SRI / Mojolicious-9.39 / lib / Mojo / Util.pm

```
package Mojo::Util;
use Mojo::Base -strict;

use Carp           qw(carp croak);
use Data::Dumper   ();
use Digest::MD5    qw(md5 md5_hex);
use Digest::SHA    qw(hmac_sha1_hex sha1 sha1_hex);
use Encode         qw(find_encoding);
use Exporter       qw(import);
use File::Basename qw(dirname);
use Getopt::Long   qw(GetOptionsFromArray);
use IO::Compress::Gzip;
use IO::Poll qw(POLLIN POLLPRI);
use IO::Uncompress::Gunzip;
use List::Util         qw(min);
use MIME::Base64       qw(decode_base64 encode_base64);
use Mojo::BaseUtil     qw(class_to_path monkey_patch);
use Pod::Usage         qw(pod2usage);
use Socket             qw(inet_pton AF_INET6 AF_INET);
use Symbol             qw(delete_package);
use Time::HiRes        ();
use Unicode::Normalize ();

# Encryption support requires CryptX 0.080+
use constant CRYPTX => $ENV{MOJO_NO_CRYPTX} ? 0 : !!eval {
  require CryptX;
  require Crypt::AuthEnc::ChaCha20Poly1305;
  require Crypt::KeyDerivation;
  require Crypt::Misc;
  require Crypt::PRNG;
  CryptX->VERSION('0.080');
  1;
};

# Check for monotonic clock support
use constant MONOTONIC => !!eval { Time::HiRes::clock_gettime(Time::HiRes::CLOCK_MO

# Punycode bootstring parameters
use constant {
  PC_BASE         => 36,
  PC_TMIN         => 1,
  PC_TMAX         => 26,
  PC_SKEW         => 38,
  PC_DAMP         => 700,
  PC_INITIAL_BIAS => 72,
  PC_INITIAL_N    => 128
};

# To generate a new HTML entity table run this command
# perl examples/entities.pl > lib/Mojo/resources/html_entities.txt
my %ENTITIES;
{
  # Don't use Mojo::File here due to circular dependencies
  my $path = File::Spec->catfile(dirname(__FILE__), 'resources', 'html_entities.txt

  open my $file, '<', $path or croak "Unable to open html entities file ($path): $!
  my $lines = do { local $/; <$file> };

  for my $line (split /\n/, $lines) {
    next unless $line =~ /^(\S+)\s+U\+(\S+)(?:\s+U\+(\S+))?/;
    $ENTITIES{$1} = defined $3 ? (chr(hex $2) . chr(hex $3)) : chr(hex $2);
```

```
  }
}

# Characters that should be escaped in XML
my %XML = ('&' => '&amp;', '<' => '&lt;', '>' => '&gt;', '"' => '&quot;', '\'' => '&

# "Sun, 06 Nov 1994 08:49:37 GMT" and "Sunday, 06-Nov-94 08:49:37 GMT"
my $EXPIRES_RE = qr/(\w+\W+\d+\W+\w+\W+\d+\W+\d+:\d+:\d+\W*\w+)/;

# Header key/value pairs
my $QUOTED_VALUE_RE   = qr/\G=\s*("(?:\\\\|\\"|[^"])*")/;
my $UNQUOTED_VALUE_RE = qr/\G=\s*([^;, ]*)/;

# HTML entities
my $ENTITY_RE = qr/&(?:\#((?:[0-9]{1,7}|x[0-9a-fA-F]{1,6}));|(\w+[;=]?))/;

# Encoding, encryption and pattern caches
my (%ENCODING, %ENCRYPTION, %PATTERN);

our @EXPORT_OK = (
  qw(b64_decode b64_encode camelize class_to_file class_to_path decamelize decode de
  qw(encode encrypt_cookie extract_usage generate_secret getopt gunzip gzip header_p
  qw(html_attr_unescape html_unescape humanize_bytes md5_bytes md5_sum monkey_patch
  qw(punycode_encode quote scope_guard secure_compare sha1_bytes sha1_sum slugify sp
  qw(steady_time tablify term_escape trim unindent unquote url_escape url_unescape x
);

# Aliases
monkey_patch(__PACKAGE__, 'b64_decode',    \&decode_base64);
monkey_patch(__PACKAGE__, 'b64_encode',    \&encode_base64);
monkey_patch(__PACKAGE__, 'hmac_sha1_sum', \&hmac_sha1_hex);
monkey_patch(__PACKAGE__, 'md5_bytes',     \&md5);
monkey_patch(__PACKAGE__, 'md5_sum',       \&md5_hex);
monkey_patch(__PACKAGE__, 'sha1_bytes',    \&sha1);
monkey_patch(__PACKAGE__, 'sha1_sum',      \&sha1_hex);

# Use a monotonic clock if possible
monkey_patch(__PACKAGE__, 'steady_time',
  MONOTONIC ? sub () { Time::HiRes::clock_gettime(Time::HiRes::CLOCK_MONOTONIC()) }

sub camelize {
  my $str = shift;
  return $str if $str =~ /^[A-Z]/;

  # CamelCase words
  return join '::', map {
    join('', map { ucfirst lc } split /_/)
  } split /-/, $str;
}

sub class_to_file {
  my $class = shift;
  $class =~ s/::|'//g;
  $class =~ s/([A-Z])([A-Z]*)/$1 . lc $2/ge;
  return decamelize($class);
}

sub decamelize {
  my $str = shift;
  return $str if $str !~ /^[A-Z]/;

  # snake_case words
```

```
124     return join '-', map {
125       join('_', map {lc} grep {length} split /([A-Z]{1}[^A-Z]*)/)
126     } split /::/, $str;
127   }
128
129   sub decrypt_cookie {
130     my ($value, $key, $salt) = @_;
131     croak 'CryptX 0.080+ required for encrypted cookie support' unless CRYPTX;
132
133     return undef unless $value =~ /^([^-]+)-([^-]+)-([^-]+)$/;
134     my ($ct, $iv, $tag) = ($1, $2, $3);
135     ($ct, $iv, $tag) = (Crypt::Misc::decode_b64($ct), Crypt::Misc::decode_b64($iv), C
136
137     my $dk = $ENCRYPTION{$key}{$salt} ||= Crypt::KeyDerivation::pbkdf2($key, $salt);
138     return Crypt::AuthEnc::ChaCha20Poly1305::chacha20poly1305_decrypt_verify($dk, $iv,
139   }
140
141   sub decode {
142     my ($encoding, $bytes) = @_;
143     return undef unless eval { $bytes = _encoding($encoding)->decode("$bytes", 1); 1 }
144     return $bytes;
145   }
146
147   sub deprecated {
148     local $Carp::CarpLevel = 1;
149     $ENV{MOJO_FATAL_DEPRECATIONS} ? croak @_ : carp @_;
150   }
151
152   sub dumper { Data::Dumper->new([@_])->Indent(1)->Sortkeys(1)->Terse(1)->Useqq(1)->Du
153
154   sub encode { _encoding($_[0])->encode("$_[1]", 0) }
155
156   sub encrypt_cookie {
157     my ($value, $key, $salt) = @_;
158     croak 'CryptX 0.080+ required for encrypted cookie support' unless CRYPTX;
159
160     my $dk = $ENCRYPTION{$key}{$salt} ||= Crypt::KeyDerivation::pbkdf2($key, $salt);
161     my $iv = Crypt::PRNG::random_bytes(12);
162     my ($ct, $tag) = Crypt::AuthEnc::ChaCha20Poly1305::chacha20poly1305_encrypt_auther
163
164     return join '-', Crypt::Misc::encode_b64($ct), Crypt::Misc::encode_b64($iv), Cryp
165   }
166
167   sub extract_usage {
168     my $file = @_ ? "$_[0]" : (caller)[1];
169
170     open my $handle, '>', \my $output;
171     pod2usage -exitval => 'noexit', -input => $file, -output => $handle;
172     $output =~ s/^.*\n|\n$//;
173     $output =~ s/\n$//;
174
175     return unindent($output);
176   }
177
178   sub generate_secret {
179     return Crypt::Misc::encode_b64u(Crypt::PRNG::random_bytes(128)) if CRYPTX;
180     srand;
181     return sha1_sum($$ . steady_time() . rand);
182   }
183
184   sub getopt {
185     my ($array, $opts) = map { ref $_[0] eq 'ARRAY' ? shift : $_ } \@ARGV, [];
```

```
  my $save   = Getopt::Long::Configure(qw(default no_auto_abbrev no_ignore_case), @$
  my $result = GetOptionsFromArray $array, @_;
  Getopt::Long::Configure($save);

  return $result;
}

sub gunzip {
  my $compressed = shift;
  IO::Uncompress::Gunzip::gunzip \$compressed, \my $uncompressed
    or croak "Couldn't gunzip: $IO::Uncompress::Gunzip::GzipError";
  return $uncompressed;
}

sub gzip {
  my $uncompressed = shift;
  IO::Compress::Gzip::gzip \$uncompressed, \my $compressed or croak "Couldn't gzip:
  return $compressed;
}

sub header_params {
  my $value = shift;

  my $params = {};
  while ($value =~ /\G[;\s]*([^=;, ]+)\s*/gc) {
    my $name = $1;

    # Quoted value
    if ($value =~ /$QUOTED_VALUE_RE/gco) { $params->{$name} //= unquote($1) }

    # Unquoted value
    elsif ($value =~ /$UNQUOTED_VALUE_RE/gco) { $params->{$name} //= $1 }
  }

  return ($params, substr($value, pos($value) // 0));
}

sub html_attr_unescape { _html(shift, 1) }
sub html_unescape      { _html(shift, 0) }

sub humanize_bytes {
  my $size = shift;

  my $prefix = $size < 0 ? '-' : '';

  return "$prefix${size}B"               if ($size = abs $size) < 1024;
  return $prefix . _round($size) . 'KiB' if ($size /= 1024) < 1024;
  return $prefix . _round($size) . 'MiB' if ($size /= 1024) < 1024;
  return $prefix . _round($size) . 'GiB' if ($size /= 1024) < 1024;
  return $prefix . _round($size /= 1024) . 'TiB';
}

sub network_contains {
  my ($cidr, $addr) = @_;
  return undef unless length $cidr && length $addr;

  # Parse inputs
  my ($net, $mask) = split m!/!, $cidr, 2;
  my $v6 = $net =~ /:/;
  return undef if $v6 xor $addr =~ /:/;

  # Convert addresses to binary
```

```
    return undef unless $net  = inet_pton($v6 ? AF_INET6 : AF_INET, $net);
    return undef unless $addr = inet_pton($v6 ? AF_INET6 : AF_INET, $addr);
    my $length = $v6 ? 128 : 32;

    # Apply mask if given
    $addr &= pack "B$length", '1' x $mask if defined $mask;

    # Compare
    return 0 == unpack "B$length", ($net ^ $addr);
  }

  # Direct translation of RFC 3492
  sub punycode_decode {
    my $input = shift;
    use integer;

    my ($n, $i, $bias, @output) = (PC_INITIAL_N, 0, PC_INITIAL_BIAS);

    # Consume all code points before the last delimiter
    push @output, split(//, $1) if $input =~ s/(.*)\x2d//s;

    while (length $input) {
      my ($oldi, $w) = ($i, 1);

      # Base to infinity in steps of base
      for (my $k = PC_BASE; 1; $k += PC_BASE) {
        my $digit = ord substr $input, 0, 1, '';
        $digit = $digit < 0x40 ? $digit + (26 - 0x30) : ($digit & 0x1f) - 1;
        $i += $digit * $w;
        my $t = $k - $bias;
        $t = $t < PC_TMIN ? PC_TMIN : $t > PC_TMAX ? PC_TMAX : $t;
        last if $digit < $t;
        $w *= PC_BASE - $t;
      }

      $bias = _adapt($i - $oldi, @output + 1, $oldi == 0);
      $n += $i / (@output + 1);
      $i = $i % (@output + 1);
      splice @output, $i++, 0, chr $n;
    }

    return join '', @output;
  }

  # Direct translation of RFC 3492
  sub punycode_encode {
    my $output = shift;
    use integer;

    my ($n, $delta, $bias) = (PC_INITIAL_N, 0, PC_INITIAL_BIAS);

    # Extract basic code points
    my @input = map {ord} split //, $output;
    $output =~ s/[^\x00-\x7f]+//gs;
    my $h = my $basic = length $output;
    $output .= "\x2d" if $basic > 0;

    for my $m (sort grep { $_ >= PC_INITIAL_N } @input) {
      next if $m < $n;
      $delta += ($m - $n) * ($h + 1);
      $n = $m;
```

```
    for my $c (@input) {
      (/)
      if    ($c < $n) { $delta++ }
      elsif ($c == $n) {
        my $q = $delta;

        # Base to infinity in steps of base
        for (my $k = PC_BASE; 1; $k += PC_BASE) {
          my $t = $k - $bias;
          $t = $t < PC_TMIN ? PC_TMIN : $t > PC_TMAX ? PC_TMAX : $t;
          last if $q < $t;
          my $o = $t + (($q - $t) % (PC_BASE - $t));
          $output .= chr $o + ($o < 26 ? 0x61 : 0x30 - 26);
          $q = ($q - $t) / (PC_BASE - $t);
        }

        $output .= chr $q + ($q < 26 ? 0x61 : 0x30 - 26);
        $bias  = _adapt($delta, $h + 1, $h == $basic);
        $delta = 0;
        $h++;
      }
    }

    $delta++;
    $n++;
  }

  return $output;
}

sub quote {
  my $str = shift;
  $str =~ s/(["\\])/\\$1/g;
  return qq{"$str"};
}

sub scope_guard { Mojo::Util::_Guard->new(cb => shift) }

sub secure_compare {
  my ($one, $two) = @_;
  my $r = length $one != length $two;
  $two = $one if $r;
  $r |= ord(substr $one, $_) ^ ord(substr $two, $_) for 0 .. length($one) - 1;
  return $r == 0;
}

sub slugify {
  my ($value, $allow_unicode) = @_;

  if ($allow_unicode) {

    # Force unicode semantics by upgrading string
    utf8::upgrade($value = Unicode::Normalize::NFKC($value));
    $value =~ s/[^\w\s-]+//g;
  }
  else {
    $value = Unicode::Normalize::NFKD($value);
    $value =~ s/[^a-zA-Z0-9_\p{PosixSpace}-]+//g;
  }
  (my $new = lc trim($value)) =~ s/[-\s]+/-/g;

  return $new;
```

```
372  }
373                (/)
374  sub split_cookie_header { _header(shift, 1) }
375  sub split_header        { _header(shift, 0) }
376
377  sub tablify {
378    my $rows = shift;
379
380    my @spec;
381    for my $row (@$rows) {
382      for my $i (0 .. $#$row) {
383        ($row->[$i] //= '') =~ y/\r\n//d;
384        my $len = length $row->[$i];
385        $spec[$i] = $len if $len >= ($spec[$i] // 0);
386      }
387    }
388
389    my @fm = (map({"\%-${_}s"} @spec[0 .. $#spec - 1]), '%s');
390    return join '', map { sprintf join('  ', @fm[0 .. $#$_]) . "\n", @$_ } @$rows;
391  }
392
393  sub term_escape {
394    my $str = shift;
395    $str =~ s/([\x00-\x09\x0b-\x1f\x7f\x80-\x9f])/sprintf '\\x%02x', ord $1/ge;
396    return $str;
397  }
398
399  sub trim {
400    my $str = shift;
401    $str =~ s/^\s+//;
402    $str =~ s/\s+$//;
403    return $str;
404  }
405
406  sub unindent {
407    my $str = shift;
408    my $min = min map { m/^([ \t]*)/; length $1 || () } split /\n/, $str;
409    $str =~ s/^[ \t]{0,$min}//gm if $min;
410    return $str;
411  }
412
413  sub unquote {
414    my $str = shift;
415    return $str unless $str =~ s/^"(.*)"$/$1/g;
416    $str                    =~ s/\\\\/\\/g;
417    $str                    =~ s/\\"/"/g;
418    return $str;
419  }
420
421  sub url_escape {
422    my ($str, $pattern) = @_;
423
424    if ($pattern) {
425      unless (exists $PATTERN{$pattern}) {
426        (my $quoted = $pattern) =~ s!([/\$\[])!\\$1!g;
427        $PATTERN{$pattern} = eval "sub { \$_[0] =~ s/([$quoted])/sprintf '%%%02X', ord
428      }
429      $PATTERN{$pattern}->($str);
430    }
431    else { $str =~ s/([^A-Za-z0-9\-._~])/sprintf '%%%02X', ord $1/ge }
432
433    return $str;
```

```
}

(/)
sub url_unescape {
  my $str = shift;
  $str =~ s/%([0-9a-fA-F]{2})/chr hex $1/ge;
  return $str;
}

sub xml_escape {
  return $_[0] if ref $_[0] && ref $_[0] eq 'Mojo::ByteStream';
  my $str = shift // '';
  $str =~ s/([&<>"'])/$XML{$1}/ge;
  return $str;
}

sub xor_encode {
  my ($input, $key) = @_;

  # Encode with variable key length
  my $len    = length $key;
  my $buffer = my $output = '';
  $output .= $buffer ^ $key while length($buffer = substr($input, 0, $len, '')) == $
  return $output .= $buffer ^ substr($key, 0, length $buffer, '');
}

sub _adapt {
  my ($delta, $numpoints, $firsttime) = @_;
  use integer;

  $delta = $firsttime ? $delta / PC_DAMP : $delta / 2;
  $delta += $delta / $numpoints;
  my $k = 0;
  while ($delta > ((PC_BASE - PC_TMIN) * PC_TMAX) / 2) {
    $delta /= PC_BASE - PC_TMIN;
    $k     += PC_BASE;
  }

  return $k + (((PC_BASE - PC_TMIN + 1) * $delta) / ($delta + PC_SKEW));
}

sub _encoding { $ENCODING{$_[0]} //= find_encoding($_[0]) // croak "Unknown encoding

sub _entity {
  my ($point, $name, $attr) = @_;

  # Code point
  return chr($point !~ /^x/ ? $point : hex $point) unless defined $name;

  # Named character reference
  my $rest = my $last = '';
  while (length $name) {
    return $ENTITIES{$name} . reverse $rest
      if exists $ENTITIES{$name} && (!$attr || $name =~ /;$/ || $last !~ /[A-Za-z0-9
    $rest .= $last = chop $name;
  }
  return '&' . reverse $rest;
}

sub _header {
  my ($str, $cookie) = @_;

  my (@tree, @part);
```

```
496     while ($str =~ /\G[,;\s]*([^=;, ]+)\s*/gc) {
497       push @part, $1, undef;
498       my $expires = $cookie && @part > 2 && lc $1 eq 'expires';
499
500       # Special "expires" value
501       if ($expires && $str =~ /\G=\s*$EXPIRES_RE/gco) { $part[-1] = $1 }
502
503       # Quoted value
504       elsif ($str =~ /$QUOTED_VALUE_RE/gco) { $part[-1] = unquote $1 }
505
506       # Unquoted value
507       elsif ($str =~ /$UNQUOTED_VALUE_RE/gco) { $part[-1] = $1 }
508
509       # Separator
510       next unless $str =~ /\G[;\s]*,\s*/gc;
511       push @tree, [@part];
512       @part = ();
513     }
514
515     # Take care of final part
516     return [@part ? (@tree, \@part) : @tree];
517   }
518
519   sub _html {
520     my ($str, $attr) = @_;
521     $str =~ s/$ENTITY_RE/_entity($1, $2, $attr)/geo;
522     return $str;
523   }
524
525   sub _options {
526
527     # Hash or name (one)
528     return ref $_[0] eq 'HASH' ? (undef, %{shift()}) : @_ if @_ == 1;
529
530     # Name and values (odd)
531     return shift, @_ if @_ % 2;
532
533     # Name and hash or just values (even)
534     return ref $_[1] eq 'HASH' ? (shift, %{shift()}) : (undef, @_);
535   }
536
537   # This may break in the future, but is worth it for performance
538   sub _readable { !!(IO::Poll::_poll(@_[0, 1], my $m = POLLIN | POLLPRI) > 0) }
539
540   sub _round { $_[0] < 10 ? int($_[0] * 10 + 0.5) / 10 : int($_[0] + 0.5) }
541
542   sub _stash {
543     my ($name, $object) = (shift, shift);
544
545     # Hash
546     return $object->{$name} //= {} unless @_;
547
548     # Get
549     return $object->{$name}{$_[0]} unless @_ > 1 || ref $_[0];
550
551     # Set
552     my $values = ref $_[0] ? $_[0] : {@_};
553     @{$object->{$name}}{keys %$values} = values %$values;
554
555     return $object;
556   }
557
```

```
558  sub _teardown {
559    my $class = shift;
560
561    # @ISA has to be cleared first because of circular references
562    no strict 'refs';
563    @{"${class}::ISA"} = ();
564    delete_package $class;
565  }
566
567  package Mojo::Util::_Guard;
568  use Mojo::Base -base;
569
570  sub DESTROY { shift->{cb}() }
571
572  1;
573    Show 492 lines of Pod
```