new ObscureJumble();
3 obscure.Enabled = true;
4
5 // Set the method to l33t speak
6 obscure.Method = ObscurityMethod.LeetSpeak;
7 obscure.LeetSpeak = new LeetSpeak();
8
9 // Add some l33t substitutions
10 obscure.LeetSpeak.AddLeet("e", "3");
11 obscure.LeetSpeak.AddLeet("l", "1");
12 obscure.LeetSpeak.AddLeet("ck", "x0r");
13 obscure.LeetSpeak.AddLeet("o", "0");
14
15 // Use the same bounds settings as before
16 for (int i = 0; i < 10; i++)
17     Console.WriteLine(Generator.GetJumble(wordCount, upperBound, lowerBound, whiteSpace, obscure));
```

Code 3.4: Example.3

```
b33ps fr1ll
l3an b1ds
bump asp
prad0 b0yc3
qu1nt b0ard
lumps ta0s
g00fy p0l3
m1nts b3ars
b1ng mar10
curbs a1ry
```

3.1.4 Use TextJargon to obscure the passwords

```

1 // Set the method to TextJargon and enable all
2 obscure.Method = ObscurityMethod.TextJargon;
3 obscure.UseJargonNumbers = true;
4 obscure.UseJargonSymbols = true;
5 obscure.UseJargonUppercase = true;
6
7 // Load default alphabets, digits and symbols
8 Generator.LoadDefaultJargonSettings();
9
10 // Set the ratio value that will obscure the words:
11 // "For every 6 chars, 3 are lower, 2 are upper and 1 digit"
12 Generator.PasswordRatio.Value = Ratio.Lowercases(3) + Ratio.Uppercases(2) + Ratio.Digits(1);
13
14 // Use the same bounds settings as before
15 for (int i = 0; i < 10; i++)
16     Console.WriteLine(Generator.GetJumble(wordCount, upperBound, lowerBound, whiteSpace, obscure));

```

Code 3.5: Example.4

```

jUDge wiN
sAlo9N FrOth
jeRK Golf
4nOTre s6LoP
s0laVE 6JacKy
CLu1b DriP3s
Un4Did fu7r
sArAh cRo4Wn
Go8rEn liNt
o5vAl bl4UEs

```

3.1.5 Wordlists and Bannlist - support your native language

```

1 // Clear both lists before inserting into them
2 Generator.Wordlist.Clear();
3 Generator.Bannlist.Clear();
4
5 // This could be the list used by the Generator
6 Generator.Wordlist.Add(new Word("abc"));
7 Generator.Wordlist.Add(new Word("dude"));
8 Generator.Wordlist.Add(new Word("hello"));
9 Generator.Wordlist.Add(new Word("BAD")); // (profanity)
10 Generator.Wordlist.Add(new Word("HACK")); // (profanity)
11 Generator.Wordlist.Add(new Word("FUBAR")); // (profanity)
12
13 for (int i = 0; i < 10; i++)
14     Console.WriteLine(Generator.GetJumble(wordCount, upperBound, lowerBound, whiteSpace));
15
16 // These words are profanity and should NOT be allowed!
17 Generator.Bannlist.Add(new Word("BAD"));
18 Generator.Bannlist.Add(new Word("HACK"));
19 Generator.Bannlist.Add(new Word("FUBAR"));
20
21 // All uppercase words are now gone
22 for (int i = 0; i < 10; i++)
23     Console.WriteLine(Generator.GetJumble(wordCount, upperBound, lowerBound, whiteSpace));

```

Code 3.6: Example.5

Use your own wordlists , your own language perhaps:

```
abc hello
dude abc
FUBAR HACK
dude dude
abc abc
HACK HACK
dude hello
FUBAR hello
BAD FUBAR
dude dude
```

Use "Bannlist" to prevent profanity:

```
abc hello
abc dude
dude hello
dude abc
dude hello
hello hello
abc dude
dude hello
abc dude
abc abc
```

3.2 Generate Text Jargon

```
1 // Set the alphabet, digits and symbols used in generation
2 Generator.PasswordAlphabet = "abcdefghijklmnopqrstuvwxyz";
3 Generator.PasswordDigits = "0123456789";
4 Generator.PasswordSymbols = "!#$%&=*?.,-+_@";
```

Code 3.7: Load the default alphabets before running the examples

3.2.1 Without the Ratio string configured, the "random" is not so random

```
1 for (int i = 0; i < 10; i++)
2     Console.WriteLine( Generator.GetJargon() );
```

Code 3.8: Example.1

```
dcxlyqkh
edfbtarp
lnbpamts
wreqvzsf
ywxegblh
zuotawsf
axuspetg
vjcksgxi
xzhueiyd
anxfvdep
```

3.2.2 Set Ratio string and enable shuffling to get better random passwords

```
1 // Set the ratio to default internal value
2 Generator.PasswordRatio = new Ratio( Generator.DefaultRatio );
3 Generator.ShuffleRatio = true;
4
5 for (int i = 0; i < 10; i++)
6     Console.WriteLine( Generator.GetJargon() );
```

Code 3.9: Example.2

```
Yf+42Joirut
cp1ai?k3XgW
!pqEHz7twb4
8R5ytekG.Nx
sR1gD@6pyei
Pn0Vrt6yaj_
ToaFm89hg$z
xpe_i41IADh
2y4SIhw ,Zba
ukqRv7zG=I2
```

3.2.3 It is very easy to construct your own ratio string

```
1 // Create new ratio object, and set it with 3 lower case letters ,
2 // 1 uppercase, 2 digits and one symbol
3 Ratio ratio = new Ratio();
4 ratio . Value = Ratio.Lowercases(3) + Ratio.Uppercases(1) + Ratio.Digits(2) + Ratio.Symbols(1);
5
6 // Or you can do this directly with one method
7 ratio . SetRatio(3, 1, 2, 1);
8
9 // Or you can do this as hard-coded string (not recommended though)
10 ratio . Value = "LLLUDDS";
11
12 // Set the generator with this ratio and disable shuffling
13 // so you can see better how it looks un-shuffled
14 Generator.PasswordRatio = ratio;
15 Generator.ShuffleRatio = false;
16
17 for (int i = 0; i < 10; i++)
18     Console.WriteLine( Generator.GetJargon() );
```

Code 3.10: Example.3

```
ylnW56 ,
tmfC89$
uxlP38=
csbA85-
uycE74%
pcsZ04-
bayE26+
qvzM03!
mbpG67-
bceR37 .
```

3.3 Evaluate Ratio Strings

```
1 Evaluation eval = new Evaluation();
2 Ratio ratio = new Ratio();
3
4 // The passwords that will be tested here
5 string[] passwords = new string[]
6 {
7     "bob", "hello", "HomerJSimp", "IsThis4Me?",
8     "V3ryStr0ngPass-w0rd!"
9 };
```

Code 3.11: Create instances and define passwords before running the examples

3.3.1 Evaluate password strength using internal checker

```
1 foreach (string pass in passwords)
2 {
3     eval.Password = pass;
4
5     Console.WriteLine(" " + eval.GetStrengthString().ToUpper());
6     Console.WriteLine(" (" + pass.Length.ToString());
7     Console.WriteLine(")\t: " + pass);
8 }
```

Code 3.12: Example.1

```
USELESS (3)      : bob
WEAK (5)         : hello
MEDIUM (10)     : HomerJSimp
STRONG (10)     : IsThis4Me?
BEST (20)       : V3ryStr0ngPass-w0rd!
```

3.3.2 Ratio objects can "accept" strings based on pattern

```
1 // 3 Lowercase, 1 Uppercase and 1 Digit
2 ratio = new Ratio("LLLUDD");
3
4 // Or set the ratio like this
5 ratio.Value = Ratio.Lowercases(3) + Ratio.Uppercases(1) + Ratio.Digits(2);
6
7 foreach (string pass in passwords)
8 {
9     if (ratio.Accepts(pass))
10        Console.WriteLine(" ACCEPTED! : " + pass);
11    else
12        Console.WriteLine(" rejected : " + pass);
13 }
```

Code 3.13: Example.2

```
rejected : bob
rejected : hello
rejected : HomerJSimp
rejected : IsThis4Me?
ACCEPTED! : V3ryStr0ngPass-w0rd!
```

3.3.3 Ratio can tell you what the string needs too be accepted

```
1 // 3 Lowercase, 1 Uppercase and 1 Digit
2 ratio = new Ratio("LLLUDD");
3
4 // Or set the ratio like this
5 ratio.SetRatio(3, 1, 2, 0);
6
7 string accepted = "";
8 string needthis = "";
9
10 foreach (string pass in passwords)
11 {
12     if (ratio.Accepts(pass, out needthis, out accepted))
13         Console.WriteLine(" ACCEPTED! : " + pass);
14     else
15     {
16         Console.WriteLine(" rejected : " + pass);
17         Console.WriteLine(" - accepted parts: " + accepted);
18         Console.WriteLine(" - what is needed: " + needthis);
19         Console.WriteLine();
20     }
21 }
```

Code 3.14: Example.3

```
rejected : bob
- accepted parts: LLL
- what is needed: UDD

rejected : hello
- accepted parts: LLL
- what is needed: UDD

rejected : HomerJSimp
- accepted parts: LLLU
- what is needed: DD

rejected : IsThis4Me?
- accepted parts: LLLUD
- what is needed: D

ACCEPTED! : V3ryStr0ngPass-w0rd!
```

3.3.4 Use OR-brackets to allow characters to be of more than one type

```

1 // 3 Lowercase and with two OR brackets, you allow: UU, DD or UD
2 //   - 2 Uppercase OR 2 Digits, OR 1 Uppercase and 1 Digit
3 ratio . Value = "LLL[UD][UD]";
4
5 // Or set it this way
6 char[] chars = new char[]
7 {
8     // Singular are "char constants"
9     Ratio.Lowercase, Ratio.Lowercase, Ratio.Lowercase,
10    Ratio.OrBegin, Ratio.Uppercase, Ratio.Digit, Ratio.OrEnds,
11    Ratio.OrBegin, Ratio.Uppercase, Ratio.Digit, Ratio.OrEnds
12 };
13 ratio . Value = new String(chars);
14
15 // Now more strings will be accepted because of wider acceptance
16 foreach (string pass in passwords)
17 {
18     if (ratio . Accepts(pass, out needthis, out accepted))
19         Console.WriteLine(" ACCEPTED! : " + pass);
20     else
21     {
22         Console.WriteLine(" rejected : " + pass);
23         Console.WriteLine(" - accepted parts: " + accepted);
24         Console.WriteLine(" - what is needed: " + needthis);
25         Console.WriteLine();
26     }
27 }

```

Code 3.15: Example.4

```

rejected : bob
- accepted parts: LLL
- what is needed: [UD][UD]

rejected : hello
- accepted parts: LLL
- what is needed: [UD][UD]

ACCEPTED! : HomerJSimp
ACCEPTED! : IsThis4Me?
ACCEPTED! : V3ryStr0ngPass-w0rd!

```

3.3.5 Let Ratio only care about length and symbols

```

1 // 1 Symbol and 6 Anything (can be any character)
2 ratio . Value = "AAAAAAS";
3
4 // Or set it this way
5 ratio . Value = Ratio.Anythings(6) + Ratio.Symbols(1);
6
7 // Length must be 7 and contain one symbol
8 foreach (string pass in passwords)
9 {
10     // "A" is turned into something else when "accepted", so
11     // the accepted string never contains "A" values!!
12     if (ratio . Accepts(pass, out needthis, out accepted))

```

```
13     Console.WriteLine(" ACCEPTED! : " + pass);
14     else
15     {
16         Console.WriteLine(" rejected : " + pass);
17         Console.WriteLine(" - accepted parts: " + accepted);
18         Console.WriteLine(" - what is needed: " + needthis);
19         Console.WriteLine();
20     }
21 }
```

Code 3.16: Example.5

```
rejected : bob
- accepted parts: LLL
- what is needed: SAAA

rejected : hello
- accepted parts: LLLLLL
- what is needed: SA

rejected : HomerJSimp
- accepted parts: LLLLLLU
- what is needed: S

ACCEPTED! : IsThis4Me?
ACCEPTED! : V3ryStr0ngPass-w0rd!
```

3.4 Evaluate Password Policies

```

1 PasswordPolicy policy = new PasswordPolicy();
2 policy.LoadDefaults();
3
4 // The passwords that will be tested here
5 string[] passwords = new string[]
6 {
7     "bob", "hello", "HomerJSimp", "IsThis4Me?",
8     "V3ryStr0ngPass-w0rd!"
9 };

```

Code 3.17: Create instances and define passwords before running the examples

3.4.1 Evaluate password strength using Password Policy

```

1 foreach (string pass in passwords)
2 {
3     Strength strength = policy.GetStrength(pass);
4
5     Console.Write(" " + strength.ToString().ToUpper());
6     Console.Write(" (" + pass.Length.ToString());
7     Console.WriteLine("\t: " + pass);
8 }

```

Code 3.18: Example.1

```

USELESS (3)      : bob
WEAK (5)        : hello
MEDIUM (10)     : HomerJSimp
STRONG (10)     : IsThis4Me?
BEST (20)       : V3ryStr0ngPass-w0rd !

```

3.4.2 Policies are built on Ratio objects with assigned strengths

```

1 foreach (PolicyRatio pRatio in policy.Items)
2 {
3     Console.Write(" " + pRatio.Strength.ToString() + " (" + pRatio.Length.ToString() + ")");
4     Console.WriteLine("\t: " + pRatio.Value.ToUpper() + "\n");
5 }

```

Code 3.19: Example.2

```

NotRated (0)    :
Useless (1)     : A
Weak (5)        : AAAAA
Medium (6)      : [UDS]AAAAA
Strong (10)     : S[UD]AAAAA
Best (14)       : S[UD][UD]AAAAA

```

3.4.3 Customize the Password Policy to create your own Strength Checker

```

1 // Not ideal but proves a point
2 policy.SetPolicy(Strength.NotRated, "");
3 policy.SetPolicy(Strength.Useless, "A");
4 policy.SetPolicy(Strength.Weak, "AAA");
5 policy.SetPolicy(Strength.Medium, "AAAA");
6 policy.SetPolicy(Strength.Strong, "AAAAA");
7 policy.SetPolicy(Strength.Best, "AAAAAA");
8
9 // Or set the BEST policy like this
10 policy[Strength.Best].SetRatio(6);
11 policy[Strength.Best].Value = Ratio.Anythings(6);
12
13 // Now you get much higher strength for the passwords
14 foreach (string pass in passwords)
15 {
16     Strength strength = policy.GetStrength(pass);
17
18     Console.Write(" " + strength.ToString().ToUpper());
19     Console.Write(" (" + pass.Length.ToString());
20     Console.WriteLine("\t: " + pass);
21 }

```

Code 3.20: Example.3

```

WEAK (3)      : bob
STRONG (5)    : hello
BEST (10)     : HomerJSimp
BEST (10)     : IsThis4Me?
BEST (20)     : V3ryStr0ngPass-w0rd!

```

3.4.4 Let the Evaluator tell you what you need to improve strength

```

1 // Begin with loading default policy
2 policy.LoadDefaults();
3
4 // Now you get much higher strength for the passwords
5 foreach (string pass in passwords)
6 {
7     // Next strength from current and what is needed to get there
8     Strength next = Strength.NotRated;
9     string improve = "";
10
11     // Get current strength
12     Strength strength = policy.GetStrength(pass, out improve, out next);
13
14     // If current matches next – its the best!
15     if (strength == next)
16     {
17         Console.Write(" " + strength.ToString().ToUpper());
18         Console.Write(" (" + pass.Length.ToString() + ") \t: " + pass + "\n");
19
20         Console.WriteLine(" (best possible password for this policy)");
21     }
22
23     // Room for improvement

```

```
24     else
25     {
26         Console.Write(" " + strength.ToString(). ToUpper());
27         Console.Write(" (" + pass.Length.ToString() + ") \t: " + pass + "\n");
28
29         Console.WriteLine(" – next strength: " + next.ToString(). ToUpper());
30         Console.WriteLine(" – need improved: " + improve);
31         Console.WriteLine();
32     }
33 }
```

Code 3.21: Example.4

```
USELESS (3)      : bob
– next strength : WEAK
– need improved : AA

WEAK (5)         : hello
– next strength : MEDIUM
– need improved : [UDS]

MEDIUM (10)     : HomerJSimp
– next strength : STRONG
– need improved : S

STRONG (10)     : IsThis4Me?
– next strength : BEST
– need improved : AAAAA

BEST (20)       : V3ryStr0ngPass–w0rd!
(best possible password for this policy)
```

Part 4

Password Policies in general

Password Policy is a combination of rules that require a string to contain certain amount of different character types, and to have minimum and/or maximum length. This part introduces guidelines that users should keep in mind when creating passwords, and suggests few mental algorithms to create secure passwords that are easy to remember.

4.1 Password guidelines

A password is a secret combination of characters that is used to authenticate the user. In modern society, people are using passwords regularly on a daily basis. Many systems today rely heavily on passwords, consequently, the complexity and strength of the password has become important. Password policies and guidelines have been designed to make passwords stronger, easy to remember and harder for any unauthorized individuals to reveal it.

The following list outlines how stronger passwords are constructed:

Length minimum 8 characters or longer, 14 is ideal

Casing both upper and lower case characters

Digits and symbols intertwine digits and symbols with the characters

Spelling misspelled words are harder to guess, but avoid very common misspellings

Repetition avoid combination of the same letters or digits: abcabcabc, 111222333

Keyboard avoid keyboard combinations, such as: qwerty, asdfgh

Names avoid using your name or the username in the password

Unique never use the same password twice

Dictionary avoid dictionary words in any language

Substitution avoid common substitution¹ of letters as only obscurity, for instance [e=3]: th3dud3 and [o=0, a=@, i=1]: pr0gr@mm1ng

4.2 Human algorithms

Instead of remembering dozens of passwords for numerous logins, it is much easier to remember one or two algorithms.

- *Create passwords from passphrases*: Take the first letter from every word in a memorable sentence, include commas and exclamation, and substitute obvious letters to symbols. Examples of this is shown in table 4.1.

Passphrase	The actual password
on COPS, four guys smoked hash at Joey's	oC,4gs#@J
my two dogs, Brian and Spot run for fun	m2d,B&Sr4f
George and I donated dollars to the Red Cross	G&Id\$thRx

Table 4.1: *Passphrases turned into passwords.*

- *Jump keyboard positions*: Move the keyboard index up by one key, so as you would type simple word gibberish is written instead. Consider holding Shift down for every third character, or every other key pressed in the number row. Examples of this is shown in table 4.2.

Word(s)	The actual password
somepassword	w9j30qww294e
MyDogIsOld	J6E9t(w)oe
REdbLUegREen	\$#egO/3t\$#3h

Table 4.2: *Jumped passwords.*

- *Add uniqueness*: Adding uniqueness to above methods by adding letters from the systems or websites title or logo, and incorporate noticeable colours and/or numbers as well. The figure 4.1 on the next page is used in the examples in table 4.3, which are very short in length and could possibly be longer and/or more complicated.

- *Example.1*: The last two letters in the title and the first letter in the color.
- *Example.2*: First letter in every word, followed by the place number of the second character in the title.

¹See leet speak: <http://en.wikipedia.org/wiki/Leet>



Image 4.1: Example logos for “Add uniqueness” to stronger password construction.

Titles	Example.1	Example.2
GMail	[<i>GMail</i> + Red]: ilR	[G <i>M</i> ail]: GM2
Facebook	[<i>Facebook</i> + Blue]: okB	[F ace <i>book</i>]: Fb5
Amazon	[<i>Amazon</i> + Yellow]: onY	[<i>Ama</i> z on]: Az4

Table 4.3: Adding uniqueness to passwords.

4.3 Password strength evaluation

One way to check password strength, is to compare the password mentally to the guidelines and see how many of them are followed. That can be confusing when dealing with long passwords, and it does not give you any relative results as people can have very different opinions. The use of software that can evaluate password strength become essential in any system which handles user management.