```
# $Id: CollectorDB.pm,v 1.6 2001/12/21 22:13:01 ned Exp $

=head1 NAME

Wander::CollectorDB - WANDER Data collection sample database

=head1 SYNOPSIS

  use Wander::CollectorDB;

=head1 DESCRIPTION

  CollectorDB::Common These are common to the DB and the Cursor packages.
  CollectorDB
  CollectorDB::Cursor
  CollectorDB::DataCursor

=cut

package Wander::CollectorDB;

require 5.005_61;
use strict;
use warnings;

our $VERSION = '1.0';

# Database structure:
# KEYS: 4 bytes, big-endian, unsigned timestamps as returned from Unix time().
# VALUES: first 2 bytes: big-endian, unsigned channel number.
#         remaining bytes: data (format defined by channel)

#-------------------------------------------------------------------------
----
# These are common to the DB and the Cursor packages.
# This is not to be included by outside applications
#
package CollectorDB::Common;
use BerkeleyDB;
use vars qw($VERSION @EXPORT $Error);

BEGIN
{
    require Exporter;
    @CollectorDB::Common::ISA = 'Exporter';

    $Error = 0;
    *{CollectorDB::Common::Error} = \$BerkeleyDB::Error;

    @CollectorDB::Common::EXPORT =
      qw( _packKey _packSpecialKey
      _packChannelAndData _unpackKey _unpackSpecialKey _unpackChannelAndData
      VERSION_KEY FIRST_TIMESTAMP FIRST_TIMESTAMP_KEY LAST_TIMESTAMP
      LAST_TIMESTAMP_KEY FORMAT_VERSION $Error);
}

# Pack and unpack keys and data into DB
# Note that the default ordering is lexical in BerkeleyDB, so we have
# to use big-endian byte packing.
```

```
sub _packKey { pack( 'N', shift ) }
sub _unpackKey { unpack( 'N', shift ) }

# data starts with channel ID.
sub _packChannelAndData { pack( 'na*', @_ ) }
sub _unpackChannelAndData { unpack( 'na*', shift ) }

# special keys
sub _packSpecialKey { pack( 'n', shift ) }
sub _unpackSpecialKey { unpack( 'n', shift ) }

# Magic key values. 2-byte keys are for internal use.
use constant VERSION_KEY    => _packSpecialKey(0);
use constant FORMAT_VERSION => 1;

# assumes 32-bit return from time()
use constant FIRST_TIMESTAMP     => 1;
use constant FIRST_TIMESTAMP_KEY => _packKey(FIRST_TIMESTAMP);
use constant LAST_TIMESTAMP      => 0xffffffff;
use constant LAST_TIMESTAMP_KEY  => _packKey(LAST_TIMESTAMP);

{
    my $home;

    # Open an environment for common settings
    # (especially read/modify/write cursor behavior with
    # the DB_WRITECURSOR arg to BerkeleyDB::cursor)
    sub _environment
    {
        my $db = shift;
        croak("CollectorDB::Common::setHome not called\n")
          unless defined($home);
        my $env = BerkeleyDB::Env->new(
            -Home  => $home,    # Home is where databases with relative names go
            -Flags => DB_CREATE # create files if they don't exist
            | DB_INIT_CDB       # single-writer, multiple readers
            | DB_INIT_MPOOL,    # shared memory buffer
        ) or die "can't open environment: $Error\n";
        return $env
    }

    sub setHome
    {
        my $class = shift;
        $home = shift;
    }
}

#-------------------------------------------------------------------------
----
# The database itself.
#
package CollectorDB;
@CollectorDB::ISA = 'CollectorDB::Common';

use vars '$globalDestruction';
$CollectorDB::globalDestruction = 0;

=over 4
```

```
=cut

#-------------------------

=item END()

=cut

sub END { $CollectorDB::globalDestruction = 1 }

BEGIN { CollectorDB::Common->import(':DEFAULT'); }
use BerkeleyDB;

#-------------------------

=item error()

Return the $BerkeleyDB::Error variable

=cut

sub error
{
    return $Error;
}

# Verify format version; write it if new database.
# Also write the version at the last timestamp position.
# returns 0 if successful
sub _checkVersion
{
    my $self = shift;

    my $version;
    my $status = $self->{_db}->db_get( VERSION_KEY, $version );
    die "Bad format version $version\n"
      if !$status && $version ne FORMAT_VERSION;
    return 0 if !$status;

    if ( $status == DB_NOTFOUND )
    {
        $status = $self->{_db}->db_put( VERSION_KEY, FORMAT_VERSION )
          and return $status;
        $status = $self->{_db}->db_put( LAST_TIMESTAMP_KEY, FORMAT_VERSION );
    }
    return $status;
}

#-------------------------

=item version()

Return the database version number

=cut

sub version
{
```

```
    my $self = shift;

    my $version;
    my $status = $self->{_db}->db_get( VERSION_KEY, $version );
    return $status ? undef: $version;
}

#---------------------------

=item sync()

Sync the DB to disk

=cut

sub sync
{
    my $self = shift;
    if ( defined($self) && defined( $self->{_db} ) )
    {
        $self->{_db}->db_sync();
    }
}

#---------------------------

=item close()

=cut

sub close
{
    my $self = shift;
    $self->{_db}->db_sync();
    $self->{_db}->db_close();
}

#---------------------------

=item new()

Constructor.

=cut

sub new
{
    my $class         = shift;
    my $fileName      = shift;
    my $directoryName = shift;

    my $self = bless( {}, $class );
    my $db = BerkeleyDB::Btree->new(
        -Filename => $fileName,
        -Env      => $self->_environment(),
        -Flags    => DB_CREATE,
        -Property => DB_DUP,                    # allow duplicate keys
    ) or die "Cannot open file $fileName: $Error\n";
    $self->{_db} = $db;
```

```
    $self->_checkVersion() and die "Can't check version: $Error\n";
    return $self;
}

# Only for my friends!
sub _db
{
    shift ()->{_db};
}

#--------------------------

=item putData()

Data is put into DB verbatim, as a string.
If you want something else, use pack first.

=cut

sub putData
{
    my ( $self, $timeStamp, $channel, $data ) = @_;

    my $packedKey = _packKey($timeStamp);
    my $packedData = _packChannelAndData( $channel, $data );
    my $status;
    my $retries = 0;
    while ( ( $status = $self->{_db}->db_put( $packedKey, $packedData ) ) ==
        DB_LOCK_DEADLOCK )
    {
        print STDERR ( scalar(localtime)
          . ": Put aborted because of deadlock; retrying\n" );
        $retries++;
        last if $retries > 10;
        sleep(1);
    }
    return $status;
}

#--------------------------

=item cursor()

Return a new cursor that can be used to retrieve data

=cut

sub cursor
{
    return CollectorDB::Cursor->new(@_);
}

#--------------------------

=item dataCursor()

Return a new cursor that can be used to retrieve data
in multiple channels
```

```
=cut

sub dataCursor
{
    return CollectorDB::DataCursor->new(@_);
}

#--------------------------

=item DESTROY()

=cut

sub DESTROY
{
    my $self = shift;
    $self->sync() unless $CollectorDB::globalDestruction;
    undef( $self->{_db} );
}

#--------------------------------------------------------------------------
----
# These are returned by CollectorDB::cursor().
# They hold a position in a database.
#
package CollectorDB::Cursor;
@CollectorDB::Cursor::ISA = 'CollectorDB::Common';

BEGIN { CollectorDB::Common->import(':DEFAULT'); }
use BerkeleyDB;

#--------------------------

=item new()

=cut

sub new
{
    my $class = shift;
    my $cdb   = shift;

    my $self = bless( {
          _cdb    => $cdb,
          _db     => $cdb->{_db},
          _cursor => $cdb->{_db}->db_cursor(),
          _key    => undef,                       # current timestamp
      },
      $class
    );
    return $self;
}

#--------------------------

=item currentKey()

=cut
```

```
sub currentKey { shift->{_key} }

#------------------------

=item setBeforeKey()

Set to the first sample whose key is <= the given key.
Returns status and key (via ref)

=cut

sub setBeforeKey
{
    my $self   = shift;
    my $keyRef = shift;      # ref
    my $key    = $$keyRef;
    my $status;

    $status = $self->setToKey( \$key ) and return $status;
    while ( $key > $$keyRef )
    {
        $status = $self->priorKey( \$key ) and return $status;
    }

    $self->{_key} = $$keyRef = $key;
    return $status;
}

# Private -- call c_get
# Does operation on given key, returns new key via ref.
# Throws channel and value away:
#     $cursor->_get( \$key, DB_xxx );
# Or, you can get them:
#     $cursor->_get( \$key, DB_xxx, \$value, \$channel );
sub _get
{
    my ( $self, $keyRef, $op, $valueRef, $channelRef ) = @_;
    my $key    = _packKey($$keyRef);
    my $value  = '';
    my $status = $self->{_cursor}->c_get( $key, $value, $op );
    $self->{_key} = $$keyRef = _unpackKey($key) if !$status;
    ( $$channelRef, $$valueRef ) = _unpackChannelAndData($value)
      if defined($valueRef);
    return $status;
}

#------------------------

=item get()

$status = $cursor->get( \$key, \$value, \$channel );

=cut

sub get
{
    my ( $self, $keyRef, $valueRef, $channelRef ) = @_;
    $self->_get( $keyRef, DB_CURRENT, $valueRef, $channelRef );
}
```

```
#-------------------------

=item setToKey()

Set to the first sample whose key is >= the given key.
Returns status and key (via ref)

=cut

sub setToKey
{
    my $self   = shift;
    my $keyRef = shift;    # ref
    $self->_get( $keyRef, DB_SET_RANGE );
}

#-------------------------

=item firstTimestamp()

Set to the first non-0 timestamp value
Returns status and key (via ref)

=cut

sub firstTimestamp
{
    my $self   = shift;
    my $keyRef = shift;   # ref
                          # $$keyRef = FIRST_TIMESTAMP unless defined($$keyRef);
    Carp::confess "not a ref" unless ref($keyRef);
    $$keyRef = FIRST_TIMESTAMP;
    my $status;
    $status = $self->setToKey($keyRef) and return $status;
    $status = $self->nextKey($keyRef) if $$keyRef == FIRST_TIMESTAMP;
    return $status;
}

#-------------------------

=item lastTimestamp()

=cut

sub lastTimestamp
{
    my $self   = shift;
    my $keyRef = shift;    # ref
      # $$keyRef = LAST_TIMESTAMP unless defined($$keyRef);    # assumes that it
 exists
    Carp::confess "not a ref" unless ref($keyRef);
    $$keyRef = LAST_TIMESTAMP;    # assumes that it exists
    my $status;
    $status = $self->setToKey($keyRef) and return $status;
    $self->priorKey($keyRef);
}

#-------------------------
```

```
=item nextKey()

Returns status, and next key value.

=cut

sub nextKey
{
    my $self   = shift;
    my $keyRef = shift;    # ref
    $self->_get( $keyRef, DB_NEXT_NODUP );
}

#--------------------------

=item priorKey()

Returns status, and prior key value.

=cut

sub priorKey
{
    my $self   = shift;
    my $keyRef = shift;    # ref
    $self->_get( $keyRef, DB_PREV_NODUP );
}

package CollectorDB::DataCursor;
@CollectorDB::DataCursor::ISA = 'CollectorDB::Cursor';
BEGIN { CollectorDB::Common->import(':DEFAULT'); }
use BerkeleyDB;

#--------------------------

=item new()

=cut

sub new
{
    my $class = shift;
    $class->SUPER::new( @_, _channelMap => {} );
}

#--------------------------
#
# =item nextKey()
#
# Returns status, and next key value.
#
# =cut
#
# sub nextKey
# {
#   my $self   = shift;
#   my $keyRef = shift;    # ref
#   $self->_get( $keyRef, DB_NEXT );
```

```
# }
#
# #-------------------------
#
# =item priorKey()
#
# Returns status, and prior key value.
#
# =cut
#
# sub priorKey
# {
#   my $self   = shift;
#   my $keyRef = shift;    # ref
#   $self->_get( $keyRef, DB_PREV );
# }
#
# may have to repeat to get to a "proper" sample.
# Does operation on given key, returns new key via ref.
# Throws channel and value away:
#     $cursor->_get( \$key, DB_xxx );
# Or, you can get them:
#     $cursor->_get( \$key, DB_xxx, \$value, \$channel );
sub _get
{
    my ( $self, $keyRef, $op, $valueRef, $channelRef ) = @_;
    my $key = _packKey($$keyRef);
    my ( $value, $status, $channel, $data ) = ('');

    while ( ( $status = $self->{_cursor}->c_get( $key, $value, $op ) ) == 0 )
    {
        $self->{_key} = $$keyRef = _unpackKey($key);
        ( $channel, $data ) = _unpackChannelAndData($value);

        if ( $op == DB_SET_RANGE )
        {
            last if $$keyRef == LAST_TIMESTAMP;
            $op = DB_NEXT;
        }

        last if exists( $self->{_channelMap}->{$channel} );

        last unless $op == DB_NEXT || $op == DB_PREV;
    }
    ( $$channelRef, $$valueRef ) = ( $channel, $data )
      if !$status && defined($valueRef);
    return $status;
}

#-------------------------

=item channels()

Set channels of interest.

  $cursor->channels(3, 2, 1);

=cut
```

```
sub channels
{
    my $self = shift;

    my $pos = 0;
    my %channelMap = map { $_, $pos++ } @_;
    $self->{_channelMap} = \%channelMap;
}

#--------------------------

=item getValues()

Return the values for the channels of interest
at the current time.

  my $values = [];
  $cursor->channels( 1, 2, 5..12 );
  my $status = $cursor->getValues($values);

=cut

sub getValues
{
    my $self     = shift;
    my $valueRef = shift;    # ARRAY ref

    # make sure that all slots of valueRef are filled
    @$valueRef[ values( %{ $self->{_channelMap} } ) ] = ();

    my ( $key, $value, $channel ) = ( $self->currentKey, '', '' );
    my $status;
    my $flag = DB_SET_RANGE;    # first _get will find first valid value
    until ( $status = $self->_get( \$key, $flag, \$value, \$channel ) )
    {
        $flag = DB_NEXT_DUP;
        my $index = $self->{_channelMap}->{$channel};
        next unless defined($index);    # ignored channel: repeat
        $valueRef->[$index] = $value;
    }
    return ( $status == DB_NOTFOUND ) ? 0 : $status;
}

1;
__END__


=head2 EXPORT

None by default.

=head1 AUTHOR

Ned Konz, ned@bike-nomad.com

=head1 SEE ALSO

perl(1).
```

```
=cut
```

```
# $Id: Channel.pm,v 1.2 2001/12/20 00:49:56 ned Exp $

=head1 NAME

Wander::Channel - Support for WANDER data collection

=head1 SYNOPSIS

  use Wander::Channel;

=head1 DESCRIPTION

This provides the base support for the Wander data collection channels.
The hierarchy looks like this:

  Channel
    DummyChannel
    RegexChannel
    TimedChannel
      PollingSerialChannel (RegexChannel SerialChannel TimedChannel)
    IOChannel
    SerialChannel
    WaitingSerialChannel (RegexChannel SerialChannel Channel)
  ChannelSamplingLoop

=cut

require 5.005_62;
use strict;
use warnings;

our $VERSION = '1.0';

use Carp;
use Wander::ChannelDB;
use Time::HiRes;

use Event;
$Event::DebugLevel = 0;

=over 4

=cut

#------------------------

=item UNIVERSAL::subclassResponsibility()

This is used to stub out methods in abstract classes

=cut

sub UNIVERSAL::subclassResponsibility
{
    local $Carp::CarpLevel = 1;
    my (
        $package,   $filename, $line,       $subroutine, $hasargs,
        $wantarray, $evaltext, $is_require, $hints,      $bitmask
      )
```

```
      = caller(1);
    Carp::confess("my subclass should have defined the method $subroutine\n");
}

#--------------------------

=item main::expandEscapes()

Expand escapes in a string.

Accepts forms:
  \x0d       Hex
  \015       Octal
  \cM        Control-char

Written by japhy.

=cut

sub main::expandEscapes
{
    local $_ = shift;
    s{
    (
      \\
      (?:
        x[A-Fa-f0-9]{2}  # hex escape
        |
        [0-7]{3}         # octal escape
        |
        c.               # ctrl-escape
        |
        .                # any other
      )
    )
  }{qq["$1"]}geexs;
    return $_;
}

=pod

=back

=head2 Channel

Channel is the base of all the sampler channels.

Config keys:
  active    (1/0) (required) Whether this channel is active (sampling)
  name      (string) This channel's name
  debug     (1/0) Whether to print debug data to log
  id        (integer, not changeable) (required) Numeric channel ID
  filename  (string) (optional) Name of Perl file to use to construct this chann
el
  reserved  (number) (optional, default=0) Number of other channels to reserve

Actually, unless a channel is reserved by another it needs a filename.

=over 4
```

```
=cut

package Channel;
my @channels;
my $config;    # persistent hash of config data

#--------------------------

=item Config()

Class method
Return a ref to the config hash, if any.

=cut

sub Config     # Channel::Config
{
    $config;
}

#--------------------------

=item Channels()

Class method.

Returns all the channels that are defined

=cut

sub Channels    # Channel::Channels
{
    return wantarray ? @channels : \@channels;
}

#--------------------------

=item CreateChannels()

Class method.

This will return an array that will have undefined elements
where the definitions are undefined.

=cut

sub CreateChannels    # Channel::CreateChannels
{
    my $package            = shift;
    my @channelDefinitions = @_;

    # Load the file that describes each channel.
    # The file must return the channel itself as the last statement.
    foreach my $definition (@channelDefinitions)
    {
        my $channel = undef;
        if ( defined($definition) )
        {
```

```
            next unless defined( $definition->{filename} );
            $config = $definition;
            eval { $channel = do $definition->{filename}; };
            die ( "Error including " . $definition->{filename} . ": $@\n" )
              if $@;
            if ( UNIVERSAL::isa( $channel, 'Channel' ) )
            {
                $channels[ $channel->id ] = $channel;

                # reserve channels if the channel hasn't already done it
                my $numberReserved = $channel->saved('reserved') || 0;
                $channel->reserveChannels($numberReserved)
                  if ( $numberReserved && !@{ $channel->reservedIDs } );

                # now build reserved channels
                for my $id ( @{ $channel->reservedIDs } )
                {
                    my $c = DummyChannel->new( $channelDefinitions[$id] );
                    push ( @{ $channel->reserved }, $channels[$id] = $c );
                }
            }
            else
            {
                warn( $definition->{filename} . " did not return a Channel\n" );
                $channel = undef;
            }
        }
    }
    undef($config);
    return wantarray ? @channels : \@channels;
}

#--------------------------

=item Channel->new( {key=>value,...} )

Constructor.

=cut

sub new    # Channel::new
{
    my $package = shift;
    my $hash    = shift;    # config data from ChannelDB, persistent tied hash
    my $self    = bless {
        persistentData => $hash,
        at             => 0,
        watcher        => undef,
        repeat         => 1,
        db             => undef,
        reservedIDs    => [],               # array of reserved channel ids
        reserved       => [],               # array of reserved dummy channels
        lastActive     => $hash->{active},
    }, $package;

    return $self;
}

#--------------------------
```

```
=item reserveChannels()

Construct multiple dummy channels

=cut

sub reserveChannels    # Channel::reserveChannels
{
    my $self   = shift;
    my $number = shift;

    $self->reservedIDs( [ $self->id + 1 .. $self->id + $number ] );
}

#-------------------------

=item declareField()

This method allows you to create simple field accessors simply:
$obj->declareField('someFieldName');    # into $obj's class
Class->declareField('someFieldName');   # into Class
declareField('someFieldName');      # uses __PACKAGE__

=cut

sub declareField    # Channel::declareField
{
    my $packageName = shift;
    $packageName = ref($packageName) if ref($packageName);
    my $fieldName = shift;
    if ( !defined($fieldName) )
    {
        $fieldName   = $packageName;
        $packageName = __PACKAGE__;
    }
    my $subText = <<EOF;
sub $packageName\::$fieldName
{
    my \$self = shift;
    \@_ ? \$self->{$fieldName} = shift
    : \$self->{$fieldName}
}
EOF
    eval $subText;
}

# Simple accessors
BEGIN
{

    # This is called after initialize with the time at which to start
    declareField('at');

    # This is called after initialize with the database
    declareField('db');
    declareField('watcher');
    declareField('repeat');
    declareField('reserved');
```

```
    declareField('reservedIDs');
    declareField('lastActive');
}

#-------------------------

=item saved()

Get or set persistent data

=cut

sub saved    # Channel::saved
{
    my $self = shift;
    my $key  = lc(shift);
    @_
      ? $self->{persistentData}->{$key} = shift
      : $self->{persistentData}->{$key};
}

#-------------------------

=item name()

Set or get my name

=cut

sub name    # Channel::name
{
    shift->saved( 'name', @_ ) || '';
}

#-------------------------

=item debug()

Return true if my debug flag is set (or set it)

=cut

sub debug    # Channel::debug
{
    shift->saved( 'debug', @_ ) || 0;
}

#-------------------------

=item id()

=cut

sub id    # Channel::id
{
    my $self = shift;
    my $id;

    if (@_)
```

```
    {
        croak("can't change ID of channel\n");
    }
    else
    {
        $id = $self->saved('id');
        $id = $self->{id} unless defined($id);
    }
    return $id;
}

#--------------------------

=item fileName()

=cut

sub fileName    # Channel::fileName
{
    shift->saved( 'filename', @_ ) || '';
}

#--------------------------

=item active()

Set or get active status.
Return the old active status.
Shut down or start up as needed
This will sense when the channel DB has been changed
upon a query.

=cut

sub active    # Channel::active
{
    my $self            = shift;
    my $currentlyActive = $self->saved('active');
    my $oldState        = @_ ? $currentlyActive : $self->lastActive;
    my $newState        = @_ ? $_[0] : $currentlyActive;
    my $retval          = @_ ? $oldState : $currentlyActive;

    if ( $oldState && !$newState )
    {

        # don't shut down a channel that is needed for other channels
        my $shouldRemainActive = grep { $_->active } @{ $self->reserved };
        $self->shutDown unless $shouldRemainActive;
    }
    elsif ( $newState && !$oldState )
    {
        $self->startUp;
    }

    $self->lastActive($newState);
    $self->saved( 'active', $newState ) if $newState != $oldState;

    return $retval;
}
```

```
#-------------------------

=item Channel::initialize()

Override this as needed.

You can use the config data in Channel::Config()

=cut

sub initialize    # Channel::initialize
{
}

#-------------------------

=item sampleAndStore()

=cut

sub sampleAndStore    # Channel::sampleAndStore
{
    my $self  = shift;
    my $value = $self->sample;
    $self->store($value);
    my $watcher = $self->watcher;
    if ( $watcher->can('interval') )
    {
        my $oldInterval = $watcher->interval;
        my $newInterval = $self->interval;

        if ( $oldInterval != $newInterval )
        {
            $watcher->interval($newInterval);
        }
    }
}

#-------------------------

=item Channel::startUp()

=cut

sub startUp    # Channel::startUp
{
    shift->watcher->start;
}

#-------------------------

=item Channel::shutDown()

=cut

sub shutDown    # Channel::shutDown
{
    shift->watcher->stop;
```

```
}

#-------------------------

=item cbtime()

=cut

sub cbtime    # Channel::cbtime
{
    shift->watcher->cbtime;
}

#-------------------------

=item defaultEventOptions()

This returns default options for the Event constructor.

=cut

sub defaultEventOptions    # Channel::defaultEventOptions
{
    my $self = shift;
    return (
        parked => 1,
        cb     => [ $self, 'sampleAndStore' ],
        desc   => $self->id,
        repeat => $self->repeat || 1,
    );
}

#-------------------------

=item Channel::setUpEvent()

This must be overridden by subclasses

=cut

sub setUpEvent    # Channel::setUpEvent
{
    UNIVERSAL::subclassResponsibility();
}

#-------------------------

=item Channel::sample()

This must be overridden by subclasses

=cut

sub sample    # Channel::sample
{
    UNIVERSAL::subclassResponsibility();
}

#-------------------------
```

```
=item store()

This is a default implementation
that also handles multiple channel values at once

=cut

sub store    # Channel::store
{
    my $self  = shift;
    my $value = shift;

    if ( ref($value) )
    {
        for my $chan ( $self, @{ $self->reserved } )
        {
            my $val = shift (@$value);
            if ( $chan->active )
            {
                printf STDERR ( "%ld %d %s\n", $self->cbtime, $chan->id, $val )
                  if $self->debug;
                $self->db->putData( $self->cbtime, $chan->id, $val )
                  and Carp::croak( "putData error: " . CollectorDB::error );
            }
        }
    }
    else
    {
        printf STDERR ( "%ld %d %s\n", $self->cbtime, $self->id, $value )
          if $self->debug;
        $self->db->putData( $self->cbtime, $self->id, $value )
          and Carp::croak( "putData error: " . CollectorDB::error );
    }

    # $self->db->sync;
}

#----------------------------------------

=pod

=back

=head2 DummyChannel

DummyChannel is used for reserved channels.
It doesn't sample by itself; other channels have to do that.

=over 4

=cut

package DummyChannel;
use base 'Channel';

#------------------------

=item DummyChannel::startUp()
```

```
=cut

sub startUp     # DummyChannel::startUp
{
}

#-------------------------

=item DummyChannel::shutDown()

=cut

sub shutDown     # DummyChannel::shutDown
{
}

#-------------------------

=item DummyChannel::setUpEvent()

=cut

sub setUpEvent     # DummyChannel::setUpEvent
{
}

#-------------------------

=item DummyChannel::initialize()

=cut

sub initialize     # DummyChannel::initialize
{
}

#-----------------------------------------
#

=pod

=back

=head2 TimedChannel

TimedChannel samples every <interval> seconds.
This is a mix-in.
You must override sample()

Config keys:
  interval    (number) (required) Seconds between samples

=over 4

=cut

package TimedChannel;
use base 'Channel';
```

```
#------------------------

=item interval()

=cut

sub interval       # TimedChannel::interval
{
    my $self = shift;
    $self->saved( 'interval', @_ )
      or croak(
        "must define a non-zero interval for channel " . $self->id . "\n" );
}

#------------------------

=item TimedChannel::setUpEvent()

(opt) interval => 10,
(opt) at => $masterTime,

=cut

sub setUpEvent     # TimedChannel::setUpEvent
{
    my $self = shift;
    $self->watcher(
        Event->timer( $self->defaultEventOptions, hard => 1,
            at => $self->at || time,
            interval => $self->interval || 60,    # default=1/min
        )
    );
}

#----------------------------------------

=pod

=back

=head2 RegexChannel

RegexChannel is a mix-in for channels that need to process regexes.
You must provide getData()

Config keys:
  regex     (string) (required) Perl regular expression

You must use parentheses around the part(s) of the regular expression that you
want to save.

=over 4

=cut

package RegexChannel;
use base 'Channel';
```

```
BEGIN
{
    RegexChannel->declareField('leftover');
    RegexChannel->declareField('lastMatch');
    RegexChannel->declareField('fields');
    RegexChannel->declareField('regex');
}

#-------------------------

=item RegexChannel::sample()

By default, all the user fields get stored in subsequent
reserved channels if there are any parentheses in the user RE,
or the entire user RE if there are no fields.
If there are reserved fields, and if enough fields were captured,
then these are stored.
Override this if you want something more sophisticated.

=cut

sub sample    # RegexChannel::sample
{
    my $self   = shift;
    my $fields = $self->fields;
    return $fields if ( @{ $self->fields } == @{ $self->reserved } + 1 );
    return @$fields ? $fields->[0] : $self->lastMatch;
}

#-------------------------

=item RegexChannel::getData()

=cut

sub getData    # RegexChannel::getData
{
    UNIVERSAL::subclassResponsibility();
}

#-------------------------

=item initializeRegexChannel()

Pre-compile the user's RE

=cut

sub initializeRegexChannel    # RegexChannel::initializeRegexChannel
{
    my $self = shift;
    my $re   = $self->saved('regex');
    $self->regex( qr{
            .*          # ignored
            ( $re )     # user regex
            ( .* )      # trailing leftovers
        }xms
    );
}
```

```
# Suitable for use as a callback
sub _filterReceivedData    # RegexChannel::_filterReceivedData
{
    my $self   = shift;
    my $event  = shift;
    my $data   = $self->getData($event);    # TODO do we need the arg?
    my $buffer = $self->leftover . $data;

    # match the last occurrence of RE
    if ( my @fields = ( $buffer =~ $self->regex ) )
    {
        $self->leftover( pop @fields );       # last parens
        $self->lastMatch( shift @fields );    # first parens
        $self->fields( \@fields );
        print STDERR ( $self->id, " got fields:|", join( '|', @fields ), "|\n" )
            if $self->debug;
        $self->sampleAndStore($event);    # call user callback and add to table
    }
    else
    {
        print STDERR ( $self->id, " no matches\n" ) if $self->debug;
        $self->leftover($buffer);
        $self->lastMatch(undef);
        $self->fields( [] );
    }
}

#----------------------------------------

=pod

=back

=head2 IOChannel

IOChannel samples when data is ready.
This is a mix-in.
You must override sample().

=over 4

=cut

package IOChannel;
use base 'Channel';

BEGIN
{
    IOChannel->declareField('fd');
}

#-------------------------

=item IOChannel::setUpEvent()

fd => $fd,
poll => 'rwe'
opt timeout => $seconds,
```

```
opt hard => $bool,
opt timeout_cb => \&code

=cut

sub setUpEvent    # IOChannel::setUpEvent
{
    my $self = shift;
    $self->watcher(
        Event->io(
            $self->defaultEventOptions, fd => $self->fd,
            poll => 're',
        )
    );
}

#-----------------------------------------

=pod

=back

=head2 SerialChannel

SerialChannel is a mix-in for channels that use serial ports.

Config keys:
  port      (string) (default=/dev/ttyS0)
  baudrate  (number) (default=9600)
  parity    (string: none,even,odd) (default=none)
  databits  (number: 5,6,7,8) (default=8)
  stopbits  (number: 1,2) (default=1)
  handshake (string: none,xoff,rts) (default=none)

=over 4

=cut

package SerialChannel;
use base 'Channel';
use Device::SerialPort;

BEGIN
{
    SerialChannel->declareField('port');
    SerialChannel->declareField('fd');
}

#-------------------------

=item initializeSerialChannel()

Saved (persistent) values have highest priority,
then settings hash argument,
then these defaults:
    {
        port      => '/dev/ttyS0',
        baudrate  => 9600,
        parity    => 'none',
```

```
        databits  => 8,
        stopbits  => 1,
        handshake => 'none'
    }

=cut

sub initializeSerialChannel    # SerialChannel::initializeSerialChannel
{
    my $self     = shift;
    my $settings = shift || {};    # hashref
    my $port     =
      Device::SerialPort->new( $self->saved('port')
        || $settings->{port}
        || '/dev/ttyS0' );
    $port->baudrate( $self->saved('baudrate')
        || $settings->{baudrate}
        || 9600 );
    $port->parity( $self->saved('parity') || $settings->{parity} || 'none' );
    $port->parity_enable( $port->parity ne 'none' );
    $port->databits( $self->saved('databits') || $settings->{databits} || 8 );
    $port->stopbits( $self->saved('stopbits') || $settings->{stopbits} || 1 );
    $port->handshake( $self->saved('handshake')
        || $settings->{handshake}
        || 'none' );
    $port->write_settings();
    $self->port($port);
    $self->fd( IO::Handle->new_from_fd( $port->{FD}, "r" ) );
}

#-----------------------------------------

=pod

=back

=head2 WaitingSerialChannel

WaitingSerialChannel
waits for data to be available and processes it
using a regex as soon as it matches the desired pattern.

=over 4

=cut

package WaitingSerialChannel;
use base qw(RegexChannel SerialChannel Channel);

#-------------------------

=item WaitingSerialChannel::getData()

=cut

sub getData    # WaitingSerialChannel::getData
{
    my $self = shift;
    $self->port->input;
```

```
}

#-------------------------

=item WaitingSerialChannel::setUpEvent()

Must be set from initialize:
port => Device::SerialPort->new(...);
regex => "regex"

fd => $fd,
poll => 'rwe'
opt timeout => $seconds,
opt hard => $bool,
opt timeout_cb => \&code

=cut

sub setUpEvent    # WaitingSerialChannel::setUpEvent
{
    my $self = shift;
    $self->leftover('');
    $self->watcher(
        Event->io( $self->defaultEventOptions, fd => $self->fd,
            poll   => 're',
            cb     => [ $self, '_filterReceivedData' ],    # override callback
            repeat => 1,                                        # default
                # timeout => $self->{timeout},
        )
    );
}

#-------------------------

=item WaitingSerialChannel::initialize()

=cut

sub initialize    # WaitingSerialChannel::initialize
{
    my $self = shift;
    $self->initializeRegexChannel;
    $self->initializeSerialChannel;
}

#-----------------------------------------

=pod

=back

=head2 PollingSerialChannel

PollingSerialChannel
uses a timer to check the input port periodically.
The buffered data will be examined

Config keys:
  prompt        (string) (optional) What to send before getting data
```

```
  responsedelay  (number) (optional) Seconds to wait after prompt
  flushdelay     (number) (optional) Seconds before sampling to flush buffers
  multiline      (0/1)    (optional) When true, splits prompt and replies up

If flushdelay is set, another timer will be set to
wake up that many seconds before the sample time
and flush the serial port buffers.

=over 4

=cut

package PollingSerialChannel;
use base qw(RegexChannel SerialChannel TimedChannel);

BEGIN
{
    PollingSerialChannel->declareField('prompt');
    PollingSerialChannel->declareField('prompts');
    PollingSerialChannel->declareField('delim');
    PollingSerialChannel->declareField('responseDelay');
}

#-------------------------

=item preFlushDelay()

Returns the pre-flush delay in seconds, or 0 if unset.
Persistent.

=cut

sub preFlushDelay    # PollingSerialChannel::preFlushDelay
{
    shift->saved('preflush') || 0;
}

#-------------------------

=item preFlush()

=cut

sub preFlush    # PollingSerialChannel::preFlush
{
    my $self = shift;
    $self->port->purge_rx;
}

#-------------------------

=item PollingSerialChannel::startUp()

=cut

sub startUp    # PollingSerialChannel::startUp
{
    my $self = shift;
    $self->SUPER::startUp;
```

```
    $self->{preFlushWatcher}->start if defined( $self->{preFlushWatcher} );
}

#--------------------------

=item PollingSerialChannel::shutDown()

=cut

sub shutDown    # PollingSerialChannel::shutDown
{
    my $self = shift;
    $self->SUPER::shutDown;
    $self->{preFlushWatcher}->stop if defined( $self->{preFlushWatcher} );
}

#--------------------------

=item PollingSerialChannel::setUpEvent()

$port->FILENO is file descriptor; read can call select.

fd => $fd,
poll => 'rwe'
opt timeout => $seconds,
opt hard => $bool,
opt timeout_cb => \&code

=cut

sub setUpEvent    # PollingSerialChannel
{
    my $self = shift;

    $self->TimedChannel::setUpEvent(@_);
    $self->watcher->cb( [ $self, '_filterReceivedData' ] );  # override callback
    if ( $self->preFlushDelay )
    {
        $self->{preFlushWatcher} = Event->timer(
            hard     => 1,
            parked   => 1,
            cb       => [ $self, 'preFlush' ],
            interval => $self->interval,
            at       => $self->at + $self->interval - $self->preFlushDelay,
            repeat   => 1,
        );
    }
}

#--------------------------

=item PollingSerialChannel::getData()

=cut

sub getData    # PollingSerialChannel::getData
{
    my $self = shift;
    my $data = '';
```

```
    my $rin  = '';
    vec( $rin, $self->port->{FD}, 1 ) = 1;
    if ( $self->saved('multiline') )
    {
        foreach my $prompt ( @{ $self->prompts } )
        {
            my $thisLine = '';
            $self->port->write($prompt);    # send prompt
            print STDERR $self->id, " prompt:|$prompt|\n" if $self->debug;
            select( $rin, undef, undef, $self->responseDelay );
            while ( (my $read = $self->port->input) ne '' )
            {
                $thisLine .= $read;
                select( $rin, undef, undef, $self->responseDelay );
            }
            $data .= $thisLine;
            print STDERR $self->id, " got:|$thisLine|\n" if $self->debug;
        }
    }
    else
    {
        my $prompt = $self->prompt;
        $self->port->write($prompt) if ($prompt);
        print STDERR $self->id, " prompt:|$prompt|\n" if $self->debug;
        select( $rin, undef, undef, $self->responseDelay );
        while ( (my $read = $self->port->input) ne '' )
        {
            $data .= $read;
            select( $rin, undef, undef, $self->responseDelay );
        }
        print STDERR $self->id, " got:|$data|\n" if $self->debug;
    }
    $data;
}

#-------------------------

=item PollingSerialChannel::initialize()

=cut

sub initialize    # PollingSerialChannel::initialize
{
    my $self = shift;
    $self->initializeRegexChannel;
    $self->initializeSerialChannel;

    my $prompt = $self->saved('prompt');

    if ($prompt)
    {
        $prompt = main::expandEscapes($prompt);
        $self->prompt($prompt);
    }

    if ( $self->saved('multiline') )
    {
        my $delim = $self->saved('regex');
        $delim =~ s/.*?(\\[rn](?:\\[rn])?)+$/$1/;
```

```
        $delim .= '$';
        $self->delim(qr($delim));
        my @lines = $prompt =~ m/([^\r\n]*[\r\n]+)/g;
        $self->prompts( [@lines] );
    }

    $self->responseDelay( $self->saved('responsedelay') || 0 );
}

#-----------------------------------------

=pod

=back

=head2 ChannelSamplingLoop

ChannelSamplingLoop is the loop that samples channels.

=over 4

=cut

package ChannelSamplingLoop;

#-------------------------

=item new()

=cut

sub new    # ChannelSamplingLoop::new
{
    bless {
        channels => [],
      },
      shift;
}

#-------------------------

=item add()

Add a Sampler

=cut

sub add    # ChannelSamplingLoop::add
{
    my $self    = shift;
    my $channel = shift;
    push ( @{ $self->{channels} }, $channel );
    $channel->setUpEvent if defined($channel);
    $self;
}

#-------------------------

=item ChannelSamplingLoop::startUp()
```

```
=cut

sub startUp    # ChannelSamplingLoop::startUp
{
    my $self = shift;
    my $channel;
    foreach $channel ( @{ $self->{channels} } )
    {
        $channel->startUp
          if defined($channel) && $channel->active;
    }
    Event::loop;
    foreach $channel ( @{ $self->{channels} } )
    {
        $channel->shutDown if defined($channel);
    }
}

#--------------------------

=item ChannelSamplingLoop::shutDown()

=cut

sub shutDown    # ChannelSamplingLoop::shutDown
{
    Event::unloop( $_[1] );
}

1;
__END__




=pod

=back

=head1 EXPORT

None by default.

=head1 AUTHOR

Ned Konz, ned@bike-nomad.com

=head1 SEE ALSO

perl(1).

=cut
```

```
# $Id: Graph.pm,v 1.1 2001/12/13 20:45:22 ned Exp $
=head1 NAME

Wander::Graph - Draw simple graphs from WANDER data

=head1 SYNOPSIS

  use Wander::Graph;

=head1 DESCRIPTION

=cut

package Wander::Graph;

require 5.005_61;
use strict;
use warnings;

use Wander::CollectorDB;
use Wander::ChannelDB;
use GD;
use Chart::Plot;

our $VERSION = '1.0';

my @options = (
    'Black',
    'Red',
    'Blue',
    'Green',
    'Black dashedline',
    'Blue dashedline',
    'Green dashedline',
    'Black nopoints',
    'Red nopoints',
    'Blue nopoints',
    'Green nopoints',
    'Black dashedline nopoints',
    'Blue dashedline nopoints',
    'Green dashedline nopoints',
);

=over 4

=cut

#-------------------------

=item new()

new( $db, $channelID [, ... ])

=cut

sub new
{
    my $class      = shift;
    my $db         = shift;
```

```
    my @channelIDs = @_;
    my $cursor     = $db->dataCursor();
    $cursor->channels(@channelIDs);
    my $self = bless {
        channelIDs     => \@channelIDs,
        cursor         => $cursor,
        firstTimestamp => 0,
        lastTimestamp  => 0xffffffff,
        options => [ @options ],
    }, $class;
    $self;
}

#-------------------------

=item options()

=cut

sub options
{
    my $self = shift;
    $self->{options} = [ @_ ];
}

#-------------------------

=item plot()

Returns a Chart::Plot object

=cut

sub plot
{
    my $self       = shift;
    my $label      = shift;
    my $plotWidth  = shift || 350;
    my $plotHeight = shift || 200;

    my @xValues;
    my @y;
    push ( @y, [] ) foreach ( 1 .. @{ $self->{channelIDs} } );
    $self->data( \@xValues, @y );

    # normalize X values to start at 0.
    my $firstX = $xValues[0];
    my $lastX  = $xValues[-1];
    my $nX     = scalar(@xValues);
    foreach my $i ( 0 .. $nX - 1 )
    {
        $xValues[$i] -= $firstX;
    }

    # construct some X labels with dates.
    my $nLabels         = 4;
    my $secondsPerLabel = ($lastX - $firstX) / ($nLabels - 1);
    my %labels;
```

```
    my $x = 0;
    for my $index (0 .. $#xValues)
    {
        if ($xValues[$index] >= $x)
        {
            my ( $sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst ) =
              localtime( $xValues[$index] + $firstX );
            my $label =
              sprintf( "%2d/%2d %2d:%02d", $mon + 1, $mday, $hour, $min );
            $labels{$xValues[$index]} = $label;
            $x += $secondsPerLabel;
        }
    }

    my $img = Chart::Plot->new( $plotWidth, $plotHeight );

    foreach my $series ( 0 .. $#y )
    {
        my ( @x, @filteredY, $seriesY );
        $seriesY = $y[$series];
        foreach my $i ( 0 .. $#xValues )
        {
            my $y = $seriesY->[$i];
            my $x = $xValues[$i];
            next if !defined($y) || !defined($x);
            push ( @x,         $x );
            push ( @filteredY, $y );
        }
        $img->setData( \@x, \@filteredY, $self->{options}->[$series] );
    }

    $img->setGraphOptions(
        xTickLabels     => \%labels,
        horGraphOffset  => 40,
        vertGraphOffset => 40,
    );

    if ( defined($label) )
    {
        my ( $GDobject, $black, $white, $red, $green, $blue ) =
          $img->getGDobject();
        $GDobject->string( gdSmallFont, ( $plotWidth - length($label) * 8 ) / 2,
            ( $plotHeight - 20 ), $label, $black );
    }
    return $img;
}

#-------------------------

=item data()

Returns an array with my data.
$status = $wanderGraph->data( \@x, \@y );
$status = $wanderGraph->data( \@x, \@y1, \@y2 );

=cut

sub data
{
```

```
    my $self        = shift;
    my $xRef        = shift;
    my @yRefs       = @_;
    my $lastChannel = $#yRefs;
    my $x = $self->{firstTimestamp};
    my $status;
    $status = $self->{cursor}->setToKey( \$x ) and return $status;

    my @values;

    while ( ( $status = $self->{cursor}->getValues( \@values ) ) == 0 )
    {
        push ( @$xRef, $x );
        push ( @{ $yRefs[$_] }, $values[$_] ) for ( 0 .. $lastChannel );
        last
          if ( $status = $self->{cursor}->nextKey( \$x ) )
          or $x >= $self->{lastTimestamp};
        @values = ();
    }
}

#-------------------------

=item firstTimestamp()

=cut

sub firstTimestamp
{
    my $self   = shift;
    my $keyRef = shift;
    if (ref($keyRef))
    {
        my $status = $self->{cursor}->firstTimestamp($keyRef);
        $self->{firstTimestamp} = $$keyRef unless $status;
        return $status;
    }
    else    # scalar: just set
    {
        $self->{firstTimestamp} = $keyRef;
        return 0;
    }
}

#-------------------------

=item lastTimestamp()

if arg is a ref, gets last timestamp.
if a scalar, sets it.

=cut

sub lastTimestamp
{
    my $self   = shift;
    my $keyRef = shift;
    if (ref($keyRef))
    {
```

```
        my $status = $self->{cursor}->lastTimestamp($keyRef);
        $self->{lastTimestamp} = $$keyRef unless $status;
        return $status;
    }
    else
    {
        $self->{lastTimestamp} = $keyRef;
        return 0;
    }
}

1;
__END__
```

***=head2 EXPORT***

***None by default.***

***=head1 AUTHOR***

***Ned Konz, ned@bike-nomad.com***

***=head1 SEE ALSO***

***perl(1).***

***=cut***

```
# $Id: ChannelDB.pm,v 1.1 2001/12/13 20:45:19 ned Exp $

package ChannelDB;

=head1 NAME

Wander::ChannelDB -- Persistent storage for WANDER data collection
channel configuration.

=head1 SYNOPSIS

  use Wander::ChannelDB;

=head1 DESCRIPTION

Wander::ChannelDB provides two packages:

  ChannelDB
  ChannelDB::Channel

=head2 ChannelDB

=cut

require 5.005_61;
use strict;
use warnings;
our $VERSION = '1.0';

use Wander::CollectorDB;
use BerkeleyDB;
use Carp qw(croak carp);

use base 'CollectorDB::Common';
BEGIN { CollectorDB::Common->import(':DEFAULT'); }

my $idKey = 'id';
my %db;    # db handles keyed by fileName

=over 4

=cut

#--------------------------

=item new()

=cut

sub new
{
    my $class    = shift;
    my $fileName = shift;
    my $db       = $db{$fileName};
    if ( !defined($db) )
    {
        $db = BerkeleyDB::Btree->new(
            -Filename => $fileName,
            -Env      => __PACKAGE__->_environment(),
```

```
            -Flags    => DB_CREATE,
        ) or croak "Cannot open file $fileName: $Error\n";
        $db{$fileName} = $db;
    }
    bless { _db => $db, _fileName => $fileName }, $class;
}

#-------------------------

=item DESTROY()

=cut

sub DESTROY
{
    my $self = shift;
    # $self->_db->close();
    # delete $db{ $self->fileName };
}

sub _db { shift->{_db} }

#-------------------------

=item fileName()

=cut

sub fileName { shift->{_fileName} }

sub _cursor
{
    my $self  = shift;
    my $flags = shift || 0;
    $self->{_db}->db_cursor($flags);
}

#-------------------------

=item firstChannel()

Return the ID# of the first channel in this DB
via a ref

=cut

sub firstChannel
{
    my $self  = shift;
    my $idRef = shift;
    my $key   = '';
    my $value;
    my $status = $self->_cursor->c_get( $key, $value, DB_FIRST );
    return $status if $status;
    $$idRef = 0 + substr( $key, 0, &ChannelDB::Channel::CHANNEL_ID_WIDTH );
    return $status;
}

#-------------------------
```

```
=item lastChannel()

Return the ID# of the last channel in this DB
via a ref

=cut

sub lastChannel
{
    my $self  = shift;
    my $idRef = shift;
    my ($key, $value) = ('', '');
    my $status = $self->_cursor->c_get( $key, $value, DB_LAST );
    if ($status == DB_NOTFOUND)
    {
        $$idRef = 0;
        return 0;
    }
    return $status if $status;
    $$idRef = 0 + substr( $key, 0, &ChannelDB::Channel::CHANNEL_ID_WIDTH );
    return $status;
}

#-------------------------

=item allChannelsThat()

Return a list of all the defined channels satisfying a predicate
Remaining data will be passed to predicate.

=cut

sub allChannelsThat
{
    my $self      = shift;
    my $predicate = shift;

    my ( $firstChannel, $lastChannel, @channels );
    my $status = $self->firstChannel( \$firstChannel ) and return;
    $status = $self->lastChannel( \$lastChannel ) and return;
    for ( my $id = $firstChannel ; $id <= $lastChannel ; $id++ )
    {
        my $channel = ChannelDB::Channel->new( $self, $id );
        push ( @channels, $channel ) if $predicate->( $channel, @_ );
    }
    return wantarray ? @channels : \@channels;
}

#-------------------------

=item allChannels()

=cut

sub allChannels
{
    shift->allChannelsThat( sub { exists(shift->{active}) } );
}
```

```
=head2 ChannelDB::Channel

=cut

package ChannelDB::Channel;
use BerkeleyDB;
use Wander::CollectorDB;
# use Storable qw(freeze thaw);
use Carp qw(croak carp);

use base 'CollectorDB::Common';
BEGIN { CollectorDB::Common->import(':DEFAULT'); }

# Keys have 5 characters, right-justified channel#
# followed by string value of keyname.
# i.e.: "    1units"
use constant CHANNEL_ID_WIDTH     => 5;
use constant CHANNEL_KEY_FORMAT    => '%' . CHANNEL_ID_WIDTH . 'd';

#--------------------------

=item new()

=cut

sub new
{
    my ( $class, $db, $id ) = @_;
    my $self = {
        _id     => $id,
        _db     => $db->_db,
        _prefix => sprintf( CHANNEL_KEY_FORMAT, $id ),
    };
    bless $self, $class;
    my $chan = bless {}, $class;
    tie %$chan, __PACKAGE__, $self;
    return $chan;
}

#--------------------------

=item TIEHASH()

=cut

sub TIEHASH { $_[1] }

#--------------------------

=item STORE()

Attempt to not store duplicates here.

=cut

sub STORE
{
    my ( $self, $key, $value ) = @_;
```

```
    die "can't set id of channel " . $self->{_id} . " to $value\n"
      if $key eq $idKey;
    $key   = $self->{_prefix} . $key;
    #   $value = freeze( \$value );
    my $cursor = $self->{_db}->db_cursor(DB_WRITECURSOR);
    die "can't make cursor: $Error\n" unless $cursor;
    my ( $oldKey, $oldValue ) = ( $key, '' );
    my $status = $cursor->c_get( $oldKey, $oldValue, DB_SET );

    if ( not $status )
    {
        $cursor->c_put( $key, $value, DB_CURRENT )
          and die "can't store $value in $key: $Error\n";
    }
    elsif ( $status == DB_NOTFOUND )
    {
        $self->{_db}->db_put( $key, $value )
          and die "can't store $value in $key: $Error\n";
    }
}

#------------------------

=item FETCH()

=cut

sub FETCH
{
    my ( $self, $key ) = @_;
    return $self->{_id} if $key eq $idKey;
    my $value;
    my $status = $self->{_db}->db_get( $self->{_prefix} . $key, $value );
    return undef if $status;
    # $value = thaw($value);
    # return $$value;
    return $value;
}

#------------------------

=item EXISTS()

=cut

sub EXISTS
{
    my ( $self, $key, $value ) = @_;
    $self->{_db}->db_get( $self->{_prefix} . $key, $value ) == 0;
}

#------------------------

=item DELETE()

=cut

sub DELETE
{
```

```
    my ( $self, $key ) = @_;
    $self->{_db}->db_del( $self->{_prefix} . $key );
}

#--------------------------

=item CLEAR()

=cut

sub CLEAR
{
    my $self = shift;
    my ( $key, $value ) = ( $self->{_prefix}, '' );
    my $cursor = $self->{_db}->db_cursor(DB_WRITECURSOR);
    my $flag   = DB_SET_RANGE;
    my $status;

    while ( ( $status = $cursor->c_get( $key, $value, $flag ) ) == 0 )
    {
        last if substr($key, 0, CHANNEL_ID_WIDTH) ne $self->{_prefix};
        $cursor->c_del() and die "Can't delete $key: $Error";
        $flag = DB_NEXT;
    }
    return $status;
}

#--------------------------

=item FIRSTKEY()

=cut

sub FIRSTKEY
{
    return $idKey;
}

#--------------------------

=item NEXTKEY()

=cut

sub NEXTKEY
{
    my ( $self, $lastkey ) = @_;
    $lastkey = '' if $lastkey eq $idKey;
    $lastkey = $self->{_prefix} . $lastkey;
    my $key    = $lastkey;
    my $value  = undef;
    my $cursor = $self->{_db}->db_cursor();
    $cursor->c_get( $key, $value, DB_SET_RANGE ) and return undef;
    return undef unless substr( $key, 0, CHANNEL_ID_WIDTH ) eq $self->{_prefix};

    if ( $key eq $lastkey )
    {
        $cursor->c_get( $key, $value, DB_NEXT ) and return undef;
        return undef unless substr( $key, 0, CHANNEL_ID_WIDTH ) eq $self->{_pref
```

```
ix};
    }

    return substr( $key, CHANNEL_ID_WIDTH );
}

1;
__END__

=head2 EXPORT

None by default.

=head1 AUTHOR

Ned Konz, ned@bike-nomad.com

=head1 SEE ALSO

perl(1).

=cut
```

```
package Wander;

=head1 NAME

Wander - Perl wrapper module for commonly used WANDER functions

=head1 SYNOPSIS

  use Wander;
  my %config = Wander::readConfig();
  my $db = Wander::openDatabase();
  my @channelDefinitions = Wander::channelDefinitions();

=head1 DESCRIPTION

This is used by a number of different modules in the WANDER data acquisition
system.

It reads the config file (typically /etc/wander.cfg) and provides it
and some utility functions:

=cut
require 5.005_61;
use strict;
use warnings;

require Exporter;

our @ISA = qw(Exporter);

our %EXPORT_TAGS = (
    'ALL' => [
        qw(
        readConfig openDatabase dbError channelDefinitions addChannel setChannel
        syncCollector killCollector startCollector
        )
    ]
);

our @EXPORT_OK = ( @{ $EXPORT_TAGS{'ALL'} } );

our @EXPORT = qw(

);
our $VERSION = '1.0';

use Wander::CollectorDB;
use Wander::ChannelDB;
use Carp qw(croak carp);

my $defaultConfigFile = "/etc/wander.cfg";

my %config;

my @requiredKeys = qw( root dbfile channelfile channeldirectory
  dbdirectory filename templatedirectory );

=pod
```

```
=over 4

=cut
#-------------------------

=item readConfig()

Read config file and expand ${...} env var refs
Returns a hash or hash ref.
Can be passed an optional config file, otherwise uses
the default one.

my %config = Wander::readConfig();

=cut
sub readConfig
{
    my $fileName = shift || $defaultConfigFile;
    if ( keys(%config) == 0 || $fileName ne $config{filename} )
    {
        open( CONFIGFILE, $fileName ) or croak "can't open $fileName\: $!\n";
        my $line;

        while ( defined( $line = <CONFIGFILE> ) )
        {
            chomp($line);
            next if $line =~ /^\s*(?:#.*)?$/;
            if ( $line =~ /^\s*(\S+?)\s*=\s*(\S.*?)\s*$/ )
            {
                my $key   = lc($1);
                my $value = $2;
                $value =~ s/\${([^}]+)}/$ENV{$1} || $config{$1} || eval($1)/gex;
                $config{$key} = $value;
            }
            else
            {
                carp("bad format of config line $line\n");
            }
        }
        $config{filename} = $fileName;
        CollectorDB->setHome( $config{dbdirectory} );
    }

    foreach my $requiredKey (@requiredKeys)
    {
        croak "missing config key $requiredKey in $fileName\n"
          unless exists( $config{$requiredKey} );
    }
    return wantarray ? %config : \%config;
}

#-------------------------

=item openDatabase()

Opens the default collector database;
returns the CollectorDB object.

=cut
```

```
sub openDatabase
{
    CollectorDB->new( $config{dbfile} )
      or dbError( '', "Couldn't open data file" );
}

#-------------------------

=item dbError()

Reports a database error; can be passed a message.

=cut
sub dbError
{
    my $msg = shift || "Database error";
    croak("$msg\: $CollectorDB::Error\n");
}

#-------------------------

=item channelDefinitions()

read all channel definitions from database
Channels look like hashes
Return the channels.


=cut
sub channelDefinitions
{
    my $file      = $config{channelfile};
    my $channelDb = ChannelDB->new($file)
      or dbError("Couldn't open channel file $file");
    my @channels = $channelDb->allChannels();
    my @retval;

    foreach my $channel (@channels)
    {
        $retval[ $channel->{id} ] = $channel;
    }
    return wantarray ? @retval : \@retval;
}

#-------------------------

=item addChannel()

Adds a channel to the configured channel DB.
Returns the new channel.
Can also set values if they're passed in as a hash.

=cut
sub addChannel
{
    my $file      = $config{channelfile};
    my $channelDb = ChannelDB->new($file)
      or dbError("Couldn't open channel file $file");
    my $lastChannel;
```

```
    $channelDb->lastChannel( \$lastChannel )
      and dbError("Can't get last channel");
    my $channel = ChannelDB::Channel->new( $channelDb, ++$lastChannel );
    %$channel = (@_);
    $channel;
}

#-------------------------

=item setChannel()

change the settings in a possibly existing channel DB
setChannel($id, %settings)

=cut
sub setChannel
{
    my $id        = +shift;
    my $file      = $config{channelfile};
    my $channelDb = ChannelDB->new($file)
      or dbError("Couldn't open channel file $file");
    my $channel = ChannelDB::Channel->new( $channelDb, $id );
    %$channel = (@_);
    $channel;
}

#-------------------------

=item valueOfChannels()

Return the value(s) of the channels
whose numbers were passed in as arguments

  my @channelValues = Wander::valueOfChannels(1, 2, 3);

=cut
sub valueOfChannels
{
    my $db     = openDatabase();
    my $cursor = $db->dataCursor;
    $cursor->channels(@_);
    my $time;
    $cursor->lastTimestamp( \$time );
    my @retval;
    $cursor->getValues( \@retval );
    return wantarray ? @retval : \@retval;
}

#-------------------------

=item syncCollector()

Syncs the data collector using the /etc/init.d script

=cut
sub syncCollector
{
    readConfig();
    system( "/etc/init.d/" . ( $config{collectorname} || 'wanderCollector' ),
```

```
      'sync' );
}

#--------------------------

=item killCollector()

Kills the data collector using the /etc/init.d script

=cut
sub killCollector
{
    readConfig();
    system( "/etc/init.d/"
      . ( $config{collectorname} || 'wanderCollector' ), 'stop' );
}

#--------------------------

=item startCollector()

Starts the data collector using the /etc/init.d script

=cut
sub startCollector
{
    readConfig();
    system( "/etc/init.d/"
      . ( $config{collectorname} || 'wanderCollector' ), 'start' );
}

1;

__END__

=pod

=back

=head2 EXPORT

None by default.

You can export any of the following:
readConfig openDatabase dbError channelDefinitions addChannel setChannel
or all of them using
use Wander ':ALL';

=head1 AUTHOR

Ned Konz, ned@bike-nomad.com

=head1 SEE ALSO

perl(1).

=cut
```

```
#!/usr/bin/perl -w
# Reads a windows-like config file
# $Id: dumpWanderChannels,v 1.2 2001/12/21 22:13:10 ned Exp $
use strict;
$^W++;
use Wander ':ALL';
use Data::Dumper;

$SIG{INT} = 'IGNORE';
$SIG{TERM} = 'IGNORE';

my %config = readConfig();
print Data::Dumper->Dump( [ \%config ], [ 'config' ] );
my @channelDefinitions = channelDefinitions();
print Data::Dumper->Dump( \@channelDefinitions, [ 'c000' .. sprintf("c%03d", $#ch
annelDefinitions )]);
```

```
#!/usr/bin/perl -w
# Mails from stdin or file arg
# $Id: emailWander,v 1.2 2001/12/21 22:13:10 ned Exp $
use strict;
use Net::SMTP;
use Getopt::Std;
use Wander ':ALL';

my %config = readConfig();

my %opts;
getopt('t:', \%opts);

my $sender    = $config{mailsender}    || 'wander@wander';
my $domain    = $config{maildomain}    || 'qualcomm.com';
my $recipient = $opts{t} || $config{mailrecipient} || 'ned@bike-nomad.com';
my $mailhost  = $config{mailhost}      || 'qualcomm.com';
my $smtp = Net::SMTP->new( $mailhost, Hello => $domain, Timeout => 60 );


$smtp->mail($sender);

$smtp->to(split(',', $recipient));

$smtp->data();
$smtp->datasend("From: $sender\n");
$smtp->datasend("To: $recipient\n");
$smtp->datasend("Subject: Wander Data as of " . scalar( localtime(time) ) . "\n" );

my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);
my @days = qw(Sun Mon Tue Wed Thu Fri Sat);
my @months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);

# Sat, 15 Jan 2000 13:12:51 -0500
# urk: fake the timezone
my $date = sprintf("%s, %02d %s %04d %02d:%02d:%02d -0800",
    $days[$wday], $mday, $months[$mon], $year + 1900,
    $hour, $min, $sec );

$smtp->datasend("Date: $date\n");
$smtp->datasend("\n");

while (<>)
{
    $smtp->datasend($_);
}

$smtp->dataend();
$smtp->quit;
```

```
#!/usr/bin/perl -w
# command line script to export data ranges for mailing, etc.
# $Id: exportWanderData,v 1.4 2001/12/21 22:13:10 ned Exp $
use strict;
use IO::File;
use Getopt::Std;

use Wander ':ALL';
Wander::readConfig();

# Wander::syncCollector();

sub usage
{
    print STDERR <<EOF;
usage: $0 [-h] [-f file] [-t secs] [-c c1,c2,c3...] [-o file]
    -f sets first time (seconds) (default: earliest) from file
       note: this also updates file afterwards
    -t sets first time (seconds) (default: earliest)
       if secs is negative, gives the last secs seconds
    -c sets export channel list (default: all active)
    -o sets output file (default: stdout)
    -h prints this summary
EOF
    exit(shift);
}

# process cmdline args
my %opt;
getopt('hc:f:t:o:', \%opt);

usage(1) if exists($opt{h});

my $lastSent = $opt{t} || 0;
$lastSent += time() if $lastSent < 0;

if (exists($opt{f}))
{
    if (open(LASTSENT, "<$opt{f}"))
    {
        $lastSent = <LASTSENT>;
        chomp($lastSent);
        close(LASTSENT);
    }
    else
    {
        warn "can't open timestamp file $opt{f} for read: $!\n";
    }
}

my @channelDefinitions = channelDefinitions();
my @activeChannels = exists($opt{c})
    ? split(/,/, $opt{c})
    : map { $_->{id} } grep { defined($_) && $_->{active} } @channelDefinitions;

my $outputFilename = $opt{o} || 'standard output';
my $outputFile = $opt{o}
    ? IO::File->new( ">$opt{o}" )
    : IO::Handle->new_from_fd(1, "w");
```

```
die "can't open $outputFilename: $!\n" unless defined($outputFile);

$SIG{INT} = 'IGNORE';
$SIG{TERM} = 'IGNORE';

my $db     = openDatabase();
my $cursor = $db->dataCursor;

$cursor->channels(@activeChannels);
my @names = map { $channelDefinitions[$_]->{name} } @activeChannels;

my $lastTimestamp;
my $x;

$cursor->lastTimestamp( \$lastTimestamp ) and die "Error: ", $db->error, "\n";

if ( $lastSent )
{
    $x = $lastSent;
    $x = CollectorDB::Common::FIRST_TIMESTAMP()
      if ( $x < CollectorDB::Common::FIRST_TIMESTAMP() );
    $cursor->setToKey( \$x ) and die "Error: ", $db->error, "\n";
}
else
{
    $cursor->firstTimestamp( \$x ) and die "Error: ", $db->error, "\n";
}

my @values;
my $quote = '"';
my $sep = "\t";
my $nl = "\r\n";

# quote args
sub quote { map { $quote . (defined($_) ? $_ : '') . $quote } @_ }

sub printRecord { $outputFile->print(join($sep, quote(@_)), $nl) }

sub formatDate
{
    my $time = shift;
    my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime($time);
    return sprintf("%02d/%02d/%04d %02d:%02d:%02d",
        $mon+1, $mday, $year+1900,
        $hour, $min, $sec);
}

printRecord(formatDate($x), "to:", formatDate($lastTimestamp) );

printRecord("Time", @names);

my $lastOutput = $x;
while ( $cursor->getValues( \@values ) == 0 )
{
    die("database corrupt at $x\n") if $x > $cursor->currentKey;
    printRecord(formatDate($x), @values );
    $lastOutput = $x;
    $cursor->nextKey( \$x ) and last;
```

```
    last if $x >= $lastTimestamp;
}

if (exists($opt{f}))
{
    open(LASTSENT, ">$opt{f}") or die "can't open timestamp file $opt{f} for write: $!\n";
    print LASTSENT "$lastOutput\n";
    close(LASTSENT);
}
```

```
#!/usr/bin/perl -w
# This program attempts a database cleanup on the BerkeleyDB files used by WANDE
R
# $Id: wanderCleanup,v 1.1 2001/12/20 21:42:35 ned Exp $
use Wander;
$|++;

my %config;

# returns an array of all processes with files in the dbdirectory open.
sub runningProcesses
{
    open PIDS, "/usr/sbin/lsof -t +D $config{dbdirectory} |";
    my @pids = <PIDS>;
    close PIDS;
    chomp @pids;
    return @pids;
}

# returns the command line of the given process
sub cmdline
{
    my $pid = shift;
    open CMDLINE, "/proc/$pid/cmdline" or warn "can't open $pid cmdline: $!\n";
    my $cmdline = <CMDLINE>;
    $cmdline =~ tr/\0/ /;
    close CMDLINE;
    return $cmdline;
}

# kill one process given its PID. Try a few signals.
sub killPid
{
    my $pid = shift;
    foreach my $sig ('INT', 'TERM', 'KILL')
    {
        print $sig;
        if (kill($sig, $pid))
        {
            sleep(5);
            last unless kill(0, $pid);
            print " ";
        }
        else
        {
            print "Can't kill $pid : $!\n";
            return 0;
        }
    }
    return 1;
}

# Run the BerkeleyDB database tools to clean up
sub cleanup
{
    my $filename = shift;
    if (system("db_verify -h $config{dbdirectory} $filename"))
    {
        print "Recovering database file $filename\n";
```

```
        system("db_dump -h $config{dbdirectory} $filename > $filename\.dump");
        rename($filename, "$filename\.old");
        system("db_load -h $config{dbdirectory} $filename < $filename\.dump");
    }
    else
    {
        print "$filename verifies OK\n"
    }
}

# main program.

%config = Wander::readConfig();
# $config{dbdirectory}
# $config{dbfile}
# $config{channelfile}

die "$config{dbdirectory} isn't writable by you\n" unless -w $config{dbdirectory};

my @killedCmdlines;

foreach my $pid (runningProcesses())
{
    my $cmdline = cmdline($pid);
    print "killing pid $pid [$cmdline] ... ";
    push(@killedCmdlines, $cmdline) if killPid($pid);
    print "\n";
}

# clean up/delete the environment
system("db_recover -h $config{dbdirectory}");

cleanup($config{dbfile});
cleanup($config{channelfile});

# start all these in the background even if they daemonize
foreach my $cmdline (@killedCmdlines)
{
    print "starting $cmdline\n";
    system("$cmdline &");
}
```

```
#!/usr/bin/perl -w

# $Revision: 1.1 $
# This is the heart of the collection loop.
use strict;
use Wander ':ALL';
use Wander::Channel;

my %config = readConfig();

if ( defined( $ARGV[0] ) && $ARGV[0] eq '-d' )    # Fork
{
    my $pidFile = $config{collectorpid} || '/var/run/wander/collector.pid';

    # kill other collection process if any
    if ( open( PIDFILE, $pidFile ) )
    {
        my $pid = <PIDFILE>;
        close PIDFILE;
        chomp($pid);
        kill( 'SIGTERM', $pid ) if $pid;
    }
    my $pid = fork;
    if ($pid)    # parent: save PID
    {
        open PIDFILE, ">$pidFile" or die "can't write $pidFile: $!\n";
        print PIDFILE $pid;
        close PIDFILE;
        exit 0;
    }
    else
    {
        my $logfile = $config{collectorlog} || '/var/log/wander/collect.log';
        umask 0;
        open STDOUT, ">/dev/null";
        open STDERR, ">>$logfile";
        open STDIN,  "</dev/null";
        print STDERR "\n---\nStarted "
          . scalar( localtime( time() ) ) . "\n";
    }
}

my $db                  = openDatabase();
my @channelDefinitions = channelDefinitions();
my @channels           = Channel->CreateChannels(@channelDefinitions);

my $intWatcher = Event->signal(
    cb     => sub { $db->sync; $DB::single = 1 },
    desc   => 'SIGINT signal watcher',
    signal => 'INT'
);
my $quitWatcher = Event->signal(
    cb     => sub { $db->close; exit },
    desc   => 'SIGQUIT signal watcher',
    signal => 'QUIT'
);

my $usr1Watcher = Event->signal(
    cb     => sub { $db->sync },
```

```
    desc   => 'SIGUSR1 signal watcher',
    signal => 'USR1'
);
my $termWatcher = Event->signal(
    cb     => sub { $db->close; exit },
    desc   => 'SIGTERM signal watcher',
    signal => 'TERM'
);

$DB::single = $DB::single;

my $loop       = ChannelSamplingLoop->new();
my $masterTime = time;

for my $id ( 0 .. $#channels )
{
    my $channel = $channels[$id];
    next if !$channel;
    print $channel->id, ' ', $channel->fileName, ' ', $channel, "\n";
    $channel->initialize();
    $channel->at($masterTime);
    $channel->db($db);
    $loop->add($channel);
}

exit( $loop->startUp() );
```

```
#!/usr/bin/perl -w
# $Id: wanderLCD,v 1.2 2001/12/21 22:13:10 ned Exp $
# LCD monitor program for WANDER
use strict;

use Wander;
use Device::SerialPort 0.11;
use Time::HiRes;
use Event;
use Text::Wrap;

# Debugging
my $debug = 0;
# use Data::Dumper;
# $Data::Dumper::Terse = 1;

my %config = Wander::readConfig();

if ( defined( $ARGV[0] ) && $ARGV[0] eq '-d' )    # Fork
{
    my $pidFile = $config{lcdpidfile} || '/var/run/wander/wanderLCD.pid';

    # kill other monitor process if any
    if ( open( PIDFILE, $pidFile ) )
    {
        my $pid = <PIDFILE>;
        close PIDFILE;
        chomp($pid);
        kill( 'SIGTERM', $pid ) if $pid;
    }

    my $pid = fork;
    if ($pid)    # parent: save PID
    {
        open PIDFILE, ">$pidFile" or die "can't write $pidFile: $!\n";
        print PIDFILE $pid;
        close PIDFILE;
        print STDERR "Started monitor process $pid\n";
        exit 0;
    }
    else    # child: close files
    {
        umask 0;
        open STDOUT, ">/dev/null";
        open STDERR, ">/dev/null";
        open STDIN,  "</dev/null";
    }
}

my $serialPortName = $config{lcdport} || '/dev/ttyS3';
my $serialPort;


# LCD Layout (20x4)
# upper left=1,1
# lower right=20,4
#
# ....................
# ....................
```

```
# ....................
# ....................
#
# Keyboard layout (5x4)
#
# E D C B A
# J I H G F
# O N M L K
# T S R Q P
#
# Keypad  Key
# Legend  Code
# A       E
# B       D
# C       C
# D       B
# E       A
# F       J
# G       I
# H       H
# I       G
# J       F
# K       O
# L       N
# M       M
# N       L
# O       K
# Up      T
# Down    S
# Left    R
# Right   Q
# ?       P

# init, deinit, pause, update, update period
my %keyboardInit = (
# code=> [ init,              deinit,  pause, update,     period ], # Legend
    # status screens
    P => [ \&help,               undef, 0, \&seeMore,        7 ],   # ?
    E => [ undef,                undef, 0, \&samplingStatus, 7 ],   # A
    D => [ undef,                undef, 0, \&onlineStatus,   7 ],   # B
    C => [ \&initChargeStatus,   undef, 0, \&seeMore,        7 ],   # C
    B => [ \&initChannelSummary, undef, 0, \&seeMore,        7 ],   # D

    # actions
    J => [ \&initGoOnline,      undef, 1, \&onlineStatus,   20 ],   # F
    I => [ \&initGoOffline,     undef, 1, \&onlineStatus,   20 ],   # G
    H => [ \&initStartSampling, undef, 1, \&samplingStatus, 10 ],   # H
    G => [ \&initStopSampling,  undef, 1, \&samplingStatus, 10 ],   # I
);

# wrap text and save it

my @wrappedLines;
my $wrappedFirstLine;
my $wrappedLinesPerPage;
my $totalPages;
my $currentPage;
my $seeMore;
```

```
sub wrapToScreen
{
    my $text = shift;
    $text =~ s/\n/ /g;
    $wrappedFirstLine    = shift || 0;
    $wrappedLinesPerPage = shift || 4;
    $Text::Wrap::columns = 21;            # TODO: 20?
    $text         = Text::Wrap::wrap( '', '', $text );
    @wrappedLines = split ( "\n", $text );
    $totalPages   = int(( @wrappedLines + $wrappedLinesPerPage - 1 )
      / $wrappedLinesPerPage);
    $currentPage = 0;
    print STDERR "Total wrapped lines: " . scalar(@wrappedLines) . "\n" if $debug;
}

# display given page (0-relative) of the last wrapped text
sub displayPage
{
    my $page  = shift || 0;
    my $firstIndex = $wrappedLinesPerPage * $page ;
    my $lastIndex =  $firstIndex + $wrappedLinesPerPage - 1;
    my $y = $wrappedFirstLine;
    clearScreen();
    foreach my $line (@wrappedLines[$firstIndex .. $lastIndex])
    {
        last unless defined($line);
        writeStringsAt(0, $y++, $line);
    }
}

# If $seeMore is set, call it for refreshing the pages.
sub seeMore
{
    print STDERR ("seeMore $currentPage\n") if $debug;
    # did we just display last page?
    if ($currentPage == 0 && defined($seeMore))
    {
        print STDERR ("calling $seeMore\n") if $debug;
        $seeMore->();
    }
    displayPage($currentPage);
    $currentPage = 0 if ++$currentPage >= $totalPages;
}

# Channel summary screen

# copied from Wander.pm
sub valueOfChannels
{
    my $db     = Wander::openDatabase();
    my $cursor = $db->dataCursor;
    $cursor->channels(@_);
    my $time;
    $cursor->lastTimestamp( \$time );
    my @retval;
    $cursor->getValues( \@retval );
    print STDERR Dumper(\$time, \@retval) if $debug;

    return wantarray ? @retval : \@retval;
```

```
}

my @displayedChannels;

sub initChannelSummary
{
    my @channelDefinitions = Wander::channelDefinitions();
    @displayedChannels =
        map { { name => $_->{name}, id => $_->{id} } }
        grep { defined($_) && $_->{active} } @channelDefinitions;
    $currentPage = 0;
    $seeMore = \&getChannelValues;
}

sub getChannelValues
{
    my @latest = valueOfChannels(map { $_->{id} } @displayedChannels);
    my $text = '';
    foreach my $i (0 .. $#displayedChannels)
    {
        my $dc = $displayedChannels[$i];
        next if $latest[$i] eq "";
        $text .= $dc->{name} . "=" . $latest[$i] . " ";
    }
    wrapToScreen($text);
}

# Charge status screen

my @chargeStatusChannels;
my $chargeStatusText;

sub getChargeStatus
{
    my @latest = valueOfChannels( @chargeStatusChannels );
    my $text = sprintf($chargeStatusText, @latest);
    wrapToScreen($text);
}

sub initChargeStatus
{
    my $cs = $config{chargestatus};
    (my $channels = $cs) =~ s/[^=]+={(\d+)}/$1 /g;
    @chargeStatusChannels = split(' ', $channels);
    print STDERR (Dumper(\@chargeStatusChannels)) if $debug;
    ($chargeStatusText = $cs) =~ s/{\d+}/%s/g;
    print STDERR ("CST=$chargeStatusText\n") if $debug;
    $seeMore = \&getChargeStatus;
    $currentPage = 0;
}

# Help screen
sub help
{
#0000000011111111112
#2345678901234567890
    wrapToScreen <<EOF;
A=Sampling Status
B=Online Status C=Charge Status
```

```
D=Channel Summary
F=Go Online G=Go Offline
H=Start Sampling
I=Stop Sampling
?=Help
EOF
    undef($seeMore);
    displayPage(0);
}

# Sampling on/off

sub initStopSampling
{
    clearScreen();
    if ( collectorPID() )
    {
        writeStringsAt( 0, 0, "Stopping sampling..." );
        Wander::killCollector();
    }
    else
    {
        writeStringsAt( 0, 0, "Not sampling." );
    }
}

sub initStartSampling
{
    clearScreen();
    if ( collectorPID() )
    {
        writeStringsAt( 0, 0, "Already sampling." );
    }
    else
    {
        writeStringsAt( 0, 0, "Starting sampling..." );
        Wander::startCollector();
    }
}

# Main Status screen

# return 0 or PID of collector
sub collectorPID
{
    my $pidFileName = $config{collectorpid} || '/var/run/wanderCollector.pid';
    if (open(PIDFILE, $pidFileName))
    {
        my $pid = <PIDFILE>;
        close PIDFILE;
        chomp($pid);
        return $pid if -e "/proc/$pid";
    }
    return 0;
}

sub samplingStatus
{
    clearScreen();
```

```
    my $pid = collectorPID();
    writeStringsAt( 0, 0, $pid ? "Sampling (PID $pid)" : "Not Sampling" );
}

# Online status screen
# line-by-line log
sub initGoOnline
{
    my @a = onlineData();
    if ( $a[0] )
    {
        writeStringsAt( 0, 0, "Already online." );
    }
    else
    {
        writeStringsAt( 0, 0, "Going online..." );
        system("pon");
    }
}

sub initGoOffline
{
    my @a = onlineData();
    if ( $a[0] )
    {
        writeStringsAt( 0, 0, "Going offline..." );
        system("poff");
    }
    else
    {
        writeStringsAt( 0, 0, "Already offline." );
    }
}

sub onlineStatus
{
    clearScreen();

    my ( $up, $addr, $peer, $rxpackets, $txpackets ) = onlineData();
    writeStringsAt( 0, 0, $up ? 'IP: ' : 'OFFLINE' );
    writeStringsAt( 0,  1, "RX:", $rxpackets );
    writeStringsAt( 10, 1, "TX:", $txpackets );
    if ($up)
    {
        writeStringsAt( 3, 0, $addr );
        writeStringsAt( 0, 2, "GW:", $peer );
    }
}

# ned@wander:/home/wander/wander/LCD$ /sbin/ifconfig ppp0
# ppp0      Link encap:Point-to-Point Protocol
#           inet addr:10.153.238.202  P-t-P:10.153.1.1  Mask:255.255.255.255
#           UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
#           RX packets:9 errors:0 dropped:0 overruns:0 frame:0
#           TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
#           collisions:0 txqueuelen:10
# ned@wander:/home/wander/wander/LCD$ /sbin/ifconfig ppp0
# ppp0      Link encap:Point-to-Point Protocol
#           POINTOPOINT NOARP MULTICAST  MTU:1500  Metric:1
```

```
#           RX packets:11 errors:0 dropped:0 overruns:0 frame:0
#           TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
#           collisions:0 txqueuelen:10

sub onlineData
{
    open( PIPE, "/sbin/ifconfig ppp0 |" );
    my @lines = <PIPE>;
    close PIPE;
    chomp(@lines);
    delete $lines[-1] unless $lines[-1];
    my ( $up, $addr, $peer, $rxpackets, $txpackets ) = ('') x 5;
    if (@lines)
    {

        if ( $up = +( @lines == 6 ) )
        {
            ( $addr, $peer ) = ( $lines[1] =~ /addr:(\S+)\s+P-t-P:(\S+)/ );
        }
        $lines[-3] =~ /packets:(\d+)/;
        $rxpackets = $1;
        $lines[-2] =~ /packets:(\d+)/;
        $txpackets = $1;
    }
    return ( $up, $addr, $peer, $rxpackets, $txpackets );
}

# Sampling screen: channels being sampled.
# For each channel:
# channel ID, name, interval, active/inactive

# Communications screen:
# IP address, DHCP/static, modem connection active
# email queue size

# Alarms screen (?)

# Support routines

sub clearScreen
{
    $serialPort->write("\xfe\x58");    # clear display
}

# position to given position (0-relative) and write joined strings
sub writeStringsAt
{
    my $x = shift;
    my $y = shift;
    $serialPort->write( pack( 'a2c2', "\xfe\x47", $x + 1, $y + 1 ) );
    $serialPort->write("");
    $serialPort->write( substr( join ( '', @_ ), 0, 20 - $y ) );
}

# process the last byte from the serial port
sub decodeKey
{
    my ( $count, $string ) = $serialPort->read(200);
    my $key = substr( $string, -1, 1 );
```

```
    processKey($key);
}

my $timer;

# used to keep track of timer watchers
my ( $onEntry, $onExit, $pause, $doRoutine, $interval );
my $screenCallbacks;
my $lastKey = '';

sub processKey
{
    my $key = shift;
    print STDERR "process $key\n" if $debug;
    return unless exists( $keyboardInit{$key} );

    $serialPort->write("\xfe\x42\x01");    # backlight on 1 min

    if ($key eq $lastKey)
    {
        $timer->now if defined($timer);
        return;
    }

    # stop old one
    if (defined($onExit))
    {
        $onExit->();
        print STDERR "calling onExit\n" if $debug;
    }

    # and cancel timer
    if ( defined($timer) )
    {
        $timer->cancel;
        undef $timer;
        print STDERR "cancelling timer\n" if $debug;
    }

    ( $onEntry, $onExit, $pause, $doRoutine, $interval ) =
      @{ $keyboardInit{$key} };

    clearScreen();

    if (defined($onEntry))
    {
        $onEntry->();
        print STDERR "calling onEntry $onEntry\n" if $debug;
    }

    if ( defined($doRoutine) && $interval )
    {
        print STDERR "polling $doRoutine at $interval\n" if $debug;
        $timer = Event->timer(
            at       => time + $pause,
            interval => $interval,
            cb       => $doRoutine,
            repeat   => 1,
        );
```

```
    }

    # go back to help after 5 mins
    Event->timer(
        at => time + 300,   # in 5 minutes
        repeat => 0,
        cb => sub { processKey('P') }
    );

    $lastKey = $key;
}

# Called when we receive a SIGTERM signal.
sub terminate
{
    my $event = shift;
    $serialPort->write("\xfe\x46\xfe\x54");    # backlight off
    $serialPort->write("stopped");
    print STDERR "stopping because of signal " . $event->w->signal . "\n" if $debug;
    exit;
}

# MAIN PROGRAM

# Init serial port
# print STDERR "open serial port\n";
$serialPort = Device::SerialPort->new($serialPortName)
  or die "can't open $serialPortName : $!\n";

$serialPort->baudrate(19200);

# print STDERR "set up LCD\n";
$serialPort->write("\xfe\x45");             # clear key buffer
$serialPort->write("\xfe\x43\xfe\x51");    # line-wrap on, auto-scroll on
$serialPort->write("\xfe\x46\xfe\x54");    # backlight off, cursor off

Event->signal( cb => \&terminate, signal => 'TERM' );
Event->signal( cb => \&terminate, signal => 'QUIT' );
Event->signal( cb => \&terminate, signal => 'INT' );

Event->io(
    cb   => \&decodeKey,
    fd   => $serialPort->FILENO,
    poll => 'r',
);

processKey('P');    # show help screen first

exit( Event::loop() );
```

```
#!/usr/bin/perl
# Webmin module-level index for WANDER
# $Id: index.cgi,v 1.8 2001/12/21 22:31:30 ned Exp $
require 'wander-lib.pl';

header( 'WANDER Data Collection System', 'logo.png', undef, 1 );

# print '<div align="center"><img src="logo.png" width="245" height="36"></div>'
;

# $tb table header background color
# $cb table body background color
# %config, %gconfig, %in, $module_config_directory
# %wanderConfig @channelDefinitions

print "<hr><h3>Data Access</h3>";

icons_table(
    [
        'display_data.cgi?mode=csv', 'display_data.cgi?mode=graph',
        'display_data.cgi?mode=mail'
    ],
    [
        'Export collected data', 'Graph collected data',
        'Mail collected data'
    ],
    [ 'images/icon.gif', 'images/icon.gif', 'images/icon.gif' ]
);

print "<hr><h3>Data Collection Configuration</h3>";

icons_table(
    [ 'start_stop.cgi', 'edit_channel.cgi', 'edit_configuration.cgi' ],
    [
        'Start, stop or sync collection', 'Add/Delete/Edit Channels',
        'WANDER master configuration'
    ],
    [ 'images/icon.gif', 'images/icon.gif', 'images/icon.gif' ]
);

print "<hr><h3>Documentation</h3>";

icons_table(
    [
        'help/Channel_Field_Editing_Help.html',
        'help/Global_Wander_Configuration_Settings.html'
    ],
    [ 'Channel Field Editing Help', 'Global WANDER Configuration' ],
    [ 'images/icon.gif',                'images/icon.gif' ]
);

print "<hr>";
```

```
#!/usr/bin/perl
# This is a Webmin CGI for graphing or displaying data
# depending on what mode it's used in.
# if the "mode" parameter is
# "graph"   then it graphs the data
# "csv"     then it displays the data as a CSV output
# "mail"    then it mails the CSV data to someone
#
# $Id: display_data.cgi,v 1.3 2001/12/11 03:21:27 ned Exp $
#
require 'wander-lib.pl';

syncCollector();

# $tb table header background color
# $cb table body background color
# %config, %gconfig, %in, $module_config_directory
# %wanderConfig @channelDefinitions

# use Data::Dumper;
# print "<pre>", html_escape(Dumper(\%wanderConfig)), "</pre><br>";

# Emit a (sub)form for the selection of dates
sub timeRangeSubForm
{
    my $db = openDatabase();
    my $cursor = $db->cursor;
    my $lastTime;
    $cursor->lastTimestamp( \$lastTime );
    my $firstTime;
    $cursor->firstTimestamp( \$firstTime );
    my $span = ( $lastTime - $firstTime ) / 3600.0;    # hours
    print "Total data range is ", sprintf( "%0.2f", $span ),
      " hours,<br>from ", scalar( localtime($firstTime) ), ' to ',
      scalar( localtime($lastTime) );
    print <<EOF;
<br>Hours of data displayed:
<input name="span" value="3" size="6" maxlength="50"></input><br>
Hours from last data to end of graph:
<input name="offset" value="0" size="6" maxlength="50"></input><br>
EOF
}

# Emit a table row for one channel given the definition hash
sub channelCheckbox
{
    my $def = shift;
    print <<EOF;
<tr $cb>
    <td align="center">$def->{id}</td>
    <td align="center"><input type="checkbox" name="channel" value="$def->{id}"></input
></td>
    <td align="center">$def->{active}</td>
    <td>$def->{name}</td>
    <td>
EOF
    foreach my $key (qw(interval units))
    {
        print "$key=$def->{$key} " if defined($def->{$key});
```

```
    }
    print "</td>\n</tr>\n";
}

# Emit a form for selection of global options
sub graphOptionsForm
{
    print <<EOF;
X Size <input name="xsize" value="500" size="5"></input>
 Y Size <input name="ysize" value="300" size="5"></input>
   same graph <input type="checkbox" name="samegraph" value="1"></inp
ut>
<br>
EOF
}

# Emit a form for selection of mailto address
sub mailForm
{
    my $toWhom = $wanderConfig{mailrecipient};
    print <<EOF;
Mail to: <input name="mailto" value="$toWhom" size="40"></input>
<br>
EOF
}

sub csvOptionsForm
{
    print <<EOF;
Excel format: <input type="checkbox" name="excel" value="1"></input>
<br>
EOF
}

# Emit a form for selection of which channels are displayed.
sub channelRequestForm
{
    my @enabledChannels = @_;
    print <<EOF;
<table border="1" padding="2" align="center">
<tr $tb>
    <td>ID</td>
    <td>enable</td>
    <td>active</td>
    <td align="center">name</td>
    <td align="center">settings</td>
</tr>
EOF
    foreach my $def( @channelDefinitions[ 1 .. $#channelDefinitions ] )
    {
        channelCheckbox($def)
    }

    print <<EOF;
<tr $cb>
    <td colspan="5" align="center">
        <input type="submit" name="display graph"></input>
    </td>
</tr>
```

```
</table>
EOF
}

# main program.

my $mode = $in{mode} || 'graph';
my @channels = split(/\0/, $in{'channel'});

if ($mode eq 'graph')
{
    header( 'Graph Data', '', 'graph_data', 1 );
    print '<form action="show_graph.cgi" align="center">';
}
elsif ($mode eq 'csv')
{
    header( 'Export Data', '', 'export_data', 1 );
    print '<form action="csv_export.cgi" align="center">';
}
elsif ($mode eq 'mail')
{
    header( 'Mail Data', '', 'mail_data', 1 );
    print '<form action="csv_export.cgi" align="center">';
}

csvOptionsForm() if $mode eq 'csv';
mailForm() if $mode eq 'mail';
graphOptionsForm() if $mode eq 'graph';

timeRangeSubForm();

channelRequestForm(@channels);
print '</form>';

footer();

1;
```

```
#!/usr/bin/perl
# Webmin start/stop data collection tool for WANDER
# $Id: start_stop.cgi,v 1.6 2001/12/21 22:31:30 ned Exp $
#
require 'wander-lib.pl';

header( 'Start, Stop, or Sync Collection', '', undef, 1 );

# $tb table header background color
# $cb table body background color
# %config, %gconfig, %in, $module_config_directory
# %wanderConfig @channelDefinitions

# use Data::Dumper;
# print "<pre>", html_escape(Dumper(\%in)), "</pre><br>";

undef(@channelDefinitions);
my @processes = collectorPID();

if ($in{dbaction})
{
    if ($in{dbaction} eq 'rename database')
    {
        killCollector();
        my $retval = rename($wanderConfig{dbfile}, $in{newname});
        print $retval
            ? "Database file $wanderConfig{dbfile} renamed as $in{newname}"
            : "Can't rename database $wanderConfig{dbfile} as $in{newname} : $!";
    }
    elsif ($in{dbaction} eq 'delete database')
    {
        killCollector();
        my $retval = unlink($wanderConfig{dbfile});
        print $retval
            ? "Database file $wanderConfig{dbfile} deleted"
            : "Can't delete database $wanderConfig{dbfile}\: $!";
    }
    elsif ($in{dbaction} eq 'cleanup database')
    {
        system("/usr/local/wander/bin/wanderCleanup");
    }
}

if ($in{action})
{
    if ( $in{action} eq 'stop it' && @processes )
    {
        killCollector();
        print "<h3>Collection stopped</h3>";
    }
    elsif ( $in{action} eq 'sync' && @processes )
    {
        syncCollector();
        print "<h3>synced to disk</h3>";
    }
    elsif ( $in{action} eq 'start collection' && !@processes )
    {
        startCollector();
        print "<h3>Collection started</h3>";
```

```
    }

    @processes = find_byname( $wanderConfig{collectorname} );
}

if (@processes)
{
    print "<table width=\"100%\" align=\"center\">
    <tr $tb align=\"center\"><td><h3>Collection process with PID("
        . join(", ", @processes) . ") is running.</h3></td></tr>\n";
    print <<EOF;
    <tr $cb align="center">
        <td><form action="start_stop.cgi">
        <input type="submit" name="action" value="stop it"></input>  
        <input type="submit" name="action" value="sync"></input>
        </form></td>
    </tr>
</table>
EOF
}
else
{
    print <<EOF;
<table align="center" width="100%">
    <tr $tb align="center"><td><h3>No collection process is running.</h3></td></t
r>
    <tr $cb align="center">
        <td><form action="start_stop.cgi">
            <input type="submit" name="action" value="start collection"></input>
            </form></td>
    </tr>
</table>
EOF
}

print <<EOF;
<table align="center" width="100%">
    <tr $tb align="center"><td><h3>Database Management (will stop collector)</h3>
</td></tr>
    <tr $cb align="center">
        <td>Rename database file<form action="start_stop.cgi">
            <input type="submit" name="dbaction" value="rename database"></input>
              New Name: <input name="newname" size="30">
            </form></td></tr>
    <tr $cb align="center">
        <td>Delete database (NO UNDELETE POSSIBLE!)<form action="start_stop.cgi">
            <input type="submit" name="dbaction" value="delete database"></input>
            </form></td></tr>
    </tr>
    <tr $cb align="center">
        <td>Clean up database (in emergencies) <form action="start_stop.cgi">
            <input type="submit" name="dbaction" value="cleanup database"></input>
            </form></td></tr>
    </tr>
</table>
EOF
```

```
#!/usr/bin/perl
# This is a Webmin CGI for editing the channel.db used by the Wander data
# collection service.
# $Id: edit_channel.cgi,v 1.6 2001/12/21 22:31:30 ned Exp $
#
require 'wander-lib.pl';

header( 'Add/Delete/Edit Channels', '', 'Channel_Field_Editing_Help', 1 );

# $tb table header background color
# $cb table body background color
# %config, %gconfig, %in, $module_config_directory
# %wanderConfig @channelDefinitions

# use Data::Dumper;
# print "<pre>", html_escape(Dumper(\%wanderConfig)), "</pre><br>";

# return a hash of description->filename
sub getTemplates
{
    my $dir = $wanderConfig{templatedirectory};
    my %retval;
    if (opendir(D, $dir))
    {
        while (my $f = readdir(D))
        {
            my %template;
            next if $f =~ /(^\.+|~)$/;
            next unless -f "$dir/$f";
            read_env_file("$dir/$f", \%template);
            next unless defined($template{description}) && $template{description
};
            $retval{$template{description}} = $f;
        }
    }
    return \%retval;
}

sub addChannelsFromTemplate
{
    my $newName      = shift;
    my $templateFile = shift;
    my $dir          = $wanderConfig{templatedirectory};
    my $cdir         = $wanderConfig{channeldirectory};
    my %template;
    read_env_file( "$dir/$templateFile", \%template );
    $template{filename} = $cdir . '/' . $template{filename};
    my $newChannel = addChannel( %template, name => $newName );
    $channelDefinitions[ $newChannel->{id} ] = $newChannel;

    foreach my $reservedOffset ( 1 .. $newChannel->{reserved} )
    {
        my $reservedChannel = addChannel(
            name   => "$newName reserved $reservedOffset",
            active => 0
        );
        $channelDefinitions[ $reservedChannel->{id} ] = $reservedChannel;
    }
}
```

```
if ( $in{_action} )
{
    my $chan = $in{_channel};
    my $cd   = $channelDefinitions[$chan];
    my $newActive = $in{active} || 0;
    my @keys = grep { $_ ne 'id' } keys(%$cd);

    if ( $in{_action} eq 'change fields' )
    {
        foreach my $key (@keys)
        {
            $cd->{ $key } = $in{ $key };
        }
        $cd->{active} = $newActive;
    }
    elsif ( $in{_action} eq 'delete channel' )
    {
        foreach my $key (@keys)
        {
            delete $cd->{ $key };
        }
        $cd->{deleted} = 1;

        delete $channelDefinitions[ $in{_channel} ];
    }
    elsif ( $in{_action} eq 'add field' && $in{_newname} ne '')
    {
        $cd->{ lc($in{_newname}) } = $in{_newvalue};
    }
    elsif ( $in{_action} eq 'delete field' && $in{_newname} ne '')
    {
        delete $cd->{ lc($in{_newname}) };
    }
    elsif ( $in{_action} eq 'add channel' )
    {
        addChannelsFromTemplate( $in{_newname}, $in{_template} );
    }

    print <<EOF;
<h3><i><b>Note:</i></b> if you added or deleted a channel,
or if you changed any fields other than the <i>active</i>
or <i>interval</i> fields, you will have to restart
data collection for the changes to take effect.</h3>
EOF
}

my $templates = getTemplates();
# print "<pre>", html_escape(Dumper($templates)), "</pre><br>";

print <<EOF;
<table align="center">
<tr $tb><td>Add a new channel</td></tr>
<tr $cb><td><form action="edit_channel.cgi#end">
Kind of channel: <select name="_template">
EOF

foreach my $desc (keys(%$templates))
{
```

```
    my $templateFile = $templates->{$desc};
    print "<option value=\"$templateFile\">$desc</option>\n";
}

print <<EOF;
</select><br>
New channel name: <input name="_newname"></input>
 <input type="submit" name="_action" value="add channel">
</form></td>
</tr>
EOF

foreach my $cd (@channelDefinitions)
{
    next unless defined($cd);
    my $id = $cd->{id} || next;
    print <<EOF;
<tr $tb>
    <td><a name=$id>Channel $id</a></td>
</tr>
<tr $cb>
    <td><form action="edit_channel.cgi#$id">
    <table>
        <input type="hidden" name="_channel" value="$id"></input>
EOF
    while ( my ( $key, $value ) = each(%$cd) )
    {
        next if $key eq 'id';
        if ( $key eq 'active' )
        {
            my $val = ( $value ? 'CHECKED' : '' );
            print <<EOF;
<tr>
    <td>$key</td>
    <td><input type="checkbox" value="1" name="$key" $val></td>
</tr>
EOF
        }
        else
        {
            print <<EOF;
<tr>
    <td>$key</td>
    <td><input name="$key" size=50 value="$cd->{$key}"></td>
</tr>
EOF
        }
    }
    print <<EOF;
<tr>
    <td></td>
    <td align="justify">
        <input type="submit" name="_action" value="change fields"></input>
        <input type="submit" name="_action" value="delete channel"></input>
    </td>
</tr>
<tr>
    <td></td>
    <td>
```

```
        <input type="submit" name="_action" value="add field"></input>
        <input type="submit" name="_action" value="delete field"></input>
        Field Name: <input name="_newname"></input><br>
        Field value: <input name="_newvalue"></input>
    </td>
</tr>
</table></form></td></tr>
EOF
}
print "</table><a name=\"end\"> </a>\n";

footer();
```

```
#!/usr/bin/perl
# outputs a graph of one or more channels
# graph.cgi?channel=1&channel=2&channel=3
# $Id: graph.cgi,v 1.4 2001/12/21 22:31:30 ned Exp $
#
BEGIN { require 'wander-lib.pl' }
use Wander::Graph;

my $db = openDatabase();

$SIG{TERM} = 'IGNORE';
$SIG{INT} = 'IGNORE';

my @activeChannels = split ( /\0/, $in{'channel'} );
my $xSize = $in{'xsize'} || 400;
my $ySize = $in{'ysize'} || 300;
my $label = $in{'label'};
my $span   = ( $in{'span'}   || 0 ) * 3600.0;
my $offset = ( $in{'offset'} || 0 ) * 3600.0;

if (@activeChannels)    # display graph
{
    my $graph = Wander::Graph->new( $db, @activeChannels );
    my $firstTimestamp;
    $graph->firstTimestamp( \$firstTimestamp );
    my $lastTimestamp;
    $graph->lastTimestamp( \$lastTimestamp );
    if ($offset)
    {
        $lastTimestamp -= $offset;
        $lastTimestamp = CollectorDB::Common::LAST_TIMESTAMP()
          if ( $lastTimestamp > CollectorDB::Common::LAST_TIMESTAMP() );
    }
    if ($span)
    {
        $firstTimestamp = $lastTimestamp - $span;
        $firstTimestamp = CollectorDB::Common::FIRST_TIMESTAMP()
          if ( $firstTimestamp < CollectorDB::Common::FIRST_TIMESTAMP() );
    }
    $graph->firstTimestamp($firstTimestamp);
    $graph->lastTimestamp($lastTimestamp);
    my $img = $graph->plot( $label, $xSize, $ySize );
    my $png = $img->draw;
    $|++;
    unless ( $in{'noheader'} )
    {
        print "Content-Type: image/png\r\n", "Content-Length: ", length($png),
          "\r\n", "Pragma: no-cache\r\n", "Expires: 0\r\n", "\r\n";
    }
    print STDOUT $png;
}

1;
```

```
#!/usr/bin/perl
# This is a Webmin CGI for graphing data
# $Id: show_graph.cgi,v 1.1 2001/12/10 07:14:21 ned Exp $
#
require 'wander-lib.pl';
use URI;

# $tb table header background color
# $cb table body background color
# %config, %gconfig, %in, $module_config_directory
# %wanderConfig @channelDefinitions

# Display one or more graphs in a table
sub showGraphs
{
    my @channels = @_;
    my $target   = URI->new( 'graph.cgi', 'http' );
    my $xsize    = $in{'xsize'};
    my $ysize    = $in{'ysize'};
    my $span     = $in{'span'};
    my $offset   = $in{'offset'};

    if ( $in{'samegraph'} )
    {
        my $label =
          join ( ',', map { $_->{name} } @channelDefinitions[@channels] );
        $target->query_form(
            ( map { ( 'channel', $_ ) } @channels ),
            label => $label,
            xsize  => $xsize,
            ysize  => $ysize,
            span   => $span,
            offset => $offset,
        );
        print "<div align=\"center\"><img width=\"$xsize\" height=\"$ysize\" src=\"",
          $target->path_query, "\"></div>\n";
    }
    else
    {
        print "\n<table align=\"center\" border=\"0\">";
        map {
            $target->query_form(
                channel => $_,
                label   => $channelDefinitions[$_]->{name},
                xsize   => $xsize,
                ysize   => $ysize,
                span    => $span,
                offset  => $offset,
            );
            print '<tr><td><img width=', $xsize, ' height=', $ysize, ' src=',
              $target->path_query, "></td></tr>\n";
        } @channels;
        print "</table>\n";
    }
}

# main program.

my @channels = split(/\0/, $in{'channel'});
```

```
header( 'Collected Data', '', undef, 1 );
showGraphs( @channels ) if @channels;
footer();
1;
```

```
#!/usr/bin/perl
# outputs a CSV export of one or more channels
# csv_export.cgi?channel=1&channel=2&channel=3
# span      span in hours (default 0 = whole data set)
# offset    offset from end (default 0)
# mailto    send to mail recipient instead of stdout
# download  use content-type application/octet-stream
# excel     try to force M$ Excel to open it
#
# $Id: csv_export.cgi,v 1.6 2001/12/21 22:31:30 ned Exp $
#
require 'wander-lib.pl';
use IO::Handle;
use IO::File;

my @activeChannels = sort { $a > $b } ( split ( /\0/, $in{channel} ) );

unless (@activeChannels)
{
    error("no channels requested");
    exit;
}

my $tempname;
my $tempfile;
my $oldStdout;

if ( $in{mailto} )
{
    header( "Mailing data", '', undef, 1 );
    print("<hr><h3>Mailed data to $in{mailto}</h3>");

    $tempname  = tempname();
    $tempfile  = IO::File->new( $tempname, "w+" );
    $oldStdout = select($tempfile);
}
elsif ( $in{download} )
{
    print "Content-Type: application/octet-stream\r\n";
    print "Content-disposition: attachment; filename=\"wanderData.csv\"\r\n";
    print "\r\n";
}
elsif ( $in{excel} )
{
    my $filename = "wanderData" . time() . ".xls";
    print "Content-Type: application/vnd.ms-excel\r\n";
    print "Content-disposition: filename=\"$filename\"\r\n";
    print "\r\n";
}
else
{
    print "Content-Type: text/plain\r\n\r\n";
}

my $span   = ( $in{span}   || 0 ) * 3600.0;
my $offset = ( $in{offset} || 0 ) * 3600.0;


my $db     = openDatabase();
```

```
my $cursor = $db->dataCursor;
$cursor->channels(@activeChannels);
my @names = map { $channelDefinitions[$_]->{name} } @activeChannels;

sub terminate
{
    undef($db);
    undef($cursor);
    exit;
}

$SIG{TERM} = \&terminate;
$SIG{INT} = \&terminate;

my $lastTimestamp;
my $x;

$cursor->lastTimestamp( \$lastTimestamp ) and die "Error: ", $db->error, "\n";

if ($offset)
{
    $lastTimestamp -= $offset;
    $lastTimestamp = CollectorDB::Common::LAST_TIMESTAMP()
      if ( $lastTimestamp > CollectorDB::Common::LAST_TIMESTAMP() );
}

if ($span)
{
    $x = $lastTimestamp - $span;
    $x = CollectorDB::Common::FIRST_TIMESTAMP()
      if ( $x < CollectorDB::Common::FIRST_TIMESTAMP() );
    $cursor->setToKey( \$x );
}
else
{
    $cursor->firstTimestamp( \$x ) and die "Error: ", $db->error, "\n";
}

my @values;
my $quote = '"';
my $sep   = ',';
my $nl    = "\n";

if ( $in{excel} )
{
    $sep = "\t";
    $nl  = "\r\n";
}
if ( $in{mailto} )
{
    $nl = "\r\n";
}

# quote args
sub quote
{
    map { $quote . $_ . $quote } @_;
}
```

```
sub printRecord
{
    print( join ( $sep, quote(@_) ), $nl );
}

sub formatDate
{
    my $time = shift;
    return scalar( localtime($time) ) unless $in{excel};
    my ( $sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst ) =
      localtime($time);
    return
      sprintf( "%02d/%02d/%04d %02d:%02d:%02d", $mon + 1, $mday, $year + 1900,
        $hour, $min, $sec );
}

printRecord( "From: "
    . scalar( localtime($x) ) . " to: "
    . scalar( localtime($lastTimestamp) ) );

printRecord( "Time", @names );

while ( $cursor->getValues( \@values ) == 0 )
{
    die("database corrupt\n") if $x > $cursor->currentKey;
    printRecord( formatDate($x), @values );
    $cursor->nextKey( \$x ) and last;
    last if $x >= $lastTimestamp;
}

if ( $in{mailto} )
{
    select($oldStdout);
    $tempfile->close;
    my $mailer = $wanderConfig{mailer} || die "no mailer in config";
    system("$mailer $in{mailto} $tempname")
      and error("can't execute $mailer $in{mailto} $tempname\: $!");
    unlink($tempname);
    footer();
}

1;
```

```
#!/usr/bin/perl
# This is a Webmin CGI for configuring the Wander config file
#
# $Id: edit_configuration.cgi,v 1.3 2001/12/21 22:31:30 ned Exp $
#
require 'wander-lib.pl';

# $tb table header background color
# $cb table body background color
# %config, %gconfig, %in, $module_config_directory
# %wanderConfig @channelDefinitions

my $configFile = $wanderConfig{filename} || '/etc/wander.cfg';

header("Edit WANDER configuration", '', 'Global_Wander_Configuration_Settings', 1);

if ($in{text})
{
    my $text = $in{text} . "\n";
    $text =~ tr/\r\n/\n/s;
    open(CONFIGFILE, ">$configFile") or error("can't open $configFile: $!");
    print CONFIGFILE $text;
    close(CONFIGFILE);

    print <<EOF;
<h3>New configuration saved. Remember you may have to restart collection and/or
other WANDER tasks if you have changed directories, etc.</h3>
<pre>
$in{text}
</pre>
EOF
}
else
{
    open(CONFIGFILE, "$configFile") or error("can't open $configFile: $!");
    my $fileContents;
    {
        local $/ = undef;
        $fileContents = <CONFIGFILE>;
    }
    close(CONFIGFILE);

    print <<EOF;
<form action="edit_configuration.cgi" method="POST" align="center">
<textarea name="text" rows="30" cols="60">$fileContents</textarea><br>
<input type="submit"></input>
</form>
EOF
}
```

```
// PIC-based Solar charger controller (CCS C)
// Copyright (C) 2002 by Ned Konz
// ned@bike-nomad.com
// 27305 83rd Avenue NW, Stanwood WA 98292 USA
//
//  This program is free software; you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation; either version 2 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 General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program; if not, write to the Free Software
//  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
//
// $Id: batmon.c,v 1.20 2001/12/16 21:22:41 ned Exp $
// Yes, this uses floating point. It turned out to be faster to do a floating
// point multiply than a 16x16->32 multiply and a 32/16 divide.
// Note that ints are unsigned in CCS C
//

#include <16f876.h>

#define SCOPE_DEBUG 0

// defines for debugging:
#undef PRINT_ID

// Note: RA4 (open drain) is set up to drive an LED and so requires
// a pullup.

#define CLOCK_FREQ      4000000 /* Crystal/ceramic resonator frequency */
#define ATOD_RESOLUTION 10      /* right-just 10 bits */
#define RS232_BAUD      9600    /* any faster and WANDER loses it */

#pragma device  ADC=ATOD_RESOLUTION *=8 ICD=TRUE
#pragma fuses   XT,WDT,NOPROTECT,NOLVP,PUT,BROWNOUT

#pragma nolist
#include <stdlib.h>
#pragma list

#pragma use fast_io(A)
#pragma use fast_io(B)
#pragma use fast_io(C)

// Digital input bits
#define CTS_PIN             PIN_B1
#define RXD_PIN             PIN_C7

// Jumper in=WANDER mode (ext pwr sw OFF at system power up)
// Jumper out=external computer mode (ext pwr sw ON at system power up)
#define EXT_MODE_JUMPER_PIN PIN_B4      /* must have cut/jumper */

// LOW=system ON
```

```
#define POWER_SWITCH_PIN    PIN_B0      /* must have added resistor and jumper *
/

#define WANDER_MODE         (!INPUT(EXT_MODE_JUMPER_PIN))
#define POWER_SWITCH_IS_ON  (!INPUT(POWER_SWITCH_PIN))

// Digital output bits
#define LED_PIN             PIN_A4
#define DTR_PIN             PIN_B2
#define LOAD_PIN            PIN_C1      /* CCP2 (PWM) usually (after cut/jumper)
 */
#define SYSTEM_POWER_PIN    PIN_B5
#define BATT_LOW_PIN        PIN_C0
#define SOLAR_PIN           PIN_C2      /* CCP1 (PWM) usually */
#define TXD_PIN             PIN_C6

#define TURN_SYSTEM_OFF     OUTPUT_HIGH(SYSTEM_POWER_PIN)
#define TURN_SYSTEM_ON      OUTPUT_LOW(SYSTEM_POWER_PIN)

#define LED_ON              OUTPUT_LOW(LED_PIN);
#define LED_OFF             OUTPUT_HIGH(LED_PIN);

// Direction registers, 1=input
#define DIRECTION_A         0b11101111
#define DIRECTION_B         0b00010011
#define DIRECTION_C         0b10000000

#pragma use delay(clock=CLOCK_FREQ)

#pragma use rs232(baud=RS232_BAUD, xmit=TXD_PIN, rcv=RXD_PIN)

// If SCOPE_DEBUG is defined, these macros produce digital waveforms
// on the given pins. Otherwise they do nothing.
#if defined SCOPE_DEBUG
#   define  DEBUG_PINPULSE(pin, usec)   OUTPUT_BIT(pin, 1); DELAY_US(usec); OUTP
UT_BIT(pin, 0);
#   define  DEBUG_PINON(pin)            OUTPUT_BIT(pin, 1);
#   define  DEBUG_PINOFF(pin)           OUTPUT_BIT(pin, 0);
#else
#   define  DEBUG_PINPULSE(pin, usec)
#   define  DEBUG_PINON(pin)
#   define  DEBUG_PINOFF(pin)
#endif

#if defined(PRINT_ID)
char const idString[] = "$Revision: 1.20 $";
char const dateString[] = __DATE__;
#endif

// Actually, second period is 217*18*256usec or 0.999936 seconds
// 3906: 2*3*3*7*31 or 217*18
//
#define RTCC_POSTSCALE 18
// #define RTCC_SECOND_PERIOD 217
// NOTE: changed to 200 to make the percent math simpler
#define RTCC_SECOND_PERIOD 200

// 10 second counter, 0..9
int8 tenSecondCounter = 0;
```

```
// A/D variables
#define BATTERY_CHANNEL         0   /* RA0/AN0 */
float batteryVoltage;

// 200 second average
float averageBatteryVoltage;

#define SOLAR_CHANNEL           1   /* RA1/AN1 */
float solarVoltage;

#define EXTERNAL_CHANNEL        2   /* RA2/AN2 */
float externalAnalogVoltage;

#define REFERENCE_CHANNEL       3
// float referenceVoltage;          // RA3/AN3, full-scale hw ref 4.1V

#define TEMPERATURE_CHANNEL     4   /* RA5/AN4 */
float temperature;

struct _scaleFactors
{
    float batteryScale;
    float solarScale;
    float externalAnalogScale;
    float temperatureScale;
    int8 temperatureOffset;
};

struct _scaleFactors scaleFactors;

// Calculate the nominal scale factors from the resistor values
// Full scale (1024) is 4.1V, for 4mV per bit actual resolution (10 bits)
// Readings go from 0 .. 1023 (right justified)
// Offsets are subtracted from original count; then remaining count is
// multiplied by scale.
#define FULL_SCALE_COUNTS   (1 << ATOD_RESOLUTION)
#define REFERENCE_VOLTAGE   4.1

#define BATTERY_SCALE   ((REFERENCE_VOLTAGE * 12.49)/(FULL_SCALE_COUNTS * 2.49))
#define SOLAR_SCALE     ((REFERENCE_VOLTAGE * 12)/(FULL_SCALE_COUNTS * 2))
#define EXTERNAL_SCALE  ((REFERENCE_VOLTAGE * 16.65)/(FULL_SCALE_COUNTS * 6.65))

// LM62: 15.6mV/degC plus 480mV
// 25 degrees C = 0.39V + 0.48V = 0.87V
// or a reading of 217 (0.87/0.004)
#define TEMPERATURE_SCALE   ((REFERENCE_VOLTAGE)/(FULL_SCALE_COUNTS * 0.0156))

// round to nearest int16
#define ROUND(n)    ((int16)((n) + 0.5))

// Offset is 0.48/0.004
// should be 123
#define TEMPERATURE_OFFSET  ROUND(0.48 * FULL_SCALE_COUNTS / REFERENCE_VOLTAGE)

struct _voltageThresholds
{
    float systemOff;
    float systemOn;
```

```
    float floatVoltage;
    float overchargeVoltage;
};

struct _voltageThresholds voltageThresholds;

// initial values (25deg C):
#define SYSTEM_OFF_THRESHOLD    10.5
#define SYSTEM_ON_THRESHOLD     11.5
#define FLOAT_VOLTAGE           13.7
#define OVERCHARGE_VOLTAGE      14.7

// EEPROM:
// flag, scaleFactors, voltageThresholds
#pragma rom 0x2100={ 0x55 }

// set by interrupt handler:
// length must be mult. of 2
// (and must correspond to pointer length above in #device)
#define MAX_SERIAL_COMMAND_LENGTH   32
int8 serialCommand[MAX_SERIAL_COMMAND_LENGTH];
int8 nextCommandByte = 0;
BOOLEAN haveACommand = FALSE;

signed int8 solarPercent = 0;
void setSolarPWM(signed int8 percent);
void setExternalPWM(signed int8 percent);

signed int8 externalPercent;

#define STATE_TICK_SECONDS  2
#define SECONDS(n)  ((n)/STATE_TICK_SECONDS)
#define MINUTES(n)  ((n*60)/STATE_TICK_SECONDS)
#define HOURS(n)    ((n*60*60)/STATE_TICK_SECONDS)

#define MAXIMUM_OVERCHARGE_TIME     HOURS(3)
#define MINIMUM_OVERCHARGE_TIME     MINUTES(5)
#define WARNING_TIME                MINUTES(3)
#define OVERRIDE_WARNING_TIME       SECONDS(30)
#define MINIMUM_OFF_TIME            MINUTES(10)
#define BASELINE_TIME               100 /* 1% per tick */

typedef enum
{
    CS_WAITING_FOR_SUN,     // 0
    CS_BULK_CHARGE,         // 1
    CS_OVERCHARGE,          // 2
    CS_FLOAT_CHARGE,        // 3
    CS_GET_BASELINE_PWM     // 4
} _chargingState;

_chargingState chargingState = CS_WAITING_FOR_SUN;

int16 timeInChargingState = 0;  // ticks
signed int8 baselinePWM = 0;

typedef enum
{
    SP_WAITING_FOR_SYSTEM_ON,   // 0
```

```
    SP_SYSTEM_ON,               // 1
    SP_WARNING,                 // 2
    SP_SYSTEM_OFF,              // 3
    SP_OFF_DELAY                // 4 waiting on delay to be turned back on
} _systemPowerState;

_systemPowerState systemPowerState = SP_WAITING_FOR_SYSTEM_ON;

int16 timeInSystemPowerState = 0;
int32 timeUntilPowerChange = 0;         // decremented if non-zero (timeout)
BOOLEAN powerSwitchOverride = FALSE;    // on to force system off
BOOLEAN powerSwitchIsOn = FALSE;

// this figure is taken from the Panasonic battery data sheet
float temperatureCorrectedFloatVoltage(void)
{
    float tempCorrectionFactor;
    tempCorrectionFactor = (temperature >= 25.0) ? 0.020 : -0.016;
    return voltageThresholds.floatVoltage - tempCorrectionFactor * (temperature
- 25.0);
}

// this figure is taken from the Panasonic battery data sheet
float temperatureCorrectedOverchargeVoltage(void)
{
    float tempCorrectionFactor;
    tempCorrectionFactor = (temperature >= 25.0) ? 0.033 : -0.028;
    return voltageThresholds.overchargeVoltage - tempCorrectionFactor * (tempera
ture - 25.0);
}

// this uses a fixed value
float temperatureCorrect(float voltage)
{
    return voltage - (temperature - 25.0) * 23.4e-3;
}

// ramp up or down to maintain a voltage
// rate is 1% every 2 seconds
void maintainVoltage(float voltage)
{
    if (batteryVoltage > voltage)
        setSolarPWM( solarPercent - 1 );
    else
        setSolarPWM( solarPercent + 1 );
}

void printChargingState(void)
{
    printf("CS=%u CT=%lu V=%6.3f %%=%u BL=%u",
        chargingState,
        timeInChargingState * STATE_TICK_SECONDS,
        batteryVoltage,
        solarPercent,
        baselinePWM);

    printf(" SS=%u ST=%lu", systemPowerState, timeInSystemPowerState * STATE_TICK
_SECONDS);
```

```
    printf(" SD=%lu OR=%u PS=%u",
        timeUntilPowerChange * STATE_TICK_SECONDS,
        powerSwitchOverride,
        powerSwitchIsOn);
}

void newChargeState(_chargingState cs)
{
    chargingState=cs;
    // set this so that the increment will roll over to 0
    timeInChargingState = (int16)~0UL;
}

void chargeControl(void)
{
    float Voc, V12, Vf, V31;
    static int8 averageSolarPercent;

    // we have to compute these periodically
    // because the temperature can change.
    Voc = temperatureCorrectedOverchargeVoltage();  // overcharge voltage
    V12 = Voc * 0.95;   // bulk to overcharge threshold
    Vf = temperatureCorrectedFloatVoltage();    // float voltage
    V31 = Vf * 0.9;     // float to bulk threshold

    switch (chargingState)
    {
        // We remain in this state until the open circuit solar voltage
        // gets 0.4V above the battery voltage (and so has the
        // possibility of charging the battery). When this happens, the
        // next state is determined by the battery voltage. If the
        // battery voltage is in excess of 90% of the float voltage, we
        // enter the CS_FLOAT_CHARGE state, otherwise we enter the
        // CS_BULK_CHARGE state.
        case CS_WAITING_FOR_SUN:
            if (solarVoltage >= batteryVoltage + 0.4)
            {
                // here comes the sun...
                if (batteryVoltage > V31)
                    newChargeState(CS_FLOAT_CHARGE);
                else
                    newChargeState(CS_BULK_CHARGE);
            }
            break;

        // In this state, full charging current is applied to the
        // battery until the battery voltage until the voltage gets to
        // 95% of the overcharge voltage. When this happens, we enter
        // the CS_GET_BASELINE_PWM state. There is no timeout in this
        // state, as the transition voltage should be low enough to
        // prevent gassing in the cells (worst case the battery would
        // stay at 2.33V per cell, which is below the gassing level of
        // 2.35 to 2.40V per cell).
        case CS_BULK_CHARGE:
            if (timeInChargingState == 0)
                setSolarPWM(100);
            else if (batteryVoltage >= V12)
            {
                newChargeState(CS_GET_BASELINE_PWM);
```

```
                averageSolarPercent = solarPercent;
            }
            break;

        // In this state, the 95% of overcharge voltage level is
        // maintained for 200 seconds while the average PWM is
        // recorded. At the end of the time period, the average PWM
        // level is recorded as the baseline PWM. Because of the
        // averaging, the average will probably be somewhat higher than the
        // actual PWM at that point (worst case, at 0% PWM, the average
        // will be 4%). Next state is CS_OVERCHARGE.
        case CS_GET_BASELINE_PWM:
            averageSolarPercent += (int8)solarPercent / 8;
            averageSolarPercent = ((int16)averageSolarPercent * 8UL) / 9;
            if (timeInChargingState < BASELINE_TIME)    // 1% per tick max
                maintainVoltage(V12);
            else
            {
                baselinePWM = averageSolarPercent;
                newChargeState(CS_OVERCHARGE);
            }
            break;

        // Now we have a reference PWM percentage to use as an indication
        // that the battery has charged up. Of course, if the load or supply
        // fluctuates much during this time, we'll get spurious results, so
        // also provide a minimum and maximum overcharge time.
        // Maintain the overcharge voltage by adjusting the PWM.
        // The current (and hence the PWM) required to maintain this voltage
        // should decrease as the battery charges up.
        // If we see that the PWM has gotten down to less than 12.5% over the
        // baseline PWM (an easy division by 8), then figure we're charged.
        case CS_OVERCHARGE:
            if (timeInChargingState > MINIMUM_OVERCHARGE_TIME
                && (solarPercent − baselinePWM < baselinePWM / 8
                    || timeInChargingState > MAXIMUM_OVERCHARGE_TIME))
                        newChargeState(CS_FLOAT_CHARGE);
            else
                maintainVoltage(Voc);
            break;

        // In this state, the battery voltage is maintained at a safe
        // float level of 13.7V (25C). We stay in this state until the
        // battery voltage sags below 90% of the float voltage, at
        // which time we enter the CS_BULK_CHARGE state again for
        // another charging cycle. It's possible that we'll enter the
        // CS_BULK_CHARGE state when the sun is gone, but that won't
        // be a problem.
        case CS_FLOAT_CHARGE:
            if (batteryVoltage <= V31)
                newChargeState(CS_BULK_CHARGE);
            else
                maintainVoltage(Vf);
            break;
    }

    timeInChargingState++;
}
```

```
void newSystemPowerState(_systemPowerState cs)
{
    systemPowerState=cs;
    // set this so that the increment will roll over to 0
    timeInSystemPowerState = (int16)~0UL;
}

void systemPowerControl(void)
{
    powerSwitchIsOn = POWER_SWITCH_IS_ON;

    switch (systemPowerState)
    {
        // In this state, we wait for the system power switch to be
        // turned on and the battery to get above the system power on
        // threshold of 11.5V at 25C. On entry, we reset the override
        // flag. The next state is SP_SYSTEM_ON.
        case SP_WAITING_FOR_SYSTEM_ON:
            powerSwitchOverride = FALSE;
            if (powerSwitchIsOn
                  && batteryVoltage >= temperatureCorrect(SYSTEM_ON_THRESHOLD))
                newSystemPowerState(SP_SYSTEM_ON);
            break;

        // In this state, we turn off the batteryLow output and turn on
        // the system using either the external power supply control
        // output or the external load PWM output. We wait until the
        // battery goes below the system off threshold of 10.5V (25C)
        // or until the power switch gets turned off, or until the
        // powerSwitchOverride flag gets set by the serial command. The
        // next state is SP_WARNING.
        case SP_SYSTEM_ON:
            if (timeInSystemPowerState == 0)
            {
                OUTPUT_LOW(BATT_LOW_PIN);
                TURN_SYSTEM_ON;
                if (!WANDER_MODE)
                    setExternalPWM(100);
            }
            else if (batteryVoltage <= temperatureCorrect(SYSTEM_OFF_THRESHOLD)
                      || !powerSwitchIsOn
                      || powerSwitchOverride)
                newSystemPowerState(SP_WARNING);
            break;

        // On entry to this state, we turn on the batteryLow output and
        // wait for 5 minutes. If the voltage goes above the system off
        // threshold and the power switch is still on and there is no
        // powerSwitchOverride flag set, we return to the SP_SYSTEM_ON
        // state. Otherwise, we go to the SP_SYSTEM_OFF state.
        case SP_WARNING:
            if (timeInSystemPowerState == 0)
                OUTPUT_HIGH(BATT_LOW_PIN);
            else if (powerSwitchIsOn
                      && !powerSwitchOverride
                      && batteryVoltage > temperatureCorrect(SYSTEM_OFF_THRESHOL
D))
                newSystemPowerState(SP_SYSTEM_ON);
            else if ((powerSwitchOverride && timeInSystemPowerState >= OVERRIDE_
```

```
WARNING_TIME)
                    || timeInSystemPowerState >= WARNING_TIME)
                newSystemPowerState(SP_SYSTEM_OFF);
            break;

        // In this state, we turn the attached load off using the
        // external load control output or the external load PWM
        // output. If there is no offDelay set by the serial command,
        // we enter the SP_WAITING_FOR_SYSTEM_ON state after 10
        // minutes, otherwise we enter the SP_OFF_DELAY state.
        case SP_SYSTEM_OFF:
            if (timeInSystemPowerState == 0)
            {
                TURN_SYSTEM_OFF;
                setExternalPWM(0);
            }
            else if (timeUntilPowerChange)
                newSystemPowerState(SP_OFF_DELAY);
            else if (timeInSystemPowerState > MINIMUM_OFF_TIME)
                newSystemPowerState(SP_WAITING_FOR_SYSTEM_ON);
            break;

        // This state is when we're waiting for the delay to time out
        // so the system can be turned back on. The delay is set with
        // the P00 command. At the end of the delay, we enter the
        // SP_WAITING_FOR_SYSTEM_ON state, which will cause the system
        // to be turned back on if the power switch is still ON and the
        // battery has a high enough voltage.
        case SP_OFF_DELAY:
            // exit when the counter is decremented to 0
            if (timeUntilPowerChange)
                --timeUntilPowerChange;
            else
                newSystemPowerState(SP_WAITING_FOR_SYSTEM_ON);
            break;
    }

    timeInSystemPowerState++;
}

void copyStructToEEPROM(char *from, int8 address, int8 size)
{
    signed int8 i;
    for (i = size - 1; i >= 0; i--)
    {
        WRITE_EEPROM(i + address, from[i]);
        RESTART_WDT();
    }
}

void copyStructFromEEPROM(char *to, int8 address, int8 size)
{
    signed int8 i;
    for (i = size - 1; i >= 0; i--)
    {
        to[i] = READ_EEPROM(i + address);
        RESTART_WDT();
    }
}
```

```
// Set nominal values into EEPROM and RAM
// First int8 starts out 0x55 and is programmed to 0xAA
// when rest of eeprom is set up.
void initializeScaleFactors(BOOLEAN force)
{
    char magic;

    copyStructFromEEPROM(&magic, 0, 1);

    if (!force && (magic == 0xAA))
    {
        // EEPROM is set up; read offsets and scales from EEPROM
        copyStructFromEEPROM((char*)&scaleFactors,
                1,
                sizeof(scaleFactors));
        copyStructFromEEPROM((char*)&voltageThresholds,
                1+sizeof(scaleFactors),
                sizeof(voltageThresholds));
    }
    else
    {
        // Note that I have to do member-by-member setting because const struct
        // copying is broken in CCS C 3.059.
        scaleFactors.batteryScale = BATTERY_SCALE;
        scaleFactors.solarScale = SOLAR_SCALE;
        scaleFactors.externalAnalogScale = EXTERNAL_SCALE;
        scaleFactors.temperatureScale = TEMPERATURE_SCALE;
        scaleFactors.temperatureOffset = TEMPERATURE_OFFSET;

        voltageThresholds.systemOff = SYSTEM_OFF_THRESHOLD;
        voltageThresholds.systemOn = SYSTEM_ON_THRESHOLD;
        voltageThresholds.floatVoltage = FLOAT_VOLTAGE;
        voltageThresholds.overchargeVoltage = OVERCHARGE_VOLTAGE;

        magic = 0xAA;
        copyStructToEEPROM(&magic, 0, 1);
        copyStructToEEPROM((char*)&scaleFactors,
                1,
                sizeof(scaleFactors));
        copyStructToEEPROM((char*)&voltageThresholds,
                1+sizeof(scaleFactors),
                sizeof(voltageThresholds));
    }
}

#pragma int_rda
void isrReceivedData(void)
{
    int8 c;

    while (kbhit())
    {
        c = getc();

        if (c == '\r')
        {
            c = 0;
            haveACommand = TRUE;
```

```
        }
        serialCommand[nextCommandByte] = c;
        if (c == 0)
            nextCommandByte = 0;
        else
        {
            nextCommandByte++;
            nextCommandByte &= MAX_SERIAL_COMMAND_LENGTH - 1;
        }
    }
}

// This will be at a 55.55msec rate (18hz)
#pragma int_timer2
void isrTimer2()
{
    static int8 postscaler;

    if (++postscaler >= RTCC_POSTSCALE)
    {
        // every second
        postscaler = 0;
        if (++tenSecondCounter >= 10)
        {
            // every 10 seconds
            tenSecondCounter = 0;
        }
    }
}

void setSolarPWM(signed int8 percent)
{
    if (percent > 100) percent = 100;
    else if (percent < 0) percent = 0;

    // SET_PWM1_DUTY((int8)(((int16)percent * (RTCC_SECOND_PERIOD+1)) / (int16)1
00 ));
    SET_PWM1_DUTY(percent << 1);
    solarPercent = percent;
}

void setExternalPWM(signed int8 percent)
{
    if (percent > 100) percent = 100;
    else if (percent < 0) percent = 0;

    // SET_PWM2_DUTY((int8)(((int16)percent * (RTCC_SECOND_PERIOD+1)) / (int16)1
00 ));
    SET_PWM2_DUTY(percent << 1);
    externalPercent = percent;
}

// initialize WDT
// Note that post-scaler is not available if it is being used by Timer0
#pragma byte OPTION=0x81

void initializeWDT(void)
{
    RESTART_WDT();
```

```
    SETUP_TIMER_0(RTCC_INTERNAL);
    OPTION = (OPTION & 0xF0) | 0x0F;    // PSA=1 (prescalar for WDT), period 2.3
04s
}

void initializeFromPowerUp(void)
{
    TURN_SYSTEM_OFF;    // turn PIN_B5 on

    // initialize port direction registers
    SET_TRIS_A(DIRECTION_A);
    SET_TRIS_B(DIRECTION_B);
    SET_TRIS_C(DIRECTION_C);
    PORT_B_PULLUPS(TRUE);   // for jumper pullup

    // initialize ADC converter and RAM values
    SETUP_ADC_PORTS(A_ANALOG_RA3_REF);
    SETUP_ADC(ADC_CLOCK_DIV_8); // minimum 1.6usec (this gives 2usec at 4MHz)

    initializeScaleFactors(FALSE);

    // initialize PWM (CCP1, output to RC2)
    // 1. Establish the PWM period by writing to the PR2 register.
    // 288 Hz rate, 8 bits max; clock is 1usec; therefore
    // use x16 prescalar (16usec) and 217 divisor
    // We also get a timer interrupt every 18Hz (55.555 msec)
    // PR2 should be 217 (0xd9)
    SETUP_TIMER_2(T2_DIV_BY_16, RTCC_SECOND_PERIOD-1, 16);
    // 2. Establish the PWM duty cycle by writing to the DCxB9:DCxB0 bits.
    setSolarPWM(100);
    setExternalPWM(0);
    // 3. Make the CCPx pin an output by clearing the appropriate TRIS bit.
    // (see above SET_TRIS_C)
    // 4. Establish the TMR2 prescale value and enable Timer2
    // by writing to T2CON.
    // (see above step 1)
    // 5. Configure the CCP module for PWM operation.
    SETUP_CCP1(CCP_PWM);
    SETUP_CCP2(CCP_PWM);

    // initialize Timer1 (16 bit)
    // Timer1 can be reset by CCPx outputs
    SETUP_TIMER_1(T1_DISABLED);

    // initialize interrupts
    // ENABLE_INTERRUPTS(GLOBAL);   // later...
    ENABLE_INTERRUPTS(INT_RDA);     // received data
    ENABLE_INTERRUPTS(INT_TIMER2);
}

float scaleAtoD(int8 channel, float scale)
{
    SET_ADC_CHANNEL(channel);
    DELAY_US(30);
    return READ_ADC() * scale;
}

float scaleAtoDWithOffset(int8 channel, float scale, int8 offset)
{
```

```
    SET_ADC_CHANNEL(channel);
    DELAY_US(30);
    return ((signed int16)READ_ADC() - (signed int16)offset) * scale;
}

void readADC(void)
{
    int16 reading;

    batteryVoltage = scaleAtoD(BATTERY_CHANNEL, scaleFactors.batteryScale);
    solarVoltage = scaleAtoD(SOLAR_CHANNEL, scaleFactors.solarScale);
    externalAnalogVoltage = scaleAtoD(EXTERNAL_CHANNEL, scaleFactors.externalAna
logScale);
    temperature = scaleAtoDWithOffset(TEMPERATURE_CHANNEL, scaleFactors.temperat
ureScale,
        scaleFactors.temperatureOffset);
}

// Serial commands:
// C\r      dump charging state
// Ennn\r   set external PWM percent to nnn (0..100)
// F\r      display scale factors (B,S,E,T)
// I\r      initialize scale factors
// Pnn [nnnnnn]\r   turn system power off (n=00), specify optional turn-on delay
// R\r      display raw A/D readings (B,S,E,T)
// Snnn\r   set solar PWM percent to nnn (0..100)
// T\r      dump thresholds
// V\r      display voltages (B,S,E,T)

void processSerialCommand(void)
{
    int8 i;
    int8 lastPercent;

    switch (serialCommand[0])
    {
        // C\r
        // Dump charging state
        case 'C':
            printChargingState();
            break;

        // Ennn\r
        // set external PWM percent to nnn (0..100)
        // Assumes that cut/jumper has been made connecting RC1/CCP2 (pin 12)
        // to gate of Q5 (instead of RB4, pin 25).
        case 'E':
            i = atoi(serialCommand+1);
            setExternalPWM(i);
            break;

        // F\r
        // display scale constants (B,S,E,T)
        case 'F':
            printf("B=%6.4f S=%6.4f E=%6.4f T=%6.4f %u",
                    scaleFactors.batteryScale, scaleFactors.solarScale,
                    scaleFactors.externalAnalogScale, scaleFactors.temperatureSc
ale,
                    scaleFactors.temperatureOffset);
```

```
            break;

        // I\r
        // re-initialize EEPROM to defaults
        case 'I':
            initializeScaleFactors(TRUE);
            break;

        // P00 [nnnn]\r
        // turn system power off if waiting to turn it off
        // optionally, provide a time value in seconds
        // to turn the power back on
        case 'P':
            if (serialCommand[1] == '0' && serialCommand[2] == '0')
            {
                if (serialCommand[3] == ' ')
                    timeUntilPowerChange = atol(serialCommand + 4) >> 1;
                powerSwitchOverride = TRUE;
            }
            break;

        // R\r
        // display raw A/D readings (B,S,E,ref,T)
        // Turns off the solar PWM for 0.5 seconds first
        case 'R':
            lastPercent = solarPercent;
            setSolarPWM(0);
            DELAY_MS(500);
            for (i = BATTERY_CHANNEL; i <= TEMPERATURE_CHANNEL; i++)
            {
                SET_ADC_CHANNEL(i);
                DELAY_US(30);
                printf("%lu ", READ_ADC());
            }
            setSolarPWM(lastPercent);
            break;

        // Snnn\r
        // set solar PWM percent to nnn (0..100)
        case 'S':
            if (serialCommand[1])
            {
                i = atoi(serialCommand+1);
                setSolarPWM(i);
            }
            break;

        // T\r
        // Dump thresholds
        case 'T':
            // work around a bug in CCS C:
            printf("F=%6.2f ", temperatureCorrectedFloatVoltage());
            printf("OC=%6.2f ", temperatureCorrectedOverchargeVoltage());
            printf("OFF=%6.2f ", temperatureCorrect(voltageThresholds.systemOff))
;
            printf("ON=%6.2f ", temperatureCorrect(voltageThresholds.systemOn));
            printf("T=%5.2f", temperature);
            break;
```

```
        // V\r
        // display voltages (B,S,E,T)
        case 'V':
            printf("B=%5.2f (%5.2f) S=%5.2f E=%5.2f T=%6.2f",
                    batteryVoltage, averageBatteryVoltage,
                    solarVoltage, externalAnalogVoltage, temperature);
            break;

        default:
            putc('?');
    }
    putc('\r');
    putc('\n');
}

// #pragma zero_ram
void main(void)
{
    static BOOLEAN sampled = FALSE;
    static int8 lastTenSecondCounter = 0;
    int8 restart;

    restart = RESTART_CAUSE();
    initializeWDT();

#if defined(PRINT_ID)
    printf("%s %s %x\r\n", idString, dateString, restart);
#endif

    switch (restart)
    {
        case NORMAL_POWER_UP:
        case MCLR_FROM_SLEEP:
            initializeFromPowerUp();
            break;

        case WDT_FROM_SLEEP:
        case WDT_TIMEOUT:
            TURN_SYSTEM_OFF;
            setSolarPWM(0);
            setExternalPWM(0);
            LED_OFF;
            while (1);      // endless loop is easier to catch with ICD
    }

    // flush receive queue before enabling interrupts
    while (kbhit())
        getc();

    ENABLE_INTERRUPTS(GLOBAL);

    // main loop (background task)

    readADC();
    averageBatteryVoltage = batteryVoltage;

    while (1)
    {
        if (haveACommand)
```

```
        {
            LED_ON;
            processSerialCommand();
            haveACommand = FALSE;
            LED_OFF;
        }

        if (lastTenSecondCounter != tenSecondCounter)   // edge detect
        {
            LED_ON;
            lastTenSecondCounter = tenSecondCounter;

            if (tenSecondCounter & 1)   // every other second
            {
                chargeControl();
                systemPowerControl();
            }
            else
            {
                readADC();
                // update 200-second average
                averageBatteryVoltage += batteryVoltage * 0.01;
                averageBatteryVoltage *= 1.0/1.01;
            }
            LED_OFF;
        }

        RESTART_WDT();
    }
}

/* NOTES
 * No string pasting: "v1.0" __DATE__
 *
 * This gets error 35, "Number of bits is out of range"
 * even though the book says that the :expr can be 1-8 bits:
 *  struct
 *  {
 *      int8 finteger: 5;
 *      int8 fraction: 8;
 *  };
 *
 * Can't pass a struct as a param
 *
 * this doesn't work (generates 8x8 mul, div):
 * SET_PWM2_DUTY((int8)((percent * (int16)(RTCC_SECOND_PERIOD+1)) / 100 ));
 *
 * SETUP_WDT(WDT_2304MS); seems to read from wrong RAM
 *
 * struct copies from const (ROM) to RAM repeat the first member
 *
 * This creates really bad code that skips the next couple of lines:
 *
 * XXX: creates buggy jump
 * float targetVoltage;
 * float temperature;
 * targetVoltage -= (temperature >= 25.0)
 *      ? 0.020 * (temperature - 25.0)
 *      : 0.016 * (25.0 - temperature);
```

```
 *
 *  This complains about undefined identifier lastPercent:
 *
        case 'R':
            {
                int8 lastPercent;
                lastPercent = solarPercent;
                setSolarPWM(lastPercent);
            }
            break;

 *
 * Can't use the same identifiers for struct and typedef tags, or for
 * struct/typedef tags and variable names. i.e.
 * struct a { ... }; typedef struct a a;
 * or
 * struct a { } a;
 * don't work
 *
 * This doesn't work right (prints the same voltage 4 times):
 *
            printf("F=%6.2f OC=%6.2f OFF=%6.2f ON=%6.2f T=%5.2f",
                    temperatureCorrectedFloatVoltage(),
                    temperatureCorrectedOverchargeVoltage(),
                    temperatureCorrect(SYSTEM_OFF_THRESHOLD),
                    temperatureCorrect(SYSTEM_ON_THRESHOLD),
                    temperature);
 *
 * This const causes the compiler to barf:
 * void copyStructToEEPROM(char const *from, int8 address, int8 size)
 */

// vim: ts=4 sw=4 columns=100
```

```
/* Program to monitor EZ-IO on Octagon board
 * and shut down system when it finds a bit set
 * $Id: powerMonitor.c,v 1.1 2001/12/11 20:16:10 ned Exp $
 */
#include <sys/io.h>
#include <stdio.h>
#include <unistd.h>

char const * const defaultResponse = "echo 'going down!'";

#define BASE 0x330
#define NOTIFY_PORT     PORT_C
#define NOTIFY_MASK     0x01
#define NOTIFY_STATE    0x01

enum
{
    PORT_A = BASE,
    PORT_B,
    PORT_C,
    CONTROL
};

void usage(char const * progname, int exitcode)
{
    fprintf(stderr, "usage: %s [-c cmd]\n", progname);
    exit(exitcode);
}

int main( int argc, char * const argv[] )
{
    char const * response = defaultResponse;
    int opt;

    while ((opt = getopt(argc, argv, "hc:")) != -1)
    {
        switch (opt)
        {
            case 'h':
            case '?':
                usage(argv[0], 1);
                break;
            case 'c':
                response = optarg;
                break;
            default:
                break;
        }
    }
    /* now optind points to the next non-option arg */

    if (ioperm(BASE, 4, 1))
    {
        perror("Can't ioperm");
        exit(2);
    }
    while (1)
    {
        if (inb(NOTIFY_PORT) & NOTIFY_MASK == NOTIFY_STATE)
```

```
        {
            int retval = system(response);

            switch (retval)
            {
                case 127:
                case -1:
                    fprintf(stderr, "system call (%s) failed\n", response);
                    perror("");
                    exit(3);
                    break;
                case 0:
                    exit(0);
                    break;
                default:
                    retval = (retval & 0xff00) >> 8;
                    fprintf(stderr,
                            "system call (%s) returns %d\n",
                            response, retval);
                    exit(retval);
                    break;
            }
        }
        sleep(10);
    }
}

/* vim: ts=4 sw=4
*/
```

```
# ${whatever} expands first as an environment variable
# and then as a prior config value
# and then (failing that) is evaluated as a Perl expression
root = /usr/local/wander
dbdirectory = ${root}/data
dbfile = ${dbdirectory}/collectedData.db
channeldirectory = ${root}/channels
channelfile = ${channeldirectory}/channels.db
templatedirectory = ${root}/templates
mailsender = wander@wander.com
maildomain = wander.com
mailrecipient = ned@whidbey.net
mailhost = mail.whidbey.net
mailer = ${root}/bin/emailWander −t
exporter = ${root}/bin/exportData
# do not use expansions in lastmailed:
lastmailed = /var/run/wander/lastmailed
collectorname = wanderCollector
collectorpid = /var/run/wander/wanderCollector.pid
collectorlog = /var/log/wander/collect.log
lcdmonitorpid = /var/run/wander/wanderLCD.pid
lcdport = /dev/ttyS3
# Each of the {##} below refers to a channel value.
chargestatus = BattV={14} SolarV={15} Temp={17} ChgState={4} ExtV={16}
```

```
#! /bin/sh

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
DAEMON=/usr/local/wander/bin/wanderCollector
NAME=wanderCollector
DESC="Wander data collector"
DB_PATH=/usr/local/wander/data
PIDFILE=/var/run/wander/$NAME.pid
SSDFLAGS=--quiet
# SSDFLAGS=--verbose

test -f $DAEMON || exit 0

set -e

case "$1" in
  start)
    echo -n "Starting $DESC: "
    start-stop-daemon --start $SSDFLAGS --pidfile $PIDFILE \
        --exec /usr/bin/perl --startas $DAEMON -- -d
    # start-stop-daemon --start --background \
    #   --exec /usr/local/bin/db_deadlock -- -t 120 -h $DB_PATH
    echo "$NAME."
    ;;
  stop)
    echo -n "Stopping $DESC: "
    start-stop-daemon --stop $SSDFLAGS --pidfile $PIDFILE \
        --exec /usr/bin/perl --startas $DAEMON
    # start-stop-daemon --stop --exec /usr/local/bin/db_deadlock
    # /usr/local/bin/db_recover -h $DB_PATH
    echo "$NAME."
    ;;
  sync)
    echo -n "Syncing $DESC: "
    start-stop-daemon --stop --signal USR1 $SSDFLAGS --pidfile $PIDFILE \
        --exec /usr/bin/perl --startas $DAEMON
    echo "$NAME."
    ;;
  restart|force-reload)
    echo -n "Restarting $DESC: "
    start-stop-daemon --stop $SSDFLAGS --pidfile \
        $PIDFILE --exec /usr/bin/perl --startas $DAEMON
    sleep 1
    start-stop-daemon --start $SSDFLAGS --pidfile \
        $PIDFILE --exec /usr/bin/perl --startas $DAEMON -- -d
    echo "$NAME."
    ;;
  *)
    N=/etc/init.d/$NAME
    # echo "Usage: $N {start|stop|restart|reload|force-reload}" >&2
    echo "Usage: $N {start|stop|restart|force-reload}" >&2
    exit 1
    ;;
esac

exit 0
```

```
#! /bin/sh

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
DAEMON=/usr/local/wander/bin/wanderLCD
NAME=wanderLCD
DESC="Wander LCD driver"
PIDFILE=/var/run/wander/$NAME.pid
SSDFLAGS=--quiet

test -f $DAEMON || exit 0

set -e

case "$1" in
  start)
    echo -n "Starting $DESC: "
    start-stop-daemon --start $SSDFLAGS --pidfile $PIDFILE --exec /usr/bin/perl
--startas $DAEMON -- -d
    echo "$NAME."
    ;;
  stop)
    echo -n "Stopping $DESC: "
    # --quiet
    start-stop-daemon --stop $SSDFLAGS --signal 15 --pidfile $PIDFILE --exec /us
r/bin/perl --startas $DAEMON
    echo "$NAME."
    ;;
  restart|force-reload)
    echo -n "Restarting $DESC: "
    start-stop-daemon --stop $SSDFLAGS --pidfile $PIDFILE --exec /usr/bin/perl -
-startas $DAEMON
    sleep 1
    start-stop-daemon --start $SSDFLAGS --pidfile $PIDFILE --exec /usr/bin/perl
--startas $DAEMON -- -d
    echo "$NAME."
    ;;
  *)
    N=/etc/init.d/$NAME
    # echo "Usage: $N {start|stop|restart|reload|force-reload}" >&2
    echo "Usage: $N {start|stop|restart|force-reload}" >&2
    exit 1
    ;;
esac

exit 0
```

```
#!/bin/sh
# Watches for pending shutdown and kills system.

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
DAEMON=/usr/local/wander/bin/powerMonitor
NAME=wanderPowerMonitor
DESC="Wander Power Monitor"
# FLAGS=--quiet

test -f $DAEMON || exit 0

set -e

case "$1" in
  start)
    echo -n "Starting $DESC: "
    start-stop-daemon --background --start $FLAGS --exec $DAEMON --startas $DAEM
ON -- -c '/sbin/poweroff -h'
    echo "$NAME."
    ;;
  stop)
    echo -n "Stopping $DESC: "
    start-stop-daemon --stop $FLAGS --exec $DAEMON
    echo "$NAME."
    ;;
  restart|force-reload)
    echo -n "Restarting $DESC: "
    start-stop-daemon --stop $FLAGS --exec $DAEMON
    sleep 1
    start-stop-daemon --background --start $FLAGS --exec $DAEMON --startas $DAEM
ON -- -c '/sbin/poweroff -h'
    echo "$NAME."
    ;;
  *)
    N=/etc/init.d/$NAME
    echo "Usage: $N {start|stop|restart|force-reload}" >&2
    exit 1
    ;;
esac

exit 0
```

```
#!/bin/bash
PATH=/usr/local/wander/bin:/sbin:/usr/sbin:/usr/local/bin:/usr/bin:/bin
lastmailed=$(grep '^lastmailed' /etc/wander.cfg | sed 's/.*= *//')
exportWanderData -f $lastmailed | emailWander
```