

Towards First-Aid IdM-Toolkit

Florian Löffler¹, Krasimir Zhelev², Frank Tröger³, [Hendrik Eggers](mailto:Hendrik.Eggers@rrze.uni-erlangen.de)⁴

¹Friedrich-Alexander-University Erlangen-Nuremberg, Regional Computing Centre, Martensstr. 1, 91058 Erlangen, Germany, Florian.Loeffler@rrze.uni-erlangen.de. ²Friedrich-Alexander-University Erlangen-Nuremberg, Regional Computing Centre, Martensstr. 1, 91058 Erlangen, Germany, Krasimir.Zhelev@rrze.uni-erlangen.de. ³Friedrich-Alexander-University Erlangen-Nuremberg, Regional Computing Centre, Martensstr. 1, 91058 Erlangen, Germany, Frank.Troeger@rrze.uni-erlangen.de. ⁴Friedrich-Alexander-University Erlangen-Nuremberg, Regional Computing Centre, Martensstr. 1, 91058 Erlangen, Germany, Hendrik.Eggers@rrze.uni-erlangen.de.

Keywords

identity management, identifiers, passwords, tools, Java, open source, system administration

1. EXECUTIVE SUMMARY

As a cornerstone of the *First-Aid IdM-Toolkit* this paper introduces the two Java tools `jpwgen` (`jpwgen`) – a password generator – and `jidgen` (`jidgen`) – a generator for identifiers – developed at Regional Computing Centre Erlangen (aka RRZE).

Both tools face a common objective to address standard problems in the field of identity management. Instead of programmers defining policies within hard-coded solutions and always “reinventing the wheel”, the `j*gen` tools offer a domain specific solution which makes policies configurable.

`Jidgen` is an easy-to-use but powerful Java-based generator for identifiers. It utilizes a template language and has an in-built collision detection to avoid duplicate identifiers.

`Jpwgen` is a Java-based password generator whose functionality extends the popular “`pwgen`” (T'so 2001) program.

Both tools can be used on the command line and can be embedded as a library as well.

1.1. Background

While doing research on Identity Management and conceptualizing systems for organizations, we produce small powerful tools to address the needs for the “one server one guy show”. The tools offer standardized, stand-alone solutions for simplified out-of-the-box usage.

1.2. Conclusions

Two tools from the First-Aid IdM-Toolkit were introduced to release developers from the vendor-lock.

2. Introduction

The Regional Computing Centre Erlangen (RRZE) at Friedrich-Alexander University Erlangen-Nuremberg started the IDMone (IDMone; Eggers 2007) project with the goal to reconstruct the existent user management system. While analyzing the requirements a reoccurring topic was the user names composition. It soon became clear that different types of user names have to be generated. Some of these types will change over time, new types will be imposed and other types will remain because these are semantically bounded. After a short research no suitable solution was found that could satisfy the variety of restrictions. Thus a decision was made to go for a general solution.

3. Motivation

In the process of developing complex systems it always happens that requirements change over time and/or additional ones are introduced, so flexible solutions have to be introduced as early as possible in order to avoid later refactoring. From our point of view a reasonably flexible solution should be following one of the major Unix design principles - to "do one thing and do it well" (Wikipedia - Unix philosophy). These solutions should be easily embeddable in composite systems but should also address the needs of the "one server, one guy show".

4. Background

This paper introduces two of our most valuable tools which follow the above mentioned concept. The first is about providing object identifiers and the second about generating diverse passwords.

4.1. Identifiers

In the field of identity management the most visible identifiers are the ones given to the persons for authentication. Most of the time these identifiers, given in the form of user names, are in conformance with management-defined rules. These rules determine the structural design of the identifiers.

For example the first two characters may contain the user's affiliation to the organization followed by a mixture of characters from the user's real name and a consecutive number to resolve conflicts.

A common requirement is uniqueness. Ensuring uniqueness implies real-time interaction with the servers building up the identity management domain.

4.2. Passwords

Unlike identifiers, the structural design of passwords is usually not so strictly defined. However, passwords should comply to certain policies. These policies are introduced by management or security experts and aim at ensuring a certain level of complexity so that a password can be considered secure enough.

Password policies limit the set of allowable passwords by imposing certain types of restrictions. A typical type of restriction are *positive restrictions* which ensures certain criteria are met. These criteria are usually related to length, inclusions of digits, non-alphanumerical characters and so on. The other type of restrictions are the *negative restrictions*, which forbid some predefined conditions. Typical examples are password restrictions that filter out passwords which would otherwise contain the first name of the user or contain repeating parts.

Thus the password complexity can be seen as a unification of all the imposed restrictions by the password policy. Password policies should be constructed in such a manner that they ensure secure passwords but do not excessively limit the number of possible fits.

A serious concern when it comes to passwords is keeping them safe. Excessively long and complicated passwords are not out-of-the-box more secure. Such passwords often end up written on sheets of paper lying on the desk or in files saved on hard-disks. Thus security can be easily compromised. A popular solution to this problem is to generate passwords which are easy to remember and this can be achieved by the concept of pronounceable passwords (Schneier 2007). Pronounceable password can be generated by arranging consonants and vowel in a manner similar to the way these are used in common languages. Additionally, by incorporating diphthongs (like "ei") or phonemes (like "th") for various languages can make passwords even easier to remember.

Readability is also an important aspect of passwords. This aspect is normally handled by simply avoiding ambiguous looking characters like "O-0" or "1-l".

4.3. Common problems

A problem shared by both - identifier and password generation is the requirement for a social acceptance filter. In the process of building random sequences of characters there always exists the possibility of generating a sequence that resembles a vulgar, insulting or politically unacceptable word. The problem can be partially solved by a blacklist against which all generated sequences are checked. However, it is practically infeasible to assemble a list of all such words and combinations which somehow resemble them. Such list are often related to country, language, ethnic group, race and so on and thus left as a users responsibility.

5. Goals

By considering the above mentioned problems related to trivial tasks such as generating identifiers and passwords it soon became clear that the overall complexity of the whole identity management project will be overwhelming. Thus a decision was made to break up the project into small blocks of manageable code, that could be changed within predictable time terms in case required. Blocks whose quality and reliability could be easily tested. Our goal was to build robust and flexible tools which can be used in different situations and under various conditions. These tools are currently embedded in our bigger frameworks but run as well on the command line and help us in our day-to-day activities.

6. j*gen Tools

A general purpose programming language with wide-spread usage and support was identified as the natural choice for the tools under development. This would allow us to built components which can easily be embedded into other components or allow us to use them standalone or even from other scripting languages. Due to the above mentioned reasons the choice fall on Java. For fast scripting, using the libraries from within Groovy is recommended.

The "j" in the tools' names originates from Java and the "gen" part comes from the purpose they are used for - namely: generating. A significant effort was made to keep the tools as small as possible and free of third party libraries. Completely avoiding helper libraries was not an option due to time constraints, however we limited the included ones as much as we could. We aim at developing FOSS (short: Free and Open Source Software) solutions thus all project dependencies are open source as well.

6.1. jidgen

This tool was initially inspired by a collection of 15 ID generation algorithms used in German universities assembled by Klaus Rosifka - Friedrich Schiller-University of Jena.

The rules for identifier generation are usually provided in the form of an informal description, as the example given in 4.1. This textual description can be easily transformed in a compact template describing accurately the composition of the identifier. The main idea is to automate the process of identifiers generation by providing a simple and straightforward template language for defining the structural composition of the required identifiers.

The template language is composed of multiple elements separated by colons. If an element should only become active to resolve a conflict (e.g. the id already exists) it can be marked as a resolver element by enclosing it in squared brackets. An EBNF-like ([ISO/IEC 14977 1996](#)) description of jidgen's template language looks like this:

```
template      :=  element { ":" element }
element       :=  resolverElement | standardElement
resolverElement :=  "[" standardElement "]"
standardElement :=  static | basic | substring | counter | random
static        :=  ...
...

```

Identifiers uniqueness and blacklists are both supported in jidgen. This is achieved by the utilisation of filters. jidgen is equipped with a filter chain. All filters within the chain are sequentially invoked and the returned result is evaluated. The filter chain is exclusive - meaning if a single filter rejects the identifier a new one has to be generated.

Currently two collision filters are distributed with jidgen and two other under development. The first of the distributed ones is used to start a Unix script and evaluate its result, the second one filters out users listed in the `"/etc/passwd"` or arbitrary identifiers in a given file. The filters under development are one LDAP filter and one SQL filter that can handle different search criteria.

The filling of the blacklist filter is left to the users of the tool.

Usage

A detailed usage description of the jidgen tool and its template language can be found on its web-site (jidgen). In this paper only short examples will be provided.

In jidgen a variable is defined by the `-T[a-z]` options, which indicate the variables are part of the template. ASCII symbols from `"a-z"` are reserved for variable names, these symbols carry no semantic and are free to use as suitable. The following example defines and initializes three variables, named correspondingly `"d"`, `"f"`, and `"l"`:

```
-Td IT -Tf John -Tl Doe
```

Variable values have to be set by the external environment and will be handled within the template which is designated by the `-T` option, for example:

```
-T 2d:2f:3l:NNN++
```

The above shown template will take the first two characters of the “*d*” variable, the first two characters of the “*f*” variable, the first three characters of the “*l*” variable and will add a three digit consecutive number at the end.

To better illustrate the template language an example call on the command line interface is show below together with the corresponding result:

```
> java -jar jidgen.jar -Td IT -Tf John -Tl Doe -T 2d:2f:3l:NNN++
itjodoe000
```

A code snippet with a slightly modified template is shown below to demonstrate how jidgen can be used as an embedded library:

```
IdGenerator idGen = new IdGenerator("-Tf John -Tl Doe -T 1f:l:[N2++]");
List <String> ids = idGen.generateIDs(5);
```

On the first line an instance of the *IdGenerator* is created. The variables and the template are provided as constructor parameters. On the second line five distinct identifiers are generated. It has to be noted that the *IdGenerator* preserves its state in order to keep its counter variables intact. Unless the generation has to continue from the last identifier, a new instance with all variables and the template has to be created. This problem can be circumvented by utilizing a proper filter but a decision was made to leave the implementation so for the time being to avoid unnecessary user confusion.

In the second example the counter element at the end of the template is marked as a resolver. A resolver only becomes active if otherwise the generated identifiers would not be distinct. The obtained result of the above call looks like:

```
ids = {
    "jdoe", "jdoe00", "jdoe01", "jdeo02", "jdoe03"
}
```

6.2. jpwgen

Passwords are generated in compliance to password policies created by management or security experts. These password policies have to be later translated into code. jpwgen helps in the process of this translation by mapping the most commonly used imposed and dis-allowance restrictions to program parameters.

The functionality of jpwgen resembles the one provided by the popular unix pwgen (T'so 2001) program. While pwgen is an extremely useful utility it concentrates mainly on imposed restrictions, jpwgen goes one step further and provides also dis-allowance restrictions handling. This is implemented by using regular expressions for checking the different types of conditions. Various checks are included by default, for example whether a password starts with a symbol or not, with a digit or not and so on.

jpwgen uses a slightly modified version of pwgen's algorithm for generation of pronounceable passwords (see 4.2. Passwords). The algorithm utilizes a predefined set of vowels and consonants that are glued together by successive iterations. Diphthongs and phonemes are also inserted in the generated passwords in order to make them easier to remember.

Furthermore jpwgen can utilize all secure random generators that are registered within the running instance of the Java virtual machine(JVM). The standard random generator can still be used in case there are entropy accumulation problems on the JVM, where jpwgen runs.

jpwgen is equipped with an exclusive filter chain which functions similar to the one used by jidgen. Several filters can be registered within the password generator and these are sequentially invoked and the results are evaluated. Two default filters are registered at instantiation time one of which is intended to be used as a blacklist. Filters are to be used in case default jpwgen parameters are not enough to fully define a given password policy. In some cases a filter can even make calls to a server and try to set a test password to a dummy object in order to check password conformance.

Usage

A detailed description of all jpwgen parameters and a comparison with pwgen can be found on jpwgen's web site (jpwgen). In this paper only short examples will be provided.

The following example demonstrates how to use jpwgen from the command line. It generates 60 passwords (-N 60) each with a size of 10 characters (-s 10), prints them in columns to the terminal (-C), does not use any special symbols (-Y) and uses the SHA1 pseudo random number generator together with the SUN security provider.

```
> java -jar jpwgen.jar -N 60 -s 10 -C -Y -S SHA1PRNG SUN
choob2Iquo oquo6eIfu7 qUee1jieth vah5oenooL eiyie4oHsh lahN5yohti
pi5aexahCo mooz1Hie6r quah9Cia5p leaghoo7Ba lol5CoGaib aka2Enei8f
...
```

The next example show jpwgen's usage as an embedded library. The code snippet fills the password policy description in a string, initializes the blacklist filter with the word "bad_word" and then starts the generation process with the provided parameters.

```
String flags = "-N 50 -M 100 -y -o -s 16 -m -q -i";
String[] args = flags.split(" ");
PwGenerator generator = new PwGenerator();
generator.getDefaultBlacklistFilter().addToBlacklist("bad_word");
List <String> passwords = generator.process(args);
```

The parameters provided in this example would generate 50 passwords (-N 50) with a maximum of 100 attempts per password to avoid deadlock (-M 100), including exactly one special symbol (-y -o), with a size of 16 characters (-s 16), including exactly one upper case letter (-m) and exactly one digit (-q). None of the passwords will start with a digit because of the "-i" flag.

Checking for deadlock can be extremely important in cases where wrong parameters are fed into the password generator. The password generator does not check for colliding parameters and it can happen the process generation is initialized with a password policy description that is infeasible. For example, a password that does not start with a letter, neither with a digit nor with a non-alphanumeric character - in this case the password generator will iterate endless in search of a non-existent password if it was not for the deadlock check. If not set a default deadlock value is used which is reasonably high.

The result obtained from the last example will be something like:

```
passwords = {
    "queijoh6Tiung(ae", "Peeneeboo$n3oopa", "zoorae/peshooN8d",
    "phohtah9menohPo+", "iwaid1ohtuYeewo@", "echai7ceoPooshie]",
    "rohj~eech2cahjee", ...
}
```

7. Summary

Although there is almost no system without the need of identifiers and passwords, no flexible and comprehensive solutions exist for their generation. Of course, a lot of enterprise applications include out-of-the-box generators, these are usually not flexible enough to satisfy all use cases.

Two tools from the First-Aid IdM-Toolkit were introduced to release developers from the vendor-lock. Other small and robust tools following the same concept will be published soon.

8. REFERENCES

Apache Commons. <http://commons.apache.org/> (last visited 05.02.2009).

Eggers, H. (2007). Six months more - about another ambiguous identity management project. EUNIS 2007, Grenoble,
<http://www.eunis.org/events/congresses/eunis2007/CD/pdf/papers/p97.pdf> (last visit 03.02.2009).

IDMone. http://www.rrze.uni-erlangen.de/forschung/abgeschlossene-projekte/idm_en.shtml (last visited 27.05.2009).

[ISO/IEC 14977 \(1996\).](http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip)

[http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996\(E\).zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip) (last visited 27.05.2009).

jidgen. <http://jidgen.berlios.de/> (last visited 05.02.2009).

jpwgen. <http://jpwgen.berlios.de/> (last visited 05.02.2009).

Schneier, B. (2007). Choosing Secure Passwords.

http://www.schneier.com/blog/archives/2007/01/choosing_secure.html (last visited 27.05.2009).

Ts'o, Theodore (2001). pwgen - Password Generator.

<http://sourceforge.net/projects/pwgen/> (last visited 26.05.2009).

Wikipedia - Unix philosophy. http://en.wikipedia.org/wiki/Unix_philosophy (last visited 14.05.2009).