

# ccnum.scm

## Credit Card Number Utilities in Scheme

Version 0.2  
2005-03-29

Neil W. Van Dyke  
neil@neilvandyke.org

<http://www.neilvandyke.org/ccnum-scm/>

Copyright © 2004 - 2005 Neil W. Van Dyke. This program is Free Software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU Lesser General Public License [LGPL] for details. For other license options and consulting, contact the author.

## 1 Introduction

This is a Scheme library of a few utilities for validating and formatting credit card numbers. Credit card numbers are represented as strings containing digits and arbitrary whitespace. The procedures are based on information gleaned from dozens of written artifacts of credit card number oral tradition, including [Bradbury], [Gilleland], and [Hippy]. The author invites free copies of authoritative documentation.

This library should work with any R5RS-compliant Scheme implementation that has an **error** procedure similar to that in [SRFI-23].

Achtung! Do not use this library as anything other than a novelty unless you understand the code thoroughly and can invest in validation of it. (The same caution applies to all the other credit card number checking routines the author has seen in other languages, most of which are surprisingly inefficient and otherwise do not instill confidence.)

## 2 Validation

The following procedures provide different ways of validating credit card numbers. Most applications will use `credit-card-number-check-digit-ok?` or `credit-card-number-seems-ok?`.

`check-credit-card-number` *str* [Procedure]  
Performs a partial validation of the credit card number in *str*. If the check digit is incorrect, then **#f** is yielded:

```
(check-credit-card-number "4408041234567890") ⇒ #f
```

If the check digit is correct, but the issuer cannot be determined, then an integer representing the digit count is yielded:

```
(check-credit-card-number "1234567890123452") ⇒ 16
```

If the check digit is correct and issuer can be determined, then a list of three elements is returned. The first element is a boolean value for whether or not the digit count matches what is known about how many digits the issuer uses for this class of cards. The second element is the digit count. The third element is a symbol loosely identifying the issuer. For example:

```
(check-credit-card-number "5551 2121 9") ⇒ (#f 9 mastercard)
```

```
(check-credit-card-number "4408041234567893") ⇒ (#t 16 visa)
```

`credit-card-number-check-digit-ok?` *str* [Procedure]

Predicate for whether or not the check digit of credit card number *str* is correct.

```
(credit-card-number-check-digit-ok? "4408 0412 3456 7893") ⇒ #t
```

```
(credit-card-number-check-digit-ok? "4408 0412 3456 7890") ⇒ #f
```

```
(credit-card-number-check-digit-ok? "trump") ⇒ #f
```

`credit-card-number-seems-ok?` *str* [Procedure]

Predicate for whether or not the credit card number *str* “seems” to be valid. For a credit card number to “seem” valid, the check digit must be correct, the issuer must be identified, and the digit count must match what is known about issuer digit counts. In the following example the check digit is correct, and the issuer (MasterCard) has been identified, but the digit count is too low for a MasterCard number:

```
(credit-card-number-check-digit-ok? "5551 2121 9") ⇒ #t
```

```
(credit-card-number-seems-ok? "5551 2121 9") ⇒ #f
```

### 3 Formatting

Two procedures are provided for formatting credit card numbers.

`write-formatted-credit-card-number` *str port* [Procedure]

Writes credit card number *str* to output port *port*, using a format similar to that used on many credit cards. In the current version of this package, the format is always groups of four digits separated by single space characters, although a future version might mimic the format used by the issuer.

```
(write-formatted-credit-card-number " 1 23 456 7890 12345 6 "  
                                     (current-output-port))
```

```
→ 1234 5678 9012 3456
```

`formatted-credit-card-number` *str* [Procedure]

Yields a formatted string representation of credit card number *str* like that written by `write-formatted-credit-card-number`.

```
(formatted-credit-card-number "1234567890123456")
```

```
⇒ "1234 5678 9012 3456"
```

```
(formatted-credit-card-number " 12 34 56 7890 1234 56")  
⇒ "1234 5678 9012 3456"
```

```
(formatted-credit-card-number "123 abc") ⇒ #f
```

Note that `(write-formatted-credit-card-number n p)` is more efficient than `(display (formatted-credit-card-number n) p)`.

## History

Version 0.2 — 2005-03-29

Minus characters (`#\-`) are now accepted as blanks in credit card numbers.

Version 0.1 — 2004-05-15

First release.

## References

[Bradbury]

Jeremy Scott Bradbury, “Credit Card Check Digit,” Web page, viewed 2004-05-15.

<http://www.cs.queensu.ca/~bradbury/checkdigit/creditcardcheck.htm>

[Gilleland]

Michael Gilleland, “Anatomy of Credit Card Numbers,” Web page, viewed 2004-05-15.

<http://www.merriampark.com/anatomycc.htm>

[Hippy]

Happy Hippy, “Credit Card Magic,” Web page, viewed 2004-05-15.

<http://www.hippy.freemove.co.uk/credcard.htm>

[LGPL]

Free Software Foundation, “GNU Lesser General Public License,” Version 2.1, 1999-02, 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

<http://www.gnu.org/copyleft/lesser.html>

[SRFI-23]

Stephan Houben, “Error reporting mechanism,” SRFI 23, 2001-04-26.

<http://srfi.schemers.org/srfi-23/srfi-23.html>