nor-id-num

Norwegian Identity Numbers

This project is for validating and generating Norwegian identity numbers. There are 3 kinds supported:

- Organization numbers
- Birth numbers
- D-numbers

Organization numbers

Organization numbers identify commercial organizations as well as non-profit organizations.

The identifying numbers consist of 9 digits. The first digit is either 8 or 9. The last digit is a checksum.

Birth numbers

Birth numbers identify individual persons. Every Norwegian citizen is given a unique birth number within a few days after being born. Foreigners may acquire a birth number e.g. when granted citizenship.

The numbers consist of 11 digits. The first 6 digits are the birth date (DDMMYY), the next 3 digits constitute an individual number for the particular date, and the last 2 digits are both checksum digits. Ranges within the individual number determine the century. The last digit of the individual number determines gender (even numbers for female individuals and odd numbers for male).

D-numbers

D-numbers is used to identify foreign individuals that don't qualify for a birth number.

The D-number is constructed exactly the same way as birth numbers, but the value 4 is added to the very first digit (e.g. the date 311299 turns into 711299).

Visual Studio solution

The Visual Studio solution consists of 3 projects:

- NinEngine
- NinUi
- UnitTest

NinEngine

This is the core project providing functionality for validating and generating identity numbers. This is all you need if you want to use the validation or generation functionality.

NinUi

This is just a sample user interface to play with the capabilities of the core module.

UnitTest

You guessed it – a project implementing unit tests for the core engine module.

Using the code

There are three main classes: OrganizationNumber, BirthNumber and DNumber.

Instantiating and validating

The classes all have a constructor accepting a string. If the string doesn't represent a valid identification number, an exception (a NinException object) is thrown. This object holds detailed information about what is wrong with the number.

Also, there is a static Create method, returning null if the passed number doesn't represent a valid identification number. This way you don't get any information about the reason for rejecting the number passed.

Generating one number at a time

Completely random

The static method OneRandom generates a random (valid) identification number. This will always succeed.

Matching a pattern

Another incarnation of OneRandom accepts a pattern and optionally a retry count. Using this method you can force particular digits in particular positions and let other positions have a random digit. For each position, provide either a digit where you want to force or a question mark where you want to randomize.

This may or may not produce a valid identification number. For instance, if you force the digits 99 in the two positions representing month in a birth number pattern, it is not possible to generate a valid number. The method will give up after a number of failed retries, returning null.

Date boundaries and gender

For birth numbers and D-numbers, there is yet another variation of the OneRandom method, accepting two dates and a gender request. The generated number will always represent a birth date between (inclusive) the two dates. Also, if you request a particular gender, the generated number will match that.

Generating several numbers at once

All possible

The statis method AllPossible returns all possible numbers in an `IEnumerable<>` collection. The number of legal possibilities are found in the constant `PossibleLegalVariations` (18,181,818 for organization numbers, 26,412,179 for birth numbers, and 26,412,204 for D-numbers).

Warning: Generating all those numbers may consume a considerable amount of time! Be patient, the method will eventually finish.

As many as you want

The static method ManyRandom accepts a count and returns that many identity numbers in an IEnumberable<> collection. First all possible numbers are generated, then the requested number are picked from those generated. This way, generating 10 numbers takes (almost) as long as generating 10,000,000.