

**Beginning Perl Programming (X52.9543.001)****Handout 8: July 13, 2000****Readme file for CSV.pm package**

Module: Text::CSV

**Description:**

Text::CSV provides facilities for the composition and decomposition of comma-separated values. An instance of the Text::CSV class can combine fields into a CSV string and parse a CSV string into fields.

**Copying:**

Copyright (c) 1997 Alan Citterman. All rights reserved. This program is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

**Prerequisites:**

perl 5.002

**Build/Installation:**

Standard build/installation supported by ExtUtils::MakeMaker(3)...

```
perl Makefile.PL
make
make test
make install
```

**Recent Changes:**

Version 0.01 06/05/1997  
original version

**Planned Enhancements:**

+ extend constructor to accept parse() configuration to support other variations of CSV

**Author:**

Alan Citterman <alan@mfgrtl.com>

**Source code for CSV.pm**

```
package Text::CSV;
```

```
# Copyright (c) 1997 Alan Citterman. All rights reserved.
# This program is free software; you can redistribute it and/or
# modify it under the same terms as Perl itself.
```

```
#####
# HISTORY
#
# Written by:
#   Alan Citterman <alan@mfgrtl.com>
#
# Version 0.01 06/05/1997
#   original version
#####
```

```

require 5.002;

use strict;

BEGIN {
    use Exporter    ();
    use AutoLoader qw(AUTOLOAD);
    use vars       qw($VERSION @ISA @EXPORT @EXPORT_OK %EXPORT_TAGS);
    $VERSION =     '0.01';
    @ISA =         qw(Exporter AutoLoader);
    @EXPORT =      qw();
    @EXPORT_OK =   qw();
    %EXPORT_TAGS = qw();
}

1;

#####
# version
#
#   class/object method expecting no arguments and returning the version number
#   of Text::CSV.  there are no side-effects.
#####
sub version {
    return $VERSION;
}

#####
# new
#
#   class/object method expecting no arguments and returning a reference to a
#   newly created Text::CSV object.
#####
sub new {
    my $proto = shift;
    my $class = ref($proto) || $proto;
    my $self = {};
    $self->{'_STATUS'} = undef;
    $self->{'_ERROR_INPUT'} = undef;
    $self->{'_STRING'} = undef;
    $self->{'_FIELDS'} = undef;
    bless $self, $class;
    return $self;
}

#####
# status
#
#   object method returning the success or failure of the most recent combine()
#   or parse().  there are no side-effects.
#####
sub status {
    my $self = shift;
    return $self->{'_STATUS'};
}

#####
# error_input
#
#   object method returning the first invalid argument to the most recent

```

```

#   combine() or parse().  there are no side-effects.
#####
sub error_input {
    my $self = shift;
    return $self->{'_ERROR_INPUT'};
}

#####
# string
#
#   object method returning the result of the most recent combine() or the
#   input to the most recent parse(), whichever is more recent.  there are no
#   side-effects.
#####
sub string {
    my $self = shift;
    return $self->{'_STRING'};
}

#####
# fields
#
#   object method returning the result of the most recent parse() or the input
#   to the most recent combine(), whichever is more recent.  there are no
#   side-effects.
#####
sub fields {
    my $self = shift;
    if (ref($self->{'_FIELDS'})) {
        return @{$self->{'_FIELDS'}};
    }
    return undef;
}

#####
# combine
#
#   object method returning success or failure.  the given arguments are
#   combined into a single comma-separated value.  failure can be the result of
#   no arguments or an argument containing an invalid character.  side-effects
#   include:
#       setting status()
#       setting fields()
#       setting string()
#       setting error_input()
#####
sub combine {
    my $self = shift;
    my @part = @_;
    $self->{'_FIELDS'} = \@part;
    $self->{'_ERROR_INPUT'} = undef;
    $self->{'_STATUS'} = 0;
    $self->{'_STRING'} = '';
    my $column = '';
    my $combination = '';
    my $skip_comma = 1;
    if ($#part >= 0) {

        # at least one argument was given for "combining"...
        for $column (@part) {

```

```

    if ($column =~ /[^\t\040-\176]/) {

        # an argument contained an invalid character...
        $self->{'_ERROR_INPUT'} = $column;
        return $self->{'_STATUS'};
    }
    if ($skip_comma) {

        # do not put a comma before the first argument...
        $skip_comma = 0;
    } else {

        # do put a comma before all arguments except the first argument...
        $combination .= ',';
    }
    $column =~ s/\042/\042\042/go;
    $combination .= "\042";
    $combination .= $column;
    $combination .= "\042";
}
$self->{'_STRING'} = $combination;
$self->{'_STATUS'} = 1;
}
return $self->{'_STATUS'};
}

#####
# parse
#
# object method returning success or failure. the given argument is expected
# to be a valid comma-separated value. failure can be the result of
# no arguments or an argument containing an invalid sequence of characters.
# side-effects include:
# setting status()
# setting fields()
# setting string()
# setting error_input()
#####
sub parse {
    my $self = shift;
    $self->{'_STRING'} = shift;
    $self->{'_FIELDS'} = undef;
    $self->{'_ERROR_INPUT'} = $self->{'_STRING'};
    $self->{'_STATUS'} = 0;
    if (!defined($self->{'_STRING'})) {
        return $self->{'_STATUS'};
    }
    my $keep_biting = 1;
    my $palatable = 0;
    my $line = $self->{'_STRING'};
    if ($line =~ /\n$/) {
        chop($line);
        if ($line =~ /\r$/) {
            chop($line);
        }
    }
    my $mouthful = '';
    my @part = ();
    while ($keep_biting and
        ($palatable = $self->bite($line, $mouthful, $keep_biting))) {

```

```

    push(@part, $mouthful);
}
if ($palatable) {
    $self->{'_ERROR_INPUT'} = undef;
    $self->{'_FIELDS'} = \@part;
}
return $self->{'_STATUS'} = $palatable;
}

```

```

#####
# _bite
#
# *private* class/object method returning success or failure.  the arguments
# are:
#   - a reference to a comma-separated value string
#   - a reference to a return string
#   - a reference to a return boolean
# upon success the first comma-separated value of the csv string is
# transferred to the return string and the boolean is set to true if a comma
# followed that value.  in other words, "bite" one value off of csv
# returning the remaining string, the "piece" bitten, and if there's any
# more.  failure can be the result of the csv string containing an invalid
# sequence of characters.
#
# from the csv string and
# to be a valid comma-separated value.  failure can be the result of
# no arguments or an argument containing an invalid sequence of characters.
# side-effects include:
#   setting status()
#   setting fields()
#   setting string()
#   setting error_input()
#####

```

```

sub _bite {
    my ($self, $line_ref, $piece_ref, $bite_again_ref) = @_;
    my $in_quotes = 0;
    my $ok = 0;
    $$piece_ref = '';
    $$bite_again_ref = 0;
    while (1) {
        if (length($$line_ref) < 1) {

            # end of string...
            if ($in_quotes) {

                # end of string, missing closing double-quote...
                last;
            } else {

                # proper end of string...
                $ok = 1;
                last;
            }
        } elsif ($$line_ref =~ /\^\042/) {

            # double-quote...
            if ($in_quotes) {
                if (length($$line_ref) == 1) {

                    # closing double-quote at end of string...

```

```

    substr($$line_ref, 0, 1) = '';
    $ok = 1;
    last;
} elsif ($$line_ref =~ /^\\042\\042/) {

    # an embedded double-quote...
    $$piece_ref .= "\\042";
    substr($$line_ref, 0, 2) = '';
} elsif ($$line_ref =~ /^\\042,/ ) {

    # closing double-quote followed by a comma...
    substr($$line_ref, 0, 2) = '';
    $$bite_again_ref = 1;
    $ok = 1;
    last;
} else {

    # double-quote, followed by undesirable character (bad character sequence)...
    last;
}
} else {
if (length($$piece_ref) < 1) {

    # starting double-quote at beginning of string
    $in_quotes = 1;
    substr($$line_ref, 0, 1) = '';
} else {

    # double-quote, outside of double-quotes (bad character sequence)...
    last;
}
}
} elsif ($$line_ref =~ /^,/ ) {

    # comma...
    if ($in_quotes) {

        # a comma, inside double-quotes...
        $$piece_ref .= substr($$line_ref, 0 ,1);
        substr($$line_ref, 0, 1) = '';
    } else {

        # a comma, which separates values...
        substr($$line_ref, 0, 1) = '';
        $$bite_again_ref = 1;
        $ok = 1;
        last;
    }
} elsif ($$line_ref =~ /^[\t\\040-\\176]/) {

    # a tab, space, or printable...
    $$piece_ref .= substr($$line_ref, 0 ,1);
    substr($$line_ref, 0, 1) = '';
} else {

    # an undesirable character...
    last;
}
}
return $ok;

```

```
}
```

```
__END__
```

## **=head1 NAME**

Text::CSV - comma-separated values manipulation routines

## **=head1 SYNOPSIS**

```
use Text::CSV;

$version = Text::CSV->version();      # get the module version

$csv = Text::CSV->new();              # create a new object

$status = $csv->combine(@columns);    # combine columns into a string
$line = $csv->string();               # get the combined string

$status = $csv->parse($line);         # parse a CSV string into fields
@columns = $csv->fields();            # get the parsed fields

$status = $csv->status();              # get the most recent status
$bad_argument = $csv->error_input();  # get the most recent bad argument
```

## **=head1 DESCRIPTION**

Text::CSV provides facilities for the composition and decomposition of comma-separated values. An instance of the Text::CSV class can combine fields into a CSV string and parse a CSV string into fields.

## **=head1 FUNCTIONS**

=over 4

=item version

```
$version = Text::CSV->version();
```

This function may be called as a class or an object method. It returns the current module version.

=item new

```
$csv = Text::CSV->new();
```

This function may be called as a class or an object method. It returns a reference to a newly created Text::CSV object.

=item combine

```
$status = $csv->combine(@columns);
```

This object function constructs a CSV string from the arguments, returning success or failure. Failure can result from lack of arguments or an argument containing an invalid character. Upon success, C<string()> can be called to retrieve the resultant CSV string. Upon failure, the value returned by C<string()> is undefined and C<error\_input()> can be called to retrieve an invalid argument.

=item string

```
$line = $csv->string();
```

This object function returns the input to C<parse()> or the resultant CSV string of C<combine()>, whichever was called more recently.

=item parse

```
$status = $csv->parse($line);
```

This object function decomposes a CSV string into fields, returning success or failure. Failure can result from a lack of argument or the given CSV string is improperly formatted. Upon success, C<fields()> can be called to retrieve the decomposed fields. Upon failure, the value returned by C<fields()> is undefined and C<error\_input()> can be called to retrieve the invalid argument.

=item fields

```
@columns = $csv->fields();
```

This object function returns the input to C<combine()> or the resultant decomposed fields of C<parse()>, whichever was called more recently.

=item status

```
$status = $csv->status();
```

This object function returns success (or failure) of C<combine()> or C<parse()>, whichever was called more recently.

=item error\_input

```
$bad_argument = $csv->error_input();
```

This object function returns the erroneous argument (if it exists) of C<combine()> or C<parse()>, whichever was called more recently.

=back

## =head1 EXAMPLE

```
require Text::CSV;

my $csv = Text::CSV->new;

my $column = '';
my $sample_input_string = 'I said, "Hi!""",Yes,"",2.34,, "1.09"';
if ($csv->parse($sample_input_string)) {
    my @field = $csv->fields;
    my $count = 0;
    for $column (@field) {
        print ++$count, " => ", $column, "\n";
    }
    print "\n";
} else {
    my $err = $csv->error_input;
    print "parse() failed on argument: ", $err, "\n";
}
```

```

my @sample_input_fields = ('You said, "Hello!"',
    5.67,
    'Surely',
    '',
    '3.14159');
if ($csv->combine(@sample_input_fields)) {
    my $string = $csv->string;
    print $string, "\n";
} else {
    my $err = $csv->error_input;
    print "combine() failed on argument: ", $err, "\n";
}

```

## **=head1 CAVEATS**

This module is based upon a working definition of CSV format which may not be the most general.

=over 4

=item 1

Allowable characters within a CSV field include 0x09 (tab) and the inclusive range of 0x20 (space) through 0x7E (tilde).

=item 2

A field within CSV may be surrounded by double-quotes.

=item 3

A field within CSV must be surrounded by double-quotes to contain a comma.

=item 4

A field within CSV must be surrounded by double-quotes to contain an embedded double-quote, represented by a pair of consecutive double-quotes.

=item 5

A CSV string may be terminated by 0x0A (line feed) or by 0x0D,0x0A (carriage return, line feed).

## **=head1 AUTHOR**

Alan Citterman F<E<lt>alan@mfgRTL.comE<gt>>

## **=head1 SEE ALSO**

perl(1)

=cut

## ***Homework Assignment***

1. Work on your final project!
2. Download the handout for Session 9. This should be available at <http://www.goss.com/perl> by Wednesday, July 19 at 9AM. Bring a printout to Session 9.<sup>1</sup>

Notes: <sup>1</sup>Not to be handed in.

**This handout is Copyright © 2000 Clint Goss, All Rights Reserved.**