```
#!/usr/bin/perl
# $Source: $
# $Revision: $
# $Date: $
# $Author: $
##############################################################################

package Eigenstate::CGI::Utils::BitCalc;

use strict;
use warnings;
use CGI qw( :cgi );
use CGI::Carp qw( fatalsToBrowser );
use Text::TagTemplate;
our $VERSION = 0.04;
our( $PARSER, %CACHE );  ## no critic # used to memoize output under mod_perl

our $BITS_IN_A_BYTE = 8; # bits
our $TEMPLATE_DIR = "$ENV{DOCUMENT_ROOT}/bitcalc";
my $EMPTY_STRING = q{};
if ( ! defined $PARSER) {
    $PARSER = Text::TagTemplate->new;
    $PARSER->unknown_action( $EMPTY_STRING );
    $PARSER->template_file( "$TEMPLATE_DIR/index.html" );
    $PARSER->add_tags( +{ RESULTS            => $EMPTY_STRING,
                          AMOUNT             => $EMPTY_STRING,
                          UNITS              => $EMPTY_STRING,
                          NOTATION           => $EMPTY_STRING,
                          NOTATION_MESSAGE   => $EMPTY_STRING,
                        });
}

my $r = CGI->new;
if ( $r->param('source') ) {
    print $r->header( -type=>'text/plain');
    _print_source();
    exit;
}

print $r->header();

my $input_units = ( $r->param('input_units') or 'megabits' );

my $input_amount = $r->param('input_amount') || 1;
$input_amount =~ s/ [^\d.]+ //xmg;  # strip everything but digits and .
$input_amount ||= 1;

my $notation_hash = _get_notation_hash($r);
my $kilo = $notation_hash->{kilo};

my $cache_key = join q{:}, $kilo,$input_amount,$input_units;
if ( $CACHE{$cache_key} ) {
    print $CACHE{$cache_key};
    exit;
}

$PARSER->add_tag( AMOUNT => "$input_amount" );
$PARSER->add_tag( NOTATION_MESSAGE => $notation_hash->{message} );

my $select;
$PARSER->add_tag( SELECT => sub {
    my($attrs) = @_;
    $select = $attrs->{NAME};
    return qq{<select name="$select">};
} );

$PARSER->add_tag( OPTION => sub {
    my($attrs) = @_;
    my $selected;
    my $value = defined $attrs->{VALUE}
                ? $attrs->{VALUE}
                : $EMPTY_STRING;
    my $select = defined $r->param($select)
                 ? $r->param($select)
                 : $EMPTY_STRING;
    if ( $select eq $value ) {
        $selected = 'SELECTED';
    } else {
        $selected = $EMPTY_STRING;
    }
    return qq{<OPTION VALUE="$value" $selected>};
} );

if ( ! $input_units && $input_amount ) {
    print $PARSER->parse_file();
    exit;
}

my %output;
my @units = qw( bits bytes kilobits kilobytes megabits
                megabytes gigabits gigabytes terabytes petabytes
              );

if ( "$input_units" eq 'bits' ) {       ## no critic - Would a hash be better?
    $output{'bits'} = $input_amount;
} elsif ( "$input_units" eq 'bytes' ) {
    $output{'bits'} = $input_amount * $BITS_IN_A_BYTE;
} elsif ( "$input_units" eq 'kilobits' ) {
    $output{'bits'} = $input_amount * $kilo;
} elsif ( "$input_units" eq 'kilobytes' ) {
    $output{'bits'} = $input_amount * $BITS_IN_A_BYTE * $kilo;
} elsif ( "$input_units" eq 'megabits' ) {
    $output{'bits'} = $input_amount * $kilo * $kilo ;
} elsif ( "$input_units" eq 'megabytes' ) {
    $output{'bits'} = $input_amount * $kilo * $kilo * $BITS_IN_A_BYTE;
} elsif ( "$input_units" eq 'gigabits' ) {
    $output{'bits'} = $input_amount * $kilo * $kilo * $kilo;
} elsif ( "$input_units" eq 'gigabytes' ) {
```

```
    $output{'bits'} = $input_amount *$kilo * $kilo * $kilo * $BITS_IN_A_BYTE;
} elsif ( "$input_units" eq 'terabytes' ) {
    $output{'bits'} = $input_amount *$kilo * $kilo * $kilo * $kilo * $BITS_IN_A_BYTE;
} elsif ( "$input_units" eq 'terabits' ) {
    $output{'bits'} = $input_amount * $kilo * $kilo * $kilo * $kilo;
} elsif ( "$input_units" eq 'petabytes' ) {
    $output{'bits'} = $input_amount *$kilo * $kilo * $kilo * $kilo * $kilo * $BITS_IN_A_BYTE;
} elsif ( "$input_units" eq 'petabits' ) {
    $output{'bits'} = $input_amount * $kilo * $kilo * $kilo * $kilo * $kilo;
} else {
    # uh oh
    # shouldn't ever get here
}

$output{'bytes'} = $output{'bits'} / $BITS_IN_A_BYTE;
$output{'kilobits'} = $output{'bits'} / $kilo,
$output{'kilobytes'} = $output{'kilobits'} / $BITS_IN_A_BYTE,
$output{'megabits'} = $output{'kilobits'} / $kilo,
$output{'megabytes'} = $output{'megabits'} / $BITS_IN_A_BYTE,
$output{'gigabits'} = $output{'megabits'} / $kilo,
$output{'gigabytes'} = $output{'gigabits'} / $BITS_IN_A_BYTE,
$output{'terabits'} = $output{'gigabits'} / $kilo,
$output{'terabytes'} = $output{'terabits'} / $BITS_IN_A_BYTE,
$output{'petabits'} = $output{'terabits'} / $kilo,
$output{'petabytes'} = $output{'petabits'} / $BITS_IN_A_BYTE,

$PARSER->list( @units );
$PARSER->entry_file("$TEMPLATE_DIR/results.htmlf");

$PARSER->entry_callback( sub {
    my $unit = shift;
    my $tags = +{
        UNIT   => "$unit",
        AMOUNT => "$output{$unit}"
    };
   return $tags;
});

$PARSER->add_tags( +{ RESULTS =>  $PARSER->parse_list_files,
                      AMOUNT  => "$input_amount",
                      UNITS   => "$input_units",
                    }
                 );
# Memoize it
$cache_key = join q{:}, $kilo,$input_amount,$input_units;
if ( ! defined $CACHE{$cache_key} ) {
    $CACHE{$cache_key} = $PARSER->parse_file;
}

print $CACHE{$cache_key};

exit;

################################################################################
#  Utility subroutines
################################################################################

sub _get_notation_hash {
    my $r = shift || Carp::confess('No CGI object passed.');
    my %notation_info = (
        message    => '(informal notation: kilobyte = 1024 bytes)',
        type       => 'legacy',
        kilo       => 1024,  # legacy
    );
    my $notation = $r->param('notation') || 'legacy';
    my $type = lc $notation;
    return \%notation_info if ( $type eq 'legacy' ); # default
    return \%notation_info if ( ! defined $type );

    if ( $type eq 'ieee' ) {
        $notation_info{message} = '(IEEE notation: kilobyte = 1000 bytes)';
        $notation_info{type} = $type;
        $notation_info{kilo} = 1000;
    }
    return \%notation_info;
}


sub _print_source {
    open my $fh, '<', "$ENV{SCRIPT_FILENAME}";
    while (<$fh>) {
        print;
    }
    return close $fh;
}
```