EzCapture

(Interactive measurement and
display of anaesthetic records)

Version 0.37

J.M. van Schalkwyk, D. Lowes

January 1, 2009

Contents

1 Introduction

This is a suite of programs that accomplishes the following:

- Reliable measurement of hand-written anaesthetic records, combined with recording of these measurements in the form of database (CSV) files that can be read by most database and analysis software;
- Partial automation of the above measurement process, with relevant checks;
- Translation of CSV files into a data format (.DAT) readable by the SAFER-sleep 'IDAS' program;
- Sequential display of IDAS .DAT files in a standard format familiar to anaesthetists who use the program for everyday patient data recording and display;
- Sequential (minute-by-minute) capture of display screens from IDAS, and storing of captured screens as .PNG data files;
- Sequential presentation of .PNG images (in real-time or, if desired, faster) by serving them up in a web-page. This application allows the anaesthetist to annotate the anaesthetic as it progresses.
- Storage of annotations to an internet database for subsequent analysis. The database is fully described, as is the PHP code that links the web-page (and interacting anaesthetist) to the database.

The suite is thus divided into the following sections:

1. Programs to capture data from a pair of digital calipers (Section 2);
2. Translation utilities (Section 4) that translate from IAR to XML, XML to CSV, and CSV to DAT formats;
3. Programs to replay data through IDAS, and screen-grab the resulting display (Section 5);
4. Web display and database routines (Sections 6–8) including SQL code and PHP code.

1.1 A note on the directory structure

This has become somewhat complex, so in Table 1 we show the relationships between noteworthy directories and sub-directories. In select subdirectories, we've also listed important files, in parenthesis.

```
ez-.-- ez-captur--->(ez-captur.pl; ez-gui.au3)
   |
   |-- ez-xlate---->(ez-xml2csv.pl;
   |                 ez-csv2dat.pl;
   |                 AnaestheticViewer.exe)
   |
   |-- ez-shw---.-- html.-->(mainpage.htm)
   |            |       |-- css -->(cascading style sheet)
   |            |        \- images --> (images to display)
   |            |
   |            |-- php -->(php scripts)
   |             \- sql -->(sql script)
   |
   |-- ez-replay-.->(ez-replay.au3)
   |              \-capture  -->(captured PNG images)
   |
   |-- ez-info--->(basic case info)
   |-- rawdata--->(raw csv files exported from Excel)
   |-- iar------->(grab encrypted IAR files from IDAS)
   |-- xml------->(files exported from IAR to XML)
   |-- csv------->(CSV files translated form XML)
   |-- dat------->(DAT files translated from CSV)
   |-- log ------>(error logging)
    \- images---->(images referred to by ez.TEX)
```

Table 1: Directory structure of EZ

The directories *ez-captur*, *ez-xlate*, *ez-shw* and *ez-replay* contain the relevant Perl and/or AutoIt executables that perform capture and formatting of raw data obtained from the digital calipers, translation between various formats, display (as HTML pages) and replay of .DAT files through SAFERsleep.

1.2 DOS execution

We have created a comprehensive Graphical User Interface (GUI) that is described in Section 2.4. However, we also have created a number of useful DOS batch files used to run components of our program suite. The main advantages of this console execution are simplicity and clarity, especially when it comes to debugging. The batch files include:

Raw data capture and processing

- **captur.bat** : Capture data from digital calipers (Section **??**) using an AutoIt script;

Translation between formats

Conversion of IAR to XML format is now accomplished within the main GUI program (written in AutoIt) — we simply invoke the IDAS viewer, and write the resulting output to an XML file.[1] Other translations are:

- **xml2csv.bat** : Invokes a Perl program to translate the source XML file (first argument, in */ez/xml*) to a CSV file stored in the *csv* subdirectory of */ez*. The batch file **test-x2c.bat** performs this transformation on a test file (Section 4.1.2);
- **csv2dat.bat** : Invokes a Perl program to translate source file to a DAT file stored in */ez/dat*. [FIX ME] The batch file **test-c2d.bat** does this on a sample file (Section 4.2.2).

The preceding files therefore mediate two pathways of data acquisition. The first is *manually* via measurements obtained from digital calipers, stored as CSV files in */ez/rawdata*, and then translated to processed CSV files (stored in */ez/csv*), followed by translation to SAFERsleep DAT files for submission to IDAS. The second is automatically, stored in SAFERsleep, retrieved from the server as IAR files (stored in */ez/iar*), decoded to XML (in */ez/xml*), and then translated sequentially to CSV and DAT files.

Replay a data file through IDAS

- **replay.bat** : Replay a file through SAFERsleep (Section 5.2) using AutoIt, and screen capture sequential images (every minute).

[1] Once we've overcome the paranoid security features??

2 ez-capture: Manual data capture

It is tedious to repeatedly measure values on an anaesthetic record chart, but these measurements are necessary when capturing data from manually recorded charts, so that they can be compared with automated records. In this section we will introduce our approach to data capture, use 'pseudocode' to sketch the code we will use to process data, and then construct a Perl program that does this processing.

2.1 Introductory notes

We initially thought that the process might work along the following lines:

1. Connect the Digital Calipers to the USB interface module, and insert the USB interface module into the USB port of the PC (running Windows);[2]
2. Press the *on* button on the Calipers;
3. Run the driving software [DETAILS], ensuring that the COM port etc are correctly set up;
4. Capture data acquired using the calipers;
5. Save the record to Excel (don't try other methods, as this may crash the PC);
6. Save the Excel spreadsheet as a CSV file, with a unique, sequential number;
7. Invoke a Perl script to:
   (a) Convert all digital caliper-recorded values to mmHg, rates;
   (b) Request input for SpO2 and EtCO2;
   (c) Write a unified, time-based record of all data as a CSV file.
8. Repeat record capture as required;
9. Exit, turn off calipers, shut down PC, and remove USB interface module and calipers.

We have partially automated the above process. This automation involves acquiring and saving basic information about the record before initiating capture; capturing separate caliper data for the systolic blood pressure, diastolic blood pressure, and heart rate, and entering SpO_2 and $ETCO_2$ data from the keyboard

[2] We used XP and cannot guarantee that this will work under Vista.

prior to invoking a Perl script to integrate the raw data into a final CSV file. Automation is achieved using the Windows program AutoIt, for which we've written scripts described in Section 2.4.

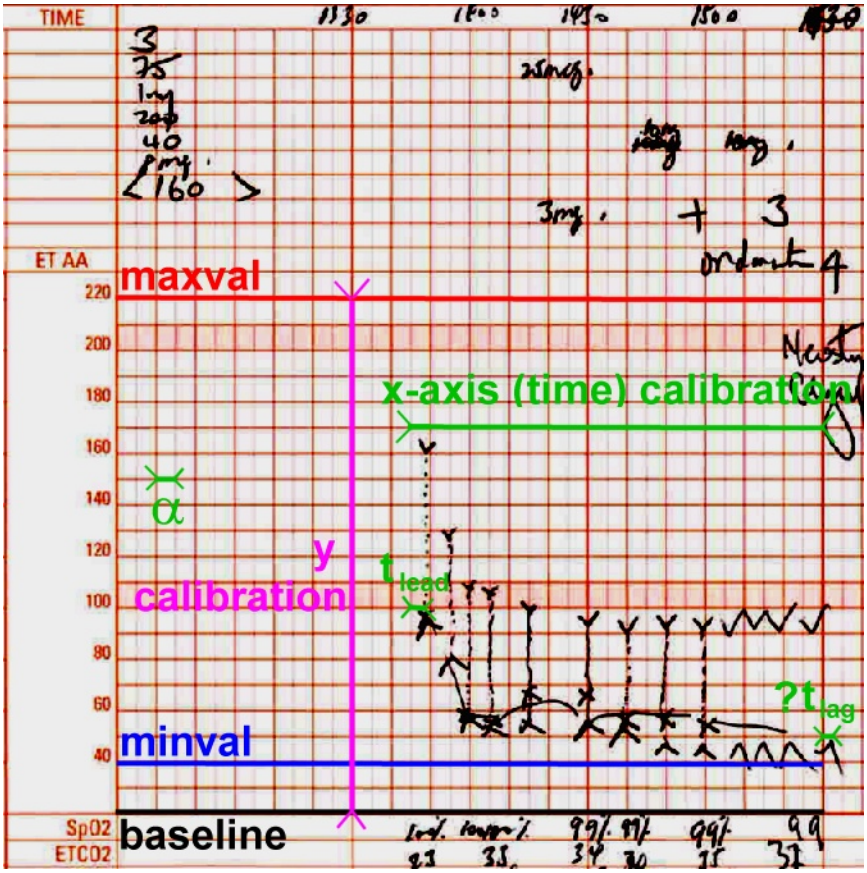

Figure 1: Constant values

The process of data entry uses certain short-cuts to signal detailed structure of the record, calibration and zeroing, and check-measurements. Note the following values:

- *ZERO* is a value in the range of -0.05 to 0.05 mm.
- *MAXVAL* signals a value in one of three ranges: 59–61 mm, 56.3–58.3 mm, and 83.9–85.9 mm, all being offsets from the baseline, with a centre value that is the average of the lower and upper limits. Each of these ranges refers to a different type of record, but each is equivalent to an 'actual' value of 220 mmHg, or a heart rate of 220 min^{-1}. *MAXVAL* is used as a calibration

signal. We will refer to the three types of records corresponding to the three ranges used above as type A, type B and type C records, respectively.

- *MINVAL* signals a measured value in the range of *MAXVAL*/10. *MINVAL* corresponds to a measurement extending from the baseline (at 20 mmHg, or a heart rate of 20 min^{-1}) to the line signalling a value of 40 (mmHg or min^{-1}).

- The *x-calibration value*. This is a measurement of duration from the start of the anaesthetic to the end of the anaesthetic, but with two caveats:

  1. Measurement must start from a pre-printed vertical line on the chart (not a user-defined mark), that is one of the vertical lines used to mark the time. This is normally a 5-minute mark, unless the user has chosen a different time-base. (Use the smallest printed interval). If the first datum recorded is not on a pre-printed vertical line, then choose the line *before* the first user mark, *NOT* afterwards. (When recording data as described below, signal the first recording as a missing value, then measure the offset, also described below).
  2. Measurement of the x-axis calibration must likewise end *at* a vertical pre-printed line, but if the last user marking is between two such lines, only measure to the vertical pre-printed line *just before* this line. If the last user marking coincides with a pre-printed vertical line, measure *to* this line.

- The value α is the distance between two (minor) vertical time lines. Unfortunately, this too varies depending on the type of record. For Types A–C, the values (in mm) are 2.15–2.35, 4.43/2–4.63/2 and 3.11–3.31. Note that in the type B record, the value of 4.53 usually refers to a ten-minute interval, but we always mandate a five-minute measuring interval! When we determine x-axis (time) offsets, we will use measured values to determine more precise values for alpha for a particular record. To make things even more complex, some anaesthetists choose to re-scale the x-axis, so we check for this by asking the user to enter start and end times.

MAXVAL (red), MINVAL (blue), alpha and the x- and y-calibration measurements are shown in Fig. 1, together with the baseline (in black) from which all y-measurements are made. In addition, this chart shows an initial delay (t_{lead}) between the point at which x-axis measurement started (a pre-printed vertical line) and the first user mark. Next look at the end of the anaesthetic. Although it initially seems that the anaesthetic actually ends *after* a pre-printed vertical line, this

is only if you look at the charting of diastolic blood pressure. If anything, the systolic mark is before the time line. Here we have chosen to use common sense (making the interpretation that the anaesthetist signalled a recording on the time line), so the t_{lag} can be disregarded. However, note that even if we had accepted this lag time, the x-axis calibration value would have extended to the same vertical pre-printed line, as per our rules above.

We signal data capture as follows [SEE USER MANUAL FOR MORE DETAIL]:

1. An initial *ZERO* must always be present, indicating that the Calipers have been zeroed (close the calipers so that they read zero, or very close to zero (e.g. 0.01) but do not press the ZERO button!);

2. The initial *ZERO* is always followed by a measurement in the range of *MAXVAL*, indicating the initial Y-axis calibration value (20–200 mmHg)

3. Then we enter the X-axis (time) calibration measurement, measured as described above;

4. Here follow measurements of *Systolic BP*. Each value represents an offset in millimeters from the 20 $\{$mmHg or min$^{-1}\}$ baseline, taken at 5 min intervals. The value should never be below *MINVAL*. Special values are discussed below.

5. After the last measurement, simply stop recording (Export to Excel) — our AutoIt script will take you back to the main GUI.

6. For the diastolic BP series, zero and Y-calibrate once more, but do not perform the X-calibration at any time. Just terminate after the last DBP reading;

7. For the heart rate series, zero and Y-calibrate, and acquire readings as usual *but* ...

8. Finally, as part of the heart rate series, repeat the zero, the Y-calibration, and as a last reading, repeat the X-axis calibration measurement (according to the previously specified rules).

The following data values are special:

- As already mentioned, a value of *ZERO* followed by a large number signals a zeroing—calibration pair.

- A double *ZERO* is entered for *every* missing 5-minute reading. The two values must be identical.
- A *ZERO* followed by a value between *ZERO* and *MINVAL* is used to signal the special case where a recorded reading is under 40, for example a diastolic BP under 40 or a heart rate under 40. This way we easily encode interpolated time values, but still preserve the information (such as it is) for the rare cases where extremely low values are recorded.
- A value greater than *ZERO* and less than the maximum value of α, signals an added reading between 5-minute intervals. After these two values we insert the Y offset of the point in question. Note than an offset of *ZERO* should never occur, as this is on the line, and a conventionally recorded datum.
- A value between *alpha* and *MINVAL* signals an error[3]
- A value of *MAXVAL* or greater signals that the measured value exceeds the maximum range! Note that this value should not be interpreted as an actual reading, but merely a signal that the range is exceeded, or 'out of the calibration range'.

We will soon explore some 'pseudocode' as an overview of our approach, and follow this by actual Perl code. Finally, we will describe how to automate parts of this process using AutoIt.

Rationale

The above approach might seem baroque, but has the following advantages:

1. We provide proof of the zero value and Y calibration values for the calipers at the start of every set of measurements;
2. At the start and end we confirm the duration of the record, and also provide a means of confirming the value of α, so that measurements taken between 'normal' time intervals (eg. at intervals of under five minutes) are reliably indicated;
3. Measurement is intuitive and quick. The person doing the measuring can simply measure offsets from baseline; where they encounter a measurement between normal time intervals, they simply measure the offset and then record the measurement as usual;

[3] Although what probably happened was the user forgot to signal a low value under 40, and simply measured the record. We will issue a warning here!

4. There is a system for recording unusually low values (under 40 mmHg or under 40 min^{-1}). This is robust in the sense that simply measuring this value will signal an error, unless the value is exceptionally low, in the range of alpha, where an 'interpolated' value will incorrectly be recorded, ultimately resulting in an error in the apparent duration of the record.

5. Measurements above the calibration range will be recorded as 'outside the range' rather than extrapolated values.

2.2 Algorithm (pseudocode)

We here develop some sketchy pseudocode that interprets data read in as a CSV file. Even if you are very adept at Perl, you might still wish to glace through this simple pseudocode before you move to the Perl code in the next section, as this code is very simply written.

Each CSV file has an initial header line that can be ignored followed by any number of lines until end of file. As this is DOS, each line ends in CR+LF (hexadecimal 0D, 0A). Each line is numbered sequentially followed by a comma and a single numeric value.

Main routine

Let's create a pseudocode representation of the process. The overall plan is to initialise certain variables, check the basic calibration measurements, and then process each data line in turn.

```
variables <- ('sbp', 'dbp', 'hr', 'spo2', 'etco2')
idx <- 0   # index into variables
SetInitialValues()
datumA <- ReadLine
t <- 0
if not datumA in ZERO -> fail(Bad initial zeroing)
StoreZero(t, datumA)
datumB <- ReadLine
MAXVAL, ALPHA, MINVAL <- TestYCal(datumB)
datumC <- ReadLine
DURATION <- TesttCal(datumC)
linecount <- 0
until end-of-file:
  ReadLine(datum);
  t <- ProcessDatum(t, datum); # t is an integer
end until
datumC <- ReadLine
TesttCal(datumC)   # final check of data.
```

```
                  # Might also compare result to DURATION?)
ReadFiO2andEtCO2()
PostProcess() or fail(Bad post-processing)
WriteAllData() or fail(Data write failed)
```

Initialisation

Here's the initialisation process. ALLDATA contains descriptions of data at any time, indexed by time as an integer.

```
routine SetInitialValues():
clear ALLDATA
clear ZEROES  # stack of zeroes
clear YCALS   # stack of Y calibration values
clear tCAL    # stack of time calibration values

# reference values:
CALA <- (59.0, 61.0)
CALB <- (56.3, 58.3)
CALC <- (83.8, 85.9)

ZERO-lo  <- -0.05;
ZERO-hi  <- +0.05;
NA       <- -1;   # signals 'NA'
UNCLEAR  <- -2;   # signals illegible datum
ZERO   <-(ZERO-lo, ZERO-hi)
end routine SetInitialValues
```

Utilities

Here's how we read a line:

```
routine ReadLine(datum; implicit:linecount):
  read-line;
  linecount ++;   # increment linecount
  read line-number;
  if (line-number != linecount) -> fail(Bad line number);
  read datum from line;
  return(datum)
end routine ReadLine
```

We require a Store routine:

```
routine Store (t,y):
  x <- ALLDATA[t]
  msg <- concatenate ("," variables[idx] "=")
  ALLDATA[t] <- concatenate (x msg y) # append value
end routine Store
```

We concatenate in a value, to sort out the potential (if infrequent) problem where two data values are stored against the same time. We might later test for and warn about this anomaly!

Similar are StoreZero and StoreYCal, and a smaller stack to store the few x-axis (time) calibrations we do:

```
routine StoreZero (x,y):
  push(ZEROES, x)
  push(ZEROES, y)
end routine StoreZero


routine StoreYCal (x,y):
  push(YCALS, x)
  push(YCALS, y)
end routine StoreYCal


routine StoretCal (x,y):
  push(tCALS, x)
  push(tCALS, y)
end routine StoretCal
```

Tests of calibration

We must be able to test Y calibration. At the same time, based on the Y calibration value we determine the type of record, and hence set the alpha value! We also return the MINVAL.

```
routine TestYCal (t, c):
  StoreYCal(t,c)
  tiny <- 1; # plusminus value for range
  alphaAlo <- 2.15
  alphaAhi <- 2.35
  alphaBlo <- 4.43
  alphaBhi <- 4.63
  alphaClo <- 3.11
  alphaChi <- 3.31
  minlo <- -tiny + t/10;
  minhi <-  tiny + t/10;
```

```
  case (c):
    value in CALA -> return (c-tiny, c+tiny,
                             alphaAlo, alphaAhi,
                             minlo, minhi)
    value in CALB -> return (c-tiny, c+tiny,
                             alphaBlo, alphaBhi,
                             minlo, minhi)
    value in CALC -> return (c-tiny, c+tiny,
                             alphaClo, alphaBhi,
                             minlo, minhi)
    default: fail(Bad calibration value)
end routine TestYCal


routine TesttCal (t, c):
  StoretCal(t,c)
  if (t < 5) or (t > 1000) -> fail(Bad time base) # arbitrary Nos.
end routine TesttCal
```

Main data processing

Here's the main data-processing and storing routine:

```
routine ProcessDatum (t, y)
  if y in ZERO ->
     y2 <- ReadLine
     if y2 in ZERO ->  # 0,0 signals datum NA
        Store(t, NA)
        return (t+5)
     if y2 in MAXVAL -> # 0, MAXVAL = end of run, recalibrate
        Recalibrate()
        idx <- idx +1  # global: signal new variable
        return (0)   # back to start time!
     if y2 > MAXVAL -> # meaningless
        fail (Bad Y calibration value)
     if y2 <= MINVAL ->  # special signal to store LOW values:
        Store(t, y2)
     default -> fail (Bad special datum value)

  if y in ALPHA ->
     y2 <- ReadLine
     ALPHAm <- (ALPHAh + ALPHAl)/2
     toff <- round( (5*y2)/ALPHAm )
     Store(t + toff, y2)
     return(t) # do NOT advance
```

```
  if y < MINVAL ->
     fail (Bad data value)

  if y >= MAXVAL ->
     MAXVALm <- (MAXVALh + MAXVALl)/2
     Store(t, MAXVALm)
     return(t+5)

  default ->
    Store (t, y)
    return (t+5);
```

FiO_2, $EtCO_2$

After determining the number of values to acquire, which is simply the DURATION measurement divided by alpha [check me], and identical to the maximum time value (noting that if the last time value is not on a 5-minute mark, the user *must* enter a 0,0), we acquire first FiO2 and then EtCO2 values (if present), allowing the user to enter NA values where appropriate. (What about 'oopses'?)

```
routine ReadFiO2andEtCO2()
  # here determine maximum index of ALLDATA, validate vs DURATION
  c <- int((DURATION+4)/5)
  t <- 0
  while (t <= c)
    sp <- acquire  # allow NA or integer 0--100
    Store (t, sp)
    t <- t+5
  end while
  t <- 0
  while (t <= c)
    et <- acquire # as above
    Store (t, et)
    t <- t+5
  end while
end routine ReadFiO2andEtCO2
```

Write data

All data should now have been stored in the single-dimensional array ALLDATA, indexed by time in minutes. (The first time is zero, signalling the start of data recording). There may be missing data for a particular time, and there is the possibility that two closely-recorded values are registered under the same time! We will next simply write all time data in order, as a CSV file. The format of the

output file is shown in Table 2. In order to output numeric values (and what about precision???) we need to make the appropriate conversions using the calibration values.

Time	*NBP-s*	*NBP-d*	*HR*	SpO_2	$ETCO_2$	*NBP-m*	*rawSBP*	*rawDBP*	*rawHR*
2008-12-14T10:23:00	120	70	72	99	NA	88	a	b	c
2008-12-14T10:28:00	123	66	72	NA	33	86	a	b	c

Table 2: Output table format (also CSV)

Note that in Table 2 we have renamed the systolic blood pressure ‘NBP-s’ and so forth. This is for subsequent compatibility (see code below).

```
routine WriteAllData()
  caly <- GetYCalValue(YCALS)
  for each value and time in ALLDATA
    WarnAndExcludeDuplicates(value)
    (rsbp, rdbp, rhr, spo2, etco2) <- ExtractData(value)
    sbp = rsbp*caly
    dbp = rdbp*caly
    hr = rhr*caly
    WriteCsvDatum(time, sbp, dbp, hr, spo2, etco2, rsbp, rdbp, rhr)
  end for
end routine WriteAllData
```

2.3 Perl code: raw data to CSV

This code is somewhat more complex than the simple schema just outlined. In Perl, it's probably wise to declare 'globals' in the main routine, so we don't have a separate Initialisation section.

2.3.1 Main routine

We start with the usual Perl shebang, and a whole lot of nasty global variables, not calculated to win friends.

Note that we do NOT at present create or check for the existence of the *log*, *rawdata* and *csv* sub-directories. Creation MUST be done manually. We also assume that the working directory is *not* the directory in which this program (*ez-captur.pl*) is located, but the main *ez* directory, containing the destination *csv* sub-directory.[4]

```
#!/usr/local/bin/perl -w
use POSIX qw(floor); # for date calculations

my ($DEBUG) = 2;     # 0 to turn off debugging,
                     #  1=overview, 2=detail, 3=nitpicking
my ($STORECOUNT)=1;  # also used for debugging
my ($LOGPRINT) = 1;  # print to log
my ($WARNINGS) = 0;  # see usage
my ($EZDIR) = '/ez/';
my $EXITCODE = 0;    # code on exiting Perl
  # exit code reports defects: 0:ok, 1:sbp, 2:dbp, 4:hr, 8:spo2, 16:etco2
  #
my $BASELINE = 20;   # (20mmHg or bpm)

my (@VARIABLES) = ('sbp', 'dbp', 'hr', 'spo2', 'etco2');
                     # note the order
my $IDX = 0;
my $MAXIDX = 3;
my $TIME = 0;
my $LINECOUNT = 1;   # will skip first line
my $SERIESCOUNT = 1; # doubled for each new series!

my ($RAWFILENAME) = $ARGV[0]; # source filename from command line
                              # NO suffix, NO path.
my ($TIMESTAMP)= $ARGV[1];    # timestamp "DD/MM/YYYY HH:MM"
my ($TRUEDURATN) = $ARGV[2];  # ADDED in v 0.37
my ($PARAMS) = $ARGV[3];      # eg "debug=3,logprint=0"
```

[4] This is clumsy and a bit nasty. Consider submitting the path to the destination directory as an extra command-line parameter to the Perl program!

```
if ($PARAMS =~ /debug=(\d)/ ) # command-line control!
   { $DEBUG = $1;
   };
if ($PARAMS =~ /logprint=(\d)/ )
   { $LOGPRINT = $1;
   };

&Print ("\n Opening file: <$RAWFILENAME> \
  with timestamp $TIMESTAMP, arguments $PARAMS", 0);
&OpenLog($LOGPRINT, $EZDIR, $RAWFILENAME, $TIMESTAMP); # start writing to (error/d

# HERE we sequentially read in _three_ files:
#  $RAWFILENAME-sbp.csv
#  $RAWFILENAME-dbp.csv
#  $RAWFILENAME-hr.csv
#  ... we then append these:

# 1. sbp
open(FILE, $EZDIR . "rawdata/$RAWFILENAME-sbp.csv") or &Die("Unable to open sbp fi
my (@INDATA) = <FILE>;
close(FILE);
# print LOGFILE "\n\nDebugging A: <@INDATA>\n\n";

# 2. dbp
open(FILE2, $EZDIR . "rawdata/$RAWFILENAME-dbp.csv") or &Die("Unable to open dbp f
my ($hdr);
   $hdr = <FILE2>;  # discard first line
my (@IN2) = <FILE2>;
close(FILE2);
# print LOGFILE "\n\nDebugging B: <@IN2>\n\n";

# 3. hr
open(FILE3, $EZDIR . "rawdata/$RAWFILENAME-hr.csv") or &Die("Unable to open hr fil
   $hdr = <FILE3>;  # discard first line
my (@IN3) = <FILE3>;
close(FILE3);
# print LOGFILE "\n\nDebugging C: <@IN3>\n\n";

(@INDATA) = (@INDATA, @IN2, @IN3); # concatenate
my @ALLDATA = ();
my @ZEROES = ();
my @YCALS = ();
my @tCAL = ();

# reference values:
my $CALAlo = 59.0;
my $CALAhi = 61.0;
my $CALBlo = 56.3;
```

```
my $CALBhi = 58.3;
my $CALClo = 83.8;
my $CALChi = 85.9;
my $ZEROlo = -0.02;  # 20 microns!
my $ZEROhi  = +0.02;
my (@CALA) = ($CALAlo, $CALAhi);
my (@CALB) = ($CALBlo, $CALBhi);
my (@CALC) = ($CALClo, $CALChi);
my (@ZERO) = ($ZEROlo, $ZEROhi);

my $NA = -1;
my $UNCLEAR = -2;

my ($datumA) = &ReadLine();
if (! &ISIN ($datumA, @ZERO))
   { &Die ("Bad initial zeroing, $datumA");
   };
&StoreZero($TIME, $datumA);

my ($datumB) = &ReadLine();
 &Print( "\n Debug: Y cal value is $datumB", 1 );

my ($MAXVALlo, $MAXVALhi,
    $ALPHAlo, $ALPHAhi,
    $MINVALlo, $MINVALhi) = &TestYCal($TIME, $datumB);
my (@MINVAL) =($MINVALlo,$MINVALhi);
my (@MAXVAL) =($MAXVALlo,$MAXVALhi);
my ($MAXVALm)=($MAXVALlo+$MAXVALhi)/2;

my ($datumC) = &ReadLine();
 &Print( "\n Debug: X cal value is $datumC", 1 );
my ($DURATION) = &TesttCal($TIME, $datumC);
 &Print ("\n Duration = $DURATION", 1);
my ($truealpha) = $datumC/int($DURATION/5);
my (@ALPHA)  =($ZEROhi, $truealpha+0.10);  # 0.10 is tolerance
my ($ALPHAm) =$truealpha;
 &Print ("\n True alpha = $truealpha\n", 1);
# here check derived $DURATION vs $TRUEDURATN:
# as $TRUEDURATN is in minutes, and $truealpha corresponds to 5 min,
# ($datumC/$truealpha) should equal int ( ($TRUEDURATN+4)/5 )
# but we can simply say:
if ( $DURATION/5  != int( ($TRUEDURATN+4)/5 ) )
   { &Print("\n **WARNING** duration mismatch: $DURATION: $TRUEDURATN", 0);
     $WARNINGS ++;
   };

while ($IDX < $MAXIDX)
  {  my ($d) = &ReadLine();
```

```
     $TIME = &ProcessDatum($TIME, $d);
  };
my ($datumC) = &ReadLine();
&TesttFinal($TIME, $datumC);

&ReadSpO2andETCO2($EZDIR, $RAWFILENAME, $DURATION);
&Print("\n END of data acquisition \n\n", 1);

&WriteAllData($TIMESTAMP);
if ($WARNINGS > 0)
   { &Print("\n ***NOTE*** There was/were $WARNINGS warning(s). \
 Please consult the log.", 0);
     $EXITCODE |= 32;
   };
 print("\n Finished!\n");
&CloseLog($LOGPRINT);
exit $EXITCODE;
```

A simple routine to check whether something (x) is within a range

```
sub ISIN
{ my ($x, @v);
     ($x, @v)=@_;
  if ( ($x >= $v[0]) && ($x <= $v[1]) )
     { &Print( "\n  Check: $x is between $v[0] and $v[1]", 3 );
       return(1);
     };
  &Print( "\n  Check: $x is NOT between $v[0] and $v[1]", 3 );
  return(0);
}
```

A debugging print routine:

```
sub Print
{ my ($p, $level); # level is debugging level.
        # 0=none, 1=overview, 2=detail, 3=picky
     ($p, $level)=@_;

  if ($DEBUG >= $level) # if level is 0, always print!
     {  if ($LOGPRINT)
           { print LOGFILE $p;
             return;
           };
        print $p; # console print
     };
}
```

2.3.2 Utilities

A death routine for fatal errors:

```
sub Die
{ my ($e);
     ($e)=@_;
  my ($l) = $LINECOUNT-1;
  &Print ("\n\n DIED: Data line = $l, \
    Index was $IDX \n Message: $e", 0);
  $EXITCODE += 128;   # 'fatal'
  &Print("\n\nExit code +128", 0);
  exit $EXITCODE;
  # caution. SEE: http://perldoc.perl.org/functions/exit.html
  # die "\n Fatal";
}
```

Here's how we read a line — we use the INDATA array previously shlurped in.

```
sub ReadLine
{ $_ = $INDATA[$LINECOUNT];
  $LINECOUNT ++;
  chomp; # get rid of cr/lf
  &Print( "\n  Check: L$LINECOUNT datum is <$_>", 2);
  if (! /^(\d+),(-?\d+\.?\d*)/ )
     { &Die ("\n Bad datum at line $LINECOUNT, value is <$_>");
     };
  my ($ln) = $1;
  my ($d) = $2;
  if ($ln != $LINECOUNT-1)
     { # &Print ("\n Bad line number $LINECOUNT, value is <$_>", 0);
       # $WARNINGS ++;
       # temporarily remmed out.
     };
  return($d)
}
```

We require a Store routine, where we append data to the relevant time within ALLDATA.

```
sub Store
{ my ($t, $y);
     ($t, $y)=@_;
  my ($a) = '';
  if (length $ALLDATA[$t] > 0)
     { $a = $ALLDATA[$t];
     } else
```

```
     { $a = "time=$t"; # start with time!
     };
  my ($v) = $VARIABLES[$IDX];
  $ALLDATA[$t] = "$a,$v=$y";
  &Print ( "\n >$STORECOUNT Store: $y at time $t, type is $v", 2 );
  $STORECOUNT ++;
}
```

We concatenate in a value, to sort out the potential (if infrequent) problem where two data values are stored against the same time. We might later test for and warn about this anomaly!

Similar are StoreZero and StoreYCal, and a smaller stack to store the few x-axis (time) calibrations we do:

```
sub StoreZero
{ my ($t, $z);
     ($t, $z)=@_;
  &Print( "\n Debug: ZERO is $z" , 3);
  $ZEROES[$t] = $z;
}

sub StoreYCal
{ my ($t, $y);
     ($t, $y)=@_;
  &Print( "\n Debug: Y cal is $y" , 3);
  # $YCALS[$t] = $y;
  push (@YCALS, $y); # discard $t
}

sub StoretCal
{ my ($t, $x);
     ($t, $x)=@_;
  &Print( "\n Debug: X cal is $x\n" , 3);
  $tCAL[$t] = $x;
}
```

2.3.3 Tests of calibration

We must be able to test Y calibration. At the same time, based on the Y calibration value we determine the type of record, and hence set the alpha value! We also return the MINVAL. TestYCal returns three pairs of values, corresponding to low and high values for maximum, alpha, and minimum values for one of three possible anaesthetic forms.

```
sub TestYCal
{ my ($t, $c);
```

```
     ($t, $c)=@_;

  &StoreYCal($t,$c);
  my ($tiny) = 1;
  my ($alphaAlo) = 2.25-0.10;
  my ($alphaAhi) = 2.25+0.10;
  my ($alphaBlo) = 4.43/2;   # Not 10 min. Must have 5 min.
  my ($alphaBhi) = 4.63/2;
  my ($alphaClo) = 3.11;
  my ($alphaChi) = 3.31;
  my ($minlo) = -$tiny + $t/10;
  my ($minhi) =  $tiny + $t/10;

  if ( &ISIN ($c, @CALA) )
     { &Print( "\n Debug: record type is A", 1 );
       return ( $c-$tiny, $c+$tiny,
                $alphaAlo, $alphaAhi,
                $minlo, $minhi);
     }
  elsif ( &ISIN ($c, @CALB) )
     { &Print( "\n Debug: record type is B", 1 );
       return ( $c-$tiny, $c+$tiny,
                $alphaBlo, $alphaBhi,
                $minlo, $minhi);
     }
  elsif ( &ISIN ($c, @CALC) )
     { &Print( "\n Debug: record type is C", 1 );
       return ( $c-$tiny, $c+$tiny,
                $alphaClo, $alphaChi,
                $minlo, $minhi);
     }
  else { &Die ("\n Bad Y calibration value $c");
       };
}
```

Check the x (time) axis value:

```
sub TesttCal
{ my ($t, $x);
     ($t, $x)=@_;
  &StoretCal($t,$x);
  # (? sanity checks here, e.g. > 10 min and < (?) 8 hours)

  # next, determine duration in minutes:
  my ($avgalpha) = ($ALPHAhi+$ALPHAlo)/2; # avg nominal alpha range
  my ($d) = 5 * int (($x + $avgalpha/5)/$avgalpha);
  return ($d);
}
```

In determining the duration, we add a little bit to $x in order to prevent a truncation error. We repeat a similar test at the end (the final datum):

```
sub TesttFinal
{ my ($t, $x);
     ($t, $x)=@_;
  &StoretCal($t,$x);

  # next, determine duration in minutes:
  my ($avgalpha) = ($ALPHAhi+$ALPHAlo)/2; # avg nominal alpha range
  my ($d) = 5 * int (($x + $avgalpha/5)/$avgalpha);
  if ($d != $DURATION)
     { &Print ("\n WARNING: initial ($DURATION) and \
          final duration ($d) don't match", 0);
       $WARNINGS ++;
       $EXITCODE |= 32; # bad x-duration match
       &Print("\n\nExit code +32", 0);
     };
}
```

2.3.4 Main data processing

Here's the main data-processing and storing routine:

```
sub ProcessDatum
{ my ($t, $y);
     ($t, $y)=@_;
  my ($y2, $y3);

  if (&ISIN ($y, @ZERO))
     { $y2 = &ReadLine();
       if (&ISIN ($y2, @ZERO))
          { &Print ("\n Storing NA value..", 2);
            &Store($t, $NA);
            return ($t+5);
          }
       elsif (&ISIN ($y2, @MAXVAL))
          { &Print ("\n Moving to next series..", 2);
            &NextSeries($t, $y, $y2);
            $SERIESCOUNT *= 2; # double
            return (0);
          }
       elsif ($y2 > $MAXVALhi)
          { &Die ("\n Bad calibration value $y2");
          }
       elsif ($y2 < $MINVALlo)
          { &Print ("\n Special low value..", 2);
            &Store($t, $y2); # special low value
```

```
            return ($t+5);
          }
       else { &Die ("\n Bad special datum value $y2");
            };
    }
  elsif (&ISIN ($y, @ALPHA))
    { &Print ("\n Special offset: $y ...", 2);
      my($toff) = int( (5*$y)/$ALPHAm ) ;
      # FOR NOW, TRUNCATE. SHOULD PROBABLY ROUND [FIX ME]
      $y2 = &ReadLine();
      &Store($t+$toff-5, $y2);
      # store offset value, -5 as is past PRIOR time!!
      return($t);  # do NOT advance
    }
  elsif ($y < $MINVALlo)
    { &Die ("\n Bad y data value $y");
    }
  elsif ($y > $MAXVALhi)
    { &Print ("\n Value above upper cal range..", 2);
      &Store($t, $MAXVALm); # signal above upper cal.
      return($t+5);
    }
  else { &Store($t, $y);
         return ($t+5);
       };
}
```

The movement between data series (e.g. from SBP to DBP) is so important, we define the function:

```
sub NextSeries
{ my ($t, $z, $y2);
     ($t, $z, $y2)=@_;
  &Recalibrate($t, $z, $y2);
  $IDX ++;
  $STORECOUNT = 1; # for debugging
  &Print( "\n\n\n DEBUG: transition($IDX) at line $LINECOUNT", 2 );
}
```

Here’s the subsidiary routine:

```
sub Recalibrate
{ my ($t, $z, $y2);
     ($t, $z, $y2)=@_;
  &StoreZero($t, $z);
  &StoreYCal($t,$y2);
  if (! &ISIN ($z, @ZERO))
     { &Die ("Repeat zeroing failed: $z"); # can happen??
```

```
     };
  if (! &ISIN ($y2, @MAXVAL) )
     { &Die ("Repeat Y calibration failed: $y2"); # can happen??
     };
  my ($tcorr) = $t-5;        # +5 was added by default
  if ($tcorr != $DURATION)
     { &Print ("\n WARNING: Durations don't match. \
         Baseline is $DURATION, this is $tcorr", 0);
       $WARNINGS ++;
       $EXITCODE |= $SERIESCOUNT;
       &Print("\n\nExit code +$SERIESCOUNT", 0);
       # durations don't match for given epoch
     }; # hmm.
}
```

2.3.5 FiO_2, $EtCO_2$

After determining the number of values to acquire, which is simply the DURATION measurement divided by 5, we acquire first SpO2 and then EtCO2 values (if present), allowing the user to enter NA values where appropriate.

```
sub ReadSpO2andETCO2
{ my ($EZDIR, $RAWFILENAME, $DURATION);
     ($EZDIR, $RAWFILENAME, $DURATION)=@_;

  my ($numobs) = 1+int( ($DURATION+4)/5 );
  &Print ("\n\n OBS COUNT = $numobs ($DURATION)", 1);

  # 1. spo2:
  $IDX = 3;  # signal SpO2
  open(FILE, $EZDIR . "rawdata/$RAWFILENAME-spo2.csv") or
     &Die("Unable to open spo2 file rawdata/$RAWFILENAME");
  my ($hdr);
  $hdr = <FILE>;
  my (@SPO2) = <FILE>;  # first line will be skipped
  close(FILE);
  my ($sz) = 1+$#SPO2;
  if ($sz != $numobs)  # not +1 as $# balances header line!
     { $EXITCODE |= 8;  # signal problem with spo2 data(count)
                        # or use $SERIESCOUNT ??
       &Print("\n\nExit code +8, observations=$numobs ($sz)", 0);
     };
  &XferData(@SPO2);

  # 2. etco2:
  $IDX = 4;  # signal ETCO2
  open(FILE, $EZDIR . "rawdata/$RAWFILENAME-etco2.csv") or
     &Die("Unable to open etco2 file rawdata/$RAWFILENAME");
```

```
  $hdr = <FILE>;
  my (@ETCO2) = <FILE>;
  close(FILE);
  my ($sz) = 1+$#SPO2;
  if ($sz != $numobs)
     { $EXITCODE |= 16;  # signal problem with etco2 data
                         # or use $SERIESCOUNT ??
       &Print("\n\nExit code +16, observations=$numobs ($sz)", 0);
     };
  &XferData(@ETCO2);
}

sub XferData
{ my (@X);
     (@X)=@_;
  my ($d, $c, $t, $inp);
  $c = 0;
  foreach $d (@X)
    { chomp($d);
      if (length $d > 1) # ignore blank lines
         { $t = 5*$c;   # time in minutes = 5* count
           if ($d =~ /,-1/)
              { &Store($t, -1); # signals 'NA'
              }
           elsif ($d !~ /^\w*(\d+),(\d+\.?\d*)\w*/ )
              { &Die ("Bad data: line $c value <$d>");
              } else
              { $inp = $2;
                if ($1 != $c)
                   { # &Print( "\n Warning! time mismatch: $c ($1)", 0);  # TEMP F
                     # $WARNINGS ++;
                   };
                &Store($t, $inp); # store value
              };
           $c ++;  # bump count
         };
     }
}
```

2.3.6 Write data

Note that in the following, we should perhaps round the output data to four significant digits, but at present we don't adulterate them.

```
sub WriteAllData
{ my ($TIMESTAMP);
     ($TIMESTAMP)=@_;
```

```
  my($d, $c);
  # 0. for starters, simply print each datum to the log:
  $c = 1;
  if ($DEBUG)
     { &Print ("\n Debug: raw data", 1);
       foreach $d (@ALLDATA)
         {
          if (length $d > 0)
             { &Print ("\n >$c: $d", 1);
               $c ++;
             };
         };
     };

  # 1. get a solid Y calibration value
  #     (what about subtracting avg(zeroes) ???)
  my ($ycal) = &GetYCal();

  # 2. Pull out data values using tags:
  # first, open file for writing:
  open (OUTFILE, ">csv/$RAWFILENAME-m.csv") or &Die (
     "*CRASH* Could not open OUTPUT FILE: csv\out$RAWFILENAME :$!\n");
     # -m suffix indicates it's a manual file ( -s is SAFERsleep)
  print OUTFILE
       "Time,NBP-s,NBP-d,HR,SpO2,ETCO2,NBP-m,rawSBP,rawDBP,rawHR";
  # names correspond to SAFERsleep .DAT names. NBP-m is calculated
  # below

  foreach $d (@ALLDATA)
    {
      if (length $d > 0)
         { &Print ("\n\n Extracting: <$d>", 3);
           if ($d !~ /,$/ ) # if no terminal comma
              { $d = "$d,"; # ensure terminal comma
              };
           &PullLine($d, $ycal, $TIMESTAMP);
         };
    };
  close OUTFILE;
};
```

Here's the routine to extract all data for one ALLDATA line, and write to output. We first extract the data to an array, v. Because we also wish to record the mean blood pressure (a value calculated from the systolic and diastolic), and the raw values, we introduce the variable $NASTYOFFSET, which is the offset in v of the mean pressure. Raw values are stored sequentially to the right of this

offset.[5]

```
sub PullLine
{ my ($d, $ycal, $TIMESTAMP);
     ($d, $ycal, $TIMESTAMP) = @_;

  my ($time);
  my ($NASTYOFFSET) = 5; # use to offset raw datum!
  # at NASTYOFFSET we have NBP-m, followed by rawSBP,rawDBP,rawHR

  if ($d !~ /time=(\d+),(.*)/)
     { &Die ("\n Bad timestamp in line: <$d>");
     };
  $time = $1;
  $d = $2;

  my ($target);
  my (@v);
  my ($i) = 0;
  my ($val) = '';
  my ($NIBs, $NIBd, $NIBm);
  foreach $target (@VARIABLES)
     { ($val, $d) = &PullDatum ($target, $d);
       #### here pull out values for later mean BP determination:
       if ($target eq 'sbp')
          { $NIBs = $val;
          }
       elsif ($target eq 'dbp')
          { $NIBd = $val;
          }
       #### end mean values.
       &Print ("\n Datum: $val leaves<$d>,i=$i target=$target",3);
       if ($i < 3) # for the first three values:
          { $v[$i+$NASTYOFFSET+1] = $val; # keep raw copy
            if ($val >= 0)
               { $val *= $ycal;
                 $val += $BASELINE; # add 20
               };
          };
       if ($val < 0)
          { $val = 'NA'; # only option, for now.
          };
       $v[$i] = $val;
       $i ++;
     };
  # HERE OBTAIN THE MEAN BLOOD PRESSURE:
    if ( ($NIBs eq 'NA')
```

[5] We might even store the relative time offset, but don't bother.

```
        ||($NIBd eq 'NA')
        )
        { $NIBm = 'NA';
        } else # m=d+1/3 of pulse pressure:
        { $NIBm = $NIBd + (($NIBs - $NIBd)/3);
          $NIBm *= $ycal;
          $NIBm += $BASELINE; # add 20.
        };
    $v[$NASTYOFFSET] = $NIBm; # store value in correct column
  # END OBTAIN MEAN BP.

  if ( length $d > 1)  # 1 allows for final comma?? [HMM CHECK ME]
     { &Print ("\n Warning: extraneous data in line <$d>", 0);
       $WARNINGS ++;
     };

  my ($p);
  my ($modtime) = &KiwiAddMinutes ($TIMESTAMP, $time);
  $modtime =~ s/ /T/; # SAFERsleep format has T IPO " "
  print OUTFILE "\n$modtime";
  foreach $p (@v)
    { print OUTFILE ",$p";
    };
};
```

Here's a simple routine that pulls out a datum, given the name. It also returns the original string, with the entire specification for that datum seamlessly excised! Note that for the routine to work reliably, the string should end with a comma.

```
sub PullDatum
{
  my ($t, $orig);      # $t is target 'string=etc,' to search for
     ($t, $orig) =@_;  # $orig = original string to search IN
  my ($dat, $modif);
  $dat = '';
  $modif = ''; # keep Perl happy

  if ( $orig =~ /(.*)$t=(.*?),(.*)/ )
     { $modif = "$1$3";
       $dat = $2;
     } else
     { $modif = $orig;
       $dat = -1;
     };

  return ($dat, $modif);
};
```

To obtain the Y calibration value, we take all the values in YCALS, and simply average them. [HMM, CHECK ME]. The Y calibration value in mm represents 200 units, so to get a conversion factor, divide 200 by the average value.

```
sub GetYCal
{
  my($d, $tot, $n);
  $tot = 0.0;
  $n = 0;
  foreach $d (@YCALS)
    { if (length $d > 0)  # should always succeed.
         { $tot += $d;
           $n ++;
         };
    };
  if ($n < 1) # most strange
     { &Die ("Odd! No Y calibration values"); # can this occur??
     };
  &Print("\n Y calibration total=$tot, count=$n", 1);
  return ( 200.0/($tot/$n) );
};
```

We also need time-manipulation routines. Julian returns a Julian day (float) given year, month, day etc. Gregorian does the reverse.

Note that IDAS doesn't seem to address issues of daylight saving (i.e. the hour that is lost, and the groundhog hour).

```
sub KiwiAddMinutes  # input format is DD/MM/YYYY HH:MM (aagh)
                    # output as international date
{ my ($ts, $min);
     ($ts, $min)=@_;

  my ($yy, $mm, $dd, $hr, $mi, $ss);
  if (  $ts !~ /(\d{2})\/(\d{2})\/(\d{4}) (\d{2}):(\d{2})/  )
     {
       &Die( "Bad Date $ts (Kiwi format DD/MM/YYYY HH:MM)"  );
     };
  ($dd, $mm, $yy, $hr, $mi) = ($1, $2, $3, $4, $5);
  my ($gooddate) = "$yy-$mm-$dd $hr:$mi:00";
  &Print ("\n Input date <$gooddate>, adding $min min", 3);
  $f = &Julian($yy, $mm, $dd, $hr, $mi, 0, 0);
  $f += $min/(24*60);  # minutes as a day fraction
  &Print ("\n Julian: <$f>", 3);
  ($yy, $mm, $dd, $hr, $mi, $ss) = &Gregorian($f);
  &Print ("\n Gregorian: $yy $mm $dd $hr $mi $ss",3 );
  return ( "$yy-$mm-$dd $hr:$mi:00" );
}
```

```
sub Julian
{ my ($fy, $fm, $fd, $fh, $fmi, $fs, $ff);
     ($fy, $fm, $fd, $fh, $fmi, $fs, $ff)=@_;

  my ($f);
  $f= 367*$fy - int(7*($fy+int(($fm+9)/12))/4)
              - int(3*(int(($fy+($fm-9)/7)/100)+1)/4)
              + int(275*$fm/9)+$fd+1721028.5
              + ($fh + ($fmi + ($fs+ "0.$ff")/60)/60)/24;
  return $f;
}

sub Gregorian
{ my ($jd);
     ($jd)=@_;

  my ($EPSILON) = 0.000001;
  $jd += $EPSILON;

  my ($Z, $R, $G, $A, $B, $C);
  my ($year, $month, $day);

  $Z = floor($jd - 1721118.5);
  $R = $jd - 1721118.5 - $Z;
  $G = $Z - 0.25;
  $A = floor($G / 36524.25);
  $B = $A - floor($A / 4);
  $year = floor(($B+$G) / 365.25);
  $C = $B + $Z - floor(365.25 * $year);
  $month = int((5 * $C + 456) / 153);
  $day = $C - int((153 * $month - 457) / 5) + $R;
  if ($month > 12)
     { $year = $year + 1;
       $month = $month - 12;
     };

    my ($gd) = 0.5 + $jd - int($jd);
     if ($gd > 1)   # Julian starts at midday.
        { $gd -= 1;
        };

    my($gh, $gmi, $gs);
            $gh = $gd; # clumsy
            $gh *= 24;
            $gmi = $gh;
            $gh = int($gh);
            $gmi -= $gh;
```

```
            $gmi *= 60;
            $gs = $gmi;
            $gmi = int($gmi);
            $gs -= $gmi;
            $gs *= 60;

  return ($year, &DoubleDigit($month),
                  &DoubleDigit(int($day)),
                  &DoubleDigit($gh),
                  &DoubleDigit($gmi),
                  &DoubleDigit(int($gs))
          );
}

sub DoubleDigit
{
   my ($i);
      ($i) = @_;

   if (length $i > 1)
      { return $i;
      };
   return "0$i"; # concatenate.
}
```

2.3.7 Error log

We need the error log for ‘forensic’ purposes, and debugging. All debugging is written here, unless logging is turned off, in which case it’s written to the console.

```
sub OpenLog
{ my ($islog, $EZDIR, $RAWFILENAME, $TIMESTAMP);
     ($islog, $EZDIR, $RAWFILENAME, $TIMESTAMP)=@_;
  if (! $islog) { return; }; # if not logging.
  my $TODAY = &GetLocalTime();
  my $logfile= $EZDIR . "log/$RAWFILENAME-capt-$TODAY.LOG";
  open (LOGFILE,
    ">$logfile") or die "*CRASH* Could not open LOG $logfile :$!\n";
  print "\n Writing to log file: $logfile";
}

sub CloseLog
{ my ($islog);
     ($islog)=@_;
  if ($islog)
     { close LOGFILE;
     };
}
```

Here are the subsidiary routines:

```
sub GetLocalTime
{ my ($sec, $min, $hour,
      $mday, $mon, $year,
      $wday, $yday, $isdst);
  ($sec, $min, $hour,
      $mday, $mon, $year,
      $wday, $yday, $isdst) = CORE::localtime(time);
  $year += 1900;      #fix y2k.
  $sec  = &DoubleDigit($sec );
  $min  = &DoubleDigit($min );
  $hour = &DoubleDigit($hour);
  $mday = &DoubleDigit($mday);
  $mon  = &DoubleDigit($mon );
  $mon ++;            #january is zero!
  return ("$year$mon$mday-$hour$min$sec");
}

sub DoubleDigit
{
   my ($i);
      ($i) = @_;
   if (length $i > 1)
      { return $i;
      };
   return "0$i"; # concatenate.
}
```

2.4 A comprehensive GUI

We here describe an AutoIt GUI that allows the user to perform all the necessary capture and format translation, both from SAFERsleep automated records, and from manual (caliper) data.

Here's the initialisation code:

```
#include <GUIConstants.au3>
#include <WindowsConstants.au3>
 $userdll = DllOpen("user32.dll") ; for frequent use of dll
 ;; FAKEDCMS suppresses activation of DCMS, gives time to
 ;; open Excel and paste data into Excel. [nasty]
 $FAKEDCMS = 0;  # use numeric (seconds) for Excel pause!!
 ;; $FAKEDCMS = 30;  # use numeric (seconds) for Excel pause!!

;; ----------------useful constants:------------------- ;;
Const $ONTOP = 0x40000;

Const $EZ_CASENUMBER = 1;
Const $EZ_CASEDATE = 2;
Const $EZ_CASETIME = 3;
Const $EZ_VALIDFLAGS = 4;
Const $EZ_ISLISTED = 5;
Const $EZ_CASEEND = 6;
Const $EZ_SURGERY = 7;
Const $EZ_DESCRIPTION = 8;
Const $EZ_DURATION = 9;

Const $EZ_MAXLINES = 9;  ;; counts the above constants

 ; here, request patient ID:
 $more = 1
 While $more = 1
   $nad = XGetFileAndTimestamp();
   $FILENAME = $nad[$EZ_CASENUMBER];
   $FILENAME = StringUpper($FILENAME);
   $DATETIME = $nad[$EZ_CASEDATE] & " " & $nad[$EZ_CASETIME];
   $VALIDITY = $nad[$EZ_VALIDFLAGS];
   $ISLISTED = $nad[$EZ_ISLISTED];
   $DURATION = $nad[$EZ_DURATION];
   $more = MainInterface($userdll, $FILENAME, $DATETIME, _
                         $VALIDITY, $ISLISTED, $DURATION);
 Wend

 DllClose($userdll);   ;; tidy up
 Exit;
```

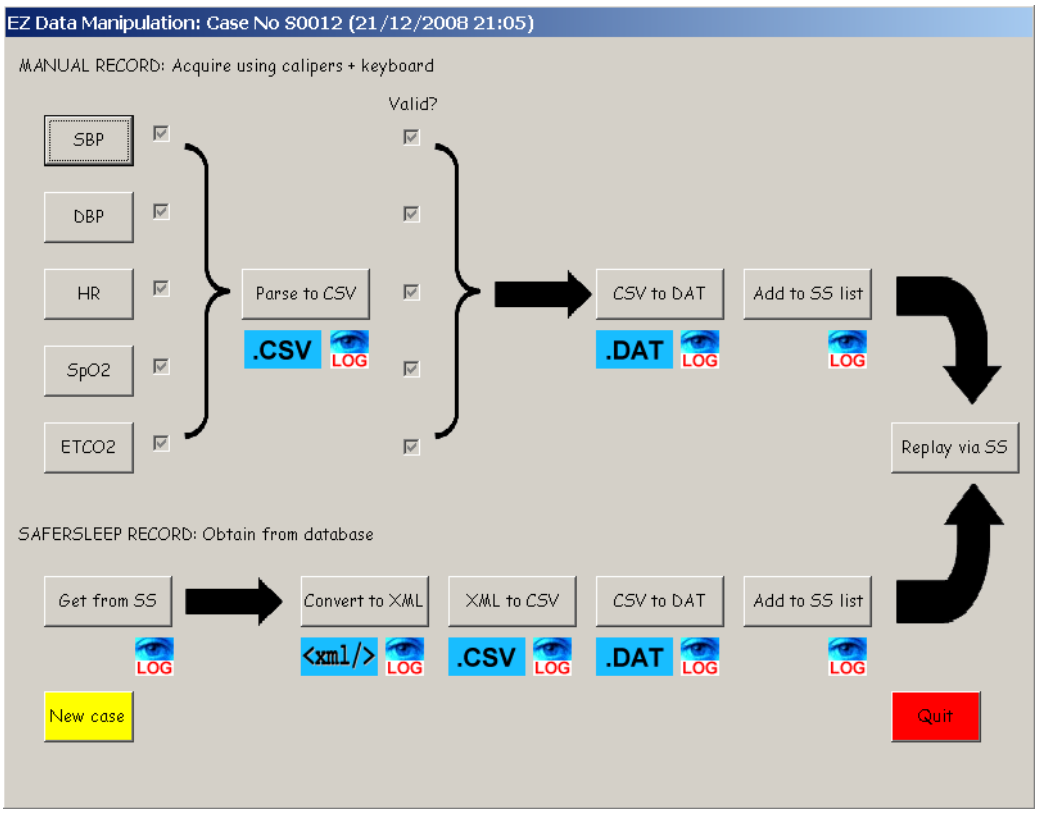

Figure 2: Ez Graphical user interface

A sample screenshot of the main page is shown in Fig. 2, and the AutoIt code that creates it is as follows:

```
;; MainInterface: make a comprehensive GUI:
Func MainInterface($userdll, $FILENAME, $DATETIME, _
                   $GOODITEMS, $ISLISTED, $DURATION)
  $hskip = 115;
  $pitch = 60;
  $h = 40;
  $w = 70;
  $bigw = 100;
  $labw = 200;
  $x = 30;
  $EZPATH = "\ez\";

  ;; the following are flags that allow enabling of various buttons:
  $FIVEITEMS = 0;  ;; bits set = data ARE in! contrast with:
  $mISCSV = 0;
  $mISDAT = 0;
  $mISSS = 0;   ;; is manual name in SAFERsleep submission list
```

```
  $sISIAR = 0;  ;; does IAR exist?
  $sISXML = 0;
  $sISCSV = 0;
  $sISDAT = 0;
  $sISSS = 0;   ;; is automatic name in SS submission list?

; make the GUI:
Dim $mygui = GuiCreate("EZ Data Manipulation: Case No " &  _
       $FILENAME & ' (' & $DATETIME & ')' , _
       800, 600, 30, 10, $WS_CAPTION, $WS_EX_TOPMOST);
  ; FONT:
  $font="Comic Sans MS"           ;
  GUISetFont (8, 400, 1, $font)  ; Size 8 font

; labels:
GuiCtrlCreateLabel("MANUAL RECORD: Acquire using calipers + keyboard", _
                   10,12,2*$labw,$h);
GuiCtrlCreateLabel("SAFERSLEEP RECORD: Obtain from database", _
                   10,6.3*$pitch,2*$labw,$h);
GuiCtrlCreateLabel("Valid?", 2.6*$hskip,0.7*$pitch, $labw,$h);

; buttons:
Dim $bSBP  = GuiCtrlCreateButton("SBP",  $x, 1*$pitch, $w, $h);
Dim $bDBP  = GuiCtrlCreateButton("DBP",  $x, 2*$pitch, $w, $h);
Dim $bHR   = GuiCtrlCreateButton("HR",   $x, 3*$pitch, $w, $h);
Dim $bSpO2 = GuiCtrlCreateButton("SpO2", $x, 4*$pitch, $w, $h);
Dim $bETCO2= GuiCtrlCreateButton("ETCO2",$x, 5*$pitch, $w, $h);

Dim $mCSV  = GuiCtrlCreateButton("Parse to CSV",   1.6*$hskip, 3*$pitch, $bigw,$h)
Dim $mDAT  = GuiCtrlCreateButton("CSV to DAT",     4*$hskip, 3*$pitch, $bigw, $h);
Dim $mSS   = GuiCtrlCreateButton("Add to SS list", 5*$hskip, 3*$pitch, $bigw, $h);

Dim $sGET  = GuiCtrlCreateButton("Get from SS",    $x,       7*$pitch, $bigw, $h);
Dim $sXML  = GuiCtrlCreateButton("Convert to XML", 2*$hskip, 7*$pitch, $bigw, $h);
Dim $sCSV  = GuiCtrlCreateButton("XML to CSV",     3*$hskip, 7*$pitch, $bigw, $h);
Dim $sDAT  = GuiCtrlCreateButton("CSV to DAT",     4*$hskip, 7*$pitch, $bigw, $h);
Dim $sSS   = GuiCtrlCreateButton("Add to SS list", 5*$hskip, 7*$pitch, $bigw, $h);

Dim $REPL  = GuiCtrlCreateButton("Replay via SS",  6*$hskip, 5*$pitch, $bigw, $h);

Dim $bQuit = GuiCtrlCreateButton("Quit", 6*$hskip, 8.5*$pitch, $w, $h);
GUICtrlSetBkColor ( $bQuit, 0xFF0000 )  ;; red button
Dim $bNew =  GuiCtrlCreateButton("New case", $x, 8.5*$pitch, $w, $h);
GUICtrlSetBkColor ( $bNew, 0xFFFF00 )  ;; yellow button

; tick-boxes:
Dim $xSBP  = GUICtrlCreateCheckbox("", 1*$hskip, 1*$pitch);
Dim $xDBP  = GUICtrlCreateCheckbox("", 1*$hskip, 2*$pitch);
```

```
Dim $xHR   = GUICtrlCreateCheckbox("", 1*$hskip, 3*$pitch);
Dim $xSpO2 = GUICtrlCreateCheckbox("", 1*$hskip, 4*$pitch);
Dim $xETCO2= GUICtrlCreateCheckbox("", 1*$hskip, 5*$pitch);

Dim $vSBP  = GUICtrlCreateCheckbox("", 2.7*$hskip, 1*$pitch);
Dim $vDBP  = GUICtrlCreateCheckbox("", 2.7*$hskip, 2*$pitch);
Dim $vHR   = GUICtrlCreateCheckbox("", 2.7*$hskip, 3*$pitch);
Dim $vSpO2 = GUICtrlCreateCheckbox("", 2.7*$hskip, 4*$pitch);
Dim $vETCO2= GUICtrlCreateCheckbox("", 2.7*$hskip, 5*$pitch);
# and set the Valid flags
SetGoodItems($GOODITEMS, $vSBP, $vDBP, $vHR, $vSpO2, $vETCO2);

; graphics:
;  arrow from parse/valid to CSV2DAT
GUICtrlCreatePic ("\ez\images\arrow.gif", 3.3*$hskip, 3*$pitch, 80, 40);
;  arrow from get from SS to IAR2XML
GUICtrlCreatePic ("\ez\images\arrow.gif", 1.2*$hskip, 7*$pitch, 80, 40);
; down arrow from add to replay
GUICtrlCreatePic ("\ez\images\arrowdn.gif", 6*$hskip, 3*$pitch, 90, 110);
; up arrow from add to replay
GUICtrlCreatePic ("\ez\images\arrowup.gif", 6*$hskip, 5.8*$pitch, 90, 110);

; xml
Dim $vwXML = GUICtrlCreatePic ("\ez\images\seexml.gif", 2*$hskip, 7.8*$pitch, 60,
; csv
Dim $vwCSV = GUICtrlCreatePic ("\ez\images\seecsv.gif", 1.62*$hskip, 3.8*$pitch, 6
Dim $vwCSV2 = GUICtrlCreatePic ("\ez\images\seecsv.gif", 3*$hskip, 7.8*$pitch, 60,
; dat
Dim $vwDAT = GUICtrlCreatePic ("\ez\images\seedat.gif", 4*$hskip, 3.8*$pitch, 60,
Dim $vwDAT2 = GUICtrlCreatePic ("\ez\images\seedat.gif", 4*$hskip, 7.8*$pitch, 60,
; sslist.txt
Dim $vwSS = GUICtrlCreatePic ("\ez\images\sslist.gif", 5.4*$hskip, 5.1*$pitch, 60,

; right brace for SBP etc
GUICtrlCreatePic ("\ez\images\rbrace.gif", 1.2*$hskip, 1.3*$pitch, 40, 4*$pitch);
; right brace for valid
GUICtrlCreatePic ("\ez\images\rbrace.gif", 2.9*$hskip, 1.3*$pitch, 40, 4*$pitch);

; special graphics: view Log (clickable?!)
Dim $imCSV = GUICtrlCreatePic ("\ez\images\Ilog.gif", 2.2*$hskip, 3.8*$pitch, 30,
Dim $imDAT = GUICtrlCreatePic ("\ez\images\Ilog.gif", 4.58*$hskip, 3.8*$pitch, 30,
Dim $imADD = GUICtrlCreatePic ("\ez\images\Ilog.gif", 5.58*$hskip, 3.8*$pitch, 30,

Dim $isGET = GUICtrlCreatePic ("\ez\images\Ilog.gif", 0.88*$hskip, 7.8*$pitch, 30,
Dim $isXML = GUICtrlCreatePic ("\ez\images\Ilog.gif", 2.58*$hskip, 7.8*$pitch, 30,
Dim $isCSV = GUICtrlCreatePic ("\ez\images\Ilog.gif", 3.58*$hskip, 7.8*$pitch, 30,
Dim $isDAT = GUICtrlCreatePic ("\ez\images\Ilog.gif", 4.58*$hskip, 7.8*$pitch, 30,
Dim $isADD = GUICtrlCreatePic ("\ez\images\Ilog.gif", 5.58*$hskip, 7.8*$pitch, 30,
```

```
;; check for defects: (otherwise will behave VERY oddly)
If ($vwXML = 0) OR ($vwCSV = 0) OR ($vwDAT = 0) _
   OR ($vwSS = 0) OR ($imCSV = 0) Then
  MsgBox( $ONTOP, 'Error', 'Missing images in \ez\images. Fix me (Bye)!' );
  Exit
EndIf


;; here set control states at start: (ugh)
;; first, tick-boxes are always greyed:
myDisable ($xSBP  );
myDisable ($xDBP  );
myDisable ($xHR   );
myDisable ($xSpO2 );
myDisable ($xETCO2);
; then check if raw data exist, if so, tick boxes:
$FIVEITEMS = CheckRaw($xSBP, $xDBP, $xHR, $xSpO2, $xETCO2, $FILENAME);

myDisable ($vSBP  );
myDisable ($vDBP  );
myDisable ($vHR   );
myDisable ($vSpO2 );
myDisable ($vETCO2);

;; secondly,
; Parse to CSV is only enabled when all 5 inputs are in!
If ($FIVEITEMS <> 31) Then ;; if bits 0--4 are not set
   myDisable ($mCSV );
EndIf

; CSV to DAT is enabled when all processing was valid!
myDisable ($mDAT );
TestmDAT ($mDAT, $EZPATH, $FILENAME, '-m'); ;; FIX THIS -> TestFile..
;; [NNNNNNNNNNNOOOOOOOOOO. Must check on status of Perl submission [FIX ME]]

; Add to SS list (manual) is valid when CSV to DAT succeeded
myDisable ($mSS );
TestFile($mSS, $EZPATH & 'dat/', $FILENAME, '-m.dat');

;; Replay via SS is only valid when both ADD buttons succeeded (Hmm?)
myDisable ($REPL );
 If $ISLISTED = 3 Then   ;; if both flags set
  myEnable($REPL);
 EndIf  ;; or make this a rtn.

;; Convert to XML is only valid once get from sS succeeded
myDisable ($sXML );
```

```
TestFile($sXML, $EZPATH & 'iar/', $FILENAME, '.iar');

; XML to cSV is valid once convert is ok
myDisable ($sCSV );
TestFile($sCSV, $EZPATH & 'xml/', $FILENAME, '.xml');

; ss CSV to DAT is valid once CSV made
myDisable ($sDAT );
TestFile($sDAT, $EZPATH & 'csv/', $FILENAME, '-s.csv');

; Add to sS list (ss) is valid once DAT made.
myDisable ($sSS );
TestFile($sSS, $EZPATH & 'dat/', $FILENAME, '-s.dat');

;; test for dud buttons:
  If $mCSV = 0 Then
   MsgBox($ONTOP, 'Error', 'Bad button: mCSV');
  EndIf

; GUI MESSAGE LOOP
GuiSetState()  ; display the GUI
While 1
  Dim $msg = GuiGetMsg();

  If $FIVEITEMS = 31 Then ;; if all 5 acquired
       myEnable ($mCSV );  ;; ugh.
  EndIf

Select  ;; START GIANT SELECT STATEMENT.
  Case $msg = $bQuit
    GUIDelete($mygui);
    Return(0);  ;; 0 = quit

  Case $msg = $bNew
    GUIDelete($mygui);
    Return(1);  ;; 1 = resume

  Case $msg = $bSBP
     $overw = 1;
     If IsRawFile('sbp', $FILENAME) = 1 Then
       $overw = MsgBox($ONTOP+1, "Warning", _
                  "Raw SBP data already captured! Overwrite?");
       If $overw = 1 Then  ;; (cancel=2) create single backup:
         $fnm = $EZPATH & 'rawdata\' & $FILENAME & '-sbp.csv';
         FileCopy ($fnm, $fnm & ".BAK", 1); ;;overwrite if exists
       EndIf
     EndIf
     If $overw = 1 Then
```

```
      GuiSetState(@SW_MINIMIZE);
      GetSequence ('sbp', $FILENAME, $DATETIME, $FAKEDCMS)
      myCheck($xSBP);
      $FIVEITEMS = BitOR($FIVEITEMS, 1); #say 'data acquired'
      GuiSetState(@SW_RESTORE);
    EndIf

  Case $msg = $bDBP
    $overw = 1;
    If IsRawFile('dbp', $FILENAME) = 1 Then
      $overw = MsgBox($ONTOP+1, "Warning", _
                 "Raw DBP data already captured! Overwrite?");
      If $overw = 1 Then  ;; (cancel=2) create single backup:
        $fnm = $EZPATH & 'rawdata\' & $FILENAME & '-dbp.csv';
        FileCopy ($fnm, $fnm & ".BAK", 1); ;;overwrite if exists
      EndIf
    EndIf
    If $overw = 1 Then
      GuiSetState(@SW_MINIMIZE);
      GetSequence ('dbp', $FILENAME, $DATETIME, $FAKEDCMS)
      myCheck($xDBP);
      $FIVEITEMS = BitOR($FIVEITEMS, 2);
      GuiSetState(@SW_RESTORE);
    EndIf

  Case $msg = $bHR
    $overw = 1;
    If IsRawFile('hr', $FILENAME) = 1 Then
      $overw = MsgBox($ONTOP+1, "Warning", _
                 "Raw HR data already captured! Overwrite?");
      If $overw = 1 Then  ;; (cancel=2) create single backup:
        $fnm = $EZPATH & 'rawdata\' & $FILENAME & '-hr.csv';
        FileCopy ($fnm, $fnm & ".BAK", 1); ;;overwrite if exists
      EndIf
    EndIf
    If $overw = 1 Then
      GuiSetState(@SW_MINIMIZE);
      GetSequence ('hr', $FILENAME, $DATETIME, $FAKEDCMS)
      myCheck($xHR);
      $FIVEITEMS = BitOR($FIVEITEMS, 4);
      GuiSetState(@SW_RESTORE);
    EndIf

  Case $msg = $bSpO2
    $overw = 1;
    If IsRawFile('spo2', $FILENAME) = 1 Then
      $overw = MsgBox($ONTOP+1, "Warning", _
                 "Raw SpO2 data already captured! Overwrite?");
```

```
      If $overw = 1 Then  ;; (cancel=2) create single backup:
        $fnm = $EZPATH & 'rawdata\' & $FILENAME & '-spo2.csv';
        FileCopy ($fnm, $fnm & ".BAK", 1); ;;overwrite if exists
      EndIf
    EndIf
    If $overw = 1 Then
      GuiSetState(@SW_MINIMIZE);
      If GetKbSequence ('spo2', $FILENAME, $DATETIME, $DURATION) = 1 Then
        myCheck($xSpO2);
        $FIVEITEMS = BitOR($FIVEITEMS, 8);
      EndIf
      GuiSetState(@SW_RESTORE);
    EndIf

  Case $msg = $bETCO2
    $overw = 1;
    If IsRawFile('etco2', $FILENAME) = 1 Then
      $overw = MsgBox($ONTOP+1, "Warning", _
                "Raw ETCO2 data already captured! Overwrite?");
      If $overw = 1 Then  ;; (cancel=2) create single backup:
        $fnm = $EZPATH & 'rawdata\' & $FILENAME & '-etco2.csv';
        FileCopy ($fnm, $fnm & ".BAK", 1); ;;overwrite if exists
      EndIf
    EndIf
    If $overw = 1 Then
       GuiSetState(@SW_MINIMIZE);
       If GetKbSequence ('etco2', $FILENAME, $DATETIME, $DURATION) = 1 Then
         myCheck($xETCO2);
         $FIVEITEMS = BitOR($FIVEITEMS, 16);
       EndIf
       GuiSetState(@SW_RESTORE);
    EndIf

  Case $msg = $mCSV
    $perlok = RunWait( @ComSpec & " /c perl " & _
      $EZPATH & "ez-captur\ez-captur.pl "  _
      & $FILENAME & ' "' & $DATETIME & '" ' & $DURATION , $EZPATH);
    If $perlok = 0 Then
     MsgBox ($ONTOP, "Note", "Conversion was successful");
     TestmDAT($mDAT, $EZPATH, $FILENAME, '-m');  ;; check if can enable!
    Else
    MsgBox ($ONTOP, "Debug", "Return code was " & $perlok);
    ;; here might check that target csv file exists, and
    ;; if so, enable next step:
    EndIf
    $GOODITEMS = bitAND($perlok, 31);  # last 5 bit flags
    SetGoodItems($GOODITEMS, $vSBP, $vDBP, $vHR, $vSpO2, $vETCO2);
          # bits that are set indicate errors:
```

```
    StoreParam ($EZPATH, $FILENAME, $EZ_VALIDFLAGS, $GOODITEMS); ;; Store 'valid'

  Case $msg = $mDAT
    $perlok = RunWait( @ComSpec & " /c perl " & _
      $EZPATH & "ez-xlate\ez-csv2dat.pl "  _
      & $FILENAME & '-m datrules.txt', $EZPATH);
    If $perlok = 0 Then
     MsgBox ($ONTOP, "Note", "Conversion was successful");
     TestFile($mSS, $EZPATH & 'dat/', $FILENAME, '-m.dat');
    Else
      MsgBox ($ONTOP, "Debug", "Return code was " & $perlok);
    EndIf

   Case $msg = $mSS
    If AddToSSList($EZPATH, $FILENAME, '-m', $DURATION) > 0 Then   ;; if success
       $ISLISTED = BitOR ($ISLISTED, 1);  ;; bit 0 set = manual
       StoreParam ($EZPATH, $FILENAME, $EZ_ISLISTED, $ISLISTED); ;; Store 'listed'
       MsgBox($ONTOP, "Success", "Added to play list: " & $FILENAME & '-m.DAT');
       If $ISLISTED = 3 Then   ;; if both flags set
         myEnable($REPL);
       EndIf  ;; or make this a rtn.
    Else
       MsgBox($ONTOP, "Oops", "Failed to add " & $FILENAME & " to playlist!");
    EndIf

  Case $msg = $sGET
    $perlok = RunWait( @ComSpec & " /c perl " & _
      $EZPATH & "ez-xlate\getcrypt.pl "  _
      & $FILENAME & ' connect-string.txt debug=3,logprint=1', $EZPATH);
    If $perlok = 0 Then
     MsgBox ($ONTOP, "Note", "Retrieval was successful");
     TestFile($sXML, $EZPATH & 'iar/', $FILENAME, '.iar');
    Else
      MsgBox ($ONTOP, "Problem", "See log. Return code was " & $perlok);
    EndIf

  Case $msg = $sXML
    If Iar2Xml ($EZPATH, $FILENAME) = 1 Then ;; failure
       MsgBox ($ONTOP, "Oops", "There seems to have been a problem. See the LOG");
    Else
       MsgBox ($ONTOP, "Note", "Apparent success!");
       TestFile($sCSV, $EZPATH & 'xml/', $FILENAME, '.xml');
    EndIf

  Case $msg = $sCSV
    ;;  MsgBox ($ONTOP, "Test", "XML to CSV");
    $perlok = RunWait( @ComSpec & " /c perl " & _
      $EZPATH & "ez-xlate\ez-xml2csv.pl "  _
```

```
    & $FILENAME & ' debug=2,logprint=1', $EZPATH);
  If $perlok = 0 Then
   MsgBox ($ONTOP, "Note", "Conversion was successful");
   TestFile($sDAT, $EZPATH & 'csv/', $FILENAME, '-s.csv');  ;; -s is for safersl
  Else
    MsgBox ($ONTOP, "Problem", "See log. Return code was " & $perlok);
  EndIf

Case $msg = $sDAT
  $perlok = RunWait( @ComSpec & " /c perl " & _
    $EZPATH & "ez-xlate\ez-csv2dat.pl "  _
    & $FILENAME & '-s' & ' datrules.txt', $EZPATH);
  If $perlok = 0 Then
   MsgBox ($ONTOP, "Note", "Conversion was successful");
   TestFile($sSS, $EZPATH & 'dat/', $FILENAME, '-s.dat');
  Else
    MsgBox ($ONTOP, "Problem", "See log. Return code was " & $perlok);
  EndIf

Case $msg = $sSS
  If AddToSSList($EZPATH, $FILENAME, '-s', $DURATION) > 0 Then   ;; if success
     $ISLISTED = BitOR ($ISLISTED, 2);  ;; bit 1 set = ss file
     StoreParam ($EZPATH, $FILENAME, $EZ_ISLISTED, $ISLISTED); ;; Store 'listed'
     MsgBox($ONTOP, "Success", "Added to play list: " & $FILENAME & '-s.DAT');
     If $ISLISTED = 3 Then   ;; if both flags set
       myEnable($REPL);
     EndIf  ;; or make this a rtn.
  Else
     MsgBox($ONTOP, "Oops", "Failed to add " & $FILENAME & "-s to playlist!");
  EndIf

Case $msg = $REPL
  $doit = MsgBox($ONTOP+1, "Confirm", "Do you want to replay the files?");
  If $doit = 1 Then ;; OK button pressed (2=cancel)
     GuiSetState(@SW_MINIMIZE);
     $perlok = RunWait( @ComSpec & " /c " & _
       $EZPATH & "ez-replay\ez-replay.au3" , $EZPATH);
     GuiSetState(@SW_RESTORE);
  Else
    MsgBox($ONTOP, "Cancelled", "Nothing done!");
  EndIf

;; ----------- IMAGE CLICKS ----------------
Case $msg = $imCSV
  LogNotepad ($EZPATH, $FILENAME, '-capt')

Case $msg = $imDAT
  LogNotepad ($EZPATH, $FILENAME, '-m-c2d')
```

```
  Case $msg = $imADD
    LogNotepad ($EZPATH, $FILENAME, '-m-play');

  Case $msg = $isGET
    LogNotepad ($EZPATH, $FILENAME, '-iar-')

  Case $msg = $isXML
    LogNotepad ($EZPATH, $FILENAME, '-xml')

  Case $msg = $isCSV
    LogNotepad ($EZPATH, $FILENAME, '-x2c-')

  Case $msg = $isDAT
    LogNotepad ($EZPATH, $FILENAME, '-s-c2d-')

  Case $msg = $isADD
    LogNotepad ($EZPATH, $FILENAME, '-s-play');

 ;; --- view actual resulting files in Notepad: (or even Excel)
 Case $msg = $vwCSV
    ViewNotepad ($EZPATH, 'csv/', $FILENAME & '-m', '.csv')

 Case $msg = $vwCSV2
    ViewNotepad ($EZPATH, 'csv/', $FILENAME & '-s', '.csv')

 Case $msg = $vwXML
    ViewNotepad ($EZPATH, 'xml/', $FILENAME, '.xml')

 Case $msg = $vwDAT
    ViewNotepad ($EZPATH, 'dat/', $FILENAME & '-m', '.dat')

 Case $msg = $vwDAT2
    ViewNotepad ($EZPATH, 'dat/', $FILENAME & '-s', '.dat')

 Case $msg = $vwSS
    ViewNotepad ($EZPATH, 'ez-replay/', 'SSLIST', '.txt')

EndSelect

Wend;
EndFunc
```

Simple subsidiary routines

```
;; a simple enabling routine:
Func myEnable($ctl)
  $st = bitAND (GUICtrlGetState ($ctl), $GUI_ENABLE);
```

```
  If $st = 0 Then   ;; prevent flicker
    GUICtrlSetState ($ctl, $GUI_ENABLE );
  EndIf
EndFunc

;; a simple control disabling routine:
Func myDisable($ctl)
  GUICtrlSetState ($ctl, $GUI_DISABLE );
EndFunc

;; similar check/uncheck:
Func myCheck($ctl)
  GUICtrlSetState ($ctl, $GUI_CHECKED );
EndFunc

Func myUncheck($ctl)
  GUICtrlSetState ($ctl, $GUI_UNCHECKED );
EndFunc

;; A routine to check whether a particular raw datum file exists:
;; $seq might be one of 'sbp', 'dbp' etc.

Func IsRawFile ($seq, $FILENAME)
 $EZPATH = "\ez\";
 $PATHNAME = $EZPATH & "rawdata\";
 $FILENAME = $FILENAME & '-' & $seq & ".csv";
 $WHOLEFILE = $PATHNAME & $FILENAME;
 If FileExists($WHOLEFILE) Then
    return (1); ;; success
 EndIf
return(0);      ;; fail
EndFunc

;; a routine to determine whether to tick various acquisition boxes:
Func CheckRaw ($xSBP, $xDBP, $xHR, $xSpO2, $xETCO2, $FILENAME)
 $my5 = 0;
 If IsRawFile('sbp', $FILENAME) = 1 Then
   myCheck($xSBP);
   $my5 = BitOR($my5, 1);
 EndIf
 If IsRawFile('dbp', $FILENAME) = 1 Then
   myCheck($xDBP);
   $my5 = BitOR($my5, 2);
 EndIf
 If IsRawFile('hr', $FILENAME) = 1 Then
   myCheck($xHR);
   $my5 = BitOR($my5, 4);
 EndIf
```

```
 If IsRawFile('spo2', $FILENAME) = 1 Then
   myCheck($xSpO2);
   $my5 = BitOR($my5, 8);
 EndIf
 If IsRawFile('etco2', $FILENAME) = 1 Then
   myCheck($xETCO2);
   $my5 = BitOR($my5, 16);
 EndIf
 return($my5);
EndFunc


;; routine to tick 'Valid' boxes (five) based on flags in $GOODITEMS
Func SetGoodItems ($GOODITEMS, $vSBP, $vDBP, $vHR, $vSpO2, $vETCO2)
  myUncheck($vSBP);
  If bitAND ($GOODITEMS, 1) = 0 Then
    myCheck($vSBP);
  EndIf

  myUncheck($vDBP);
  If bitAND ($GOODITEMS, 2) = 0 Then
    myCheck($vDBP);
  EndIf

  myUncheck($vHR);
  If bitAND ($GOODITEMS, 4) = 0 Then
    myCheck($vHR);
  EndIf

  myUncheck($vSpO2);
  If bitAND ($GOODITEMS, 8) = 0 Then
    myCheck($vSpO2);
  EndIf

  myUncheck($vETCO2);
  If bitAND ($GOODITEMS, 16) = 0 Then
    myCheck($vETCO2);
  EndIf
EndFunc


Func TestmDAT ($mDAT, $EZPATH, $FILENAME, $SUFFIX)
 ;; if .csv file has been made, enable DAT conversion..
 ;; [NO! rather test if successful return from Perl invocation [FIX ME]]
 If FileExists ( $EZPATH & "csv\" & $FILENAME & $SUFFIX & '.csv') Then
    myEnable($mDAT)
 EndIf
EndFunc
```

```
;; test whether file exists, and if it does
;; enable the relevant button:
Func TestFile ($btn, $WHOLEPATH, $FILENAME, $SUFFIX)
  If FileExists ($WHOLEPATH & $FILENAME & $SUFFIX) Then
     myEnable($btn)
  EndIf
EndFunc

;; store a parameter to the .INF file
Func StoreParam ($EZPATH, $FILENAME, $parmno, $value)
  $fnam = $EZPATH & 'ez-info\' & $FILENAME & '.inf';
  If Not FileExists ($fnam) Then
    MsgBox($ONTOP, "Error", "INF File doesn't exist: " & $fnam);
    return;
  EndIf
  $ARR = Slurp($fnam);
  $ARR[$parmno] = $value;
  Vomit($fnam, $ARR)   ;; REWRITE
EndFunc
```

2.4.1 Invoke Notepad

Here we look for all relevant log files, and work through them until we get the last file, which we display by opening up NotePad. We minimise the GUI until the log file closes.

```
Func LogNotepad ($EZPATH, $FILENAME, $SUFFIX)
  $fnam = $EZPATH & 'log\' & $FILENAME & $SUFFIX & '*.LOG';
  $h = FileFindFirstFile($fnam);
  If $h = -1 Then
    MsgBox($ONTOP, "Oops", "LOG File not found: " & $fnam);
    Return
  EndIf

  $tot = 0;
  DIM $FA[$tot+1];   ;; just playing, for now. later ? select from list

  $file = '';
  While 1
    $lastfile = $file;
    $file = FileFindNextFile($h);
    If @error Then ExitLoop;
    $FA[$tot] = $file;
    $tot = $tot + 1;
    ReDim $FA[$tot+1];
  WEnd
  FileClose($h);
```

```
  ;; run notepad:
    GuiSetState(@SW_MINIMIZE);
    Run ("Notepad.exe " & $lastfile, $EZPATH & 'log\' , @SW_MAXIMIZE);
    Sleep(1000);
    WinWaitClose($FILENAME & $SUFFIX);
    GuiSetState(@SW_RESTORE);
EndFunc
```

Similar but filename specific is ViewNotepad:

```
Func ViewNotepad ($EZPATH, $EZDIR, $FNAME, $SUFFIX)
  $fnam = $EZPATH & $EZDIR & $FNAME;
  If Not FileExists($fnam & $SUFFIX) Then
    MsgBox($ONTOP, "Oops", "File not found: " & $fnam & $SUFFIX);
    Return
  EndIf

  GuiSetState(@SW_MINIMIZE);
  $TARGET = $FNAME;
  $isXL = 0;

  If $SUFFIX = '.csv' Then
   ;; $XLpath =  RegRead( "HKEY_CLASSES_ROOT\.mht\OpenWithList\Microsoft Office Ex
    $XLpath =  RegRead( "HKEY_CLASSES_ROOT\Applications\EXCEL.EXE\shell\edit\comma
    if StringLen ($XLpath) > 5 Then
      $isXL = 1;
      $TARGET = "Microsoft Excel";
    EndIf
  EndIf

  If $isXL = 1 Then
    Run ($XLPATH & ' ' &  $fnam & $SUFFIX, $EZPATH & $EZDIR , @SW_MAXIMIZE);
  Else
    Run ("Notepad.exe " & $fnam & $SUFFIX, $EZPATH & $EZDIR , @SW_MAXIMIZE);
  EndIf

  Sleep(1000);
  WinWaitClose($TARGET);
  GuiSetState(@SW_RESTORE);
EndFunc
```

2.4.2 Queue entries for replay by SAFERsleep

```
;; AddToSSList:
;;   open the file 'SSLIST.TXT' and
;;   search for an entry $FILENAME$SUFFIX.DAT (suffix is -s or -m)
;;   if the entry exists, return the index of the entry
```

```
;;   if it doesn't insert the entry and return MINUS the index
;;   on failure, return 0.
;;   Also create a log file: $FILENAME$SUFFIX-play.LOG
Func AddToSSList($EZPATH, $FILENAME, $SUFFIX, $DURATION)

  $partname =  $FILENAME & $SUFFIX ;    ;; do NOT add .DAT here!
  $fpath = $EZPATH & 'ez-replay\SSLIST.TXT' ;
  $logname = $EZPATH & 'log\' & $FILENAME & $SUFFIX & '-play.LOG';

  ; might confirm $DURATION is numeric;
  ;; $epochs = 1 + Int( ($DURATION+4)/5 ); !NO we want minutes

  If FileExists($fpath) = 0 Then
    $fh = FileOpen ($fpath, 2);  ;; Create new file!
    If $fh = -1 Then
      FileWriteLine ($logname, "Error 1: Bad file " & $fpath);
      ;; ** write to log
      Return (0);  ;; FAILED HORRIBLY.
    EndIf
    FileWriteLine ($fh, "% Header line");
    FileWriteLine ($fh, $partname & ',' & $DURATION);
    FileClose($fh);  ;; might test for success?
    FileWriteLine ($logname, "New entry at line 1");
    Return (1);      ;; written to line 1.
  EndIf

  $lineno = ScanFile($fpath, $partname, $logname, 3);
  If $lineno = -1 Then
      FileWriteLine ($logname, "Error 2: Bad file " & $fpath);
      Return (0);  ;; FAILED HORRIBLY.
  EndIf
  If $lineno > 0 Then   ;; WAS found
    FileWriteLine ($logname, "Entry found at line " & $lineno);
    Return($lineno);
  EndIf

  ; not found, so append to file
  FileWriteLine ($fpath, $partname & ',' & $DURATION);
  ; clumsy beyond belief:
  $lineno = ScanFile($fpath, $partname, $logname, 3);
  If $lineno = -1 Then
      FileWriteLine ($logname, "Error 4: Bad file " & $fpath);
      Return (0);  ;; FAILED HORRIBLY.
  EndIf
  If $lineno > 0 Then   ;; FOUND
    FileWriteLine ($logname, "Entry inserted at line " & $lineno);
    Return($lineno);
  EndIf
```

```
  FileWriteLine ($logname, "Error 3: Bad file " & $fpath);
  Return (0);  ;; FAILED
EndFunc


Func ScanFile ($path,$what, $logname, $DEBUG)
  $file = FileOpen($path, 0);   ;; read only
  If $file = -1 Then
     If $DEBUG = 3 Then
       FileWriteLine ($logname, "FILE NOT OPENED!?");
     EndIf
    Return (-1);  ;; fail horribly
  EndIf

  If $DEBUG = 3 Then
    FileWriteLine ($logname, " Searching for: " & $what);
  EndIf

; Read in lines of text until the EOF is reached
$lcount = 1;
While 1
    $line = FileReadLine($file);
    If @error = -1 Then ExitLoop;  ;; end of file
    $ln = StringSplit($line, ',');
    If $ln[1] = $what Then
      If $DEBUG = 3 Then
         FileWriteLine ($logname, "Found at line " & $lcount);
      EndIf
      FileClose($file);
      Return ($lcount);
    EndIf
    If $DEBUG = 3 Then
      FileWriteLine ($logname, "  Comparing: " & $line);
    EndIf
    $lcount = $lcount + 1;
Wend

 If $DEBUG = 3 Then
   FileWriteLine ($logname, "Not found");
 EndIf
 FileClose($file);
Return(0);   ;; not found
EndFunc
```

2.4.3 Advanced Datum acquisition

Similar to our basic schema, we here obtain *just one data set* for one of SBP, DBP, or HR. We initialise data fetch, and wait for Excel. When invoked, after a sufficient wait, we save the file in the format *X0001-sbp.csv*, where 'X0001' represents the case number, and the suffix is one of sbp, dbp or hr.

We start each set with a zero and Y calibration. Note that with the final (HR) acquisition alone, we *must* perform a terminal zero, calibrate, and x-determination.

```
Func GetSequence ($seqname, $FILENAME, $DATETIME, $FAKEDCMS)
 $EZPATH = "\ez\";
 $PATHNAME = $EZPATH & "rawdata\";
  $FILENAME = $FILENAME & '-' & $seqname & ".csv";
    ;; e.g. "S0001-sbp.csv"

 OpenDCMS($FAKEDCMS);
  WinActivate("DCMS");
 ;; Data entry here..

 ;--Excel--
  WinWaitActive ("Microsoft Excel");
  WinActivate ("Microsoft Excel");
  ; if target file exists, detect and delete it!! <heh>
  $WHOLEFILE = $PATHNAME & $FILENAME;
  If FileExists($WHOLEFILE) Then
    FileDelete($WHOLEFILE);
  EndIf

  If $FAKEDCMS <> 0 Then
   MsgBox($ONTOP, "Fake Excel input", "You have " & $FAKEDCMS & " seconds for Exce
   WinActivate ("Microsoft Excel");
  EndIf

  Sleep (3000);     ;; good long wait for import
  Send ("!f");      ;; Alt+F = File Save
  Sleep(300)
  Send ("a");       ;; ... as
  Sleep (200);
  Send($WHOLEFILE);
   Sleep (200);
  Send ("{TAB}");   ;; tab to file type
  Sleep (200);
  Send ("C");       ;; CSV
  Sleep (200);
  Send ("{ENTER}"); ;;accept CSV
  Sleep (200);
  Send ("{ENTER}"); ;; save
```

```
  Sleep(300);
  If WinExists("Microsoft Excel", "The selected file type does not") Then
      Send ("{ENTER}");  ;; Windoze being helpful
  EndIf
  Sleep(300);
  If WinExists("Microsoft Excel", "not compatible with CSV") Then
      Send ("{ENTER}");  ;; Windoze being helpful
  EndIf
  Sleep (500);
  Send ("!fx");     ;; CLOSE Excel!
  Sleep (200);      ;; Windows will ask "Do you want to save"
  Send ("n");       ;; no!

  ;; next, close DCMS!
  CloseDCMS($FAKEDCMS);

EndFunc



 ;--DCMS--

Func OpenDCMS ($FAKEDCMS)
 If $FAKEDCMS <> 0 Then
   return;   ;; do nothing
 EndIf
 $DCMSEXEC = "C:\Program Files\USB DCMS FOR MEASURING TOOLS\Receive.exe";
 Run($DCMSEXEC);
 WinWaitActive("DCMS");
 ;; open port:
 ControlClick ( "DCMS", "", _
        "[CLASS:ThunderRT6CommandButton; INSTANCE:6]" )
 ;; establish connection (3 tries works!)
 ControlClick ( "DCMS", "", _
        "[CLASS:ThunderRT6CommandButton; INSTANCE:11]" )
 Sleep(300);
 ControlClick ( "DCMS", "", _
        "[CLASS:ThunderRT6CommandButton; INSTANCE:11]" )
 Sleep(300);
 ControlClick ( "DCMS", "", _
        "[CLASS:ThunderRT6CommandButton; INSTANCE:11]" )
 Sleep(300);
EndFunc

Func CloseDCMS ($FAKEDCMS)
 If $FAKEDCMS <> 0 Then
   return;   ;; do nothing
 EndIf
```

```
  ;--- Shutdown and exit!
  WinActivate("DCMS");
  ;; Close port
  ControlClick ( "DCMS", "", _
     "[CLASS:ThunderRT6CommandButton; INSTANCE:11]" )
  ;; Exit DCMS
  ControlClick ( "DCMS", "", _
     "[CLASS:ThunderRT6CommandButton; INSTANCE:10]" )
EndFunc
```

2.4.4 Keyboard input

```
Func GetKbSequence ($seqname, $FILENAME, $DATETIME, $DURATION)
 $EZPATH = "\ez\";

  $prog = @ComSpec & " /c " & "perl " & $EZPATH & "ez-captur\ez-keyboard.pl ";
  $perlok = RunWait ( $prog & $FILENAME & ' ' & $seqname & ' ' & $DURATION , _
      $EZPATH, @SW_MAXIMIZE );
    ;; run the Perl program in DOS, with relevant arguments
    ;; last Run parameter specifies working directory
 If $perlok = 0 Then
    Return 1
 EndIf

 MsgBox ($ONTOP, "OOPS", "Error in keyboard input");
EndFunc
```

2.4.5 Decrypt SAFERsleep IAR file to XML

It is also possible to retrieve automatically-captured records from SAFERsleep, but these are encrypted. A decryption tool is provided with XML output, but it is inconvenient to use this tool manually on multiple records. We therefore automate the process, again using AutoIt and Perl. Here's the code that coordinates the translation from IAR to XML

```
Func Iar2Xml ($EZPATH, $IARFILENAME)

;; ** NB ** at present we assume the existence of all of the paths listed below.
$EZPATH = "\ez\";
$XPATH = $EZPATH & "ez-xlate\";
$IARPATH = $EZPATH & "iar\";
$XMLPATH = $EZPATH & "xml\";
$ERRORLOG = $EZPATH & 'log\' & $IARFILENAME & '-xml.LOG';
```

```
ClipPut('')   ;; clear clipboard

;; activate SAFERsleep viewer: NB. batch file must be defined.
Run($XPATH & "xlat.bat " & $IARPATH & $IARFILENAME & '.IAR', $XPATH );  ;; ugh
WinWaitActive ("SAFERsleep IAR Viewer");
WinActivate ("SAFERsleep IAR Viewer");

;; * HERE MUST CHECK FOR: Error box, and if appears:
;;   simply close down, don't capture, log the error
If WinExists("Error.") Then
  $dudtext = ControlGetText ( "Error", "", "[CLASS:Static; INSTANCE:1]" ) ;; get
  FileWrite($ERRORLOG, "Error in file: " & $IARFILENAME & ", text=" & $dudtext )
  MsgBox($ONTOP, "Parse Error", "IAR viewer forced error(logged)", 5);
  WinActivate("Error.");
  Send("{ENTER}");        ;; acknowledge
  Sleep(100);
  Return (1);  ;; failure
EndIf;

Sleep(100);
ControlClick ("SAFERsleep IAR Viewer", "", "[CLASS:TcxCustomInnerMemo; INSTANCE:
Sleep(100);
Send("^a")
Sleep(100);
Send("^c")
Sleep(1000);   ;; this delay seems important

;; get data from clipboard:
$XMLDATA = ClipGet();            ;; get xml from clipboard
If StringLen($XMLDATA) < 1000 Then
  FileWrite($ERRORLOG, "Warning: tiny XML file: " & $IARFILENAME );  ;;log warni
EndIf

;; simply write to xml file:
$outn = StringSplit ($IARFILENAME, ".");          ;; get rid of suffix
$OUTNAME = $outn[1] & ".xml";                     ;; [might check for error?]
$fHandle = FileOpen( $XMLPATH & $OUTNAME, 2);     ;; write=erase prior contents
If @error <> 0 Then    ;; if failed
  MsgBox($ONTOP, "FATAL Error", "Failed to open file for writing: " & $filename)
  Exit
EndIf
FileWrite($fHandle, $XMLDATA);   ;; ????????????????????? need to do so repeatedl
FileClose($fHandle);
WinClose("SAFERsleep IAR Viewer");
WinWaitClose("SAFERsleep IAR Viewer");
Sleep(500);                     ;;  wait, just in case.
FileWrite($ERRORLOG, "Apparently successful translation: " & $IARFILENAME );  ;;
Return (0);   ;; success
```

```
EndFunc;
```

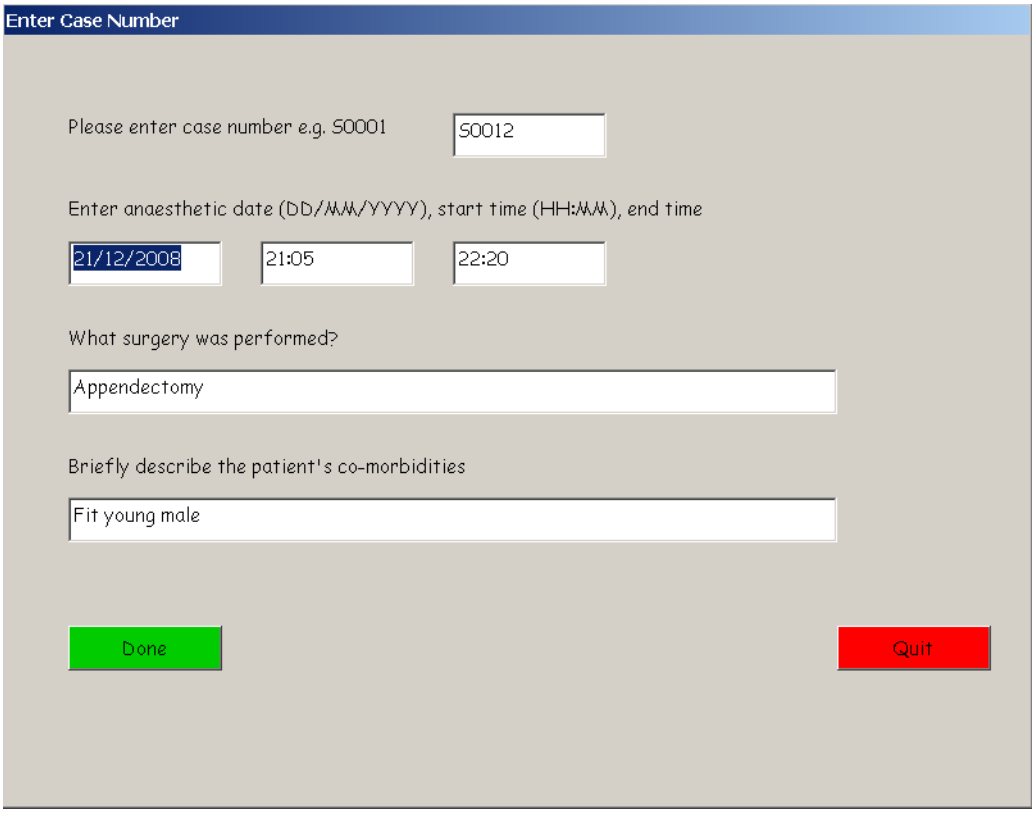

Figure 3: Initial screen for record information

2.5 Obtain basic record information

XGetFileAndTimestamp pops up a GUI, and obtains a file name, as well as a date and time, and certain other information. The date is in NZ format (DD/MM/YYYY) and the time is limited to hours and minutes (HH:MM). This interface screen is shown in Fig. 3.

```
Func XGetFileAndTimestamp()
; make a simple GUI to check the log-in
  Local $x,$y,$w,$h,$pitch,$skip;
  $x=50;
  $y=10;
  $w=120;
  $h=35;
  $skip = 150;
```

```
  $pitch = 50;

Dim $FAT[1+$EZ_MAXLINES]; ;; 'return' array
Dim $EZPATH = "\ez\";
Dim $validflags=31;    ;;set=invalid
Dim $islisted=0;       ;; 1=manual file listed (-m),
                       ;; 2=ss file listed (-s)
Dim $mygui = GuiCreate("Enter Case Number", _
       720, 600, 40, 10, $WS_CAPTION, $WS_EX_TOPMOST);
; FONT:
$font="Comic Sans MS"           ;
GUISetFont (9, 400, 1, $font)  ; Size 9 font

;; File name:


; labels:
GuiCtrlCreateLabel("Please enter case number e.g. S0001", _
       $x, $y + $pitch, $w*4, $h)
;; Date and time:
GuiCtrlCreateLabel("Enter anaesthetic date (DD/MM/YYYY), start time (HH:MM), end t
       $x, $y + 2.3*$pitch, $w*5, $h)
;; Other labels
GuiCtrlCreateLabel("What surgery was performed?", _
       $x, $y + 4.3*$pitch, $w*4, $h)
GuiCtrlCreateLabel("Briefly describe the patient's co-morbidities", _
       $x, $y + 6.3*$pitch, $w*4, $h)

; text:
Dim $caseID = GuiCtrlCreateInput("", $x+$skip*2, $y+$pitch, $w, $h) ;
Dim $myDATE = GuiCtrlCreateInput("", $x, $y+3*$pitch, $w, $h) ;
Dim $myTIME = GuiCtrlCreateInput("", $x+$skip, _
                                 $y + 3*$pitch, $w, $h);
Dim $myEND  = GuiCtrlCreateInput("", $x+2*$skip, _
                                 $y + 3*$pitch, $w, $h);

; Nature of surgery
Dim $mySURGERY = GuiCtrlCreateInput("", $x, $y + 5*$pitch, $w*5, $h) ;
; Patient description
Dim $myDESCRIPTION = GuiCtrlCreateInput("", $x, $y + 7*$pitch, $w*5, $h) ;

; Done BUTTON
Dim $buttonID = GuiCtrlCreateButton("Done", $x, $y+9*$pitch, $w, $h);
GUICtrlSetBkColor ( $buttonID, 0x00CC00 )  ;; green button
;; IMAGE (not now)
;; Dim $picID = GUICtrlCreatePic("pathtoimagegoeshere.gif", _
;;                                 550, 50, 408,308);
Dim $bQuit = GuiCtrlCreateButton("Quit", $x+3.15*$skip, $y+9*$pitch, $w, $h);
```

```
GUICtrlSetBkColor ( $bQuit, 0xFF0000 )  ;; red button


; GUI MESSAGE LOOP
GuiSetState()  ; display the GUI
While 1
  Dim $msg = GuiGetMsg();

  If $msg = $caseID Then
     $FAT[$EZ_CASENUMBER] = GUICtrlRead($caseID);
     ;; here might pre-check validity of casenumber before FileExists..;;
    If FileExists ($EZPATH & 'ez-info\' & $FAT[$EZ_CASENUMBER] & '.inf') Then
      ;; first, get all values from file
      $FAT = Slurp($EZPATH & 'ez-info/' & $FAT[$EZ_CASENUMBER] & '.inf')
      ;; next write values to controls:
      If $FAT[0] > 0 Then
        GUICtrlSetData ($myDATE, $FAT[$EZ_CASEDATE]);
        GUICtrlSetData ($myTIME, $FAT[$EZ_CASETIME]);
        GUICtrlSetData ($myEND, $FAT[$EZ_CASEEND]);
        GUICtrlSetData ($mySURGERY, $FAT[$EZ_SURGERY]);
        GUICtrlSetData ($myDESCRIPTION, $FAT[$EZ_DESCRIPTION]);
        $islisted = $FAT[$EZ_ISLISTED];
        $validflags = $FAT[$EZ_VALIDFLAGS];
      EndIf
    EndIf
  EndIf

  If $msg = $bQuit Then
    Exit
  EndIf

  If $msg = $buttonID Then     ; was 'Done' button pressed? [FIX THE FOLLOWING!]
     $FAT[$EZ_CASENUMBER] = GUICtrlRead($caseID);
     $FAT[$EZ_CASEDATE] = GUICtrlRead($myDATE);
     $FAT[$EZ_CASETIME] = GUICtrlRead($myTIME);
     $FAT[$EZ_CASEEND] = GUICtrlRead($myEND);
     $FAT[$EZ_SURGERY]= GUICtrlRead($mySURGERY);
     $FAT[$EZ_DESCRIPTION]= GUICtrlRead($myDESCRIPTION);
     $duration = ValidateAllInputs($FAT[$EZ_CASENUMBER], _
                                   $FAT[$EZ_CASEDATE], _
                                   $FAT[$EZ_CASETIME], _
                                   $FAT[$EZ_CASEEND]);
     ;; MsgBox($ONTOP, "Debug", "Duration is " & $duration);
     If $duration > 0 Then
              GUIDelete($mygui);
              $FAT[0] = $EZ_MAXLINES;  ;;number of items
              $FAT[$EZ_DURATION] = $duration;
              $FAT[$EZ_ISLISTED   ] = $islisted;
```

```
              $FAT[$EZ_VALIDFLAGS   ] = $validflags;
              ;; and here WRITE file:
              $fn = $EZPATH & 'ez-info/' & $FAT[$EZ_CASENUMBER] & '.inf';
              Vomit($fn, $FAT);  ;; write file .INF file for this case.
              Return ($FAT);
      Else
        GuiCtrlSetState($caseID,$GUI_FOCUS);  ;; fix mouse focus problem![hack]
      EndIf
  EndIf
WEnd
EndFunc
```

Subsidiary functions

The following functions are used above.

```
Func ValidateAllInputs($casenumber, $casedate, $casetime, $end)

      If (StringLen($casenumber) <> 5) Then
        MsgBox($ONTOP, "Woops!", "Case number must be 5 characters long! No blanks'
        Return(0);
      EndIf
      If (StringLen($casedate) <> 10) Then
        MsgBox($ONTOP, "Woops!", "Date must be in the form dd/mm/yyyy (" & $casedat
        Return(0);
      EndIf
      If (StringLen($casetime) <> 5) Then
        MsgBox($ONTOP, "Woops!", "Start time must be in the form HH:MM (" & $caseti
        Return(0);
      EndIf
      If (StringLen($end) <> 5) Then
        MsgBox($ONTOP, "Woops!", "End time must be in the form HH:MM (" & $end & ')
        Return(0);
      EndIf
      $jstart = ValidTimestamp ($casedate, $casetime); ;; julian date
      If $jstart = 0 Then
        MsgBox($ONTOP, "Bad date!", _
        "Please check date/time: " & $casedate & " " & $casetime);
        Return(0);
      EndIf
      $jend = ValidTimestamp ($casedate, $end);
      If $jend = 0 Then
              MsgBox($ONTOP, "Bad end time!", _
        "Please check time: " & " " & $casetime);
        Return(0);
      EndIf
  ;; here calculate duration:
    ;; MsgBox($ONTOP, "Debug", "Difference between " & $jend & ' and ' & $jstart);
```

```
    $duration = $jend - $jstart;
    If ($duration < 0) Then ;; next day
       $duration = $duration + 0.5;
    EndIf
    $duration = Int (0.5 + ($duration * 60*24) );  ;; get minutes
    Return($duration);
EndFunc


;; Slurp file data in. Format is:
;;  first line: "date=DD/MM/YYYY"
;;  second line:"time=HH:MM"
;;  third line:"valid=31" (valid flags)
Func Slurp ($fnam)
  Dim $ALEN = $EZ_MAXLINES;
  Dim $arr[$ALEN+1];
  $arr[0] = 0;   ;; 0 signals failure

   $f = FileOpen ($fnam, 0) ;; read only
   if ($f = -1) Then
     Return($arr);  ;; fail
   EndIf

   $WARNINGS = 0;
   ;; the following is extraordinarily cumbersome and needs to be revised (cf perl
      $v = FileReadLine ($f);  ;; case number
      $s = StringSplit($v, "=");
      If $s[1] = 'casenumber' Then
        $arr[$EZ_CASENUMBER] = $s[2];
      Else
        $WARNINGS = $WARNINGS + 1;
      EndIf

      $v = FileReadLine ($f);  ;; first, date
      $s = StringSplit($v, "=");
      If $s[1] = 'date' Then
        $arr[$EZ_CASEDATE] = $s[2];  ;; get date
      Else
        $WARNINGS = $WARNINGS + 1;
      EndIf

      $v = FileReadLine ($f);  ;; next, time
      $s = StringSplit($v, "=");
      If $s[1] = 'time' Then
        $arr[$EZ_CASETIME] = $s[2];  ;; and time
      Else
        $WARNINGS = $WARNINGS + 1;
      EndIf
```

```
$v = FileReadLine ($f);  ;; validity flags
$s = StringSplit($v, "=");
If $s[1] = 'valid' Then
  $arr[$EZ_VALIDFLAGS] = $s[2];
Else
  $WARNINGS = $WARNINGS + 1;
EndIf

$v = FileReadLine ($f);  ;; "listed" flags
;; might here check for success..
$s = StringSplit($v, "=");
If $s[1] = 'listed' Then
  $arr[$EZ_ISLISTED] = $s[2];
Else
  $WARNINGS = $WARNINGS + 1;
EndIf

;; parameters added in v 0.37:
$v = FileReadLine ($f);  ;; end time
$s = StringSplit($v, "=");
If $s[1] = 'end' Then
  $arr[$EZ_CASEEND] = $s[2];
Else
  $WARNINGS = $WARNINGS + 1;
EndIf

$v = FileReadLine ($f);  ;; description
$s = StringSplit($v, "=");
If $s[1] = 'surgery' Then
  $arr[$EZ_SURGERY] = $s[2];
Else
  $WARNINGS = $WARNINGS + 1;
EndIf

$v = FileReadLine ($f);  ;; surgery
$s = StringSplit($v, "=");
If $s[1] = 'description' Then
  $arr[$EZ_DESCRIPTION] = $s[2];
Else
  $WARNINGS = $WARNINGS + 1;
EndIf

$v = FileReadLine ($f);  ;; duration (derived)
$s = StringSplit($v, "=");
If $s[1] = 'duration' Then
  $arr[$EZ_DURATION] = $s[2];
Else
```

```
        $WARNINGS = $WARNINGS + 1;
      EndIf

   If $WARNINGS > 0 Then
      MsgBox($ONTOP, "Warning", "In reading file " & $fnam _
       & "there was/were " & $WARNINGS & " line(s) containing bad data!");
   EndIf
   FileClose($f);

  $arr[0] = $ALEN;          ;; size!
  Return($arr);  ;; return an ARRAY
EndFunc
```

The reverse function, Vomit, writes an array that was Slurped. Badly written and nasty.

```
Func Vomit ($fnam, $ARR)
  Dim $ALEN = 1 + $ARR[0]; ;; get item count
  ;; here might check this vs $EZ_DURATION

  $i = 1;
  $f = FileOpen ($fnam, 2) ;; rewrite
  Dim $VALS [$EZ_MAXLINES+1];  ;;cumbersome
  $VALS[$EZ_CASENUMBER ] = 'casenumber';
  $VALS[$EZ_CASEDATE   ] = 'date';
  $VALS[$EZ_CASETIME   ] = 'time';
  $VALS[$EZ_VALIDFLAGS ] = 'valid';
  $VALS[$EZ_ISLISTED   ] = 'listed';
  $VALS[$EZ_CASEEND    ] = 'end';
  $VALS[$EZ_SURGERY    ] = 'surgery';
  $VALS[$EZ_DESCRIPTION] = 'description';
  $VALS[$EZ_DURATION   ] = 'duration';

  While $i < $ALEN
    $v = $VALS[$i] & '=' & $ARR[$i];
    FileWriteLine($f, $v);
    $i = $i + 1;
  Wend

  FileClose($f);
EndFunc

Func ValidTimestamp ($d, $t)
  $f = KiwiJulian ($d, $t);
  $backdate = KiwiGregorian ($f);
  if $backdate <> ($d & " " & $t & ":00") Then
    MsgBox ($ONTOP, "Debug", _
    "We compared " & $d & " " & $t & ":00 with " & $backdate );
```

```
    Return (0);
  EndIf
 Return ($f);
EndFunc

Func KiwiJulian ($d, $t)
  $ds = StringSplit ($d, "/");
  $dd = $ds[1];   ;; day
  $mm = $ds[2];   ;; month
  $yy = $ds[3];   ;; year

  $tt = StringSplit ($t, ":");
  $hr = $tt[1];
  $mi = $tt[2];
  $ss = 0;
  return (Julian($yy,$mm,$dd,$hr,$mi,$ss));
EndFunc

Func Julian ($fy, $fm, $fd, $fh, $fmi, $fs)
  $a1 = 367*$fy;
  $a2 = Int(7*($fy+Int(($fm+9)/12))/4);
  $a3 = Int(3*(Int(($fy+($fm-9)/7)/100)+1)/4);
  $a4 = Int(275*$fm/9)+$fd+1721028.5;
  $a5 = ($fh + ($fmi + ($fs)/60)/60)/24;
  $f= $a1 - $a2 - $a3 + $a4 + $a5;
  return($f);
EndFunc

;; Given Julian date (float) make Gregorian date/time:
Func KiwiGregorian ($jd)
 $EPSILON = 0.00001;
 $jd += $EPSILON;
 $Z = Floor($jd - 1721118.5);
 $R = $jd - 1721118.5 - $Z;
 $G = $Z - 0.25;
 $A = Floor($G / 36524.25);
 $B = $A - Floor($A / 4);
 $year = Floor(($B+$G) / 365.25);
 $C = $B + $Z - Floor(365.25 * $year);
 $month = int((5 * $C + 456) / 153);
 $day = Int($C - int((153 * $month - 457) / 5) + $R);
 If $month > 12 Then
    $year = $year + 1;
    $month = $month - 12;
 EndIf

 ;; next, HH:MM:SS
 $gd = 0.5 + $jd - Int($jd);
```

```
 If $gd > 1 Then  ;; Julian starts at midday!!
   $gd = $gd - 1;
 EndIf

 $gh = $gd;   ;; clumsy
 $gh = $gh * 24;
 $gmi = $gh;
 $gh = Int($gh);
 $gmi = $gmi - $gh;
 $gmi = $gmi * 60;
 $gs = $gmi;
 $gmi = Int($gmi);
 $gs = $gs - $gmi;
 $gs = Int($gs * 60);

 return ( DoubleDigit($day) & "/" _
        & DoubleDigit($month) & "/" & $year & " " _
        & DoubleDigit($gh) & ":" _
        & DoubleDigit($gmi) & ":" _
        & DoubleDigit($gs) );
EndFunc

;; trivial function to turn 1 digit into two
Func DoubleDigit ($d)
 If StringLen($d) < 2 Then
    $d = "0" & $d;
 EndIf
 Return($d);
EndFunc
```

2.5.1 DOS invocation of ez-GUI

We simply type in *run* from within the *ez* directory:

```
echo off
cls
\ez\ez-captur\ez-gui.au3
```

Invoking the anaesthetic viewer

We do so using a DOS batch file, *xlat.bat*, stored in the */ez/ez-xlate* directory:

```
echo off
cls
AnaestheticViewer "%1"
```

This assumes that the *AnaestheticViewer.exe* is in the DOS path (see invocation).

2.6 Perl keyboard input

A simple Perl program to acquire either ETCO2 or SpO2 values from the console, and write them to a file in the format S0001-spo2 in the */ez/rawdata* directory. We submit command-line parameters, the first being the serial number, the second the type of datum (spo2 or etco2), and the third the duration of the anaesthetic in minutes. An optional fourth parameter consists of command-line controls. The program is called *ez-keyboard.pl*.

The exit code reflects success (zero) or failure (nonzero).

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

my ($DEBUG) = 3;     # 0 to turn off debugging,
                     #  1=overview, 2=detail, 3=nitpicking
my ($LOGPRINT) = 1;  # print to log
my ($WARNINGS) = 0;  # see usage
my ($EZDIR) = '/ez/';
my ($RAWPATH) = $EZDIR . 'rawdata/';
my $EXITCODE = 0;    # code on exiting Perl
my (@ALLDATA) = ();  # used to store data values. GLOBAL.
my ($STORECOUNT)=1;  # also used for debugging

my ($RAWFILENAME) = $ARGV[0]; # serial number eg S0001. NO suffix.
my ($DATTYPE) = $ARGV[1];     # etco2 or spo2
my ($DURATION) = $ARGV[2];    # duration in minutes

my ($PARAMS) = $ARGV[3];      # eg "debug=3,logprint=0"
if ($PARAMS =~ /debug=(\d)/ ) # command-line control!
   { $DEBUG = $1;
   };
if ($PARAMS =~ /logprint=(\d)/ )
   { $LOGPRINT = $1;
   };

&OpenLog($LOGPRINT, $RAWFILENAME, $DATTYPE); # start writing to (error/debug) log
&Print( "\n Command line arguments are $RAWFILENAME, $DATTYPE, $DURATION, $PARAMS'
$ARGV[0] = '-';      # avoid <stdin> bug in Perl!?

&ReadConsole($DATTYPE, $DURATION);

&Print("\n END of data acquisition \n\n", 1);

# HERE WRITE DATA TO RAW FILE:
&WriteKbData($RAWPATH, $RAWFILENAME, $DATTYPE);

if ($WARNINGS > 0)
   { print "\n ***NOTE*** There was/were $WARNINGS warning(s). \
```

```
 Please consult the log.";
   };
print("\n Finished!\n");
&CloseLog($LOGPRINT);
exit $EXITCODE;
```

Here's the logging routine:

```
sub OpenLog
{ my ($islog, $RAWFILENAME, $DATTYPE);
     ($islog, $RAWFILENAME, $DATTYPE)=@_;
  if (! $islog) { return; }; # if not logging.
  my $TODAY = &GetLocalTime();
  my $logfile= "/ez/log/$RAWFILENAME-$DATTYPE-$TODAY.LOG";
  open (LOGFILE,
    ">$logfile") or die "*CRASH* Could not open LOG $logfile :$!\n";
  print "\n Writing to log file: $logfile";
}

sub CloseLog
{ my ($islog);
     ($islog)=@_;
  if ($islog)
     { close LOGFILE;
     };
}
```

Here are the subsidiary routines:

```
sub GetLocalTime
{ my ($sec, $min, $hour,
      $mday, $mon, $year,
      $wday, $yday, $isdst);
  ($sec, $min, $hour,
      $mday, $mon, $year,
      $wday, $yday, $isdst) = CORE::localtime(time);
  $year += 1900;      #fix y2k.
  $sec  = &DoubleDigit($sec );
  $min  = &DoubleDigit($min );
  $hour = &DoubleDigit($hour);
  $mday = &DoubleDigit($mday);
  $mon  = &DoubleDigit($mon );
  $mon ++;            #january is zero!
  return ("$year$mon$mday-$hour$min$sec");
}

sub DoubleDigit
{
```

```
   my ($i);
      ($i) = @_;
   if (length $i > 1)
      { return $i;
      };
   return "0$i"; # concatenate.
}

sub Print
{ my ($p, $level); # level is debugging level.
        # 0=none, 1=overview, 2=detail, 3=picky
     ($p, $level)=@_;

  if ($DEBUG >= $level) # if level is 0, always print!
     {  if ($LOGPRINT)
           { print LOGFILE $p;
             return;
           };
        print $p; # console print
     };
}

sub Die
{ my ($e);
     ($e)=@_;
  my ($l) = $LINECOUNT-1;
  &Print ("\n\n DIED: Data line = $l, \
    Index was $IDX \n Message: $e", 0);
  $EXITCODE += 128;   # 'fatal'
  exit $EXITCODE;
  # caution. SEE: http://perldoc.perl.org/functions/exit.html
  # die "\n Fatal";
}
```

Reading console values

```
sub ReadConsole
{
  my ($dattype, $DURATION);
     ($dattype, $DURATION)=@_;

  # first, check duration:
  while ($DURATION !~ /^\d+$/ )
    { print "\n Oops! Duration of <$DURATION> doesn't seem valid.";
      print "\n\n Please enter the DURATION in minutes:";
      $DURATION = <STDIN>;
    };
  $DURATION = 5 * (int (($DURATION+4)/5) ); # round up
```

```
  my ($tot) = 1 + $DURATION/5;
  my ($c, $t);
  print "\n\n ENTER DATA. Use the letter n for 'NA'. \
    Use - to skip back, \
        x for no more, \
        n for NA. ";
  my ($badinp, $forced);

  $c = 0; # count
  $t = 0; # time
  $forced = 0;
  while ($c < $tot)
    { $t = 5 * $c;
      if (! $forced)
         { print "\n Enter $dattype value for time $t: ";
           # here will validate spo2/etco2
           $inp = <STDIN>;
           chomp ($inp); # remove cr,lf
           my ($v) = &ValidParam($inp, $dattype);
           if ($v == -1)  # SKIP BACK
              { if ($c > 0)
                   { $c --;
                   } else
                   { print "\n Can't go back. At start!";
                   };
              }
           elsif ($v == -2) # NA
              { &Store($c, -1); # -1 signals datum NA
                $c ++;
              }
           elsif ($v == -3) # forced!
              { $forced = 1;
              }
           elsif ($v == 0)
              { print "\n Oops! Invalid parameter: $inp. \n   Please try again.";
              } else
              { &Store($c, $inp); # store value
                $c ++;
              }
           } else # IS FORCED
           { &Store($c, -1);  # force rest to NA
             $c ++;
           };
    };
}

sub ValidParam
{ my ($inp, $dattype);
```

```
     ($inp, $dattype)=@_;

  if ($inp =~ /-/ )
     { return -1;
     };
  if ($inp =~ /n/i )
     { return -2;
     };
  if ($inp =~ /x/i )
     { return -3;
     };

  if ($inp !~ /^\w*\d+\.*\d*\w*$/ )  # is it numeric. allow whitespace.
     { return (0);  # fail
     };

  if ($dattype eq 'spo2')
     { if ( ($inp < 50)
          ||($inp > 100)
          )
          { return(0);
          };
     }
  elsif ($dattype eq 'etco2')
     { if ( ($inp < 1)
          ||($inp > 100)  # bugger. need to address mmHg vs kPa. [FIX ME]
          )
          { return(0);
          };  # [or warn and confirm if > nnn]
     }
  else { &Die ("Bad parameter type: $dattype");
       };
  return 1;
}
```

here are the Store and Unstore routines. The latter is redundant.

```
sub Store
{ my ($t, $y);
     ($t, $y)=@_;
  $ALLDATA[$t] = $y;
  &Print ( "\n >$STORECOUNT Store: $y at time $t", 2 );
  $STORECOUNT ++;
}

sub Unstore
{
  my ($t); # parameter is eg 'etco2', $t is time
```

```
     ($t) = @_;
  $ALLDATA[$t] = '';   # redundant
};
```

Writing data

```
sub WriteKbData
{ my ($RAWPATH, $RAWFILENAME, $DATTYPE);
     ($RAWPATH, $RAWFILENAME, $DATTYPE)=@_;
  my ($outname) = "$RAWPATH$RAWFILENAME-$DATTYPE.csv";
  open (OUTFILE, ">$outname") or &Die (
     "*CRASH* Could not open OUTPUT FILE: $outname :$!\n");
  print OUTFILE "Count,Value";
  my ($KBCOUNT) = 0;
  foreach $d (@ALLDATA)
    { $KBCOUNT ++;
      &Print ("\n\n Extracting: <$d>", 3);
       print OUTFILE "\n$KBCOUNT,$d";
    };
  close OUTFILE;
}
```

3 Retrieve files from SAFERsleep using Perl

Here's the Perl script *getcrypt.pl*, placed in the */ez/ez-xlate* directory. First we connect to SAFERsleep, then we obtain a binary image of the encrypted IAR file, and write it to the */ez/iar* directory.

```
#!/usr/local/bin/perl -w
 use strict;
 use Win32::ODBC;
my ($DEBUG) = 3;     # 0 to turn off debugging,
                     #  1=overview, 2=detail, 3=nitpicking
my ($LOGPRINT) = 1;  # print to log
my ($EZDIR) = '/ez/';
my $EXITCODE = 0;    # code on exiting Perl. global.

my ($CASENUM) = $ARGV[0];    # e.g. S0029
my ($CONFILENAME)=$ARGV[1];    # connection string file
my ($PARAMS)  = $ARGV[2];    # eg "debug=3,logprint=0"
if ($PARAMS =~ /debug=(\d)/ )# command-line control!
   { $DEBUG = $1;
   };
if ($PARAMS =~ /logprint=(\d)/ )
   { $LOGPRINT = $1;
   };
 &OpenLog($LOGPRINT, $EZDIR, $CASENUM); # for debugging
 &Print ("\n Arguments are: $CASENUM ($CONFILENAME) $PARAMS",3);

 if ($CASENUM !~ /S(\d{4})/i )
    {&Die( "\n Bad number: <$CASENUM>" );
    }
my ($REALNUM) = "SOO$1";

my ($CONSTRING) = &GetConnectionString ($EZDIR, $CONFILENAME);
if ($CONSTRING !~ /^'(.+)'$/ )
    { &Die ("\n Connection String must be quoted. <$CONSTRING>");
    } else
    { $CONSTRING = $1;
    };
 my $myODBC;     #ODBC connection..
 my ($dead) = 0;
 &Print ("\n Connection string is <$CONSTRING>", 2);
 $myODBC = new Win32::ODBC($CONSTRING) or $dead = 1;
 if ($dead)
    { &Die ("Error: " . Win32::ODBC::Error() );
    };
 my $OUTTEXT;
 unless ($myODBC->Connection)
   { &Die( "*CRASH* Failed to connect. Dearie me!\n" );
```

```
   };
 &Print( "\n ODBC: connection worked\n", 2);
 $myODBC->SetMaxBufSize(500000); # set buffer size

 my $stmt = "SELECT AnaestheticMonitorData.Data FROM AnaestheticMonitorData,
  AnaestheticPatient, Patient WHERE
  AnaestheticMonitorData.AnaestheticId = AnaestheticPatient.AnaestheticId AND
  AnaestheticPatient.PatientId = Patient.PatientId AND
    Patient.NHI = '$REALNUM'";
 &Print("\n SQL query is <$stmt>", 3);
 my ($answer) = &SQLFetchDatum($myODBC, $stmt);
 my ($anlen) = length $answer;
 if ($anlen < 256)  # arbitrary minimum ??
    { &Die ("Error. Tiny IAR file ($REALNUM) -- fetch failed. Length just $anlen.'
    };
  # here open IAR file, write, and exit:
  my ($dead) = 0;
  open (OUTFILE, ">/ez/iar/$CASENUM.IAR") or $dead = 1;
  if ($dead)
     { &Die( "*CRASH* Could not open OUTPUT FILE: /ez/iar/$CASENUM.IAR :$!\n" );
     };
  binmode OUTFILE;
  print OUTFILE $answer;
  close OUTFILE;
  $myODBC->Close();
  &CloseLog($LOGPRINT);
  exit $EXITCODE;
```

Here are the SQL-handling routines.

```
sub SQLFetchDatum
 { my ($myODBC, $SQLstmt);
      ($myODBC, $SQLstmt) = @_;
  &DoSQL ($myODBC, $SQLstmt);
  my $dat;
  $myODBC->FetchRow();
  $dat = $myODBC->Data();     #get data
  return $dat;
}

sub DoSQL
 { my ($myODBC, $SQLstmt);
      ($myODBC, $SQLstmt) = @_;
  my ($retcode);

  $retcode = ($myODBC->Sql($SQLstmt));
  if ($retcode) # if problem
     { my ($sqlErrors);
```

```
       if ($retcode < 1)
       { &Die ("ERROR SQL failed: code 'retcode' \n<$SQLstmt>" );
       } else
       { $sqlErrors = $myODBC->Error();
         if ( $sqlErrors !~ /\[911\].+\[1\].+\[0\]/ )
            { &Print("SQL problem 'retcode' \n<$SQLstmt> ($sqlErrors)\n" ) ;
              $EXITCODE |= 1;  # signal warning
            };
       };
     };
 }
```

Subsidiary routines follow in the next sections. They are 'pretty standard'. The log file is identified by -*iar*- after the file name.

```
sub Die
{ my ($e);
     ($e)=@_;
  $EXITCODE += 128;   # 'fatal'
  &Print("\n $e \nExit code +128", 0);
  exit $EXITCODE;
  # caution. SEE: http://perldoc.perl.org/functions/exit.html
  # die "\n Fatal";
}

sub Print
{ my ($p, $level); # level is debugging level.
        # 0=none, 1=overview, 2=detail, 3=picky
     ($p, $level)=@_;

  if ($DEBUG >= $level) # if level is 0, always print!
     {  if ($LOGPRINT)
           { print LOGFILE $p;
             return;
           };
        print $p; # console print
     };
}

sub OpenLog
{ my ($islog, $EZDIR, $FILENAME);
     ($islog, $EZDIR, $FILENAME)=@_;
  if (! $islog) { return; }; # if not logging.
  my $TODAY = &GetLocalTime();
  my $logfile= $EZDIR . "log/$FILENAME-iar-$TODAY.LOG";
  open (LOGFILE,
    ">$logfile") or die "*CRASH* Could not open LOG $logfile :$!\n";
  print "\n Writing to log file: $logfile";  # clumsy
}
```

```
sub CloseLog
{ my ($islog);
     ($islog)=@_;
  if ($islog)
     { close LOGFILE;
     };
}

sub GetLocalTime
{ my ($sec, $min, $hour,
      $mday, $mon, $year,
      $wday, $yday, $isdst);
  ($sec, $min, $hour,
      $mday, $mon, $year,
      $wday, $yday, $isdst) = CORE::localtime(time);
  $year += 1900;      #fix y2k.
  $sec  = &DoubleDigit($sec );
  $min  = &DoubleDigit($min );
  $hour = &DoubleDigit($hour);
  $mday = &DoubleDigit($mday);
  $mon  = &DoubleDigit($mon );
  $mon ++;            #january is zero!
  return ("$year$mon$mday-$hour$min$sec");
}

sub DoubleDigit
{
   my ($i);
      ($i) = @_;
   if (length $i > 1)
      { return $i;
      };
   return "0$i"; # concatenate.
}
```

Here’s a routine that loads the connection string and returns it. The specified file is opened, and the string retrieved — it’s the first non-null line that doesn’t start with a % sign.

```
sub GetConnectionString
{ my ($EZDIR, $CONFILENAME);
     ($EZDIR, $CONFILENAME)=@_;
  my ($conf) = $EZDIR . "$CONFILENAME";
  open(CONF, $conf) or &Die("Can’t open $conf");
  my ($seek) = 20;           # max lines
  my ($opt);
  while ($seek > 0)
    { $_ = <CONF>;           # naive
```

```
      if (  (length $_ > 2)
         && ( ! /^\w*\%/ )
         )
         { chomp;
           $opt = $_;
           &Print ("\n Connection string: <$opt>", 3);
           close CONF;
           return ($opt);      # return string
         };
      $seek --;
    }
  close CONF;
  &Die("Connection string not found in <$conf>");
}
```

3.1 The connection string

Here's the file *connect-string.txt* containing the database connection string. It is written directly to the */ez* directory.

```
% Connection string for SAFERsleep database.
'Driver=SQL Server;Database=SaferSleep;Server=AHSL242;ServerName=IDASApplicationSe
```

4 Format translation: ez-xlate

We have three tasks here:

1. Turn a SAFERsleep IAR file into an XML file (Section 2.4.5);
2. Translate an XML file into a CSV file (Section 4.1.2);
3. Turn a CSV file into a .DAT file for replay via SAFERsleep (Section 4.2);

The following sections address the last two tasks. The IAR to XML translation has already been described above (Section 2.4.5).

4.1 Translate from XML to CSV

From within the XML data, we clip out and separate *data* tags in the following format:

```
<data><time>2008-03-31T10:09:42</time>
  <label>ETCO2</label><value>0.000000</value></data>
```

(The single line has been broken into two for the sake of legibility). One approach is to identify all distinct times and create an ordered array of these times. We then identify all labels, make an array for each label, and then enter values at each time (where the label exists).

4.1.1 Perl code: translate to CSV

The following perl program accepts a source file name. The program opens the source (XML) file, and using the same name, but with a CSV rather than an XML suffix, writes the file to the destination directory. If the destination file exists, it is simply overwritten!

The plan is:

1. Get file name from the command line
2. Read in whole XML file
3. Identify all distinct times, and store the times in order in an array called ALLTIMES. It may be best to use a counter and associative array, assuming that a later time contained in the file is always later in time than any earlier one, and associating a monotonically increasing count with each new time;

4. Identify the first label, create an array for it, and work through the entire XML file, snipping out each time/label/value triplet. Look up the time in ALLTIMES, obtain an index, and store the value at that index in the array for this label. (Warn if duplicate values are encountered for a particular time);

5. Repeat the preceding step for all labels;

6. Write the header;

7. For each time, obtain values for all label arrays in turn (writing NA if no value exists), and write the time and values in CSV format;

Note that pressure values are compound, and should be split up into three components. So, for example NBP:

```
<data><time>2008-03-31T10:24:13</time>
  <label>NBP</label><value>119.000000/57.000000(77.000000)</value></data>
```

...should be turned into three: NBP-sys, NBP-dia, and NBP-mean in the final translation (one value becomes three columns). We must code for this and potentially for other similar pressure readings.

When we translate from CSV to .DAT, we will need to identify such values and ‘re-translate’ accordingly.

```
# Perl program to translate anaesthetic monitor data from XML to CSV format:
#!/usr/local/bin/perl -w

my ($EZPATH) = "/ez/";
my ($OUTPATH) = $EZPATH . 'csv/';
my ($LOCALPATH) = $EZPATH . 'xml/';
my ($EXITCODE) = 0;    # code on exiting Perl

my ($DEBUG) = 3;    # 0 debug off, 1=overview, 2=detail, 3=nitpicking
my ($LOGPRINT) = 1; # print to log
my ($WARNINGS) = 0; # see usage

# the following are 'global':
my ($LOCALFILE) = $ARGV[0]; # get filename e.g. "S0012" WITHOUT suffix;
my ($PARAMS) = $ARGV[1];      # eg "debug=3,logprint=0"
if ($PARAMS =~ /debug=(\d)/ ) # command-line control!
   { $DEBUG = $1;
   };
if ($PARAMS =~ /logprint=(\d)/ )
   { $LOGPRINT = $1;
   };
```

```
&OpenLog($LOGPRINT, $LOCALFILE);# start writing to (error/debug) log

if ($LOCALFILE !~ /^S\d{4}/i )
   { &Die("Bad file name: <$LOCALFILE>.");
   };
my ($FILENAME) = $LOCALFILE . '.xml' ; # e.g. "S0012.xml"
my($LINECOUNT)=0;

$FILENAME = $LOCALPATH . $FILENAME;
open(FILE, $FILENAME) or &Die("Unable to open file <$FILENAME>");
&Print ("Debugging file: $FILENAME", 1);
my (@INDATA) = <FILE>;
close(FILE);

my ($coutf) = "$OUTPATH$LOCALFILE-s.csv";  # note -s suffix (for sAFERsleep!)
&Print ("\n Source file: <$FILENAME>\n Output path: <$OUTPATH>, \n Local file: $LO
open (COUTFILE, ">$coutf") or &Die (
     "*CRASH* Could not open output CSV FILE: $coutf :$!\n");

# now establish associative array of times:
my ($TIMECOUNT) = 0;
my (%ALLTIMES) = ();
my ($j);
my (%LABELS) = ();  # and array of labels, each of which will refer to an unnamed

foreach $j (@INDATA)
  { $LINECOUNT ++;
    if ($j =~ /<data><time>(.+)<\/time><label>(.+)<\/label><value>(.+)<\/value><\/
       { my $t = $1;
         my $lbl = $2;
         my $v = $3;
         if (! exists $ALLTIMES{$t})
            { $ALLTIMES{$t}=$TIMECOUNT;
              $TIMECOUNT ++;
              &Print ("\n line: $t c=$TIMECOUNT", 2);
            };
         my ($now) = $ALLTIMES{$t};
         # here's the tricky bit:
         # if label not yet defined, make array;
         if (! exists $LABELS{$lbl})
            { my (@lblarry) = ();
              $LABELS{$lbl} = \@lblarry;
            };
         ${$LABELS{$lbl}}[$now] = $v;   # against this label, store value for this
         &Print ("\n Storing value $v for label $lbl at time $now", 3);
       };
  };
```

```
my %TRIPLETS = ();
$TRIPLETS{'NBP'} = 1;  # signal a pressure.

#next, (debug) recover all values. First, the headers:
# if header in 'triplets', then make THREE headers.
print COUTFILE "Time,";
my (@headers) = keys %LABELS;
my ($h);
   foreach $h (@headers)
     { if (exists $TRIPLETS{$h})
          { &Print ("\n Triplet header: $h-s, $h-d, $h-m", 3);
            print COUTFILE "$h-s,$h-d,$h-m,"; # systolic,diastolic,mean
          } else
          { &Print ("\n Header: $h", 3);
            print COUTFILE "$h,";  # print header name
          };
     };

# next, print line-by-line:
# there is a catch:
#   for names in the TRIPLETS list (pressures), we must either
#   pull out 3 values (sys,dia,mean), or (if not there), insert 3 NAs.

my ($t, $v, $n);
foreach $t (sort keys %ALLTIMES)
  { print COUTFILE ("\n$t,");
    $n = $ALLTIMES{$t};  # get sequence number of time (or could just increment fr
    foreach $h (@headers)
      { $v = ${$LABELS{$h}}[$n];
        if (exists $TRIPLETS{$h}) # pressure in format "S/D(M)":
          { if (defined $v)
               { if ( $v =~ /(.+)\/(.+)\((.+)\)/ )
                    { # might have debug stmt here.
                      print COUTFILE "$1,$2,$3,";
                    } else
                    { print COUTFILE "NA,NA,NA,";
                      &Print( "\n WARNING: bad value($h) line $n = <$v>", 0);
                      $WARNINGS ++;
                      $EXITCODE |= 0x01;  # signal warning
                    };
               } else
               { print COUTFILE "NA,NA,NA,";
               };
          } else
          { if (defined $v)
               { print COUTFILE "$v,";
               } else
```

```
                { print COUTFILE "NA,";
                };
            };
        };
    };
close (COUTFILE);

if ($WARNINGS > 0)
   { print "\n ***NOTE*** There was/were $WARNINGS warning(s). \
 Please consult the log.";
   };
print("\n Finished!\n");
&CloseLog($LOGPRINT);
exit $EXITCODE;
```

A debugging print routine:

```
sub Print
{ my ($p, $level); # level is debugging level. 0=none, 1=overview, 2=detail, 3=pic
     ($p, $level)=@_;

  if ($DEBUG >= $level) # if level is 0, always print!
     {  if ($LOGPRINT)
           { print LOGFILE $p;
             return;
           };
        print $p; # console print
     };
}
```

Here's the logging routine, with associated subroutines. The log is written to the */ez/log* directory.

```
sub OpenLog
{ my ($islog, $LOCALFILE);
     ($islog, $LOCALFILE)=@_;
  if (! $islog) { return; }; # if not logging.
  my $TODAY = &GetLocalTime();
  my $logfile= "/ez/log/$LOCALFILE" . "-x2c-$TODAY.LOG";
  open (LOGFILE, ">$logfile") or die "*CRASH* Could not open LOG $logfile :$!\n Do
}

sub GetLocalTime
{ my ($sec, $min, $hour,
      $mday, $mon, $year,
      $wday, $yday, $isdst);
  ($sec, $min, $hour,
      $mday, $mon, $year,
```

```
      $wday, $yday, $isdst) = CORE::localtime(time);
  $year += 1900;      #fix y2k.
  $sec  = &DoubleDigit($sec );
  $min  = &DoubleDigit($min );
  $hour = &DoubleDigit($hour);
  $mday = &DoubleDigit($mday);
  $mon  = &DoubleDigit($mon );
  $mon ++;            #january is zero!
  return ("$year$mon$mday-$hour$min$sec");
}

sub DoubleDigit
{
   my ($i);
      ($i) = @_;
   if (length $i > 1)
      { return $i;
      };
   return "0$i"; # concatenate.
}

sub CloseLog
{ my ($islog);
     ($islog)=@_;
  if ($islog)
     { close LOGFILE;
     };
}
```

A death routine for fatal errors:

```
sub Die
{ my ($e);
     ($e)=@_;
  my ($l) = $LINECOUNT-1;
  &Print ("\n\n DIED: Data line = $l \n Message: $e", 0);
  $EXITCODE += 128;
  exit $EXITCODE;
##  die "\n Fatal";
}
```

4.1.2 A DOS batch file: xml2csv

Here's the batch file (*xml2csv.bat*), used to invoke the XML translation from the DOS command line:

```
echo off
cls
rem Translate XML to CSV
perl ez-xlate\ez-xml2csv.pl %1
ECHO Error level:
ECHO.%ERRORLEVEL%
```

It should be placed in the *ez* directory. The sole argument is to the source (XML) file. An example is in this test file, *test-x2c.bat*:

```
echo off
cls
echo Translate XML to CSV --- test only:
perl ez-xlate\ez-xml2csv.pl "test.xml" debug=2,logprint=0
ECHO Error level:
ECHO.%ERRORLEVEL%
```

4.2 Translate from CSV to .DAT

Given CSV files, we next need to translate the columns of CSV data into a format recognised by SAFERsleep. We write a custom Perl program to make this translation, from CSV to a SAFERsleep '.DAT' file.[6]

Note that the first line of the CSV file to be converted *must* contain timestamps in the format:

```
2008-03-31T10:09:42
```

There's another potential catch — if a mean pressure is absent, it's tempting to calculate this here. We do not do so. The NBP-m value must have been worked out and inserted into the source CSV file!

There are four major wrinkles here:

1. We wish to be selective in the columns we export from the CSV file;
2. Pressures are compound in the IDAS .DAT file, so we need to amalgamate three columns to create a .DAT pressure column.[7]
3. In their manual records, most anaesthetists do not differentiate between intra-arterial blood pressures and non-invasive blood pressures. We should accordingly render all pressures as 'NBP' in the .DAT file; [EXPLORE THIS]
4. IDAS doesn't respect the timestamp in the .DAT file, simply taking data for each 'sample' as needed. Regardless of other settings, in demonstration mode IDAS seems to take a sample as originating every 30 s, unless sample timestamps are identical, in which case the samples are taken as originating at the same time. Display in demonstration mode is every minute, so if we wish to skip samples we must:
   (a) Provide dummy samples 'every 30 s', using a modality that isn't commonly displayed;[8]
   (b) Provide our actual samples as required.

[6] Although ironically enough the original XML is close enough to make direct translation from XML to .DAT attractive.

[7] A subsidiary problem we must explore is the case where either the diastolic or systolic is missing. We will determine the mean according to the 'usual physiological' formula where we add a third of the pulse pressure to the diastolic.

[8] For example, 'ST3'; or disable display of the chosen item, from within IDAS.

We address the need to be selective in our export by having, in the CSV source directory, a file (*datrules.txt*) that provides rules for column selection. The rules are formatted as follows:

```
% Conversion rules for .CSV to .DAT:
"$D[0]","SpO2","{SpO2}"
"$D[0]","HR","{HR}"
"$D[0]","ETCO2","{ETCO2}"
"$D[0]","NBP","{NBP-s}/{NBP-d}({NBP-m})"
```

In the above file, always placed in the *csv* directory,[9] and called *datrules.txt*, all lines starting with % are comments. Other lines act as templates, where $D[0] is a place-holder for the time, and values in curly braces indicate the names of columns from which values will be obtained for every line (i.e. for each particular time). After substitution of values into the template, a line is written to the destination file, however, if the value in the source is 'NA', no line is written.

4.2.1 Perl code to translate to .DAT

Let's implement the above. In addition we will render all floating point numbers with six digits after the decimal point (all zeroes) as encountered in the .DAT files. We add 'epsilon' and then brutally truncate.

```
# Perl program to translate from CSV format to IDAS .DAT format:
#!/usr/local/bin/perl -w

use POSIX qw(floor); # for date calculations

my ($DEBUG) = 3;    # 0=no debugging, 1=overview, 2=detail, 3=nitpicking
my ($LOGPRINT) = 1;          # print to log
my($LINECOUNT)=0;
my ($WARNINGS) = 0;          # see usage
my ($EPSILON) = 0.0000005;   # tiny number.

my ($EZPATH) = "/ez/";
my ($OUTPATH) = $EZPATH . 'dat/';
my ($LOCALPATH) = $EZPATH . 'csv/';
my ($EXITCODE) = 0;          # code on exiting Perl

# the following are 'global':
my ($LOCALFILE) = $ARGV[0];   # source filename e.g. "S0013-m" NO suffix
my ($RULEFILE) = $ARGV[1];
if ($RULEFILE !~ /.+\.txt$/i )# invalid rule file
   {$RULEFILE = "datrules.txt";
   };
```

[9] We might later consider submitting the file name as a separate parameter to Perl.

```
my ($PARAMS) = $ARGV[2];     # eg "debug=3,logprint=0"
if ($PARAMS =~ /debug=(\d)/ )# command-line control!
   { $DEBUG = $1;
   };
if ($PARAMS =~ /logprint=(\d)/ )
   { $LOGPRINT = $1;
   };
&OpenLog($LOGPRINT, $EZPATH, $LOCALFILE);# start writing to (error/debug) log

my ($FILENAME) = "$LOCALFILE.csv";    # e.g. "S0013-m.csv";

# here, from $LOCALPATH, load file containing translation rules:
my $datrules = $LOCALPATH . $RULEFILE;
open (DATRULES, $datrules) or &Die ("Unable to open $datrules");
my (@DATRULES) = <DATRULES>; # load all lines
close (DATRULES);

$FILENAME = $LOCALPATH . $FILENAME;
open(FILE, $FILENAME) or &Die("Unable to open source file <$FILENAME>");
&Print ("Debugging file: $FILENAME", 1);
my (@INDATA) = <FILE>;
close(FILE);

my ($doutf) = "$OUTPATH$LOCALFILE.dat";
&Print ("\n Source file: <$FILENAME>\n Output path: <$OUTPATH>," .
          "\n Local file: $LOCALFILE \n Output file: <$doutf>", 2);
open (DOUTFILE, ">$doutf") or &Die (
     "*CRASH* Could not open output DAT FILE: $doutf :$!\n");

# obtain headers from input file...
$_ = $INDATA[0];                # get header line
my (@COLNAMES) = split /,/;     # get names
my ($cc) = 0;                   # clumsy count
my ($cn);
my (%COLLOOKUP) = ();
foreach $cn (@COLNAMES)
  {
   $COLLOOKUP{$cn} = $cc;
   $cc ++;
  }; # now,given name, can get position of column!

# HERE WILL PROCESS datrules.txt into RULES array:
my (@RULES)=();
my ($RCOUNT) = 0;
my ($dr);
foreach $dr (@DATRULES)
  {
    if ($dr =~ /^\w*\%/ ) # if comment
```

```
       { #  --ignore--
       }
    elsif (length $dr < 4)
       { # ignore short lines
       }
    else
       { $RULES[$RCOUNT]= &PrepareRule($dr, %COLLOOKUP);
         # the above can DIE if bad line!
         $RCOUNT ++;
       };
  };

my ($ILEN) = $#INDATA;  # do NOT add 1.
my ($LASTTIME) = '';    # redundant
my ($THISTIME) = '';
while ($LINECOUNT < $ILEN)
  {
    $LINECOUNT ++;  # skip first line
    $_ = $INDATA[$LINECOUNT];
    my (@D) = split /,/;
    (@D) = &TruncArray(@D); # convert to 6 decimals ?!
    # here might check for dud lines
      # (not at present)
    # here fill in dummy values for IDAS:
      $LASTTIME = $THISTIME;
      $THISTIME = $D[0];
      if (length $LASTTIME > 0)
         { &InterpolateTime($THISTIME, $LASTTIME);
           # the above clumsily prints to DOUTFILE
         };
    # show progress (for command line)..
    print '.';
    my ($r);
    foreach $r (@RULES)
      {
        my ($opt) = &ExtractDAT($r, @D);
        if (length $opt > 0)
            { print DOUTFILE "$opt\n";
            };
      };
   };
close (DOUTFILE);

if ($WARNINGS > 0)
   { &Print ("\n ***NOTE*** There was/were $WARNINGS warning(s). \
 Please consult the log.", 0);
   };
print("\n Finished!\n");
```

```
&CloseLog($LOGPRINT);
exit $EXITCODE;


sub InterpolateTime
{ my ($now, $then);
     ($now, $then)=@_;
  my ($sEPSILON) = 5;  # up to 25 seconds is fine.

  if ($now !~ /(\d{4})-(\d{2})-(\d{2})T(\d{2}):(\d{2}):(\d{2})/ )
     { &Die("Bad date in CSV file: $now");
     };
  my ($jdnow) = &Julian($1,$2,$3,$4,$5,$6,0);
  if ($then !~ /(\d{4})-(\d{2})-(\d{2})T(\d{2}):(\d{2}):(\d{2})/ )
     { &Die("Bad (old) date in CSV file: $then");
     };
  my ($jdthen) = &Julian($1,$2,$3,$4,$5,$6,0);
  my ($sec) = $jdnow - $jdthen;
  $sec = int($sEPSILON + 60*60*24*$sec);
  if ($sec < 0)
     { &Die("Bad time change ($sec) in CSV: $now -> $then");
     };
  if ($sec < 60)  # sampling interval is 30s.
     { &Print("\n Sampling interval: $sec ($now, $then) -> $jdnow - $jdthen \n", 3);
       return;
     };
  if ($sec > 15*60) # ridiculuous
     { &Die("Time interval ($sec) too long! $now -> $then");
     };
  &Print("\n Sampling interval: $sec ($now, $then) -> $jdnow - $jdthen \n", 3);
  my ($delta) = 1.0/(2*60*24); # 30 seconds.
  my ($yy,$mm,$dd,$hr,$mi,$se);
  while ($sec > 59)
     { $sec -= 30;
       # here could print any old value, but try for 'honesty':
       $jdthen += $delta;
       ($yy,$mm,$dd,$hr,$mi,$se)=&Gregorian($jdthen);
       print DOUTFILE '"' . "$yy-$mm-$dd" . 'T' . "$hr:$mi:$se" . '","T3","0.000000"' . "\n" ;
     };
}
```

Here's a copy of our Julian/Gregorian subroutines, stolen from our data capture Perl script in Section 2.3.

```
sub Julian
{ my ($fy, $fm, $fd, $fh, $fmi, $fs, $ff);
     ($fy, $fm, $fd, $fh, $fmi, $fs, $ff)=@_;

  my ($f);
  $f= 367*$fy - int(7*($fy+int(($fm+9)/12))/4)
              - int(3*(int(($fy+($fm-9)/7)/100)+1)/4)
              + int(275*$fm/9)+$fd+1721028.5
              + ($fh + ($fmi + ($fs+ "0.$ff")/60)/60)/24;
  return $f;
}

sub Gregorian
{ my ($jd);
     ($jd)=@_;
```

```
  my ($EPSILON) = 0.000001;
  $jd += $EPSILON;

  my ($Z, $R, $G, $A, $B, $C);
  my ($year, $month, $day);

  $Z = floor($jd - 1721118.5);
  $R = $jd - 1721118.5 - $Z;
  $G = $Z - 0.25;
  $A = floor($G / 36524.25);
  $B = $A - floor($A / 4);
  $year = floor(($B+$G) / 365.25);
  $C = $B + $Z - floor(365.25 * $year);
  $month = int((5 * $C + 456) / 153);
  $day = $C - int((153 * $month - 457) / 5) + $R;
  if ($month > 12)
     { $year = $year + 1;
       $month = $month - 12;
     };

    my ($gd) = 0.5 + $jd - int($jd);
     if ($gd > 1)   # Julian starts at midday.
        { $gd -= 1;
        };

    my($gh, $gmi, $gs);
            $gh = $gd; # clumsy
            $gh *= 24;
            $gmi = $gh;
            $gh = int($gh);
            $gmi -= $gh;
            $gmi *= 60;
            $gs = $gmi;
            $gmi = int($gmi);
            $gs -= $gmi;
            $gs *= 60;

  return ($year, &DoubleDigit($month),
                  &DoubleDigit(int($day)),
                  &DoubleDigit($gh),
                  &DoubleDigit($gmi),
                  &DoubleDigit(int($gs))
          );
}

sub DoubleDigit
{
   my ($i);
      ($i) = @_;

   if (length $i > 1)
      { return $i;
      };
   return "0$i"; # concatenate.
}
```

Here’s the rule-processing routine. A rule is in the format:

```
$q = "|$D[0]|,|SpO2|,|$D[x]|"
```

. . . where x is an integer greater than zero. Even the following is possible:

```
$q = "|$D[0]|,|NBP|,|$D[1]/$D[2]($D[3])|"
```

Given a string along the lines of:

```
"$D[0]","SpO2","{SpO2}"
```

We turn it into a rule, by identifying all items in curly braces, and replacing each by the relevant $D[x] reference. If a column reference isn't found, we Die.

```
sub PrepareRule
{ my ($dr, %COLLOOKUP);
     ($dr, %COLLOOKUP)=@_;
  my ($ismore) = 1;
  my ($pre, $v, $post);
  while ($ismore)
    {
     if ( $dr =~ /(.*)\{(.+?)\}(.*)/ )
        { ($pre, $v, $post) = ($1, $2, $3);
          if (! exists $COLLOOKUP{$v})
             { &Print ("Fatal! Column $v not found.", 0);
               &Die ("Column $v not foudn");
             };
          $v = $COLLOOKUP{$v};  # $v is numeric
          $dr = $pre . '$D[' . $v . ']' . $post;
        } else
        { $ismore = 0;
        };
    };
  $dr =~ s/"/\|/g; # replace all quotes with pipes
  my ($r) = '$q = "' . $dr . '"';
  &Print ("\n Debug: prepared rule is <$r>", 3);
  return ($r);
}
```

And next, the template-filling routine that uses a rule. We handle a rule by:

1. Evaluating it, so that each $D[x] value is substituted with a real value;
2. Scanning it for "NA" — if this is present, we return a null string;
3. Replacing all pipes with double quotes in the string $q.

```
sub ExtractDAT
{  my ($r, @D);
      ($r, @D)=@_;
   my ($q);
```

```
  &Print ("\n Debug: applying <$r> to ( @D )", 3);
   eval ($r); # evaluate string, put result in $q !
   if ($@)    # if failed..
      { &Print("\n *Warning: Missing variable or other 'eval' failure: $@", 0);
        $WARNINGS ++;
        $EXITCODE |= 0x08;   # warning code: eval failure.
        return (''); #
      }
   # HMM. what if $D[x] doesn't exist?
   if ($q =~ /NA/ )
      { return ('');
      };
   $q =~ s/\|/"/g; # replace all pipes with quotes
   return ($q);
}
```

A clumsy truncation routine, coercing all numbers to floats with six digits:

```
sub TruncArray   # brutal
{ my (@D);
     (@D)=@_;

  my (@X) = ();
  my ($i) = 0;
  my ($f, $tr);
  &Print ("\n Trunc: ", 3);

  foreach $f (@D)
    {
      $tr = &Trunc($f);
      &Print (" <$f> -> <$tr>;", 3);
      $X[$i] = $tr;
      $i ++; # bump index
    };
  return (@X);
}

sub Trunc
{ my ($f);
     ($f) = @_;

  if ($f =~ /^(\d+\.\d{6})/ )  # if 6 digits or more
     { $f += $EPSILON;  # prevent massive truncation error
       if ($f =~ /^(\d+\.\d{6})/ ) # in case...
          { return ($1);
          };
     };
  if ($f =~ /^(\d+\.)(\d*)/ )
```

```
     { my ($pre) = $1;
       my ($post) = $2;
       while (length $post < 6)
         { $post .= '0';
         }
       return ("$pre$post");
     };
  if ($f =~ /^(\d+)$/ )
     { return ( $1 . ".000000");
     };
  return ($f);  # unchanged
}
```

A debugging print routine:

```
sub Print
{ my ($p, $level); # level is debugging level. 0=none, 1=overview, 2=detail, 3=pic
     ($p, $level)=@_;

  if ($DEBUG >= $level) # if level is 0, always print!
     {  if ($LOGPRINT)
           { print LOGFILE $p;
             return;
           };
        print $p; # console print
     };
}
```

Here are our 'standard' logging routines:

```
sub OpenLog
{ my ($islog, $EZPATH, $LOCALFILE);
     ($islog, $EZPATH, $LOCALFILE)=@_;
  if (! $islog) { return; }; # if not logging.
  my $TODAY = &GetLocalTime();
  my $logfile= $EZPATH . "log/$LOCALFILE-c2d-$TODAY.LOG";
  open (LOGFILE, ">$logfile") or die "*CRASH* Could not open LOG $logfile :$!\n Do
}

sub GetLocalTime
{ my ($sec, $min, $hour,
      $mday, $mon, $year,
      $wday, $yday, $isdst);
  ($sec, $min, $hour,
      $mday, $mon, $year,
      $wday, $yday, $isdst) = CORE::localtime(time);
  $year += 1900;      #fix y2k.
  $sec  = &DoubleDigit($sec );
```

```
  $min  = &DoubleDigit($min );
  $hour = &DoubleDigit($hour);
  $mday = &DoubleDigit($mday);
  $mon  = &DoubleDigit($mon );
  $mon ++;            #january is zero!
  return ("$year$mon$mday-$hour$min$sec");
}

sub DoubleDigit
{
   my ($i);
      ($i) = @_;
   if (length $i > 1)
      { return $i;
      };
   return "0$i"; # concatenate.
}

sub CloseLog
{ my ($islog);
     ($islog)=@_;
  if ($islog)
     { close LOGFILE;
     };
}
```

A death routine for fatal errors:

```
sub Die
{ my ($e);
     ($e)=@_;
  my ($l) = $LINECOUNT-1;
  &Print ("\n\n DIED: Data line = $l \n Message: $e", 0);
  $EXITCODE |= 0x80;
  exit $EXITCODE;
  ## die "\n Fatal";
}
```

4.2.2 A DOS batch file: csv2dat

Here's the batch file (*csv2dat.bat*), used to invoke the CSV translation from the DOS command line. It assumes a single parameter, and (redundantly) submits the standard translation rule file.

```
echo off
cls
rem Translate XML to CSV
```

```
perl ez-xlate\ez-csv2dat.pl %1 datrules.txt
ECHO Error level:
ECHO.%ERRORLEVEL%
```

It should be placed in the *ez* directory. The two arguments are respectively the path to the source file and the destination *directory*.

And here's a test file (*test-c2d.bat*) that requires no arguments:

```
echo off
cls
echo Translate CSV to SAFERsleep .DAT file: test only!
perl ez-xlate\ez-csv2dat.pl  "test" "datrules.txt" "debug=3,logprint=0"
ECHO Error level:
ECHO.%ERRORLEVEL%
```

Here's a similar test to convert *captured* data, *called test-c2d.bat* that takes the output *out-T0001.csv* from the *test-captur.bat* file, and converts it to a .DAT:

```
echo off
cls
echo Translate CSV to SAFERsleep .DAT file: a test of CAPTURED data.
perl ez-xlate\ez-csv2dat.pl  "out-T0001.csv"
ECHO Error level:
ECHO.%ERRORLEVEL%
```

5 ez-replay: Replaying data via SAFERsleep

It is vitally important to note that the following code should only be run *after* SAFERsleep has been reconfigured to point to a testing server, and *not* the live database. This prevents the actual database from becoming clogged with 'dummy' copies of records. You should also disable the antibiotic warning in the IDAS .ini file.

We launch IDAS in demonstration mode, and play the DEMO.DAT file (suitably modified by us), grabbing screens from the program every minute and storing each screen as a sequentially numbered PNG file.

5.1 AutoIt code for monitor replay

Here's the code:

```
;; ===================================================================;
; AutoIt Version: 3.0
; Program to copy a .DAT file to the IDAS file DEMO.DAT
; Then run IDAS in demonstration mode,
; and repeatedly capture screen (every minute), using AutoIt (!)
; and writing each captured image as a PNG file.
; Repeat for multiple .DAT files.

; AutoIt includes:
#Include <ScreenCapture.au3>

; miscellaneous variables:
Const $ONTOP = 0x40000;

$SLACKTIME = 300;                       ;; ms
$PSPWAIT = 2000;                        ;; ms
$PSPWAIT2 = 2000;                       ;; ms
$ONEMINUTE = 60000;                    ;; ms = 60s
;; $ONEMINUTE = 10000;                    ;; 10 s for debugging

; ---------------------------------------------------;
; PREPARATION: Load numbers etc. Identify a file
Global $CURRENTFILE = '';
Global $CAPTURECOUNT= '';

$EZPATH = "\ez\";
$REPLAYPATH = $EZPATH & "ez-replay\";
$LISTNAME = $REPLAYPATH & "SSLIST.TXT";


; get current file (if left off in middle) = CurrentFile.txt
```

```
$CURRFILENO = FileReadLine($REPLAYPATH & "CurrentFile.txt",1); ;;open|read|close
;; DEBUG ONLY:
;; $CURRFILENO = 1;
if @error <> 0 then    ;; if failed
  MsgBox($ONTOP, "FATAL Error", "Failed to open CurrentFile.txt");
  Exit
EndIf
;; at present do not rewrite this, so must be updated manually.
;;MsgBox($ONTOP, "Debug", "File index is " & $CURRFILENO,10);


$FD = GetFileList($LISTNAME, $CURRFILENO)
If $FD[0] = 0 Then
  MsgBox ($ONTOP, "Fatal Error", "Line not found in file list, No=" & $CURRFILENO)
EndIf
$CURRENTFILE = $FD[1];
$CAPTURECOUNT =$FD[2];
;; here might check the above are valid and reasonable.
;;   MsgBox($ONTOP, "Debug", "File is " & $CURRENTFILE & ", Count: " & $CAPTURECOU

$CAPTUREINTERVAL = $ONEMINUTE - $SLACKTIME -$PSPWAIT -$PSPWAIT2;
$stub = StringSplit($CURRENTFILE, "-"); ;; will remove suffix

$FAKENHI     = 'SS' & $stub[1];
;; MsgBox($ONTOP, "DEBUG", "Fake NHI is " & $FAKENHI);

$SAVEPATH = "\ez\ez-captur\images\";
$SAVENAME = $SAVEPATH & $CURRENTFILE;

; ------------------------------------------------------;
; ACTUALLY RUN THINGS: (CLUMSILY)
; run safersleep:
;; here might get path to idas from registry, but for now:
$idaspath = "C:\Program Files\Safer Sleep\IDAS Client\IDAS.exe";
If FileExists($idaspath) = 0 Then
  MsgBox($ONTOP, 'Error', "IDAS.exe not found: " & $idaspath);
  Exit;
EndIf
Run($idaspath);
If WinWaitActive("SAFERsleep","",30) = 0 Then  ;; allow 30s for boot:
  MsgBox($ONTOP, "Error", "Failed to load SAFERsleep")
  Exit
EndIf

Sleep (15000);  ;; clumsy
If WinExists ("Information", '') Then
  WinActivate("Information",'');
  Send("{ENTER}");    ;; this is for the bloody INFORMATION box
```

```
  ;; if cannot connect to server.
Else
  MsgBox($ONTOP, "Dx", "Info box NOT detected. Click OK");
  Send("{ENTER}");  ;; allow manual 'wait' (or not)
EndIf

Sleep(3000);
If WinExists ("SAFERsleep Login Screen", '') Then
;;  MsgBox($ONTOP, "Dx", "Login Screen detected");
  Send("{ENTER}");   ;; Hit enter at Login screen;
Else
  MsgBox($ONTOP, "Dx", "Login Screen NOT detected. Click OK");
Send("{ENTER}");   ;; Hit enter at Login screen;
EndIf

Sleep(3000);
If WinExists ("Information", '') Then
   ;;  MsgBox($ONTOP, "Dx", "Info box 2 detected");
   Send("{ENTER}");   ;; Hit enter at Login screen;
Else
  MsgBox($ONTOP, "dX", "Info box 2 NOT detected. Click OK");
  Send("{ENTER}");   ;; Hit enter at Login screen;
EndIf

Sleep(3000);
If WinExists ("Main Menu", '') Then
  ;; MsgBox($ONTOP, "Dx", "Main menu detected");
Send("{ENTER}");   ;; new case (default)
Else
  MsgBox($ONTOP, "Ooops!", "No main menu");
  Exit;
EndIf

;; ========================BIG OUTER LOOP FOR SINGLE ANAESTHETICS ===============
$ISMORE = 1;
While $ISMORE

; HERE WE OVERWRITE THE DEMO.DAT FILE
$currentpath = $EZPATH & 'dat\' & $CURRENTFILE & '.dat';
If FileExists($currentpath) = 0 Then
  MsgBox($ONTOP, 'Error', "Source DAT file not found: " & $currentpath);
  Exit;
EndIf
$destpath = "C:\Program Files\Safer Sleep\IDAS Client\devicedata\DEMO.DAT";
If FileExists($destpath) = 0 Then
  MsgBox($ONTOP, 'Error', "DEMO.DAT not found! " & $destpath);
  Exit;
EndIf
```

```
FileCopy($currentpath, $destpath, 1);  ;; overwrite
;; might check for success

; START NEW ANAESTHETIC:
;; MsgBox($ONTOP, "Debug", "New Anaesthetic - waiting..")
Sleep(1000)
If WinExists ("Start New Anaesthetic", "") = 0 Then
  Send("{ENTER}");
  MsgBox($ONTOP, "HMM. Try again", "Click OK to start new anaesthetic..");
Else
  ;; MsgBox($ONTOP, "Seems ok", "The menu exists??");
EndIf

WinWaitActive("Start New Anaesthetic")
Send($FAKENHI);
Sleep(500);
Send("{ENTER}");

; enter fake patient details:
;; MsgBox($ONTOP, "Debug", "Enter fake details")
WinWaitActive("Edit Patient");
Send("Demo");
Sleep(50);
Send("{TAB}");
Sleep(50);
Send($CURRENTFILE);
Sleep(50);
Send("{TAB}");
Sleep(50);
Send("26/12/1900");
Sleep(50);
Send("{TAB}");
Sleep(50);
Send("F");
Sleep(50);
Send("{TAB}");
Sleep(50);
Send("{ENTER}");
Sleep(500);

$MAINWINDOWNAME = "SAFERsleep - " & $FAKENHI;
  ;; e.g. SAFERsleep - SSS0012   Demo, S0012-s    Anaesthetic chart - ***DEMO MODE
  ;;MsgBox($ONTOP, "Debug", "Wait for main window" & $MAINWINDOWNAME)
If WinExists($MAINWINDOWNAME) = 0 Then
  MsgBox($ONTOP, "Error", "Window not found: <" & $MAINWINDOWNAME & ">");
  Exit;
Else
;;  MsgBox($ONTOP, "Success", "Window found: " & $MAINWINDOWNAME);
```

```
  Sleep(3000);   ;; just in case?
EndIf

WinActivate($MAINWINDOWNAME, '');
ControlClick($MAINWINDOWNAME, "", "[CLASS:TcxCustomComboBoxInnerEdit; INSTANCE:9]"
  ;;MouseClick("left", 132, 352);
  ;;Send("HOP");
Send("ggg");       ;; debug name on test station. ugh.
Send("{ENTER}");

; start anaesthetic:
Sleep(1000);

; --- next get parameters for main window:
$xy = WinGetPos($MAINWINDOWNAME); ;; -> x,y,w,h
$WINX = $xy[0] + 5;
$WINY = $xy[1] + 0.16*$xy[3];
$WINW = $xy[2] * 0.83;
$WINH = $xy[3] * 0.85;
$WINH2 = $xy[3];

$xy = ControlGetPos($MAINWINDOWNAME, "", "[CLASS:TdxNavBar; INSTANCE:1]");
MouseClick("left", 20+$xy[0], (0.07 * $WINH2)+$xy[1]);   ;; clumsy hack

;; ------------------------------------;;
;; LOOP: REPEATEDLY CAPTURE AND SAVE:
for $MYCOUNT = 1 to $CAPTURECOUNT
   CheckAntibiotic();  ;; ensure clear antibiotic warning

   $WHOLEFILE = $SAVENAME & "-" & LeadZero($MYCOUNT);
   If FileExists($WHOLEFILE & ".PNG") Then
     FileDelete($WHOLEFILE & ".PNG");
   EndIf

;; if we use AutoIt to capture the screen, under XP/Win2K we can write a PNG!
  Local $hbmp;
  $hbmp = _ScreenCapture_Capture('', $WINX, $WINY, $WINW, $WINH)
  _ScreenCapture_SaveImage($WHOLEFILE & ".PNG" , $hbmp);   ;; THIS MAKES A PNG!
 WinActivate($MAINWINDOWNAME);  ;; ??need

 Sleep($CAPTUREINTERVAL);
next;   ;; end of repeat capture loop
;; ------------------------------------;;

$response = MsgBox($ONTOP+1, "Debug", "Next anaesthetic?", 60);
   ;; wait 1 min or default to 'yes'
If $response = 2 Then
   $ISMORE = 0;
```

```
Else
  $CURRFILENO = $CURRFILENO + 1;
  $FD = GetFileList($LISTNAME, $CURRFILENO)
  If $FD[0] = 0 Then
     $ISMORE = 0;
  Else
    $CURRENTFILE = $FD[1];
    $CAPTURECOUNT =$FD[2];
    ;; here might check the above are valid and reasonable.
    MsgBox($ONTOP, "Debug", "File is " & $CURRENTFILE & ", Count: " & $CAPTURECOUN
    WinActivate($MAINWINDOWNAME);
    MouseClick("left", 946, 426);   ;; hack,hack!
       Sleep(100);                  ;; hack
    Send("{ENTER}");   ;; do you want to end anaesthetic YES..
       Sleep(100);                  ;; hack
    Send("{TAB}{ENTER}");  ;; do NOT print
    ;; Here, to start another anaesthetic, Send {ENTER}.
       Sleep(3000);                  ;; enough time to save etc..
    Send("{ENTER}");  ;; start new anaesthetic
  EndIf
EndIf

Wend;   ;; END BIG OUTER LOOP FOR SINGLE ANAESTHETICS

EndCapture();     ;; END OF MAIN SECTION

;; ----------------------------------------------------- ;;
;; FUNCTIONS:
;; 1. LeadZero ($n)
;; This checks length and if under 3, prepends zeroes.

Func LeadZero($n)
  While StringLen($n) < 3
      $n = "0" & $n;
  Wend
  Return($n);
EndFunc;

Func CheckAntibiotic()
  If WinExists ("Antibiotic Warning") = 1 Then
   WinActivate("Antibiotic Warning")
   Send("{ENTER}");
  EndIf
EndFunc;


;; ----------------------------------------------------- ;;
;; EndCapture: close all applications
```

```
Func EndCapture()

  MouseClick("left", 946, 426);   ;; hack,hack!
     Sleep(100);                     ;; hack
  Send("{ENTER");
     Sleep(100);                     ;; hack
  Send("{TAB}{ENTER}");
  ;; Here, to start another anaesthetic, Send {ENTER}.
  ;;; but we will..
  Send("{TAB}{ENTER}");
  Sleep(50);                     ;; hack
  Send("{ENTER}");

  ;; close MWSnap:
  WinClose("MWSnap");
Exit;                             ;; done

EndFunc;

;; ------------------------------------------------------ ;;
;; GetFileList
Func GetFileList($fname, $lineno)
; get file list with number of frames to capture
  Dim $FDAT[3];
  $FDAT[0] = 0;  ;; no data
  $ln = 1 + $lineno;  ;; first line is a header line!

  $myline = FileReadLine($fname, $ln);
  If @error <> 0 Then    ;; if failed
    Return ($FDAT);  ;; FAIL
    Exit
  EndIf
  ;;MsgBox($ONTOP, 'Debug', 'Line value for line ' & $ln & '(' & $lineno & ') is '
  $FDAT = StringSplit($myline,",");
  If $FDAT[0] < 2 Then
     Dim $FDAT[3];
     $FDAT[0] = 0;  ;; failed.
  EndIf
  Return($FDAT);
EndFunc
```

5.2 Invoke ez-replay from DOS

The file *replay.bat* can be used to invoke the AutoIt script that plays .DAT files back through IDAS and screen-captures the display every minute:

```
echo off
```

```
cls
rem Invoke AutoIt script to playback .DAT files
ez-replay\ez-replay.au3
```

It should be placed in the *ez* directory. At present it takes no arguments.

6 ez-shw: An interactive web-based application

Here we provide stand-alone HTML code that demonstrates the core functionality of our web application. Note that the actual HTML used in our project is *written dynamically* by our PHP code, but is very similar to the following code, which is intended mainly for demonstration and debugging purposes.

This application sequentially displays PNG files within a web-page that allows the anaesthetist viewing the page to scroll through the record in faster than real-time, and annotate the record where they believe they would have intervened. A record is kept of the viewing anaesthetist's progress, and these data are written to an SQL database for later analysis.

6.1 HTML code

Here we describe the pages used in displaying the records. The pages are written in HTML; user interaction and data-gathering are implemented using JavaScript ('Ecmascript').

We still need to do the following here:

1. Display the number of minutes elapsed, rather than the clumsy reminder after 1 minute ('one minute has elapsed').
2. Add in an 'artefact' button;
3. Ensure minute by minute default play continues;
4. We really need to interpolate values every minute, otherwise we are screwed by the automatic recording.

6.1.1 Main page: display and data acquisition

The main HTML page is called *mainpage.htm*:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html lang="en">
<head>
    <title>Main Page: display anaesthetic record</title>
    <link type="text/css" rel="stylesheet" href="css/eZstyle.css">
<script type="text/javascript">
    <!--

// ==================== LOAD IMAGES ========================== //
```

```
TOTALIMAGES = 75;  //  *datum1*
MYIMAGES = new Array(TOTALIMAGES);
PATIENTID = "S0012-s"; //  *datum2*
THISIMAGE = 0;
// PreloadImages();  // here might check cookie for start image

NOW = 0;
TIMESTAMP = 0;
// ONEMINUTE=15000;  // debug: every 15s.
ONEMINUTE = 60000;
ONESECOND=1000;
MINUTETIMER = '';  // vitally important GLOBAL!
AAGH = '';         // likewise

MYCLOCK = new Array(12);
TICKER = 0;
TICKTIME = ONEMINUTE/12;
QUICKTIME = ONESECOND/12;
CLOCKTICK = '';
LoadClock();

BUFFERING = 0;
MYIMAGES[0]= new Image(824,446);  // [later, taller]
MYIMAGES[0].src = 'png/START.png';
MODIM = parseInt(TOTALIMAGES/12);

addLoadEvent(PreloadImages); // set up preloading..


// -------- event loader ----------------- //

function addLoadEvent(func) {
  var oldonload = window.onload;
  if (typeof window.onload != 'function') {
    window.onload = func;
  } else {
    window.onload = function() {
      if (oldonload) {
        oldonload();
      }
      func();
    }
  }
}

// ------------------clock stuff ----------------------- //

function LoadClock()
```

```
{var i = 0;
  while (i < 12)
    { MYCLOCK[i] = new Image(48,48);
      var cname = "images/clk" + i + ".png";
      // alert("Debug: Clock image is " + myname);
      MYCLOCK[i].src = cname;
      i ++;
    }
}

function Tick4()  // don't SetClock
{ TICKER ++;
  if (TICKER > 11)
     { TICKER = 0;
     };
  document.images.clock.src = MYCLOCK[TICKER].src;
}

function Tick5()
{ TICKER ++;
  if (TICKER > 11)
     { TICKER = 0;
       TICKTIME = 5000; // revert
     };
  document.images.clock.src = MYCLOCK[TICKER].src;
  SetClock(TICKTIME);
}

function StopClock()
{
 clearTimeout(CLOCKTICK);
}

function SetClock(t)
{ CLOCKTICK=setTimeout("Tick5()", t);
}

function SpeedTime()
{ TICKTIME = QUICKTIME;
  clearTimeout(CLOCKTICK);  // in case
  SetClock(TICKTIME);
}

function ZeroClock()  // [? need]
{
  document.images.clock.src = MYCLOCK[0].src;
}
```

```
// ========================================================== //
// buffered image loading (sequential)

function DOnext()
{ if ((BUFFERING % MODIM) == 0)
     { Tick4();
     };
  if (++BUFFERING < TOTALIMAGES)
     { loadNextImage();
     } else
     { // ZeroClock();
       document.getElementById('loadingLabel').style.visibility='hidden';
       document.images.mainimage.src = MYIMAGES[0].src;
       NextOn();
     };
}

function loadNextImage()
{ MYIMAGES[BUFFERING] = new Image(824,446);  // [later, taller]
  MYIMAGES[BUFFERING].onload = DOnext;
  var serial = MakeSerial(BUFFERING, 3);  // serial number
  var myname = "png/" + PATIENTID + "-" + serial + ".PNG";
  // alert("Debug: File is " + myname);
  MYIMAGES[BUFFERING].src = myname;
}

function PreloadImages()
{ NextOff();
  IntOff();
  ArtOff();
  if ( document.getElementById('loadingLabel'))
     { document.getElementById('loadingLabel').style.visibility='visible';
     };
  BUFFERING = 1;
  loadNextImage();
}


function MakeSerial(i, lgth)
{ while (i.toString().length < lgth)
    { i = "0" + i.toString();
    };
  return (i);
}


// see: http://www.sean.co.uk/a/webdesign/javascriptdelay.shtm
function pausecomp(millis)
```

```
{
var date = new Date();
var curDate = null;

do { curDate = new Date(); }
while(curDate-date < millis);
}

function GetTimestamp()
{ var date = new Date();
  var yyyy = date.getFullYear();
  var mm   = MakeSerial(1 + date.getMonth(), 2);  // zero-based?!
  var dd   = MakeSerial(date.getDate(), 2);
  var hh   = MakeSerial(date.getHours(), 2);
  var mi   = MakeSerial(date.getMinutes(), 2);
  var ss   = MakeSerial(date.getSeconds(),2);
  return ( yyyy + '-' + mm + '-' + dd + ' ' + hh + ':' + mi + ':' + ss);
}

// ====================== TIMERS/IMAGES ============================ //

// SetMinTimer: wrapper for setTimeout.
// links timer to function RemindAfter1min.
//
function SetMinTimer(t)
{ MINUTETIMER=setTimeout("RemindAfter1min()", t);
}

// ClearMinTimer: trivial: clear timeout.
//
function ClearMinTimer()
{ clearTimeout(MINUTETIMER);
}

//function MessageOff()
//{ document.getElementById('elapsedTime').style.visibility="hidden";
//}

// Display Elapsed time:
//
function MessageOn()
{ jane = document.getElementById('elapsedTime');
  while (jane.firstChild)
    { jane.removeChild(jane.firstChild);
    };
  MYW = document.createTextNode( THISIMAGE + " min");
  jane.appendChild(MYW);
  jane.style.visibility="visible";
```

```
}

// MessagePaused: similar to the above but indicate 'pause':
//
function MessagePaused()
{ jane = document.getElementById('elapsedTime');
  while (jane.firstChild)
    { jane.removeChild(jane.firstChild);
    };
  MYW = document.createTextNode("(paused)");
  jane.appendChild(MYW);
  jane.style.visibility="visible";
}


function RemindAfter1min()
{ SkipToNextImage();
}

// SkipToNextImage: automatic movement to next image.
function SkipToNextImage()
{ HideThanks();
  ClearMinTimer();
  SetMinTimer(ONEMINUTE);
  ShowNextImage();
}

// MoveToNextImage: invoked after storing data (below)
//
function MoveToNextImage()
{ TIMESTAMP = GetTimestamp();
  ShowNextImage();
  SetMinTimer(ONEMINUTE);
  SpeedTime();
}

// DoNext: Invoked on clicking next. Simply hides thanks, moves.
function DoNext()
{ if (THISIMAGE == 0)
     { document.getElementById('elapsedLabel').style.visibility='visible';
       ArtOn();
       IntOn();
     };
  HideThanks();
  ShowNextImage();
  SpeedTime();  // advance clock quickly
  BrieflyDisableNext();
}
```

```
function NextOff()
{ clearTimeout(AAGH);  // in case next re-enabling is pending!!
  document.getElementById('nextbutton').disabled=true;
}

function NextOn()
{ document.getElementById('nextbutton').disabled=false;
}

function IntOff()
{ document.getElementById('intbutton').disabled=true;
}

function IntOn()
{ document.getElementById('intbutton').disabled=false;
}

function ArtOff()
{ document.getElementById('artbutton').disabled=true;
}

function ArtOn()
{ document.getElementById('artbutton').disabled=false;
}



function BrieflyDisableNext()
{ NextOff();
  AAGH = setTimeout("ReEnableNext()",ONESECOND);
}

function ReEnableNext()
{ ClearMinTimer(); // in order to set it in a moment.
  NextOn();
  SetMinTimer(ONEMINUTE-ONESECOND);
}

// ShowNextImage: simply display next image after incrementing count:
//
function ShowNextImage()
{ THISIMAGE ++;
  if (THISIMAGE > TOTALIMAGES)
     { ClearMinTimer();
       ArtOff();
       IntOff(); // disable
       alert("Replay completed. Thank you!");
```

```
      // HERE WILL SAVE, MOVE TO NEXT PATIENT!
      return;
    };
  MessageOn();
  document.images.mainimage.src = MYIMAGES[THISIMAGE].src;
}

function HideThanks()
{ document.getElementById('thanks1').style.visibility="hidden";
}

function ShowThanks()
{ document.getElementById('thanks1').style.visibility="visible";
}

// ==================== DATA RECORDING ========================== //
ALLDATA = new Array(200);  // hmm. what is maximum?
DATUMCOUNT = 0;

function Intervene()
{ HideThanks();
  NextOff();
  ArtOff();
  ClearMinTimer();
  StopClock();
  // setTimeout("ClearMinTimer()",2000); // just in case clear set by NEXT!! ????
  MessagePaused();
  TIMESTAMP = GetTimestamp();
  RevealTopInputs();
}

function ShowTheRest()
{
  firstcomment = NoPipes(document.myform.intervention.value);
  if (firstcomment.length < 3)
     { alert("Please describe your intervention!");
       return(false);
     };
  RevealBottomInputs();
  document.getElementById('oka').style.visibility='hidden'; // hide self!
  // alert ("Intervention recorded");
}

function CancelIntervention()
{ alert ("Intervention Cancelled");
  MessageOn();
  SetMinTimer(ONEMINUTE);
  SetClock(TICKTIME);
```

```
  ClearAllValues();
  NextOn();
  ArtOn();
  HideAllInputs();
}

function StoreAndThanks()
{ var datastring = '';
  var firstcomment = NoPipes(document.myform.intervention.value);
  var secondcomment =NoPipes(document.myform.variablecomment.value);
  var sbp = OneZero(document.myform.sbp.checked);
  var dbp = OneZero(document.myform.dbp.checked);
  var hr =  OneZero(document.myform.hr.checked);
  var spo2 =OneZero(document.myform.spo2.checked);
  var etco2=OneZero(document.myform.etco2.checked);

  datastring = "t="+TIMESTAMP +"|c1="+firstcomment +"|c2="+secondcomment +"|sbp="+
  alert ("Debug data are: " + datastring);
  ClearAllValues();
  ALLDATA[DATUMCOUNT] = datastring;
  DATUMCOUNT ++;

  ShowThanks();
  MoveToNextImage();
  // alert ("Thank you. Please continue!");
  NextOn();
  ArtOn();
  HideAllInputs();
  // alert("Should have moved");  // this would not be displayed!
}

function ClearAllValues()
{
  document.myform.intervention.value = '';
  document.myform.variablecomment.value='';
  document.myform.sbp.checked=0;
  document.myform.dbp.checked=0;
  document.myform.hr.checked=0;
  document.myform.spo2.checked=0;
  document.myform.etco2.checked=0;
}

function NoPipes(c)
{ var pindex = c.indexOf('|');
  while ( pindex > -1)
      { b4 = c.substring(0,pindex);
        aftr = c.substring(pindex+1);
        c = b4 + aftr;
```

```
        pindex = c.indexOf('|');
      };
  return(c);
}

function OneZero(v)
{ if (v == 0)
     { return (0);
     }
  return(1);
}

// ==================== GRAPHIC ALTERATIONS ============================ //

function HideAllInputs()
  { document.getElementById('texta').style.visibility='hidden';
    document.getElementById('oka').style.visibility='hidden';
    document.getElementById('cancela').style.visibility='hidden';
    document.getElementById('checkb1').style.visibility='hidden';
    document.getElementById('checkb2').style.visibility='hidden';
    document.getElementById('checkb3').style.visibility='hidden';
    document.getElementById('checkb4').style.visibility='hidden';
    document.getElementById('checkb5').style.visibility='hidden';
    document.getElementById('textb').style.visibility='hidden';
    document.getElementById('okb').style.visibility='hidden';
    document.getElementById('labela').style.visibility='hidden';
    document.getElementById('labelb').style.visibility='hidden';
    document.getElementById('labelc').style.visibility='hidden';
    document.getElementById('labeld').style.visibility='hidden';
    document.getElementById('labele').style.visibility='hidden';
    document.getElementById('labelf').style.visibility='hidden';
    document.getElementById('labelg').style.visibility='hidden';
    document.getElementById('labelh').style.visibility='hidden';

   // document.getElementById('').style.visibility='hidden';
  }

function RevealTopInputs()
  { document.getElementById('texta').style.visibility='visible';
    document.getElementById('oka').style.visibility='visible';
    document.getElementById('cancela').style.visibility='visible';
    document.getElementById('labela').style.visibility='visible';
  }

function RevealBottomInputs()
  { document.getElementById('checkb1').style.visibility='visible';
    document.getElementById('checkb2').style.visibility='visible';
    document.getElementById('checkb3').style.visibility='visible';
```

```
    document.getElementById('checkb4').style.visibility='visible';
    document.getElementById('checkb5').style.visibility='visible';
    document.getElementById('textb').style.visibility='visible';
    document.getElementById('okb').style.visibility='visible';
    document.getElementById('labelb').style.visibility='visible';
    document.getElementById('labelc').style.visibility='visible';
    document.getElementById('labeld').style.visibility='visible';
    document.getElementById('labele').style.visibility='visible';
    document.getElementById('labelf').style.visibility='visible';
    document.getElementById('labelg').style.visibility='visible';
    document.getElementById('labelh').style.visibility='visible';
  }

 //-->
</script>

</head>
<body bgcolor='#CCCCCC'>

  <FORM name='myform' action=''>  <!-- FIX ME: FILL IN ACTION -->

<table cellpadding='0' cellspacing='0' width='100%' border='0'> <!-- main table-->
 <tr>
   <td width='10%'><div align='center'>
   <span class='invisiblewarning' id='loadingLabel'>Loading..</span>
   <img name="clock" src="images/clk0.png" alt="" width="48" height="48">
   <b class='invisiblewarning' id='elapsedLabel'>Elapsed time:<br></b>
   <span class='invisiblewarning' id='elapsedTime'></span>
   </div></td>
   <td width='90%' bgcolor='#CCCCCC' colspan='3' align='center'>
   <img class='prominent' src="images/INTRO.png" alt="" width='824' height='446'
    name='mainimage' border='3'>
   </td>
</tr>

<tr><td colspan='3' bgcolor='white'><hr></td></tr>

<tr bgcolor='white'><!-- patient info at bottom-->
 <td  colspan='3' align='left'>
<b>Case: S0012</b>  Appendectomy. A well young man.     
 </td>
</tr>

<tr><td colspan='3' bgcolor='white'><hr></td></tr>

 <tr bgcolor='white'>
   <td  align='center'>
     <p><input type="button" onClick='DoNext()' id='nextbutton' value="  
```

```
    <p><input type="button" onClick='Intervene()' id='intbutton' value="INTERVENE
    <p><input type="button" onClick='alert("Here will deal with artefacts!")' id=
  </td>

  <td valign='top' align='center'>
      <!-- intervention box goes here -->
      <br><br><span class='invisible' id='labela'>State intervention:</span>
        <br><input class='invisible' type='text' size='50' name='intervention' id
        <p><input class='invisible' type='button' onClick='ShowTheRest()' id='oka
           <input class='invisible' type='button' onClick='CancelIntervention()'
 </td>

 <td>
   <table width='100%' border='0'>
     <tr><td colspan='2'><span class='invisible' id='labelb'><b>Variable(s) you m
     </td></tr>
     <tr><td width='30%'>
       <input  class='invisible' type="checkbox" name="sbp" id='checkb1' onClick=
            <span  class='invisible' id='labelc'>Systolic BP </span>
       <br> <input  class='invisible' type="checkbox" name="dbp" id='checkb2' onC
            <span  class='invisible' id='labeld'>Diastolic BP</span>
       <br> <input  class='invisible' type="checkbox" name="hr" id='checkb3' onCl
            <span  class='invisible' id='labele'>Heart Rate  </span>
       <br> <input  class='invisible' type="checkbox" name="spo2" id='checkb4' on
            <span  class='invisible' id='labelf'> SpO<sub>2</sub></span>
       <br> <input  class='invisible' type="checkbox" name="etco2" id='checkb5' c
            <span  class='invisible' id='labelg'> EtCO<sub>2</sub></span>
        </td>
        <td width='70%' align='center'>
         <br><span class='invisible' id='labelh'>Optional comment:</span>
         <br><input  class='invisible' type='text' size='50' id='textb' name='var
         <p><input  class='invisible' type='button' onClick='StoreAndThanks()' id
         <p><span class='invisible' id='thanks1'>Thank you!</span>

        </td>
       </tr>
     </table>
 </td>
</tr>
</table><!-- end main table-->

</FORM>

</body>
</html>
```

6.1.2 Cascading style sheet

Here's *eZstyle.css*:

```
/* eZstyle.css - my tiny style sheet */
/* 1. Broad brush strokes */
body { margin-left: 0%; margin-right: 0px;
       color: #00008B;
       font-family: Garamond, serif;
       text-align : left;
     }
div { margin-top : 1em;
    }
h1,
h2,
h3,
h4,
h5,
h6 { color : #000040;
     font-family: sans-serif;
     margin-bottom: 0;
   }
p {
    margin-top: 0.3em;
  }
hr { border : 1px #AAAAAA;
     border-style : dashed;
   }
/* 2. My special classes */
.invisible {
            visibility : hidden;
           }
.invisiblewarning {
             visibility : hidden;
             color : black;
             font-size : 120%;
             font-weight : bold;
           }

.prominent { border-width : thick;
           border-style : solid;
           border-color : black;
           }

.heavybox { border-width : thick;
            border-style : solid;
            border-color : #2F1D1D;
            margin-top : 1 ;
            margin-right : 1px ;
```

```
            margin-bottom : 1 ;
            margin-left : 1 ;
          }

/* 3. Useful SPAN and other tags */

/* 4. Pseudo stuff */
a:hover {color : red ;
         background-color : black;}

/* X. area for testing follows: */
/* the end of the style sheet */
```

7 Database definition

The database is written in SQL, and implemented on the Web using mySQL. We have elsewhere discussed our method for partially automating capture of data on manually written anesthetic records. Here we take these data and run them through the SaferSleep process. We capture the resulting screens as PNG graphics, and then display the graphics on the web. An anaesthetist scrolls through such a graphic record sequentially, and indicates 'points of significance' where she/he would likely have intervened, given their basic knowledge of the anaesthetic and prior data on the display. (Where someone has already intervened based on the record, a subsequent intervention should only be in the context of the previous intervention having been performed when they stated they would have intervened [EXPLAIN BETTER]).

The SQL database is loosely based on the one we created for the STrICT study. We kick off with a table of tables, which lists all of the tables within the database *in the order in which they were created*! This allows us to re-import backed up data without relational problems. The name of the table is METATABLE, and we clearly have to make it before the other tables!

```
CREATE TABLE METATABLE (
 metatable integer,
   constraint badMetatable primary key (metatable),
 TableName varchar(32)
                   );
```

Now that we've sorted that out, here's PERSON:

7.1 PERSON

```
CREATE TABLE PERSON (
  person integer,
  constraint badPerson primary key (person),
  pExpiration integer,
  pLoginName varchar(16),
  pPassword varchar(32),
  pSID varchar(32),
  pValue varchar(255),
  FirstQualified integer,
  pMade timestamp
                   );
```

We've moved the Forename, Surname and PersonRole fields from the PERSON table to the PERSDATA table below. We've also moved **FirstQualified** from the PERSDATA table to here, as it can only ever have one value, and belongs here.

The additional fields required by our PHP program are:

pExpiration is a UNIX timestamp used to handle time-out if a user overstays their welcome (at present we time out after a fixed, maximum period of 1 hour, rather than monitoring for inactivity);

pLoginName is the log-in name of a user (if appropriate);

pPassword which accommodates an md5-encoded password;

pSID is a session ID; and

pValue is used to store 'stuff' (The most recent page visited by this user).

We've also added **pMade**, a timestamp for the creation of a person entry, as a 'bookkeeping' entry.

Here's the PERSONROLE table, used below.

```
CREATE TABLE PERSONROLE (
  personrole integer,
  constraint badPersonRole primary key (personrole),
  rText varchar(32)
                    );

INSERT INTO PERSONROLE (personrole, rText)
   VALUES (0, 'anonymous'),
          (2, 'anaesthetist'),
          (192, 'superuser');
```

We add the role of 'superuser' for the database administrator to use, and we have a special value of 'nobody' just in case. Next we have the ETHNICITY table.[10]

```
CREATE TABLE ETHNICITY (
  ethnicity integer,
  constraint badEthnicity primary key (ethnicity),
  EthnicName varchar(32) );

INSERT INTO ETHNICITY (ethnicity, EthnicName)
    VALUES ( 1, 'Maori'),
           ( 2, 'NZ European'),
           ( 3, 'Asian'),
           ( 4, 'Pacific Islander'),
           ( 5, 'Other');
```

[10] It might be argued that a separate table is needed relating an individual and multiple ethnic origins, but we won't go into this level of detail.

7.2 PERSDATA

```
CREATE TABLE PERSDATA (
  persdata integer,
  constraint badPersdata primary key (persdata),
  pdCreated timestamp,
  Person integer,
  constraint badpdPerson foreign key (Person)
    references PERSON,
  pdForename varchar(32),
  pdSurname varchar(32),
  pdGender integer,
  PersonRole integer,
  constraint badpdPersonRole foreign key (PersonRole)
    references PERSONROLE,
  CurrentSpecialty integer,
    constraint BadpdCurrentSpecialty foreign key (CurrentSpecialty)
      references CURRENTSPECIALTY,
  CurrentLocation integer,
    constraint BadpdCurrentLocation foreign key (CurrentLocation)
      references CURRENTLOCATION,
  Ethnicity integer,
    constraint badPersEthnicity FOREIGN KEY (Ethnicity)
      REFERENCES ETHNICITY,
  PhysicalAddress varchar(128),
  Email varchar(128),
  Telephone varchar(32)
                   );
```

We toss in **pdGender** for reasons of common sense, and we've added a **pdCreated** field to allow us to determine when a particular entry was made. Gender codes are 1 for female and 2 for male.

CurrentSpecialty is coded as below and refers to their main anaesthetic interest (???); **CurrentLocation** is where the person mainly practises. Here are the relevant tables:

```
CREATE TABLE CURRENTSPECIALTY (
  currentspecialty integer,
  constraint badCurrentSpecialty primary key (currentspecialty),
  SpecialText varchar(32)
                   );

INSERT INTO CURRENTSPECIALTY (currentspecialty, SpecialText)
   VALUES (0, 'nil'),
          (1, 'General'),
          (2, 'Cardiothoracics'),
          (3, 'Obstetrics'),
          (4, 'other');
```

We might wish to augment the above table somewhat.

```
CREATE TABLE CURRENTLOCATION (
  currentlocation integer,
  constraint badCurrentLocation primary key (currentlocation),
  LocationText varchar(32)
                  );

INSERT INTO CURRENTLOCATION (CurrentLocation, LocationText)
   VALUES (1, 'Mainly public'),
          (2, 'Mainly private'),
          (3, 'Similar public/private');
```

7.3 ANRECORD

ANRECORD refers to an anesthetic record, captured either manually or automatically. We will create an anonymised PERSON entry that refers to a given patient, to facilitate data analysis.

```
CREATE TABLE ANRECORD (
  anrecord integer,
    constraint badAnrecord primary key (anrecord),
  Manual integer,
  PngFilename varchar(32),
  Person integer,
  constraint badAnrecPerson foreign key (Person)
    references PERSON
                  );
```

Manual is 1 for manually recorded scenario, 0 for machine-recorded. PngFilename is the name (without 001–nnn suffix) of PNG files.

7.4 ONESESSION

```
CREATE TABLE ONESESSION (
  onesession integer,
    constraint BadOnesession primary key (onesession),
  osStart TIMESTAMP,
  osEnd TIMESTAMP,
  Assessor integer,
    constraint BadosAssessor foreign key (Assessor)
      references PERSON,
  Anrecord integer,
    constraint badOsAnrecord foreign key (Anrecord)
      references ANRECORD
                    );
```

7.5 RATING

First, let's create a small table which contains the text describing each rating item. We'll call it RATING:

```
CREATE TABLE RATING (
  rating integer,
    constraint BadRating primary key (rating),
  Onesession integer,
    constraint badRatingSession foreign key (Onesession)
      references ONESESSION,
  raText varchar(255),
  raSBP   integer,
  raDBP   integer,
  raHR    integer,
  raSPO2  integer,
  raETCO2 integer,
  raTime integer
                  );
```

The five references to measured parameters are set to 1 only if these were documented as having influenced the anaesthetist's decision to intervene. The *raText* field contains the description of the decision to intervene. The raTime field contains the *frame* at which this particular intervention occurred — the count starts at 1 (for the first frame).

7.6 Frills

The SALT table is used for logging on using md5-encoded passwords:

```
CREATE TABLE SALT (
  salt integer,
    constraint badSaltkey primary key (salt),
  sTime integer,
  sValue integer
                  );
```

The **sTime** is a UNIX integer time (32 bits) used to see whether a particular salt value has expired, and the **sValue** entry is a pseudorandom number. We won't here initialise the hundred entries in the SALT table, reserving that for the first time we try to check a password.

We also create a transaction log, which details user, their IP address, a timestamp for the transaction, a code for what was done, and an optional description field. Here we go ...

```
CREATE TABLE FORENSIC (
  forensic integer,
    constraint badForensicKey primary key (forensic),
  Person integer,
    constraint badForensicPerson foreign key (Person)
      references PERSON,
  fIp varchar (15),
  fTime TIMESTAMP,
  fCode integer,
  fText varchar (128)
                    );
```

The above table is not yet in use, and we don't (yet) have a separate table for the **fCode** field.

7.7 UIDS

In this final table, we create seeds for generating unique IDs for all of the other tables. The SQL standard doesn't directly provide automatic key generation, and although dialects do provide thinks like auto-incrementing fields or permit user-defined functions to fill the gap we will not use these facilities as they vary between databases. We will rather take keys as we need them from the UIDS table. Each new primary key for a new table entry will be fetched from the UIDS table. We don't have key seed fields for tables that will not ordinarily acquire new rows.[11]

The process of updating ('auto-incrementing') these source fields is rather tricky, and database 'features' designed to prevent errors may get in our way. Problems only really arise where several users are concurrently accessing the same field in UIDS, but we must account for this eventuality. The accessory fields beginning with 'aux' are used to accommodate this necessity — they *must* start off with values identical to the corresponding key fields.

```
CREATE TABLE UIDS (
  uids integer,
   constraint badUids primary key (uids),
  uPerson integer,
  uPersdata integer,
  uAnrecord integer,
  uOnesession integer,
  uRating integer,

  auxPerson     integer,
  auxPersdata   integer,
```

[11] They would have to be manually updated within the database.

```
  auxAnrecord   integer,
  auxOnesession integer,
  auxRating     integer
                  );

INSERT INTO UIDS (uids,
                  uPerson,
                  uPersdata,
                  uAnrecord,
                  uOnesession,

                  auxPerson,
                  auxPersdata,
                  auxAnrecord,
                  auxRating)
   VALUES        (1, 1000, 1000, 1000, 1000,
                  1000, 1000, 1000, 1000);

INSERT INTO PERSON (person, pLoginName, pPassword)
   VALUES          (1, 'Superuser', 'PasswordGoesHere');

INSERT INTO PERSDATA (persdata, Person, pdForename, pdSurname,
                      PersonRole)
   VALUES            (1, 1, 'Mr', 'Superuser',
                      192);
```

The last few statements create a single super-user, who will be able to log on with the username ‘Superuser’, and the relevant password. Note that as things stand this ‘naked’ password is inserted into the database, and will allow the Superuser to log on. It is wise for him/her to then immediately alter the password; even smarter would be to insert an md5-encoded password into the above code.

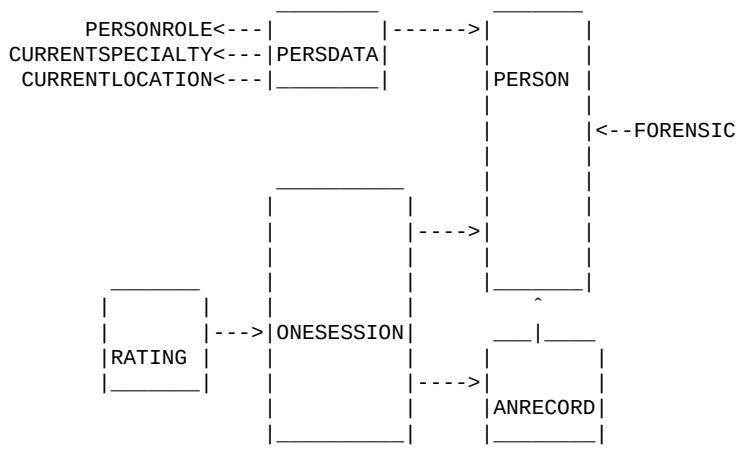

Table 3: Database table relationships

7.8 Database structure

Table 3 provides a simple overview of the table relationships in our database. An arrow from table A to table B indicates that table A *references* table B. Major tables are shown as blocks.

7.9 Testing the database

We tested the database in two phases.

Testing on a DOS machine

We first run Dogwagger 2.1 to generate the SQL code in the file *eZ.sql* in the directory */ez/ez-shw/sql/*. We create a database called eZ in a convenient SQL (for example, the old freeware Ocelot SQL[12]) and then submit the code manually.[13]

Internet testing

In our testing, it's wise to consider everything, but in particular file names and directory paths to be case-sensitive. We create a database in mySQL and upload a PHP script that submits our *eZ.sql* file, as follows.

We log onto **anaesthetist.com** using FTP commander and in the *public_html* folder, make a directory called *eZ*, as well as a subdirectory of *eZ* called *sql*. We then upload the file *eZ.sql* from our local PC directory (*ez/ez-shw/sql/*) to the *sql* directory, and the PHP script *database_create.php* described in Section 7.10 to the *eZ* directory.

Before we run the PHP script, we log on to **anaesthetist.com** using CPANEL, and create the database *anaes2_eZ* from within the mySQL database section. After creating the database, we add the system administrator with all privileges. We manually insert the administrator and password into the connection script script called *eZ_database_connect.php*. Note that there is a potentially fatal flaw in CPANEL — it's possible to delete an entire database with one click, and no confirmation!

We've already uploaded *database_create.php* so all we need to do is upload the following:

1. eZ_database_connect.php, which we place in the relevant directory *above* the HTML public directory;

2. eZ_GLOBALS.php which goes into the *ez* directory;

These are described in the next section.

[12] Install Ocelot, permitting ODBC connectivity, and in Windows go Start—Control Panel—Administrative Tools—Data Sources. Under User DSN click on the [Add] button, select Ocelot from the list, and click [Finish]. In the data source name type in the letters *eZ* and you're done.

[13] In the Ocelot example, within a DOS box, from the *ocelot* directory type in *demo32* and press Enter. Paste the SQL in sequentially after typing in *disconnect default* and then *connect to 'eZ'*. Remember to COMMIT at the end!

7.10 PHP: Creating the SQL database

Before we create our database, we have several prerequisites. The first is a script that actually connects to the database.

eZ_database_connect.php

Here's the script. It requires the existence of the global variable $DEBUGGING, which we'll consider in a moment.

```
function eZ_database_connect ()
{
 GLOBAL $DEBUGGING;
 $handDB = @mysql_pconnect ("localhost",
                       "name_sysadmin",
                       "PASSWORDHERE") or die
   ('Cannot connect to database because: ' . mysql_error());
 if ($DEBUGGING)
    { print "\n You managed to connect to the database.";
    };
 if (! mysql_select_db ("anaes2_eZ", $handDB))
    { die (" You didn't manage to connect to eZ.");
    } else
    { if ($DEBUGGING)
         { print " You are connected to eZ.";
         };
    };
 return ($handDB);
}
```

Note that the relevant values for name_sysadmin, and PASSWORDHERE must be inserted manually, and the script then stored in the directory *above* the *public_html* directory on **anaesthetist.com**.[14]

eZ_GLOBALS.php

The following are a few globals (mainly constants) required by our program. The database time-out precedes the cookie time-out, so we can identify the time-out and generate an appropriate message.

```
# 1. time-out before forcing log-off:
define ( 'eZ_TIMEOUT', 60*60);    # 1 hour
define ( 'COOKIE_EXTRA_WAIT', 10000); # before cookie expiry
```

[14] We are here taking the approach of running PHP under Apache, rather than the optimal approach of running PHP as a CGI script.

```
#2. useful constants:
define ( 'SHOW_USER', 1);
define ('MAXFETCHTRIES', 1000);# see usage
define ('MAXKEY', 100000000);  # likewise
define ('FETCHTIMEOUT', 2000); # milliseconds

#3. Used to test status of person logging on:
define ( 'eZ_ADMINISTRATOR_MIN', 128);
define ( 'eZ_ASSESSOR_MIN', 19);
define ( 'eZ_ASSESSOR_MAX', 63);
define ( 'EVERYONE', 1000);
define ( 'NOBODY', 0);
define ( 'SALTSIZE', 100);
define ( 'SALTTIMEOUT', 300); # seconds

#4. Convenient globals:
$CONNECTIONPATH = '../../eZ_database_connect.php';
# above is for anaesthetist.com setup
$handDB = '';
$USERKEY = '';
$DEBUGGING = 0;

#5. nasty hacks:
 $CLUMSY_INDEX=0;
 $CLUMSY_INDEX2=0;
```

7.10.1 Database creation script

Here we create a combined HTML/PHP script called *database_create.php*. To create the database, ensure the file *eZ.sql* is in the correct location specified below, and then simply access the file *database_create.php* with your web browser:

http://www.anaesthetist.com/eZ/database_create.php

This is a 'one-off' page which will do nothing once the database has been created. In fact, it might be wise to completely remove this page from our server once we've made the database! Once created, the database can of course be viewed using phpMyAdmin.[15]

```
<head><title>Create the eZ database</title>
</head><body>

<h2>Create a database</h2>
```

[15] It's wise to immediately alter the password using phpMyAdmin if you've not inserted an encrypted password in the SQL creation file.

```
<p>Here we read a SQL script file, parse it, and submit the CREATE
statements contained within to mySQL. We will use this facility to
create an entire database!
<p>First, let's connect to the database. We use the eZ_database_connect
script, which must be installed in the correct directory.
<?php
 GLOBAL $handDB;
 $DEBUGGING = 1; # 0=off.
 require_once('eZ_GLOBALS.php');
 require_once($CONNECTIONPATH);
 $handDB = eZ_database_connect();
 if (! is_null($handDB) )
    { print "<p><b> Connected to eZ database...</b>";
    } else
    { print "<p> Database connection *FAILED*";
    };
?>

<p>We now have an open connection to the database,
with a handle in <i>$handDB</i>. We'll test to see whether
the database creation script has already been run using
a function called <b>is_eZ_populated</b>, which checks
for the existence of the UIDS table, and if the table
doesn't exist, we run the whole creation script.
<?php
function is_eZ_populated ($handDB)
{ $query = "SELECT uids FROM UIDS WHERE uids = 1";
  $ok = mysql_query( $query,
                     $handDB);
 return($ok);
}

function batch_sql_statements($handDB, $fname)
{ GLOBAL $DEBUGGING;
  $METACOUNT = 1;
  $fdata = file($fname);         # slurp in array of lines
  # here process lines as per our usual parser:
  # In the call to FetchNextLine, $query is by reference.
  $query = '';
  $linecount = 0;
  while (FetchNextLine ($query, $fdata))
   { $linecount ++;
     if ($DEBUGGING)
        { print ("<br> Debugging: $query");
        } else
        { print '.';
        };
     if (preg_match( '/\s*CREATE\s+TABLE\s+(\w+)\W/im',
```

```
                    $query, $matches)) # ver 0.64
        { $tblname = $matches[1];
          if ($tblname != 'METATABLE')
             { $iq = "INSERT INTO METATABLE (metatable, TableName)
               VALUES ($METACOUNT, '$tblname')";
               $METACOUNT += 1;
               if (! mysql_query($iq, $handDB))
                  { print "\n Error with metadata ($tblname)";
                  };
             };
        };
     if (! mysql_query( $query, $handDB) )
        { print ( "\n Error during SQL execution: " .
            mysql_error() );
          return 0; # fail
        };
     $query = '';
   };
  print ("<br> Number of lines submitted = $linecount");
 return (1); # success
}

function FetchNextLine(&$query, $fdata)
{ STATIC $LINE; # on first pass made zero!
  while ($LINE < count($fdata))
    { $txt = $fdata[$LINE]; # get next line
      $LINE ++;
      if (! preg_match ( '/^--/', $txt) ) # if not comment
         { $query = $query . $txt;
           if (preg_match ( '/\;\s*$/', $txt)) # terminal ';'?
              { return (1);
    };   };   };
  $LINE = 0; # in case re-invoke FetchNextLine !!
  return 0; # no more lines
}
?>

The relevant SQL initialisation file is called
<i>eZ.sql</i>, and must be present in
the sql subdirectory of the current directory,
or creation will fail.

<?php
 if (! is_null($handDB) ) # provided connected
    { if (! is_eZ_populated($handDB))
         { if (batch_sql_statements($handDB, 'sql/eZ.sql'))
              { print "<p> SUCCESS! eZ database created.";
                print "<p> <a href='login.php'>GO THERE!</a>";
```

```
                # here should commit!
                # mysql_query( 'COMMIT;', $handDB);
                # but mySQL auto-commits by default (ugh)!
              } else
              { print "<p> *ERROR*.
                          Failed to create eZ database!";
              };
          } else
          { print "<p> eZ database already exists (Duh)!";
          };
      # mysql_close($handDB); # removed: keep database open...
     };
?>
```

The above should be largely self-explanatory. If the *database_create.php* script succeeded, then A link will be created at the bottom of the page, pointing to the script *login.php*. We'll consider this next.

8 PHP access coding

Web pages described in the HTML Section (Section 6.1 above) are served up using scripts written in PHP version 5. Data obtained from the HTML pages are written to the SQL database using PHP.

8.1 Logging in

login.php

Here's the main log-in script, which generates a log-in page by writing HTML code together with a 'salt' number. The user types in their username and password, which the page encrypts and then POSTs to *InitialLogon.php* (See Section 8.1.1).

Note that apart from requiring the script *eZ_GLOBALS.php*, which we've already defined (it contains $CONNECTIONPATH), this script also needs *ancillary.php*, which is described later (Section 10).

In addition, because the HTML uses some javascript, we must create the *js* subdirectory of *eZ* on **anaesthetist.com**, and place the *md5.js* script there! Ultimately we may wish to insert one or more 'pretty' images that we'll then need to upload into the *eZ/images* directory, so we might as well make this directory now.

Note that if the javascript include isn't found (you failed to upload it) then the script will silently fail to perform correctly!

```
header ('Cache-control: no-cache' );
require_once('eZ_GLOBALS.php');
require_once($CONNECTIONPATH);
require('ancillary.php');
GLOBAL $handDB;
$handDB = eZ_database_connect();
if (! $handDB)
   { ## mysql_close($handDB);
     print ( "\n Failed to connect to database!" ); # ugh!
     readfile ('rescue.htm');
     return(0);
   };

 # seed pseudorandom number generator:
 $mt = microtime();
 if (! preg_match ( '/0.(.*) /', $mt, $matches))
    { $mt = time(); # ugh.
    } else
    { $mt = $matches[1];
    };
  srand($mt);
```

```
  # find a free row to use:
  $K = SALTSIZE;  # number of salt rows (global)
  $TIMEOUT = SALTTIMEOUT; # (global constant)
  $quiet = 20;    # attempts
  while ($quiet > 0)
    { $J = rand();  # hmm
      $I = $J % $K; # index into salt table
      $now = time(); # UNIX time
      $qry = "SELECT sTime, sValue FROM SALT WHERE salt = $I";
      list($STIME, $SVAL) = GetSQL($handDB, $qry, 'get salt');
      if (strlen($STIME) > 1) # if row exists ..
         { if (($now - $STIME) > $TIMEOUT) # if timed out
              { $quiet = 0; # force exit, signal 'found row'.
              };
         } else # SET UP the SALT table: [can later remove]
         { $c = 0;
           while ($c < $K)
             { $r = rand();
               $qry = "INSERT INTO SALT (salt, sTime, sValue)
                         VALUES ($c, 100, $r)"; # force time out
               DoSQL($handDB, $qry, 'insert salt row');
               $c += 1;
             };
         };
      $quiet -= 1;   # prevent infinite loop
    };
  if (! $quiet) # 20 tries failed if $quiet==0
     { readfile('busy.htm');
       exit();
     };
  # we now have valid row $I and salt value $J:
  $qry = "UPDATE SALT SET sTime = $now,
                          sValue = $J WHERE
            salt = $I";
  DoSQL($handDB, $qry, 'store salt J value');
  $SALT = $J;
  ## print ("<p>Debug: row is $I and salt is $J<p>");


print <<<HTML0
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en">\n
<head><title>Log in to eZ</title>
<LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
  <script src='js/md5.js' type='text/javascript'></script>
  <script type='text/javascript'>
  <!--
function SubmitLogon(myform)
```

```
  { if (myform.username.value.length < 4)
       { alert ("Invalid user name. Must be at least 4 characters!");
         return(false);
       };
    password = myform.pswd.value;
    if (password.length < 6)
       { alert ("Invalid password. At least 6 characters are required");
         return(false);
       };
    salt = "$SALT"; // now, md5 as required:
    // alert("Debug: You input password " + password);
    myform.pswd.value = hex_md5(salt + hex_md5(password));
    // alert("Debug: encrypted value for " + hex_md5(password) + " is " + myform.p
    return (true);
  };
  //-->
  </script>
</head>
<body>
<div align="center">
<h2>eZ: log in to <i>live</i> database</h2>

 <FORM name="initiallogon"
   ACTION="InitialLogon.php"
   METHOD="POST"
   onSubmit="return SubmitLogon(this)" >
<input type="hidden" name="salt" value="$SALT">
<table>
<tr><td>User Name:
</td><td colspan="2">
<input type="text" name="username" size="16">
</td></tr>
<tr><td>Password:</td><td>
<input type="password" name="pswd" size="16">
</td>
<td><input type="submit" name="submit" value="Log in"></td></tr>
</table>
</form>

</div>
</body>
</html>
HTML0;
```

Formerly we used *login.htm* which is now largely just a redirection stub, however the advantage of still providing this page to users as the initial logon is that it confirms that Javascript is enabled, without which eZ won't work. Here's the file:

```
<head>
```

```
  <title>Log-in to eZ (redirection)</title>
  <script type='text/javascript'>
  <!--
   window.location.replace("login.php");
  //-->
  </script>
</HEAD>
<BODY>
<h2>Loading eZ...</h2>
<p>If nothing happens in a few seconds, click
<a href="http://www.anaesthetist.com/eZ/login.php">here</a>,
but it's likely that Javascript is disabled on your machine,
and you will need to enable it before continuing!
</BODY>
```

Here's *rescue.htm*, mentioned previously.

```
<head><title>Something bad happened?</title></head>
<body>
<h2>Woops! Database connection failed.</h2>
Click here: <a href="http://www.anaesthetist.com/">
 Return to Anaesthetist's main page</a>
</body>
```

We also require a helper HTML file *busy.htm* which we invoke if there is no login slot available after 20 tries. This is at present a stub, but later we might have a Javascript 30 second count-down, after which the user can try again to log in!

```
<HEAD>
  <title>Very busy at present...</title>
  <LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
</HEAD> <BODY> <h2>Hmm. We seem to be <i>very</i> busy...</h2>

<p>... or you left the login screen idle
for over five minutes, and we got tired
waiting :)

<p>Please wait about 30 seconds, and then click
<a href="http://www.anaesthetist.com/eZ/login.php">here</a> to
try again!
</BODY>
```

[FIX THE ABOVE TO MAKE IT MORE USER-FRIENDLY, ALSO CHECK WHY IT SOMETIMES GETS PRESENTED INAPPROPRIATELY].

8.1.1 InitialLogon.php

The following script is invoked from the form specified by login.php (Section 8.1).[16]. If the submitted parameters (username and password) are garbage, we return to *login.php*. Only after we've sanitised these values do we try to connect to the database, after which we log on (if we can) using the *processLogon* function described in Section 8.1.2.

This script requires *ValidFx.php* (Section 8.2) and on success passes control *mainpage.php* (Section 9.1). The variable $LOGGED_IN is set and passed to *mainpage.php*; otherwise mainpage will again try to confirm that the user is logged in! The function CheckCode is contained in *ancillary.php*.

The component function *Force_Html_InitialLogon* makes use of two tiny HTML files, *oops1.htm* and *oops2.htm*.

```
GLOBAL $USERKEY;
GLOBAL $handDB;
GLOBAL $THISPAGE;
$THISPAGE = '';

require_once('ValidFx.php'); # our login validation script
$username = $_POST['username'];
$pswd = $_POST['pswd'];
$salt = $_POST['salt'];
## print "<p>Debugging: password is '$pswd', salt is '$salt'";
if (  (strlen($username)>32)
    ||(strlen($pswd) > 80)
    ||(strlen($username)<4)
    ||(strlen($pswd)< 6)
   ) { ## print "<p>Bad password/user name";
       Force_Html_InitialLogon('Invalid password/username.');
       exit();
     };
Sanitise($username);
Sanitise($pswd);
CheckCode($salt, 'bad SALT value. Worrying!');

$handDB = eZ_database_connect();
$USERKEY = processLogon($username, $pswd, $handDB, $salt);

# Prepare to load main page:
$LOGGED_IN = 1;
$qry = "SELECT PersonRole, pValue
          FROM PERSON, PERSDATA
          WHERE PERSON.person = PERSDATA.Person
```

[16]But it might be accessed directly — in this case, the POST variable *username* will not be set in which case we simply direct the user to the *login.htm* file (Section 8.1)

```
            AND PERSON.person = $USERKEY";
  list ($userstatus, $OLDPAGE) =
       GetSQL($handDB, $qry, 'get status/old');
  $success =
"USER=\"$username\"|STATUS=\"$userstatus\"|OLDPAGE=\"$OLDPAGE\"";
  # clumsy duplication of validate_login (ValidFx.php):
  require ('mainpage.php');
```

The above duplication of $success is clumsy, but if we repeat the log-in attempt at *mainpage.php* we encounter a problem — the altered cookie is only seen at the next browser page load, so login fails!

If everything worked according to plan, we load the main page (Section 9.1). Here's that processLogon function, where we confirm the md5 encrypted user password using the schema described below.[17]

It is very desirable to use Javascript md5 encoding of all passwords prior to submission. We encrypt the password provided by the user in the first page, and check the submitted value in the destination page. To encrypt the password we set the password value to the md5 string provided by Paul Johnston's Javascript routine hex_md5 (See Section **??**).

Next, we implement a similar but more complex encryption of user passwords. Now, if we simply encrypt the password, then anyone intercepting this *md5-encrypted* password can always use it to log on.[18] So we must provide something ('salt') from the server side so that every new session is distinct. Salt values must also expire, and be used once only.[19] We proceed as follows.

1. We already have a way of determining whether a user is logged on or not — we store a session ID, and clear it at log-out. Sessions expire.

2. When someone requests a log-on, we provide a 'salt' value in the log-on page. The server also increments the SALT table index once the salt value has been provided.

3. We generate the salt value as a pseudorandom integer J, and take I = J modulo K, where K is the number of items in the SALT table, using I as an index into SALT.

4. If that SALT table entry hasn't expired, we try again (max of 20 times, at which time we say 'busy', encourage wait of 30s, and then allow resubmission);

[17] Note that this approach has not been (formally) cryptographically tested.
[18] There's also the matter of rainbow tables!
[19] This approach cannot protect against 'Man in the middle' attacks.

5. If SALT entry is expired (5 min is the cutoff) then we overwrite table row I with the current timestamp and the value J_I = J. We then ...
6. Submit the value J in the login form that the user sees.
7. The user responds with md5(J+md5(pwd)), login, and the user J value J_u(which should be identical to J), where '+' represents concatenation of the two strings.
8. On receiving the user response with J_u, we examine the SALT table item at index I = J_u mod K. If J_u is not equal to J_I or the time has expired, we say "invalid/timeout" and they can retry after a 30s wait. If both are ok we ...
9. Calculate md5(J+md5(pwd)) for the given user, and if not equal to the supplied value, say "invalid/timeout"; otherwise ...
10. Store md5(J+md5(pwd)) as the session ID against that user name both in the database and as a cookie at the user end, and proceed.

In order to implement the above, we need a SALT table with say 100 entries. If we were concerned about repeated, frequent logging attempts messing around with access by other users, we might log the IP address of each unsuccessful attempt, and check subsequent attempts — e.g. institute delay/lock-out for repeat offenders.

8.1.2 processLogon

The following routine must be invoked before anything is written as HTML. It either exits on failure, or returns the user key.

```
function processLogon($username, $pswd, $handDB, $salt)
{
  $K = SALTSIZE; # global constant
  $I = $salt % $K;
  ## print "<p>Debug: index of salt item is $I";
  $now = time();
  $TIMEOUT = SALTTIMEOUT; # 5min (global)
  $qry = "SELECT sTime, sValue FROM SALT WHERE salt = $I";
  list ($STIME, $SVALUE) = GetSQL($handDB, $qry, 'check salt');
  ## print "<p>Debug: time is $STIME salt is $SVALUE";
  $J2 = rand(); # [hmm]
  if ($SVALUE != $salt) # oops
     { ## print "<p>Debug: failed. Values ($SVALUE != $salt)";
       readfile('busy.htm');
       exit(); # [hmm. might differentiate this from next, or log]!
```

```
    };

  $qry = "UPDATE SALT SET sValue = $J2 WHERE salt = $I";
  DoSQL($handDB, $qry, 'prevent salt reuse');
  if (($now - $STIME) > $TIMEOUT) # timed out
     { ## print "<p>Debug: timeout. ($STIME + $TIMEOUT &lt; $now)";
       readfile('busy.htm');
       exit();
     };
  # now have valid SALT value. Get user password:
  $qry = "SELECT person, pPassword FROM PERSON
    WHERE pLoginName = '$username'";
  list ($uk, $MD5PWD) = GetSQL($handDB, $qry, 'get md5 password');
  if (strlen($MD5PWD) < 31) # hack for startup
     { $MD5PWD = md5($MD5PWD);
     };
  ## print "<p>Debug: md5 pwd for $username is $MD5PWD";
  $CHECKVAL = md5 ($salt . $MD5PWD); # concatenate, then md5
  ## print "<p>Debug: salted password md5 is $CHECKVAL";
  if ($pswd != $CHECKVAL)
     { Force_Html_InitialLogon('Invalid password/username.');
       exit();
     };
  # create expiry time and SID:
  $expiry = time() + eZ_TIMEOUT;
  $SID = $pswd; # [ugh]
  $qry = "UPDATE PERSON
            SET pSID = '$SID',
                pExpiration = $expiry
            WHERE person = $uk";
  DoSQL($handDB, $qry, 'set db SID, expirn');
  # and store SID in a cookie!
  setcookie('eZ_SID', $SID, $expiry+COOKIE_EXTRA_WAIT);
   ## $dqry = "SELECT pSID FROM PERSON WHERE person = $uk";
   ## list($SID) = GetSQL($handDB, $dqry, 'debug get sid');
   ## print ("<p>Debug: cookie set to $SID, timeout $expiry");
  return($uk); # return user key.
}
```

Force_Html_InitialLogon

All the following does is read an HTML warning file (See section 6.1) and enclose the supplied message. It would be possible to fancy things up a bit.

```
function Force_Html_InitialLogon ($msg)
{ # fancy dressing up might go here #
  # print("<p>DEBUG: Message is '$msg'");
  readfile('oops1.htm');
```

```
  print ($msg);
  readfile('oops2.htm');
};
```

HTML files: *oops1.htm* and *oops2.htm*

The following files might conveniently be combined into one file with a 'string to be substituted for the message' in the centre. Oh well here's *oops1.htm* . . .

```
<head><title>Oops!</title>
<LINK href="css/strictstyle.css" type='text/css'
  rel='stylesheet'>
</head>
<body>

<h2>Oops!</h2>

There was a problem with logging on.

<!-- message follows -->
```

Here's the second part, *oops2.htm*:

```
<p>Click <a href='login.htm'>here</a> to try again.

</body>
```

8.1.3 Javascript file: *md5.js*

Here's a Javascript file that mediates md5 encoding, stored locally in */ez/ez-shw/php/js*, and on the server in */ez/js*. Thanks to Paul Johnston for this md5 Javascript code.

```
/*
 * A JavaScript implementation of the RSA Data Security, Inc. MD5 Message
 * Digest Algorithm, as defined in RFC 1321.
 * Version 2.1 Copyright (C) Paul Johnston 1999 - 2002.
 * Other contributors: Greg Holt, Andrew Kepert, Ydnar, Lostinet
 * Distributed under the BSD License
 * See http://pajhome.org.uk/crypt/md5 for more info.
 */

/*
 * Configurable variables. You may need to tweak these to be compatible with
 * the server-side, but the defaults work in most cases.
 */
var hexcase = 0;  /* hex output format. 0 - lowercase; 1 - uppercase        */
var b64pad  = ""; /* base-64 pad character. "=" for strict RFC compliance   */
var chrsz   = 8;  /* bits per input character. 8 - ASCII; 16 - Unicode      */
```

```
/*
 * These are the functions you'll usually want to call
 * They take string arguments and return either hex or base-64 encoded strings
 */
function hex_md5(s){ return binl2hex(core_md5(str2binl(s), s.length * chrsz));}
function b64_md5(s){ return binl2b64(core_md5(str2binl(s), s.length * chrsz));}
function str_md5(s){ return binl2str(core_md5(str2binl(s), s.length * chrsz));}
function hex_hmac_md5(key, data) { return binl2hex(core_hmac_md5(key, data)); }
function b64_hmac_md5(key, data) { return binl2b64(core_hmac_md5(key, data)); }
function str_hmac_md5(key, data) { return binl2str(core_hmac_md5(key, data)); }

/*
 * Perform a simple self-test to see if the VM is working
 */
function md5_vm_test()
{
  return hex_md5("abc") == "900150983cd24fb0d6963f7d28e17f72";
}

/*
 * Calculate the MD5 of an array of little-endian words, and a bit length
 */
function core_md5(x, len)
{
  /* append padding */
  x[len >> 5] |= 0x80 << ((len) % 32);
  x[(((len + 64) >>> 9) << 4) + 14] = len;

  var a =  1732584193;
  var b = -271733879;
  var c = -1732584194;
  var d =  271733878;

  for(var i = 0; i < x.length; i += 16)
  {
    var olda = a;
    var oldb = b;
    var oldc = c;
    var oldd = d;

    a = md5_ff(a, b, c, d, x[i+ 0], 7 , -680876936);
    d = md5_ff(d, a, b, c, x[i+ 1], 12, -389564586);
    c = md5_ff(c, d, a, b, x[i+ 2], 17,  606105819);
    b = md5_ff(b, c, d, a, x[i+ 3], 22, -1044525330);
    a = md5_ff(a, b, c, d, x[i+ 4], 7 , -176418897);
    d = md5_ff(d, a, b, c, x[i+ 5], 12,  1200080426);
    c = md5_ff(c, d, a, b, x[i+ 6], 17, -1473231341);
    b = md5_ff(b, c, d, a, x[i+ 7], 22, -45705983);
    a = md5_ff(a, b, c, d, x[i+ 8], 7 ,  1770035416);
    d = md5_ff(d, a, b, c, x[i+ 9], 12, -1958414417);
    c = md5_ff(c, d, a, b, x[i+10], 17, -42063);
    b = md5_ff(b, c, d, a, x[i+11], 22, -1990404162);
    a = md5_ff(a, b, c, d, x[i+12], 7 ,  1804603682);
    d = md5_ff(d, a, b, c, x[i+13], 12, -40341101);
    c = md5_ff(c, d, a, b, x[i+14], 17, -1502002290);
    b = md5_ff(b, c, d, a, x[i+15], 22,  1236535329);

    a = md5_gg(a, b, c, d, x[i+ 1], 5 , -165796510);
    d = md5_gg(d, a, b, c, x[i+ 6], 9 , -1069501632);
    c = md5_gg(c, d, a, b, x[i+11], 14,  643717713);
    b = md5_gg(b, c, d, a, x[i+ 0], 20, -373897302);
```

```
    a = md5_gg(a, b, c, d, x[i+ 5], 5 , -701558691);
    d = md5_gg(d, a, b, c, x[i+10], 9 ,  38016083);
    c = md5_gg(c, d, a, b, x[i+15], 14, -660478335);
    b = md5_gg(b, c, d, a, x[i+ 4], 20, -405537848);
    a = md5_gg(a, b, c, d, x[i+ 9], 5 ,  568446438);
    d = md5_gg(d, a, b, c, x[i+14], 9 , -1019803690);
    c = md5_gg(c, d, a, b, x[i+ 3], 14, -187363961);
    b = md5_gg(b, c, d, a, x[i+ 8], 20,  1163531501);
    a = md5_gg(a, b, c, d, x[i+13], 5 , -1444681467);
    d = md5_gg(d, a, b, c, x[i+ 2], 9 , -51403784);
    c = md5_gg(c, d, a, b, x[i+ 7], 14,  1735328473);
    b = md5_gg(b, c, d, a, x[i+12], 20, -1926607734);

    a = md5_hh(a, b, c, d, x[i+ 5], 4 , -378558);
    d = md5_hh(d, a, b, c, x[i+ 8], 11, -2022574463);
    c = md5_hh(c, d, a, b, x[i+11], 16,  1839030562);
    b = md5_hh(b, c, d, a, x[i+14], 23, -35309556);
    a = md5_hh(a, b, c, d, x[i+ 1], 4 , -1530992060);
    d = md5_hh(d, a, b, c, x[i+ 4], 11,  1272893353);
    c = md5_hh(c, d, a, b, x[i+ 7], 16, -155497632);
    b = md5_hh(b, c, d, a, x[i+10], 23, -1094730640);
    a = md5_hh(a, b, c, d, x[i+13], 4 ,  681279174);
    d = md5_hh(d, a, b, c, x[i+ 0], 11, -358537222);
    c = md5_hh(c, d, a, b, x[i+ 3], 16, -722521979);
    b = md5_hh(b, c, d, a, x[i+ 6], 23,  76029189);
    a = md5_hh(a, b, c, d, x[i+ 9], 4 , -640364487);
    d = md5_hh(d, a, b, c, x[i+12], 11, -421815835);
    c = md5_hh(c, d, a, b, x[i+15], 16,  530742520);
    b = md5_hh(b, c, d, a, x[i+ 2], 23, -995338651);

    a = md5_ii(a, b, c, d, x[i+ 0], 6 , -198630844);
    d = md5_ii(d, a, b, c, x[i+ 7], 10,  1126891415);
    c = md5_ii(c, d, a, b, x[i+14], 15, -1416354905);
    b = md5_ii(b, c, d, a, x[i+ 5], 21, -57434055);
    a = md5_ii(a, b, c, d, x[i+12], 6 ,  1700485571);
    d = md5_ii(d, a, b, c, x[i+ 3], 10, -1894986606);
    c = md5_ii(c, d, a, b, x[i+10], 15, -1051523);
    b = md5_ii(b, c, d, a, x[i+ 1], 21, -2054922799);
    a = md5_ii(a, b, c, d, x[i+ 8], 6 ,  1873313359);
    d = md5_ii(d, a, b, c, x[i+15], 10, -30611744);
    c = md5_ii(c, d, a, b, x[i+ 6], 15, -1560198380);
    b = md5_ii(b, c, d, a, x[i+13], 21,  1309151649);
    a = md5_ii(a, b, c, d, x[i+ 4], 6 , -145523070);
    d = md5_ii(d, a, b, c, x[i+11], 10, -1120210379);
    c = md5_ii(c, d, a, b, x[i+ 2], 15,  718787259);
    b = md5_ii(b, c, d, a, x[i+ 9], 21, -343485551);

    a = safe_add(a, olda);
    b = safe_add(b, oldb);
    c = safe_add(c, oldc);
    d = safe_add(d, oldd);
  }
  return Array(a, b, c, d);

}

/*
 * These functions implement the four basic operations the algorithm uses.
 */
function md5_cmn(q, a, b, x, s, t)
{
```

```
  return safe_add(bit_rol(safe_add(safe_add(a, q), safe_add(x, t)), s),b);
}
function md5_ff(a, b, c, d, x, s, t)
{
  return md5_cmn((b & c) | ((~b) & d), a, b, x, s, t);
}
function md5_gg(a, b, c, d, x, s, t)
{
  return md5_cmn((b & d) | (c & (~d)), a, b, x, s, t);
}
function md5_hh(a, b, c, d, x, s, t)
{
  return md5_cmn(b ^ c ^ d, a, b, x, s, t);
}
function md5_ii(a, b, c, d, x, s, t)
{
  return md5_cmn(c ^ (b | (~d)), a, b, x, s, t);
}

/*
 * Calculate the HMAC-MD5, of a key and some data
 */
function core_hmac_md5(key, data)
{
  var bkey = str2binl(key);
  if(bkey.length > 16) bkey = core_md5(bkey, key.length * chrsz);

  var ipad = Array(16), opad = Array(16);
  for(var i = 0; i < 16; i++)
  {
    ipad[i] = bkey[i] ^ 0x36363636;
    opad[i] = bkey[i] ^ 0x5C5C5C5C;
  }

  var hash = core_md5(ipad.concat(str2binl(data)), 512 + data.length * chrsz);
  return core_md5(opad.concat(hash), 512 + 128);
}

/*
 * Add integers, wrapping at 2^32. This uses 16-bit operations internally
 * to work around bugs in some JS interpreters.
 */
function safe_add(x, y)
{
  var lsw = (x & 0xFFFF) + (y & 0xFFFF);
  var msw = (x >> 16) + (y >> 16) + (lsw >> 16);
  return (msw << 16) | (lsw & 0xFFFF);
}

/*
 * Bitwise rotate a 32-bit number to the left.
 */
function bit_rol(num, cnt)
{
  return (num << cnt) | (num >>> (32 - cnt));
}

/*
 * Convert a string to an array of little-endian words
 * If chrsz is ASCII, characters >255 have their hi-byte silently ignored.
 */
```

```
function str2binl(str)
{
  var bin = Array();
  var mask = (1 << chrsz) - 1;
  for(var i = 0; i < str.length * chrsz; i += chrsz)
    bin[i>>5] |= (str.charCodeAt(i / chrsz) & mask) << (i%32);
  return bin;
}

/*
 * Convert an array of little-endian words to a string
 */
function binl2str(bin)
{
  var str = "";
  var mask = (1 << chrsz) - 1;
  for(var i = 0; i < bin.length * 32; i += chrsz)
    str += String.fromCharCode((bin[i>>5] >>> (i % 32)) & mask);
  return str;
}

/*
 * Convert an array of little-endian words to a hex string.
 */
function binl2hex(binarray)
{
  var hex_tab = hexcase ? "0123456789ABCDEF" : "0123456789abcdef";
  var str = "";
  for(var i = 0; i < binarray.length * 4; i++)
  {
    str += hex_tab.charAt((binarray[i>>2] >> ((i%4)*8+4)) & 0xF) +
           hex_tab.charAt((binarray[i>>2] >> ((i%4)*8  )) & 0xF);
  }
  return str;
}

/*
 * Convert an array of little-endian words to a base-64 string
 */
function binl2b64(binarray)
{
  var tab = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
  var str = "";
  for(var i = 0; i < binarray.length * 4; i += 3)
  {
    var triplet = (((binarray[i   >> 2] >> 8 * ( i   %4)) & 0xFF) << 16)
                | (((binarray[i+1 >> 2] >> 8 * ((i+1)%4)) & 0xFF) << 8 )
                |  ((binarray[i+2 >> 2] >> 8 * ((i+2)%4)) & 0xFF);
    for(var j = 0; j < 4; j++)
    {
      if(i * 8 + j * 6 > binarray.length * 32) str += b64pad;
      else str += tab.charAt((triplet >> 6*(3-j)) & 0x3F);
    }
  }
  return str;
}
```

8.2 Validate user

Note that invocation of *ValidFx.php* implies availability of all the functions in *ancillary.php*!

The following script contains the function *validate_login* which should be invoked at the start of all PHP scripts which wish to access eZ. This function checks that the current user is actually logged on, and if so, permits access to the relevant page. If the user *isn't* logged on, then a log-in screen is popped up. Do not confuse this with the initial log-on performed by the script *InitialLogon.php*. The reason why we need to validate the user for *every page* is that HTML is stateless — we need to identify a particular user by comparing a cookie stored on their machine with a value in the database. This value is the session ID (SID).

Once the invoking page has said require_once (ValidFx.php), it can actually invoke *validate_login*. We submit the parameter value SHOW_USER to this function, although at present this value isn't checked or used. The script sets the global $USERKEY to the user primary key from the SQL database table PERSON, and $handDB becomes an open handle on the database. It's a good idea[20] to set the variable $THISPAGE prior to calling routines from ValidFx.php, although a null value (the default) is also acceptable. Here's the initial code:

```
header ('Cache-control: no-cache' );
require('eZ_GLOBALS.php');
require('ancillary.php');
require($CONNECTIONPATH);
```

validate_login

Now for the function that actually performs the log in validation. If an SID already exists for this user (stored in a cookie, and checked against the database) then validate_login succeeds, but otherwise we pop up an error (Oops) page! On success, we not only provide a handle to the database, but we also provide information about the user (including their status), and the last page they visited!

```
function validate_login ($showuser)
{
  GLOBAL $handDB;
  GLOBAL $USERKEY;
  GLOBAL $THISPAGE;

 #1. connect to database
 $handDB = eZ_database_connect();
```

[20] But it's a particularly *bad* idea to encourage a user to visit a page which was accessed using a POST method, so we routinely comment out the $USERKEY allocation on these pages!

```
 if (! $handDB)
    { ## mysql_close($handDB);
      print ( "\n Failed to connect to database!" );
            # here have rescue HTML code:
      readfile ('rescue.htm');
      exit();
    };

#2. retrieve cookie. If null, present NEW login screen
 $SID = $_COOKIE['eZ_SID'];
 # print "<p>Debug: cookie SID is $SID";
 if (strlen($SID) < 1)
    { mysql_close($handDB);
      ForceLogin('Welcome to eZ!');
      exit();
    };

#2a. debug only:
 # $dqry = "SELECT pSID FROM PERSON WHERE person=1";
 # list($i) = GetSQL($handDB, $dqry, 'debug get su sid');
 # print ("<p>Superuser SID is currently $i");

#3. Check for failure:
 $qry = "SELECT person, pExpiration, pLoginName, pValue
            FROM PERSON WHERE pSID = '$SID'";
 $handQ = mysql_query($qry, $handDB);
  if (! is_resource($handQ))
    { # debug:
      print ( "<br>DEBUG: SQL error at login:" . mysql_error() );
    };
 if (! mysql_num_rows($handQ))
    { # force cookie to evaporate:
      mysql_close($handDB);
      setcookie('eZ_SID', '', time()-100);
      ForceLogin('No match for cookie');
      exit();
    };

#4. if cookie stale, present login
  $row = mysql_fetch_assoc($handQ);
  $exptime = $row['pExpiration']; # $row is associative array
  $now = time();
  if ($exptime < $now)
     { # here force cookie to evaporate:
       mysql_close($handDB);
       setcookie('eZ_SID', '', time()-100);
 # print ("<br>Debug: exp time &lt;$exptime&gt; TIME is $now");
       ForceLogin('Session has expired');
```

```
       exit();
     };

#5. Retrieve previous page
  $OLDPAGE = $row['pValue']; # most recent page
  $USERKEY = $row['person'];
  $expiry = time() + eZ_TIMEOUT;
  if (strlen($THISPAGE) < 2) # if invalid
     { $THISPAGE = $OLDPAGE;
     };
  $qry = "UPDATE PERSON
            SET pValue = '$THISPAGE',
                pExpiration = $expiry
            WHERE person = $USERKEY";
  if (! mysql_query($qry, $handDB) )
     { print (
  "<br>DEBUG: SQL error in storing page:" . mysql_error() );
     };
  $user = $row['pLoginName'];

#6. Determine user status:
 $qry = "SELECT MAX(PersonRole) AS userStatus FROM PERSDATA
         WHERE Person = $USERKEY";
 list($userstatus) = GetSQL($handDB, $qry, 'get user status');
# print "<br>Dbg: User status for $USERKEY is $userstatus";

# 7. Set cookie, return user details
  setcookie('eZ_SID', $SID, $expiry+COOKIE_EXTRA_WAIT);
  $outvalue =
 "USER=\"$user\"|STATUS=\"$userstatus\"|OLDPAGE=\"$OLDPAGE\"";
  return($outvalue); # success
}
```

ForceLogin

This simple function reads an html file which in turn allows a re-try of *login.htm*, itself described in section 8.1. A message is printed.

```
function ForceLogin ($msg)
{ # fancy dressing up might go here #
  # print("<p>Debugging ForceLogin: '$msg'");
  $MSG = $msg; # reminder
  readfile('oops1.htm');
  print ($msg);
  readfile('oops2.htm');
};
```

8.3 Logging out: logout.php

```
 $THISPAGE = ''; # irrelevant.
 require('ValidFx.php'); # our login validation script
 $success = validate_login(SHOW_USER);  # -> $USERKEY, $handDB
 $MINUSERSTATUS = NOBODY;
 $MAXUSERSTATUS = EVERYONE;
 list ($MYHEADER, $USERSTATUS) = eZHeaderGreet($success,
                              $MINUSERSTATUS, $MAXUSERSTATUS,
                                        'Logging out...', 0);

  # now $USERKEY, $handDB set, so clean up DB (log out):
  setcookie('eZ_SID', '', time()-100); # clear cookie
  $qry = "UPDATE PERSON
            SET pSID = NULL,
                pExpiration = 0
            WHERE person = $USERKEY";
  $handleQ = mysql_query($qry, $handDB);
  if (! $handleQ) # hmm. debugging only.
     { print "<br>Logout error: user key was &lt;$USERKEY&gt; " .
        mysql_error();
     };
  mysql_close($handDB);

print <<<HTML1
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en">
<head><title>Log out from eZ</title>
<LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
</head>

$MYHEADER

<p>Logged out. Thanks for using eZ.

<p><a href='login.php'>Click here</a> to log in again!
HTML1;
```

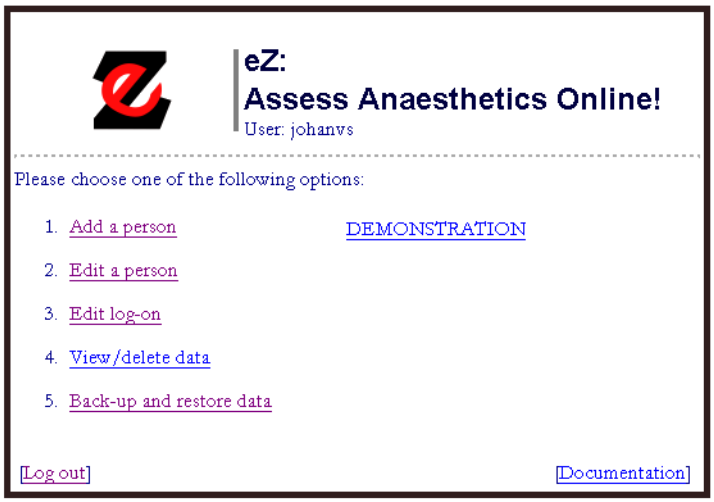

Figure 4: The main page

9 Working PHP code

9.1 The main page: mainpage.php

```
 header( 'Cache-control: no-cache' );
 require_once('ValidFx.php'); # our login validation script
# if invoked by InitialLogon.php, then IT defined $success,
# and set $LOGGED_IN; otherwise:
if (! $LOGGED_IN)
   { $success = validate_login(SHOW_USER);
        # returns $USERKEY, $handDB for our use
     if (! $success)
     { exit();
     };
   };

$MINUSERSTATUS = NOBODY;
$MAXUSERSTATUS = EVERYONE;
$THISPAGE = 'mainpage.php';
list ($MYHEADER, $USERSTATUS) = eZHeaderGreet($success,
                              $MINUSERSTATUS, $MAXUSERSTATUS,
       'eZ: Assess Anaesthetics Online!', 1);
print <<<HTML1
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en">
<head>
```

```
<title>eZ Main Page</title>
<LINK href="css/eZstyle.css" type="text/css"
      rel="stylesheet">
</head>
<body>
 <table width='100%' height='98%'>
  <tr><td class='middling' align='center'>
   <!-- OUTER TBL still no good CSS equivalent ?? -->
<table class='heavybox'><tr><td>
<!-- middle TBL -->
       $MYHEADER
HTML1;

if ($USERSTATUS < eZ_ASSESSOR_MAX)
   { print "<p><a href='eZ_assess.php'>
            Assess another anaesthetic</a>";
   } else
   {
  print <<<HTML2b
  <p>Please choose one of the following options:
   <table width='80%'><!-- inner table -->
    <tr>
     <td width='60%'>
      <ol>
        <p><li><a href="eZ_add_person.php">
        Add a person</a>
        <p><li><a href="eZ_edit_person.php">
        Edit a person</a>
        <p><li><a href="eZ_edit_logon.php">
        Edit log-on</a>
        <p><li><a href="eZ_view.php">
        View/delete data</a>
        <p><li><a href="eZ_backup.php">
        Back-up and restore data</a>
      </ol>
     </td>
     <td valign='top' width='40%'> <br><a href="eZ_demo.php">DEMONSTRATION</t
    </tr>
   </table><!-- end inner tbl -->
HTML2b;
   };

print <<<HTML3
  <p><table width='100%'>
      <tr><td>[<a href="logout.php">Log out</a>]</td>
        <td align='right'>
  [<a href="GPL.htm">Documentation</a>] </td>
      </tr>
```

```
    </table>

  </td></tr></table><!-- end middle tbl -->
 </td></tr></table><!-- end OUTER TBL -->
</body></html>
HTML3;
```

9.2 Assess an anaesthetic

9.2.1 Demonstration page

The functionality is as for the demonstration module.

1. Identification of the user, as usual;
2. Presentation of the first screen;
3. Timing, so the anaesthetic by default advances at 1 minute per minute;
4. The user can scan forward in time;
5. The user can annotate the anaesthetic at will, stating what motivated their decision to ‘intervene’.
6. The user can identify artifacts;
7. All user annotations are stored as long-lived cookies;
8. At the end, the cookie data are ‘written’ in a mock fashion to the server;
9. The user is then thanked for participating.

Note that the user can pause the assessment at any time, but they cannot go ‘back in time’. We should store the images in a separate *png* directory, that is not the same as the usual *images* directory. We might even have relevant subdirectories within the *png* directory! [CONSIDER THE -m vs -s suffix, also the merit of having the PNG images served up as BLOBS??]

9.2.2 Actual assessments

The actual user assessment is as above, but note that a block of ten anaesthetics will be randomised, and this order will be preserved. There are several other wrinkles:

- If the assessment is halted and the user exits, on return the current state will be retrieved. This suggests that we should make some indication of ‘points of assessment’ and even permit the user to mouseover these, with a popup (or overlay) of details!
- For the same anaesthetic, we will not allow the manual view to follow the automatic immediately, or vice versa. Otherwise any order is permissible.
- At the end, the assessing anaesthetist is asked whether they thought this was a manual or an automated anaesthetic record.

We must consider in some detail the implications of multiple recorded values for an automated record, versus interpolated values for the manual one. It may well be necessary to interpolate on the manual record [THINK CAREFULLY ABOUT THIS]!

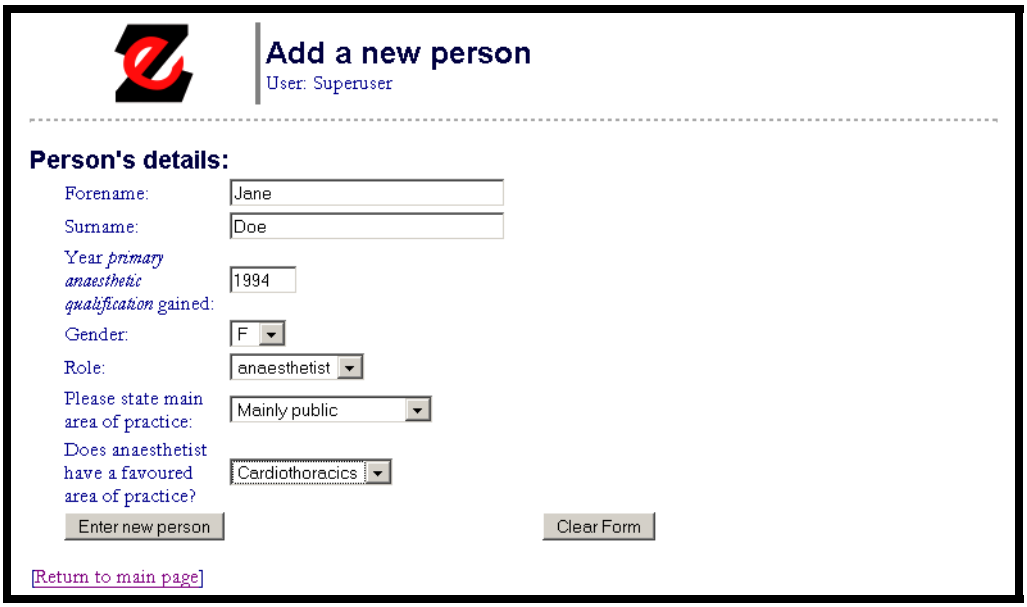

Figure 5: Adding a new anaesthetist (assessor)

9.3 Entering a person

Here we enter basic information about a person. The person might be an anaesthetist (assessor) or a superuser. When the administrator clicks on the 'Add new' button, the data are submitted as POST data to the PHP script called *eZ_admin_personadded.php*. Preliminary processing is first performed by the Javascript in the header of the page.

For superusers, we can leave out values for 'First Qualified', 'Current Specialty' and 'Current Location'. At present we won't capture ethnicity, email, telephone and physical address.

eZ_add_person.php

Full PHP/HTML code follows:

```
# Setting up a login name and password is yet another screen!

 header( 'Cache-control: no-cache' );
 GLOBAL $DEBUGGING;
 GLOBAL $handDB;

 $THISPAGE = 'eZ_add_person.php';
 require_once('ValidFx.php'); # our login validation script
 $MINUSERSTATUS = eZ_ADMINISTRATOR_MIN;
```

```
 $MAXUSERSTATUS = EVERYONE;
 $success = validate_login(SHOW_USER);

 list ($MYHEADER, $USERSTATUS) = eZHeaderGreet($success,
                              $MINUSERSTATUS, $MAXUSERSTATUS,
                                        'Add a new person', 0);
$THISYEAR = WhatYearIsIt(); # :ancillary.php. INSERTED BELOW.
  print <<<HTML0
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en">
<head><title>Administration --- Enter details of a person</title>
<LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
  <script type='text/javascript'>
  <!--
  // The following is clumsy and must be adjusted
  //   if new fields are added.
  function CheckInput(myform)
    { errorcount = 0;
      if (! ValidForename(myform.forename, 0))
         { errorcount ++;
         };
      if (! ValidSurname(myform.surname, 0))
         { errorcount ++;
         };
      if (myform.gender.selectedIndex < 1)
         { errorcount ++;
         };
      if (myform.personrole.selectedIndex < 1)
         { errorcount ++;
         };
      if (myform.personrole.selectedIndex <= 2)
         { if (! ValidStartYear(myform.startyear, 0))
              { errorcount ++;
              };
           if (myform.currentlocation.selectedIndex < 1)
              { errorcount ++;
         };   };
      //
      if (errorcount > 0)
         { if (errorcount == 1)
              { alert (
  "Please complete relevant fields! You left out one.");
              } else
              { alert (
  "Oops! Please complete relevant fields. There were " +
                    errorcount + " errors.");
              };
          return (false);
```

```
        };
      return (true);
    }

  // ValidStartYear(startyear, chirp):
  // check if valid year. If not return false.
  // If chirp is nonzero, then also give ALERT.
  function ValidStartYear(startyear, chirp)
    { thisyear = $THISYEAR ;
      if (  (startyear.value.length != 4)
          ||(startyear.value < 1950)
          ||(startyear.value > thisyear)
        ) // what if insert alphas...???
        { if (chirp)
             { alert ("Please enter valid 4 digit year e.g. 2003");
             };
         // startyear.select();
         // startyear.focus();
          return (false);
        };
       return (true);
    }

  function ValidSurname(surname, chirp)
    { if (  (surname.value.length < 2)
        ) // can add more to the above
        { if (chirp)
             { alert ("Please enter valid surname");
             };
         // surname.select();
         // surname.focus();
          return (false);
        };
       return (true);
    }

  function ValidForename(forename, chirp)
    { if (  (forename.value.length < 2)
        ) // can add more to the above
        { if (chirp)
             { alert ("Please enter valid forename");
             };
          // forename.select();
          // forename.focus();
          return (false);
        };
       return (true);
    }
```

```
function ConfirmClear ()
  { if (confirm (
 'Are you sure you want to clear the form? (Click OK to clear it!)') )
       { return true;
       };
    return false;
  }
  //-->
  </script>
</head>
<body>
$MYHEADER
HTML0;

$DEBUGGING = 1;

print <<<HTML2
<FORM name="eZ_assess"
      ACTION="eZ_admin_personadded.php"
      METHOD="POST"
      onSubmit="return CheckInput(this)" >
  <input type=hidden name="confirmation" value="0">
  <h3>Person's details:</h3>
  <table width='65%'>
    <tr><td></td><td>Forename:</td><td>
     <input type="text" name="forename" size="32"
       onChange="ValidForename(this, 1)" >
    </td></tr>
    <tr><td></td><td>Surname:</td><td>
      <input type="text" name="surname" size="32"
       onChange="ValidSurname(this, 1)" >
    </td></tr>
    <tr><td></td><td>Year <i>primary anaesthetic qualification</i> gained:</td><td
       <input type="text" name="startyear" size="5"
       onChange="ValidStartYear(this, 1)" >
    </td></tr>
    <tr><td width='5%'> </td><td width='14%'>
      Gender: </td><td width='81%'>
      <select name="gender" size="1">
               <option selected value="0">?</option>
               <option value="1">F</option>
               <option value="2">M</option>
      </select>
    </td></tr>
    <tr><td></td><td>Role: </td><td>
HTML2;
```

```
PrintPoplist ($handDB, "personrole",
  "SELECT personrole, PERSONROLE.rText from PERSONROLE
      WHERE personrole > 0
      ORDER BY personrole" );

print <<<HTML3
    </td></tr>
    <tr><td></td><td>Please state main area of practice: </td><td>
HTML3;

PrintPoplist ($handDB, "currentlocation",
  "SELECT currentlocation, CURRENTLOCATION.LocationText
    from CURRENTLOCATION WHERE currentlocation > 0
      ORDER BY currentlocation" );

print <<<HTML4
    </td></tr>
    <tr><td></td><td>Does anaesthetist have a particular anaesthetic interest? </t
HTML4;

PrintPoplist ($handDB, "currentspecialty",
  "SELECT currentspecialty, CURRENTSPECIALTY.SpecialText
      from CURRENTSPECIALTY WHERE currentspecialty > 0
      ORDER BY currentspecialty" );

print <<<HTML5
    </td></tr>
    <tr><td></td><td>
      <INPUT TYPE="submit" NAME="submit" VALUE="Enter new person">
    </td><td align='right'>
      <INPUT TYPE="reset" VALUE="Clear Form"
        onClick="return ConfirmClear()">
    </td></tr>
  </table>
</FORM>
<p>[<a href='mainpage.php'>Return to main page</a>]
</body></html>
HTML5;
```

The most interesting feature of the above page is the hidden POST variable called 'confirmation', forced to zero, so that eZ_admin_personadded.php knows that incoming data is 'raw'. The possibility exists that similar people might already exist in the database. The latter script checks for this and requires confirmation, then forcing resubmission of data to *itself*, but with the confirmation variable set to 1. If 'confirmation' is 1, then no further checking takes place!

eZ_admin_personadded.php

We process the POST data from the preceding page. In addition, we provide the option to add a user name and password.

```
## Screen layout: ##################################################
#                                                                  #
#   <Message about added person>                                   #
#                                                                  #
#   [Add user name/password]  (for reviewers/superusers)           #
#                                                                  #
#   [Back to main page]                                            #
#                                                                  #
####################################################################
#
# alternatively, if confirmation is required we get:
#
####################################################################
#   WARNING! Similar names exist in the database!                  #
#                                                                  #
#   LIST OF SIMILAR NAMES:                                         #
#                                                                  #
#   [insert list of forename, surname, etc]                        #
#                                                                  #
#   Are you sure you wish to add this person?                      #
#                                                                  #
#   [Do NOT add person]               [YES, Add person]            #
####################################################################

header( 'Cache-control: no-cache' );
 require_once('ValidFx.php'); # our login validation script
 $MINUSERSTATUS = eZ_ADMINISTRATOR_MIN;
 $MAXUSERSTATUS = EVERYONE;
 $success = validate_login(SHOW_USER);
 list ($MYHEADER, $USERSTATUS) = eZHeaderGreet($success,
                              $MINUSERSTATUS, $MAXUSERSTATUS,
                                        'Adding new person', 0);
  print <<<HTML0
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en">
<head><title>Administration --- Adding new person</title>
<LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
  <script type='text/javascript'>
  <!--
  function CheckInput ()
    { return (true);  // stub
    }
  //-->
  </script>
```

```
</head>
<body>
$MYHEADER
HTML0;

$FORENAME    = $_POST['forename'];
$SURNAME     = $_POST['surname'];
$STARTYEAR   = $_POST['startyear'];
$GENDER      = $_POST['gender'];
$PERSONROLE  = $_POST['personrole'];
$CONFIRM = $_POST['confirmation'];
$CURRENTSPECIALTY = $_POST['currentspecialty'];
$CURRENTLOCATION  = $_POST['currentlocation'];

Sanitise($FORENAME);
Sanitise($SURNAME);
CheckCode($GENDER, 'Bad gender value');
 # in the following NULL is returned on failure:
CheckCode($PERSONROLE, 'Bad role code');
CheckCodeNull($CURRENTSPECIALTY);
CheckCodeNull($CURRENTLOCATION);

$similarnames = array(); # null array
if ($CONFIRM == 1) # don't check for similar names!
   { # do nothing...
   }
elseif ($CONFIRM == 0) # check for similar names
   { $sur = $SURNAME;
     $sur = strtoupper($sur); # upper case
     $similarnames = SQLManySQL($handDB,
       "SELECT distinct Person FROM PERSDATA WHERE
         UPPER(pdSurname) = '$sur' OR
         UPPER(pdForename) = '$sur'", # check for switch!
   'get identical surnames');
      # might use more substantial algorithm (soundex or better)
   } else
   { readfile ('lostdata.htm');
     exit();
   };

# ----------now insert OR confirm-------------#
if (count ($similarnames) < 1) # if ok, simply store:
   {
    $NEWID = InsertPerson($handDB, $FORENAME, $SURNAME,
                          $STARTYEAR, $GENDER, $PERSONROLE,
                          $CURRENTSPECIALTY, $CURRENTLOCATION);
    if ($NEWID > 0)
       { print <<<HTML2A
```

```
         <p>New user added!
         <p>$FORENAME $SURNAME has been added to the database.
         (Role code: $PERSONROLE).
HTML2A;
       };
  if (strlen($STARTYEAR) > 3)
     { print " This person first qualified in $STARTYEAR.";
     };

  print <<<HTML2X
       <p>It is now wise to add LOGIN details for this user.
       Click <a href='eZ_do_editlogon.php?editlogon=$NEWID'>
       here</a> to do so!
HTML2X;

    print <<<HTML2E
<p><a href="eZ_add_person.php">Add <i>another</i> person</a>
<p>[<a href='mainpage.php'>Return to main page</a>]
HTML2E;

   } else  # ARE similar cases:
   { print <<<HTML2B
     <FORM name="eZ_assess"
      ACTION="eZ_admin_personadded.php" // myself!
      METHOD="POST"
      onSubmit="return CheckInput(this)" >
     <input type=hidden name="forename" value="$FORENAME">
     <input type=hidden name="surname" value="$SURNAME">
     <input type=hidden name="startyear" value="$STARTYEAR">
     <input type=hidden name="gender" value="$GENDER">
     <input type=hidden name="personrole" value="$PERSONROLE">
     <input type=hidden name="currentspecialty"
            value="$CURRENTSPECIALTY">
     <input type=hidden name="currentlocation"
            value="$CURRENTLOCATION">
     <input type=hidden name="confirmation" value="1">

   <h2>Please confirm ...</h2>
     The following users have names similar to the one
     you wish to add ($FORENAME $SURNAME). Please <i>confirm</i>
     that you wish to add this user by clicking on the
     CONFIRM button below. (Obviously if the person already
     exists in the database, Abort and do not confirm).
   <p><div align='center'><table width='65%' border='2'>
   <tr><td align='center' colspan='5'>
        <b>Users with similar names</b></td></tr>
   <tr><td><i>Forename</i></td>
       <td><i>Surname</i></td>
```

```
       <td><i>Current role</i></td>
       <td><i>Year started</i></td>
       <td><i>(Database) ID No</i></td></tr>
HTML2B;

  # NOTE that because $similarnames was generated by SQLManySQL,
  # it is actually an array of arrays, each sub-array
  # having one element. This is nasty. So we flatten
  # the array to 1 dimension!
    $similarnames = Flatten($similarnames);
    $users = GetUserDetails($handDB, $similarnames);
  # get forename, surname, role, year started.
  # print ('<br>Debugging: ');
  # print_r ($users);  # debug
    PrintDetailTable($users, 5); # print table with 5 columns

  print <<<HTML2C
  <tr><td colspan='2'><a href='mainpage.php'>
   Abort. Do NOT add user $FORENAME $SURNAME!</a></td>
   <td colspan='3'><INPUT TYPE="submit" NAME="submit"
       VALUE="CONFIRM! Add this person."></td></tr>
  </table></div>
HTML2C;
   };

# ---------------------------------------------------------- #
function InsertPerson($myODBC, $FORENAME, $SURNAME,
                      $STARTYEAR, $gender, $role,
                      $CURRENTSPECIALTY, $CURRENTLOCATION)
{
  if (strlen($STARTYEAR) < 4) # bad data: default to 1900 [ugh]
     { $STARTYEAR = '1900';
     };

  $now = date('Y-m-d h:i:s');
  $NEWID = FetchKey($myODBC, 'Person');   # new key for PERSON
  $qry = "INSERT INTO PERSON (person, pMade, FirstQualified)
     VALUES ($NEWID, TIMESTAMP '$now', DATE '$STARTYEAR-01-01')";
  DoSQL($myODBC, $qry, "insert new person");

  $iPERSDATA = FetchKey ($myODBC, 'Persdata');
  $qry = "INSERT INTO PERSDATA (persdata, Person,
          PersonRole, CurrentSpecialty, CurrentLocation,
          pdCreated, pdSurname, pdForename, pdGender)
      VALUES ($iPERSDATA, $NEWID,
          $role, $CURRENTSPECIALTY, $CURRENTLOCATION,
          TIMESTAMP '$now', '$SURNAME', '$FORENAME', $gender)";
  DoSQL ($myODBC, $qry, "insert person data");
```

```
  return($NEWID);  # hmm.
}

# ------------------------------------------------------- #
function GetUserDetails($handDB, $similarnames)
{ # $similarnames is array of IDs (key of PERSON table)
  # this fx is cumbersome and slow.
  # returns array of arrays, each subarray containing forename,
  # surname, role, date of 1st qualifn (as year-01-01) and ID.
  $opt = array();
  $i = 0;
  foreach ($similarnames as $p)
    { $detl = array();
      $detl[0] = FetchForename ($handDB, $p);
      $detl[1] = FetchSurname($handDB, $p);
      $detl[2] = FetchRole ($handDB, $p);
      list ($detl[3]) = GetSQL ($handDB,
        "SELECT FirstQualified
        FROM PERSON WHERE person = $p",
        'GET yr of 1st qualification');
      $detl[4] = $p;
      $opt[$i] = $detl; #
      $i += 1;
    };
  return ($opt);
}
```

HTML file: *lostdata.htm*

If odd or missing data are submitted to a PHP script which processes POST data, the following tiny page will be displayed:

```
<head><title>Lost data!</title></head>
<body>
<h2>Lost information?</h2>
<p>The POST data submitted to this form do not contain the expected
information! This suggests that you've randomly accessed this page,
or inserted really weird text in the previous one.
<p>Click
<a href='http://www.intensivist.com/strict/mainpage.php'>
here</a> to return to the main page.
<p><hr>
</body>
```

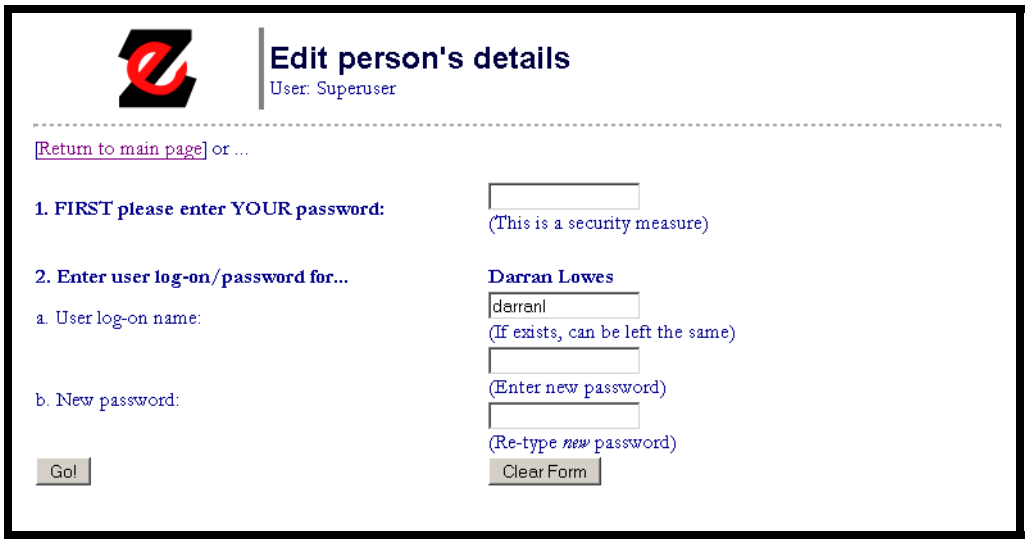

Figure 6: Editing log-on details

9.4 Editing log-on details

Here's the page (*eZ_do_editlogon.php*) that accepts new log-on details for the selected user. Once the administrator has entered the new details, we must still POST the values to *eZ_do_editlogon2.php*.

We md5-encode the administrator password *twice* to prevent it being revealed. Because the md5-encoded password being supplied for the user could be used to log-on as that user if somebody listened in to a password-changing session, we xor the password with the *singly* md5-encoded administrator password before submitting this.[21] Then the server-side script can securely pull out the new password.

```
 header( 'Cache-control: no-cache' );
# $THISPAGE = 'eZ_do_editlogon.php';
 require_once('ValidFx.php'); # our login validation script
 $MINUSERSTATUS = eZ_ADMINISTRATOR_MIN;
 $MAXUSERSTATUS = EVERYONE;
 $success = validate_login(SHOW_USER);
 list ($MYHEADER, $USERSTATUS) = eZHeaderGreet($success,
                                $MINUSERSTATUS, $MAXUSERSTATUS,
                                "Edit person's details", 0);
print <<<HTML0
```

[21] Noting that if the session were recorded and the user password subsequently was disclosed, this would compromise the current administrator password, so it's a good idea for the administrator to change their password often. However, if this session too had been captured, the new password would also be compromised. For better security, it would be wise to incorporate a set of OTPs known only to the administrator and the database, and use these for the XOR.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en"> <head> <title>Edit person's details</title>
<LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
  <script src='js/md5.js' type='text/javascript'></script>
  <script type='text/javascript'>
  <!--
function ConfirmClear ()
  { if (confirm (
 'Are you sure you want to clear the form? (Click OK to clear it!)') )
       { return true;
       };
    return false;
  }

function CheckInput (myform)
  { errorcount = 0;
    errmsg = '';
    mypwd = myform.mypswd.value;
    ulogon = myform.userlogon.value;
    pwd1 = myform.newpassword1.value;
    pwd2 = myform.newpassword2.value;
    if (NastyString(mypwd))
       { errmsg = errmsg +
  "You password does NOT contain characters \\\\ | ' '  ";
         errorcount ++;
       };
    if (NastyString(pwd1))
       { errmsg = errmsg +
  "Password CANNOT contain characters \\\\ | ' '  ";
         errorcount ++;
       };
    if ( ulogon.length < 4)
       { errmsg = errmsg +
  'Please supply user name (4 characters or more). ';
         errorcount ++;
       };
    if ( mypwd.length < 6)
       { errmsg = errmsg +
  "Your password IS 6 or more characters long. ";
         errorcount ++;
       };
    if ( pwd1.length < 6)
       { errmsg = errmsg +
  "New user password: minimum length must be 6 characters. ";
         errorcount ++;
       };
    if ( pwd2.length < 6)
       { errmsg = errmsg +
```

```
  "User password minimum length (confirmation) must be 6 characters. ";
        errorcount ++;
      };
    if ( pwd1 != pwd2)
      { errmsg = errmsg + "Confirmatory password doesn't match! ";
        errorcount ++;
      };
    if (errorcount > 0)
      { alert (errmsg);
        return (false);
      };
    // now encrypt passwords:
    mypwd = hex_md5(mypwd); // first encryption
    pwd1 = hex_md5(pwd1);
    // alert ("Debug md5 pwd is " + pwd1);
    pwd1 = XorStringJ(pwd1, mypwd);
    // alert ("Xor md5 pwd is " + pwd1);
    if (pwd1.length < 30)
      { alert ('Oops!');
        return(false);
      };    // ????
    // end xor section
    mypwd = hex_md5(mypwd); // 2nd encryption
    myform.mypswd.value = mypwd;
    myform.newpassword1.value = pwd1;
    myform.newpassword2.value = '';
    return (true);
  }

function XorStringJ(strA, strB)
{ lgth = strA.length;
  if (lgth != strB.length)
     { return ('');
     };
  outs = '';
  while (lgth > 0)
     { lgth -= 1;
       ichA = Fixch(strA.charCodeAt(lgth));
       ichB = Fixch(strB.charCodeAt(lgth));
       ichA ^= ichB;
       ichA = Unfix(ichA);
       outs = String.fromCharCode(ichA) + outs;
     };
  // here might confirm length of outs==lgth ??
  return (outs);
}

function Fixch(c) // convert hex digit to hex value
```

```
{ if (c > 90)
     { c -= 32; // force upper case
     };
  c -= 48; // 0x30
  if (c > 9)
     { c -= 7;
     };
  return (c); // actual hex value
}

function Unfix(c) // convert hex value to hex digit
{ if (c > 9)
     { c += 7+32; // force LOWER case!! (md5 rule)
     };
  c += 48;
  return(c);
}

function NastyString(pwd)
{ if (pwd.indexOf('\\\\') > -1) // quadruplicate: PHP + Javascript!
     { return (true);
     };
  if (pwd.indexOf('|') > -1)
     { return (true);
     };
  if (pwd.indexOf("'") > -1)
     { return (true);
     };
  if (pwd.indexOf('') > -1)
     { return (true);
     };
  return(false);
}
  //-->
  </script>
</head>
<body>
$MYHEADER
HTML0;

# here GET user ID or fail..
$OTHERLOGONID = $_GET['editlogon'];
CheckCode($OTHERLOGONID, 'Bad log-on'); # or fail
# hmm. We might check that the user is entitled to log-on
# privileges. For now, accept all.

$OTHERSURNAME = FetchSurname($handDB, $OTHERLOGONID);
$OTHERFORENAME = FetchForename($handDB, $OTHERLOGONID);
```

```
$qry = "SELECT pLoginName FROM PERSON WHERE person = $OTHERLOGONID";
list ($OTHERLOGIN) = GetSQL($handDB, $qry, 'get user login');

print <<<HTML2
<p>[<a href='mainpage.php'>Return to main page</a>] or ...

<FORM name="eZ_do_editlogon2"
      ACTION="eZ_do_editlogon2.php"
      METHOD="POST"
      onSubmit="return CheckInput(this)" >
 <input type=hidden name="otheruserid" value="$OTHERLOGONID">
 <table width='80%' cellpadding='1' cellspacing='1'>
   <tr><td><b>1. FIRST please enter YOUR password: </b></td>
    <td> <input type="password" name="mypswd" size="16" /> <br>
      (This is a security measure)</td></tr>
   <tr><td colspan='2'> </td></tr>
   <tr><td><b>2. Enter user log-on/password for...</b> </td>
    <td><b>$OTHERFORENAME $OTHERSURNAME</b></td>
   </tr>
   <tr><td>a. User log-on name: </td>
    <td>
      <input type='text' name='userlogon' size='16'
         value='$OTHERLOGIN'><br>
    (If exists, can be left the same)</td>
   </tr>
   <tr><td rowspan='2'>b. New password: </td>
    <td><input type='password' name='newpassword1' size='16' ><br>
      (Enter new password)</td>
   </tr>
   <tr>
    <td><input type='password' name='newpassword2' size='16' <br>
      (Re-type <i>new</i> password)</td>
   </tr>
   <tr><td><INPUT TYPE="submit" NAME="submit" VALUE="Go!"></td>
    <td><INPUT TYPE="reset" VALUE="Clear Form"
      onClick="return ConfirmClear()"></td></tr>
 </table>
</form>
  </body></html>
HTML2;
```

Figure 7: Editing id/password

eZ_do_editlogon2.php

The page which takes the new log-on data and writes them to the database. We accommodate encryption of both the supervisor password (double md5), and of the new user password (md5 xor'ed with the md5 of the supervisor password).

```
 header( 'Cache-control: no-cache' );
# $THISPAGE = 'eZ_do_editlogon2.php';
 require_once('ValidFx.php'); # our login validation script
 $MINUSERSTATUS = eZ_ADMINISTRATOR_MIN;
 $MAXUSERSTATUS = EVERYONE;
 $success = validate_login(SHOW_USER);
 list ($MYHEADER, $USERSTATUS) = eZHeaderGreet($success,
                              $MINUSERSTATUS, $MAXUSERSTATUS,
                              "Edit person's details", 0);
print <<<HTML0
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en">
<head><title>Edit person's details</title>
<LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
</head>
<body>
$MYHEADER
HTML0;

$OTHERLOGONID = $_POST['otheruserid'];
$MYPSWD  = $_POST['mypswd'];
$USERLOGON=$_POST['userlogon'];
$NEWPWD1  =$_POST['newpassword1'];

CheckCode($OTHERLOGONID, 'Error in logon'); # or fail
Sanitise($MYPSWD);
Sanitise($USERLOGON);
Sanitise($NEWPWD1);

# first check mypswd. this is md5(md5(plaintextpwd))
```

```
$qry = "SELECT pPassword FROM PERSON WHERE person = $USERKEY";
list($MD5PWD) = GetSQL($handDB, $qry, 'get encr. pwd');
# print "<p>Debug: my md5 pwd is $MD5PWD";
if (strlen($MD5PWD) < 30) # hack for startup
   { $MD5PWD = md5($MD5PWD);
     # print "<p>Debug Hack: bare password, now $MD5PWD";
   };
if ($MYPSWD != md5($MD5PWD))  # double md5!
   { # $d = md5($MD5PWD); # DEBUG
     # print "<p>Debug: this md5 is $d";
     readfile('badpassword.htm');
     exit();
   };

#print "<p>Debug: doublemd5 is $MYPSWD,
#  xor of new password md5 is $NEWPWD1,
#  user is $USERLOGON with id $OTHERLOGONID,
#  db md5 pwd is $MD5PWD;";

# extract new password using xor:
$NEWPWD1 = XorString($MD5PWD, $NEWPWD1);
# print "<p>Debug: xor extract gave: $NEWPWD1";
if (strlen($NEWPWD1) < 31)
   { readfile('badpassword.htm');
     exit();
   };

# check that new user name doesn't exist for somebody else:
# (don't use UNIQUE constraint as can be null)!
$qry = "SELECT COUNT(person) FROM PERSON WHERE
         pLoginName = '$USERLOGON'
         AND person <> $OTHERLOGONID";
list ($dups) = GetSQL($handDB, $qry, 'get dup logon');
if ($dups != 0)
   { readfile('duplicate.htm');
     exit();
   };


# write the new password:
$qry = "UPDATE PERSON SET
          pLoginName = '$USERLOGON',
          pPassword = '$NEWPWD1'
        WHERE person = $OTHERLOGONID";
DoSQL($handDB, $qry, 'update user id/password');

# fetch eye candy:
$OTHERSURNAME = FetchSurname($handDB, $OTHERLOGONID);
```

```
$OTHERFORENAME = FetchForename($handDB, $OTHERLOGONID);

print <<<HTML3
<p>Log-on/password have been updated for user
  $OTHERFORENAME $OTHERSURNAME.
Log-on name is '$USERLOGON'.
<p>[<a href='mainpage.php'>Return to main page</a>]

<p>[<a href='eZ_edit_logon.php'>Edit another logon</a>]
</body></html>
HTML3;

function XorString($strA, $strB)
{ $lgth = strlen($strA);
  if ($lgth != strlen($strB))
     { return ('');
     };
  $outs = '';
  while ($lgth > 0)
     { $lgth -= 1;
       $ichA = Fixch($strA[$lgth]);
       $ichB = Fixch($strB[$lgth]);
       $c = Unfix($ichA ^ $ichB);
       $outs = $c . $outs; # [clumsy]
     };
  return ($outs);
}

function Fixch($c) # convert hex digit to hex value
{ $i = ord($c); # convert chr to integer
  if ($i > 90) # 0x5a (if lower case)
     { $i -= 32; # 0x20
     };
  $i -= 48; # 0x30
  if ($i > 9)
     { $i -= 7;
     };
  return ($i); # actual hex value
}

function Unfix($i) // convert hex value to hex digit
{ if ($i > 9)
     { $i += 32+7;  // FORCE lower case as in md5 !!
     };
  $i += 48;
  return(chr($i));
}
```

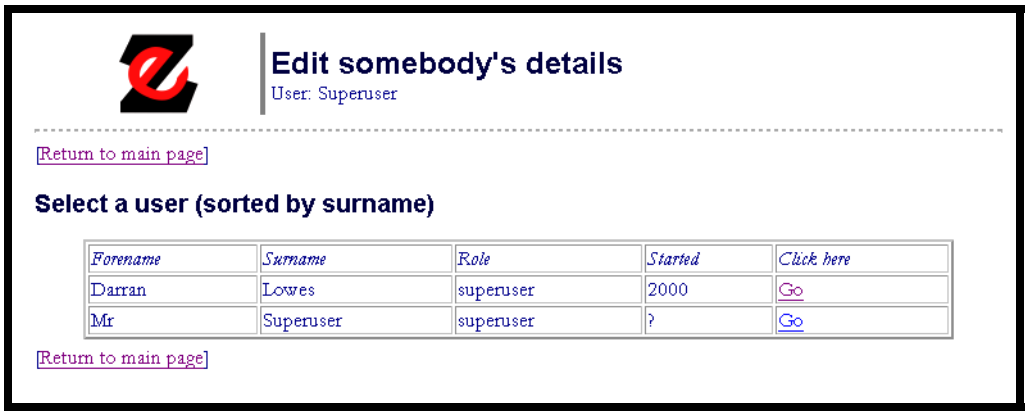

Figure 8: Editing a person's details

9.5 Editing someone's details

Here we have the facility to edit a person's details. This introductory page (eZ_edit_person.php) allows us to select a particular person. When the administrator clicks on a link, control is transferred to the page *eZ_do_edit.php*.

```
 header( 'Cache-control: no-cache' );
 $THISPAGE = 'eZ_edit_person.php';
 require_once('ValidFx.php'); # our login validation script
 $MINUSERSTATUS = eZ_ADMINISTRATOR_MIN;
 $MAXUSERSTATUS = EVERYONE;
 $success = validate_login(SHOW_USER);
 list ($MYHEADER, $USERSTATUS) = eZHeaderGreet($success,
                                $MINUSERSTATUS, $MAXUSERSTATUS,
                                 "Edit somebody's details", 0);
print <<<HTML1
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en"> <head>
<title>Edit details</title>
<LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
</head>
<body>
$MYHEADER
<p>[<a href='mainpage.php'>Return to main page</a>]
  <h3>Select a user (sorted by surname)</h3>
HTML1;

$link = "<a href='eZ_do_edit.php?editperson=USERID'>Go</a>";
PrintActiveUserlist($handDB, $link);

print <<<HTML3
  </table></div>
```

```
<p>[<a href='mainpage.php'>Return to main page</a>]
  </body></html>
HTML3;
```

Figure 9: Editing a person's details

eZ_do_edit.php

The following PHP script is deceptively short, because it uses GET and then simply invokes a much more complex script.

```
header( 'Cache-control: no-cache' );
require_once('ValidFx.php'); # our login validation script
$success = validate_login(SHOW_USER);
 # here GET user ID
$OTHERLOGONID = $_GET['editperson'];
CheckCode($OTHERLOGONID, 'Bad log-on ID'); # or fail
require ('subedit.php');
```

subedit.php

Here's the much more complicated *subedit.php*. The reason for the above shenanigans is to allow us to POST data from within *eZ_do_edit.php* to the POST version, which can then invoke itself. Why not simply use POST always? It's easier in some circumstances to submit the user ID using GET!

```
 $MINUSERSTATUS = eZ_ADMINISTRATOR_MIN;
 $MAXUSERSTATUS = EVERYONE;
 list ($MYHEADER, $USERSTATUS) = eZHeaderGreet($success,
                              $MINUSERSTATUS, $MAXUSERSTATUS,
                                        "Edit details", 0);
print <<<HTML0
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en"> <head>
<title>Edit</title>
<LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
  <script type='text/javascript'>
  <!--
function CheckInputSurname (myform)
  {   valu = myform.newsurname.value;
      if (valu.length < 2)
         { alert('Minimum surname length is 2 characters');
           // might perform other checks?
           return (false);
         };
      return (true);
  }

function CheckInputForename (myform)
  {   valu = myform.newforename.value;
      if (valu.length < 1)
         { alert('Minimum forename length is 1 character');
           // might perform other checks?
           return (false);
         };
      return (true);
  }

function CheckAllInputs(myform)
  { // we will not check the poplists.
    if (  (! CheckInputForename(myform))
        ||(! CheckInputSurname(myform))
        ) // clumsy
        { return (false);
        };
    return (true);
  };

  //-->
  </script>
</head>
<body>
$MYHEADER
HTML0;
```

```
# check user ID or fail:
$OTHERSURNAME = FetchSurname($handDB, $OTHERLOGONID);
$OTHERFORENAME = FetchForename($handDB, $OTHERLOGONID);
$qry = "SELECT pLoginName FROM PERSON WHERE person = $OTHERLOGONID";
list ($OTHERLOGIN) = GetSQL($handDB, $qry, 'get user login');
# the value in OTHERLOGIN may be null.
# in the following we assume only one valid PERSDATA entry [ugh]:
$qry = "SELECT PersonRole FROM PERSDATA WHERE Person = $OTHERLOGONID";
list ($OTHERROLE) = GetSQL($handDB, $qry, 'get role');
$qry = "SELECT CurrentSpecialty FROM PERSDATA WHERE Person = $OTHERLOGONID";
list ($OTHERSPECIALTY) = GetSQL($handDB, $qry, 'get role');
$qry = "SELECT CurrentLocation FROM PERSDATA WHERE Person = $OTHERLOGONID";
list ($OTHERLOCATION) = GetSQL($handDB, $qry, 'get role');
$qry = "SELECT FirstQualified FROM PERSON WHERE Person = $OTHERLOGONID";
list($QUALYEAR) = GetSQL($handDB, $qry, 'get qual yr');

  $qry =  "SELECT personrole, rText FROM PERSONROLE
            WHERE personrole > 0";
  $ROLEPOPLIST = TextPoplistSelected ($handDB,
    "newrole", $qry, $OTHERROLE );

  $qry =  "SELECT currentlocation, LocationText FROM CURRENTLOCATION
            WHERE currentlocation > 0";
  $LOCATIONPOPLIST = TextPoplistSelected ($handDB,
     "newlocation", $qry, $OTHERLOCATION);

  $qry =  "SELECT currentspecialty, SpecialText FROM CURRENTSPECIALTY
            WHERE currentspecialty > 0";
  $SPECIALTYPOPLIST = TextPoplistSelected ($handDB,
     "newspecialty", $qry, $OTHERSPECIALTY);

 $LGIN;
 if (strlen($OTHERLOGIN) < 1)
    {$LGIN = '';
    } else
    {$LGIN = "Login name is '$OTHERLOGIN'.";
    };

print <<<HTML3
<p>[<a href='mainpage.php'>Return to main page</a>]

<h3>Alter details for $OTHERFORENAME $OTHERSURNAME</h3>

<div class='narrow'>
$LGIN By default,
 all values are left at their current settings.
</div>
```

```
<div align='center'>
 <FORM name="eZ_update"
   ACTION="eZ_update.php"
   METHOD="POST"
   onSubmit="return CheckAllInputs(this)" >
  <input type=hidden name="editperson" value="$OTHERLOGONID">

  <table width='80%'>
   <tr><td width='30%'>a. Alter surname: </td>
       <td width='70%'>
          <input type='text' name='newsurname'
          size='16' value='$OTHERSURNAME'></td>
   </tr>
   <tr><td width='30%'>b. Alter forename: </td>
       <td width='70%'>
          <input type='text' name='newforename'
          size='16' value='$OTHERFORENAME'></td>
     </tr>
   <tr><td width='30%'>c. Change role:</td>
       <td width='70%'> $ROLEPOPLIST</td>
   </tr>
   <tr><td width='30%'>d. Change location:</td>
       <td width='70%'> $LOCATIONPOPLIST </td>
   </tr>
   <tr><td width='30%'>e. Change specialty:</td>
       <td width='70%'> $SPECIALTYPOPLIST </td>
   </tr>
   <tr><td width='30%'>e. Year of qualification:</td>
       <td width='70%'>
       <input type='text' size='5' name='qualyear' value='$QUALYEAR'>
       </td>
   </tr>
   <tr><td colspan='2'>
     <INPUT TYPE="submit" NAME="submit" VALUE="Go!"></td>
   </tr>
  </table>

 </form>
</div>
HTML3;

print <<<HTML8

<p><a href='eZ_edit_person.php'>Back to list of people</a>

<p>[<a href='mainpage.php'>Return to main page</a>]
  </body></html>
```

```
HTML8;
```

Figure 10: Updated!

eZ_update.php

We accept personal details from the preceding page, and update the database accordingly. Submitted items are otheruserid, newrole, newlocation, newspecialty, newsurname and newforename. Each of these values should be valid, with a few exceptions:

- If the new role is not a registrar, then newspecialty can (and must) be null/zero;
- If the new role is neither registrar nor nurse, then newlocation can (and must) be null/zero.

```
 header( 'Cache-control: no-cache' );
# $THISPAGE = 'eZ_update.php';
 require_once('ValidFx.php'); # our login validation script
 $MINUSERSTATUS = eZ_ADMINISTRATOR_MIN;
 $MAXUSERSTATUS = EVERYONE;
 $success = validate_login(SHOW_USER);
 list ($MYHEADER, $USERSTATUS) = eZHeaderGreet($success,
                                $MINUSERSTATUS, $MAXUSERSTATUS,
                                "Details updated", 0);
print <<<HTML0
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en">
<head><title>Details updated</title>
<LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
</head>
<body>
$MYHEADER
```

```
HTML0;

$OTHERLOGONID = $_POST['editperson'];
$NEWROLE =      $_POST['newrole'];
$NEWLOCATION =  $_POST['newlocation'];
$NEWSPECIALTY = $_POST['newspecialty'];
$NEWSURNAME =   $_POST['newsurname'];
$NEWFORENAME =  $_POST['newforename'];
$QUALYEAR    =  $_POST['qualyear'];

# we might check and forbid a new role of 'superuser'!

CheckCode($OTHERLOGONID, 'Bad logon id'); # or fail
CheckCode($NEWROLE, 'Bad role code');
CheckCodeNull($QUALYEAR);
if ($QUALYEAR != 'NULL')
   {
     $THISYEAR = WhatYearIsIt();
     if (  ($QUALYEAR < 1950)
         ||($QUALYEAR > $THISYEAR)
        )
        { $QUALYEAR = 'NULL';
        };
   };
Sanitise($NEWSURNAME);
Sanitise($NEWFORENAME);
if (  (strlen($NEWSPECIALTY) < 1)
    ||($NEWSPECIALTY == 0)
   ) { $NEWSPECIALTY = 'NULL';
     } else
     { CheckCode($NEWSPECIALTY, 'Bad specialty code');
     };
if (  (strlen($NEWLOCATION) < 1)
    ||($NEWLOCATION == 0)
   ) { $NEWLOCATION = 'NULL';
     } else
     { CheckCode($NEWLOCATION, 'Bad location code');
     };

$OTHERSURNAME = FetchSurname($handDB, $OTHERLOGONID);
$OTHERFORENAME = FetchForename($handDB, $OTHERLOGONID);

$qry = "UPDATE PERSDATA SET
          pdForename = '$NEWFORENAME',
          pdSurname  = '$NEWSURNAME',
          PersonRole = $NEWROLE,
          CurrentSpecialty = $NEWSPECIALTY,
          CurrentLocation = $NEWLOCATION
```

```
        WHERE person = $OTHERLOGONID";
DoSQL($handDB, $qry, 'update user data');

$qry = "UPDATE PERSON SET FirstQualified = $QUALYEAR
        WHERE person = $OTHERLOGONID";
DoSQL($handDB, $qry, 'set new qual yr');

list ($ROLETXT) = GetSQL ($handDB,
  "SELECT rText FROM PERSONROLE WHERE
    personrole = $NEWROLE", 'get role');

list ($SPECTXT) = GetSQL ($handDB,
  "SELECT LocationText FROM CURRENTLOCATION WHERE
    currentlocation = $NEWLOCATION", 'get role');

list ($LOCTXT) = GetSQL ($handDB,
  "SELECT SpecialText FROM CURRENTSPECIALTY WHERE
    currentspecialty = $NEWSPECIALTY", 'get role');

print <<<HTML3
<p>Data have been updated for $OTHERFORENAME $OTHERSURNAME.
<p>Name is now $NEWFORENAME $NEWSURNAME;  role, specialty
  and location are ($ROLETXT; $SPECTXT; $LOCTXT)

<p><a href='eZ_do_edit.php?editperson=$OTHERLOGONID'>
   Back to previous page</a>

<p>[<a href='mainpage.php'>Return to main page</a>]
</body></html>
HTML3;
```

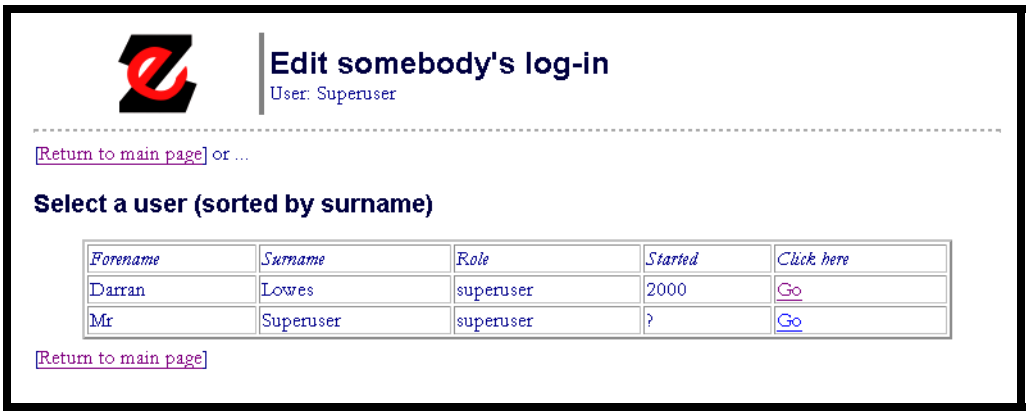

Figure 11: Editing a user log-on

9.6 Editing log-on details

The following page presents a table of users who are permitted to have log-on privileges. Clickable links are created which submit a GET to the page *eZ_do_editlogon*. The user ID submitted to that page is associated with the name *editlogon*.

eZ_edit_logon.php

```
 header( 'Cache-control: no-cache' );
 $THISPAGE = 'eZ_edit_logon.php';
 require_once('ValidFx.php'); # our login validation script
 $MINUSERSTATUS = eZ_ADMINISTRATOR_MIN;
 $MAXUSERSTATUS = EVERYONE;
 $success = validate_login(SHOW_USER);
 list ($MYHEADER, $USERSTATUS) = eZHeaderGreet($success,
                                $MINUSERSTATUS, $MAXUSERSTATUS,
                                "Edit somebody's log-in", 0);
print <<<HTML0
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en"> <head>
<title>Edit log-in</title>
<LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
</head>
<body>
$MYHEADER
<p>[<a href='mainpage.php'>Return to main page</a>] or ...
  <h3>Select a user (sorted by surname)</h3>
  <p><div align='center'>
  <table width='90%' border='2'>
  <tr><td><i>Forename</i></td>
      <td><i>Surname</i></td>
```

```
      <td><i>Role</i></td>
      <td><i>Started</i></td>
      <td><i>Click here</i></td></tr>
HTML0;

  $activeusers = array();
  $qry = "SELECT distinct Person FROM PERSDATA
     WHERE PersonRole > 0";
  $activeusers = SQLManySQL($handDB, $qry, 'get active users');
  $activeusers = Flatten($activeusers); #
  $users = GetLinkedUserDetails($handDB, $activeusers,
    "<a href='eZ_do_editlogon.php?editlogon=USERID'>Go</a>");
   MyDoubleSort($users, 1, 0); #sort by surname, then forename!
   PrintDetailTable($users, 5); # print table with 5 columns

print <<<HTML3
  </table></div>
<p>[<a href='mainpage.php'>Return to main page</a>]
  </body></html>
HTML3;
```

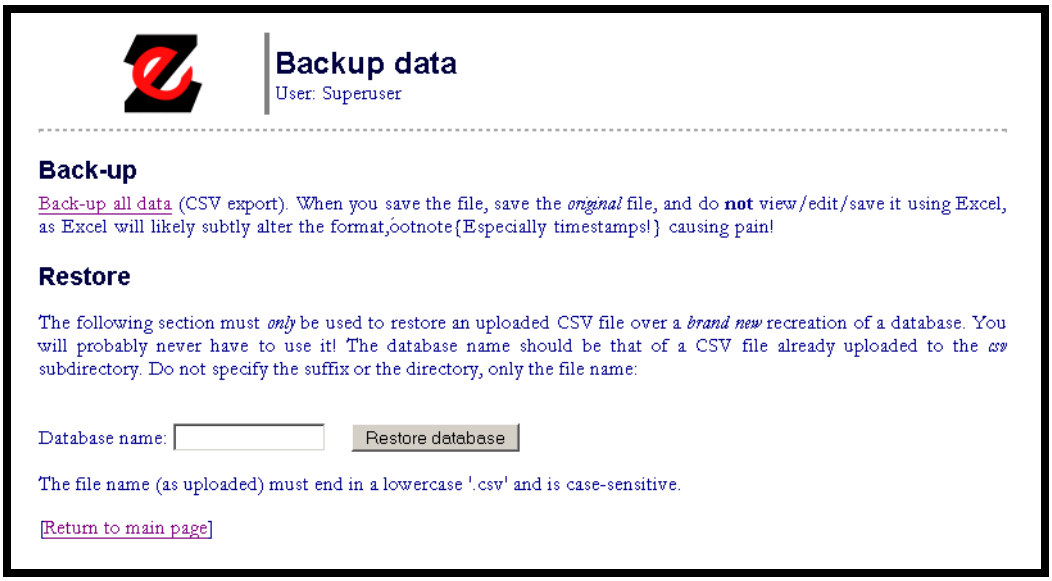

Figure 12: Backup and restore

9.7 Backup

Here we encourage easy backup of all data (apart from encrypted passwords) to a CSV file which can be re-imported if the dire need arises.

```
 header( 'Cache-control: no-cache' );
 $THISPAGE = 'eZ_backup.php';
 require_once('ValidFx.php'); # our login validation script
 $MINUSERSTATUS = eZ_ADMINISTRATOR_MIN;
 $MAXUSERSTATUS = EVERYONE;
 $success = validate_login(SHOW_USER);
 list ($MYHEADER, $USERSTATUS) = eZHeaderGreet($success,
                              $MINUSERSTATUS, $MAXUSERSTATUS,
                            'Backup data', 0);
print <<<HTML1
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en"> <head>
<title>Extract data</title>
<LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
  <script type='text/javascript'>
  <!--
 function CheckRestore(myform)
  { if (myform.csvname.value.length < 2)
       { alert ("Invalid CSV name. Need at least 1 character!");
         return(false);
       };
    if (! confirm("Restore? Are you sure?"))
```

```
       { return (false);
       };
    return (true);
  };

  //-->
  </script>
</head>
<body>
$MYHEADER
<h3>Back-up</h3>
<p><a href='backup_all.php'>Back-up all data</a> (CSV export). When you save
the file, save the <i>original</i> file, and do <b>not</b> view/edit/save it
using Excel, as Excel will likely subtly alter the format,\footnote{Especially tim

<h3>Restore</h3>
<div class='narrow'>
The following section must <i>only</i> be used to restore an uploaded
CSV file over a <i>brand new</i> recreation of a database. You will
probably never have to use it! The database name should be that
of a CSV file already uploaded to the <i>csv</i> subdirectory. Do
not specify the suffix or the directory, only the file name:

<p><FORM name="restoredatabase"
   ACTION="restore_db.php"
   METHOD="POST"
   onSubmit="return CheckRestore(this)" >
   <br> Database name: <input type="text" name="csvname" size="16">
      
   <INPUT TYPE="submit" NAME="submit" VALUE="Restore database">
 </form>

<p>The file name (as uploaded) must end in a lowercase '.csv' and is
case-sensitive.
</div>

<p>[<a href='mainpage.php'>Return to main page</a>]
</body></html>
HTML1;
```

restore_db.php

We read in a CSV and restore the database, if possible.

```
 header( 'Cache-control: no-cache' );
 require_once('ValidFx.php'); # our login validation script
 $MINUSERSTATUS = eZ_ADMINISTRATOR_MIN;
 $MAXUSERSTATUS = EVERYONE;
```

```
 $success = validate_login(SHOW_USER);
 list ($MYHEADER, $USERSTATUS) = eZHeaderGreet($success,
                            $MINUSERSTATUS, $MAXUSERSTATUS,
                           'Restoring database', 0);

print <<<HTML0
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html lang="en"> <head>
<title>View comments</title>
<LINK href="css/eZstyle.css" type='text/css' rel='stylesheet'>
</head>
<body>
$MYHEADER
HTML0;

GLOBAL $DEBUGGING;

$DEBUGGING = 1; # jvs temporary

$BIGNAME = $_POST['csvname'];
if (strlen($BIGNAME) > 0)
   { require('csv_read.php');  #
     mysql_query("START TRANSACTION");
     $ok = ReadBigCsv ($handDB, $BIGNAME);
     if ($ok > 0)
        { print "<p>There was/were $ok error(s)";
          mysql_query("ROLLBACK");
        } else
        { mysql_query("COMMIT");
        };
   } else
   { print "Bad CSV name. Nothing done!";
   };

$DEBUGGING = 0;

print <<<HTML3
<p>[<a href='mainpage.php'>Return to main page</a>]
</body></html>
HTML3;
```

A large CSV reader

This continues from the preceding PHP script. We permit a single CSV file to contain multiple data tables *in order*. To be able to do this we establish the conventions that:

- The file must contain within its first line the text:

```
%%DatabaseBackupName="eZ"
```

- Any line which starts with . . .

```
%%TableName=
```

  . . . forces termination of the previous table, and the start of the newly specified one;

- After the tablename line, there must be a ‘columndata’ line along the lines of:

```
%%ColumnData(4)=rating(int),Patient(int),Assessor(int),raText(string),
```

  The line must start with a double percent sign, followed by the word *ColumnData* with the number of columns in parenthesis and an equals sign. After this we have the names of the columns (each followed by its data type in parenthesis and a comma).

Let’s implement this functionality within *csv_read.php*, supplying the database handle and the name of the CSV file *without* a directory path and suffix. In order to completely restore a database, this file should be uploaded to the *csv* subdirectory of the *eZ* directory after the main database has been created (this creation may involve reading of other CSV files), and *before* new data have been inserted.

```
function ReadBigCsv ($handDB, $BIGNAME)
{# first check for people with an ID of 1000 or more. If so, fail!
  $qry = "SELECT COUNT(person) FROM PERSON
           WHERE person > 999";
  list($PERSCOUNT) = GetSQL($handDB, $qry, 'get new people');
  if ($PERSCOUNT > 0)
     { print "<p>There are already $PERSCOUNT new people in the
        database, so I won't restore anything!";
       return(1);
     };
  $qry = "DELETE FROM UIDS WHERE uids = 1";
  DoSQL($handDB, $qry, 'delete UIDS!');
  # allow new insertion into UIDS.
 $errcount = 0;
  $CSVFILE = "csv/$BIGNAME.csv";
  if (! @stat($CSVFILE)) # if not exists
     { print ("<br>File not found: $CSVFILE");
       return (1);      # signal 1 error
     };
```

```
  $BIGLINES = file($CSVFILE); # get all lines
  $NEWCSV = 0;
  $headarray = '';
  $csvname = '';
  $L = $BIGLINES[0];
  if (! preg_match ( '/%%DatabaseBackupName="eZ"/', $L))
     { print "<p>Woops! big bad CSV";
       return (1);
     };
  foreach ($BIGLINES as $L)
    { print "<br>Debug parsing &lt;$L&gt;";
      if (preg_match ( '/^%%/', $L))
         { if (preg_match( '/^%%TableName="(.+)"/', $L, $fx))
              { $csvname = $fx[1]; # pull out name
                print "<P>Table '$csvname' identified";
              }
  elsif (preg_match( '/^%%ColumnData\((\d+)\)=(.+),.*/', $L, $fx))
              # note the greedy match, and final comma
              { $colcount = $fx[1]; # not needed. ? a check.
                $headarray = explode(',', $fx[2]);
                $headarray = FixNewHead($headarray);
                print_r($headarray); # debug
                if (count($headarray) < 1)
                   { print "<p>Bad (null) head array";
                     return ($errcount+1);
                   };
              } else
              { # do nothing: it's just a comment
              };
         } else # likely to be data
         { if (strlen($L) > 3) # arbitrary
              { $L = rtrim($L);
                $L = rtrim($L, ','); # remove terminal comma!
                $errcount += SaveCsvLine(strtoupper($csvname),
                             $headarray,
                             $L,
                             $handDB);
    };   };   };
  return ($errcount);
}

function FixNewHead ($HA)
{ $OPT = array();
  $i = 0;
  foreach ($HA as $L) # [ugh]
    { print "<br> debug head item: '$L'";
      if (! preg_match( '/(\w+)\((\w+)\)/', $L, $fx))
         { print "\n<p>Error: Bad CSV head item: '$L'";
```

```
          return (’’);
        };
      $itm = $fx[1];
      $type = $fx[2];
      if ($type == ’int’)
         { $OPT[$i] = $itm;
         }
      elsif ($type == ’string’)
         { $OPT[$i] = "’$itm’";
         }
      elsif ($type == ’date’)
         { $OPT[$i] = "DATE ’$itm’";
         }
      elsif ($type == ’timestamp’)
         {  $OPT[$i] = "TIMESTAMP ’$itm’";
         }
      # ultimately will need more (float, time, ..)
      else { print "\n<p>Error: Bad CSV item type in ’$L’";
             return (’’);
           };
      $i += 1; # bump output index
    };
  return ($OPT);
};
```

CSV back-up: backup_all.php

This page is clumsy. We print data for all SQL tables to a single CSV file, hoping that we never have to use these data.

We constrain our export to rows where the key is $>=$ 1000. Initially we used the mySQL ‘SHOW TABLES’ functionality, but from version 0.64 we keep a list of tables as we create them (in the METATABLES table) and here we simply keep the same order in writing the tables. This approach allows us to re-create the tables without inserting items which depend on others!

```
# PHP to generate a CSV file for download:
header("Content-type: application/octet-stream");
header("Content-Disposition: attachment; filename=\"eZ-FULLBACKUP.csv\"");
header( ’Cache-control: no-cache’ );

 $THISPAGE = ’backup_all.php’;
 require_once(’ValidFx.php’); # our login validation script
 $MINUSERSTATUS = eZ_ADMINISTRATOR_MIN;
 $MAXUSERSTATUS = EVERYONE;
 $success = validate_login(SHOW_USER);  # provides $handDB

$date = date("F j, Y");
```

```
print "%%DatabaseBackupName=\"eZ\",DatePerformed=\"$date\"";
print "\n" . "%% If you wish to restore the database, you must";
print "\n" . "%%  upload this file to the csv subdirectory, ";
print "\n" . "%%  re-create a fresh database, and Restore";

 # get list of tables:
 ## $qry = "SHOW TABLES"; # good in mySQL
 ## $tbllist = SQLManySQL($handDB, $qry, 'show tables');

 $qry = "SELECT TableName FROM METATABLE ORDER BY metatable";
 $tbllist = SQLManySQL($handDB, $qry, 'get tables');

 $tbllist = Flatten($tbllist);
 foreach ($tbllist as $t)
   { $keymin = 1000;    # default
     if ($t == 'UIDS')  # vital!
        { $keymin = 1;
        }; # clumsy hack (minimum key):
     PrintTableCsvData ($handDB, $t, $keymin);
   };
 print "\n\n" . "%% ---------END OF DATA------------ ";

function PrintTableCsvData ($handDB, $tname, $keymin)
{
  ## print "\n%%DEBUGGING: Table is $tname, minimum key $keymin";
  $HDR = "\n\n" . "%%TableName=\"$tname\""; # record table name
  $keyname = strtolower($tname); # lower case key: our rule!

  # Include column meta-data here:
    $HDR .= "\n" . "%%ColumnData";
  # we have the minimum possible key, $keymin, but
  # the actual minimum might be larger, so check this!
  $qry = "SELECT MIN($keyname) FROM $tname WHERE
     $keyname >= $keymin";
  list($km) = GetSQL($handDB, $qry, 'get actual min');
  if (strlen($km) > 0)
     { $keymin = $km;
     };

  # get the very first row, so we can get field attribs:
    $qry = "SELECT * FROM $tname WHERE $keyname = $keymin";
    $RSLT = mysql_query($qry);
    $FLDS = mysql_num_fields($RSLT);
    $j = 0;
    $HDR .= "($FLDS)=";
    ## print "\n%% Debug: Number of fields is $FLDS";
    $FLDLIST = '';
    while ($j < $FLDS) # for each field
```

```
      { $fnam = mysql_field_name($RSLT, $j);
        $ftype = mysql_field_type($RSLT, $j);
        if (! preg_match( '/PASSWORD/i', $fnam)) # no password!
           { $HDR .= "$fnam($ftype),";
             $FLDLIST .= "$fnam,";
           };
        $j += 1;
      };
    $FLDLIST = rtrim($FLDLIST, ","); # trim terminal comma
    if (mysql_fetch_array($RSLT)) # if any rows:
       { print $HDR;  # print header;
                      # next, get and print all data:
         $qry = "SELECT $FLDLIST FROM $tname WHERE $keyname >= $keymin";
         ## print "\n%% debug sql: $qry";
         $ALLDATA = SQLManySQL($handDB, $qry, 'get all table data');
         foreach ($ALLDATA as $DROW)
            { print "\n"; # new row
              foreach ($DROW as $DI)
                { # replace CR within data with \n:
                  $DI = str_replace("\n", '\n', $DI);
                  # replace comma with \,
                  $DI = str_replace(",", '\,', $DI);
                  print "\"$DI\",";
                };
            };
       };
};
```

10 Ancillary code

The following ancillary functions are used throughout the database. They are all contained within *ancillary.php*. The image *tinyezlogo.png* is also required, as is the trivial *vbar.png*

10.1 General-purpose functions

10.1.1 eZHeaderGreet

Return text string, *and* add in header with image!

```
function eZHeaderGreet($success,
                          $MINUSERSTATUS,
                          $MAXUSERSTATUS,
                          $title,
                          $previous)  # $previous is 0 or 1.
{ $matches = '';
  list($USER, $USERSTATUS, $OLDPAGE) = PullOutUserInfo($success);
if (  ($USERSTATUS < $MINUSERSTATUS)
    ||($USERSTATUS > $MAXUSERSTATUS)
   )
   { readfile ('failedaccess.htm');
     exit();  # fail
   };
# print ("<br>Debug: user status is $USERSTATUS");
$OLD=''; # might try eg (Welcome!) [hmm]
if ($previous)
   { if (strlen($OLDPAGE) > 3)
        { $OLD =
           "(previously at <a href='$OLDPAGE'>$OLDPAGE</a>)";
        };
   };
$txt[0] = "<p><div align=center>
<table width='95%'>
<tr><td width='150' align='center'>
<a href='GPL.htm'><img border='0' title='Click here!' alt='eZ logo'
  src='images/tinyezlogo.png'></a></td>
<td width='4'><img alt='' width='4' height='64' src='images/vbar.png'>
</td>
<td><h2>$title</h2>
  <span class='info'>User: $USER $OLD</span></td>
</tr></table>
<hr>
</div>";
```

```
$txt[1] = $USERSTATUS;
 return ($txt);
}
```

10.1.2 CheckCode

Check that item is numeric.

```
function CheckCode(&$code, $msg)
{  if (! preg_match ( '/^\s*(\d+)\s*$/', $code, $vals) )
      { print ( "<br>Bad data item &lt;$code&gt;.
                 <p> $msg. ");
        readfile('badcode.htm');
        exit();
      };
  $code = $vals[1]; # trim whitespace, just in case.
};
```

Similar to CheckCode is CheckCodeNull, which however doesn't exit, merely returning NULL if the match fails:

```
function CheckCodeNull(&$code)
{  if (! preg_match ( '/^\s*(\d+)\s*$/', $code, $vals) )
      { $code = 'NULL';
      };
  $code = $vals[1]; # trim whitespace
};
```

10.1.3 Sanitise

Clean up data input:

```
function Sanitise (&$txt)
{
  # first remove whitespace at start, end:
  $matches = '';
  preg_match ( '/^\s*(.*)\s*$/', $txt, $matches);
  $txt = $matches[1]; # get $1; or use ltrim, rtrim.
  $txt = preg_replace( '/\|/', '', $txt); # get rid of pipes
  $txt = preg_replace( '/\`/', '', $txt); # and backticks
  $txt = preg_replace( '/"/', "'", $txt); #double quote -> single
  $txt = preg_replace( "/'/", "''", $txt);#duplicate any quote
  if ($DEBUGGING)
     {
       print "<br><b>Sanitised:</b> &lt;$txt&gt;";
     };
}  # return with altered $txt (It's by reference)
```

10.1.4 WhatYearIsIt

Simply retrieve the current year!

```
function WhatYearIsIt()
{ return ( date("Y") );
};
```

10.2 Handling lists

10.2.1 PrintPoplist

PrintPoplist obtains the desired primary key (the first item specified in the $SQL SELECT statement) and $elems extra items. ($elems is not an explicit parameter). The $elems extra items are concatenated into a single string! Each (key; $elems) concatenation pair is printed as an *option* within an HTML <SELECT> statement.

```
function PrintPoplist ($handDB, $listname, $query )
{
  print( "<select name='$listname' >
         <option selected value='0'>?</option>\n");
  $handQ = mysql_query($query, $handDB);
  if (! is_resource($handQ))
    { # debug:
      print ( "<br> SQL QUERY error: " . mysql_error() );
    };
  $count = 0;
  while ($row = mysql_fetch_array($handQ, MYSQL_NUM))
    { $mykey = $row[0];
      $myvalue = ConcatenateArray( array_slice($row, 1) );
        # concatenate remaining elements
      print ( "<option value='$mykey'>$myvalue</option>\n" );
      $count += 1;
    };
  print ('</select>');
  return($count);
}
```

10.2.2 TextPoplistSelected

This function is similar to PrintPoplist. It accepts the key of the *selected* item which is then flagged as selected.

```
function TextPoplistSelected ($handDB, $listname, $query, $k)
{ $opt = '';
```

```
  $s = '';
  if ($k < 1)
     { $s = 'selected';
     };
  $opt = $opt . sprintf( "<select name='$listname' >
  <option $s value='0'>?</option>\n");
  $handQ = mysql_query($query, $handDB);
  $count = 0;
  while ($row = mysql_fetch_array($handQ, MYSQL_NUM))
    { $mykey = $row[0];
      $myvalue = ConcatenateArray( array_slice($row, 1) );
        # concatenate remaining elements
      $s = '';
      if ($k == $mykey)
         { $s = 'selected';
         };
      $opt = $opt . sprintf (
        "<option $s value='%d'>%s</option>\n", $mykey, $myvalue );
      $count += 1;
    };
  $opt = $opt . sprintf ('</select>');
  return($opt);
}
```

10.2.3 PrintActiveUserlist

```
function PrintActiveUserlist($handDB, $link)
{
  print <<<HTMLpau
  <p><div align='center'>
  <table width='90%' border='2'>
  <tr><td><i>Forename</i></td>
      <td><i>Surname</i></td>
      <td><i>Role</i></td>
      <td><i>Started</i></td>
      <td><i>Click here</i></td></tr>
HTMLpau;

  $activeusers = array();
  $qry = 'SELECT distinct Person FROM PERSDATA
     WHERE PersonRole > 0';
  $activeusers = SQLManySQL($handDB, $qry, 'get active users');
  $activeusers = Flatten($activeusers); #
  $users = GetLinkedUserDetails($handDB, $activeusers, $link);
   MyDoubleSort($users, 1, 0); #sort by surname, then forename!
   PrintDetailTable($users, 5); # print table with 5 columns
}
```

10.3 Tables, arrays and sorting

10.3.1 MyDoubleSort

```
# Given 2-D array of rows, sort, first by the first column,
#  then by the second (if first column entries are equal):
# similar to MySort.

function MyDoubleSort(&$arry, $col1, $col2)
{
  GLOBAL $CLUMSY_INDEX;  #
  $CLUMSY_INDEX = $col1;
  GLOBAL $CLUMSY_INDEX2;  #
  $CLUMSY_INDEX2 = $col2;
  usort ($arry, 'my_dblsort');
}

function my_dblsort ($first, $second)
{
  GLOBAL $CLUMSY_INDEX;
  GLOBAL $CLUMSY_INDEX2;
  $r = strcmp ($first[$CLUMSY_INDEX], $second[$CLUMSY_INDEX]);
  if ($r)  # if nonzero (not the same):
     { return ($r);
     };
  # two items are equal, so compare sub-items:
  return ( strcmp ($first[$CLUMSY_INDEX2], $second[$CLUMSY_INDEX2]) );
}
```

10.3.2 ConcatenateArray

Join all array elements into a single string, separating elements with blanks. There is a terminal blank.

```
function ConcatenateArray ($ary)
{ $opt = '';
  foreach ($ary as $itm)
    { $opt .= "$itm ";
    };
  return($opt);
};
```

10.3.3 Flatten

Given an array of many dimensions, flatten to a unidimensional array, which is returned as a new array. The problem with the following is the deprecated Call-time pass-by-reference ...

```
function Flatten($myarray) # don't use me!
{
  $simflat = array();
  array_walk($myarray, 'flatten_array', &$simflat);
  # see flatten_array fx below!
  return ($simflat);
}

function flatten_array($value, $key, &$array)
{
  if (!is_array($value))
     { array_push($array,$value);
     } else
     { array_walk($value, 'flatten_array', &$array);
     };
}
```

...so instead we use:

```
function Flatten($myarray)
{
  $Aout = array();
  foreach ($myarray as $a)
    { if (is_array($a))
         { $J = Flatten($a);
           foreach ($J as $i)
             { array_push ($Aout, $i);
             };
         } else
         { array_push ($Aout, $a);
         }
    };
  return ($Aout);
}
```

10.3.4 PrintDetailTable

```
function PrintDetailTable($data, $cols)
{ # cols is number of colums, $data = array of data.
  foreach ($data as $d)
    {
      # print ('<br>Deeper debug: ');
      # print_r ($d);
      print ('<tr>');
      $i = 0;
      while ($i < $cols)
        { print ('<td>');
          if (strlen ($d[$i]) < 1)
```

```
            { print '?';  # should work if NULL
            } else
            { print $d[$i];
            };
          print ('</td>');
          $i += 1;
        };
      print ('</tr>');
    };
}
```

10.4 User information

10.4.1 PullOutUserInfo

Given a 'user string', pull out user and associated details. Returns a list of the three items in that order.

```
function PullOutUserInfo ($userstring)
{
  $matches = '';
  if (! preg_match ( '/USER="(.+)"\|STATUS="(.+)"\|OLDPAGE="(.*)"/',
                     $userstring, $matches) )
     { print (
 "<br>Error. Bad user data. Terminated. String was &lt;$userstring&gt;" );
       exit();
     };
  array_shift($matches);  # remove first element (which is the whole match)
  return ($matches);
}
```

10.4.2 GetLinkedUserDetails

The following is like GetUserDetails from eZ_admin_personadded.php *but* the added parameter $link is something along the lines of:

```
 "<a href='eZ_do_editlogon.php?editlogon=USERID'>Go</a>";
```

A required component is the text string USERID, replaced with the relevant key contained in the list $link. The URL represents a GET (not POST) to the page specified e.g. eZ_do_editlogon

```
function GetLinkedUserDetails($handDB, $unames, $link)
{ # $unames is array of IDs (key of PERSON table)
  # this fx is cumbersome and slow.
  # returns array of arrays, each subarray containing
```

```
  # forename, surname, role, date started practice
  # (as year-01-01) and link URL (contains ID).

  $opt = array();
  $i = 0;
  foreach ($unames as $p)
    { $detl = array();
      $detl[0] = FetchForename ($handDB, $p);
      $detl[1] = FetchSurname($handDB, $p);
      $detl[2] = FetchRole ($handDB, $p);
      list ($detl[3]) = GetSQL ($handDB, "SELECT FirstQualified
        FROM PERSON WHERE person = $p",
        'GET year of 1st qualification');
      $detl[4] = $link;
      # use str_replace rather than regex:
      $detl[4] = str_replace ('USERID', $p, $detl[4]);
      # might check for success.
      $opt[$i] = $detl;
      $i += 1;
    };
  return ($opt);
}
```

10.4.3 FetchSurname

Given a person's ID, obtain their surname.

```
function FetchSurname ($handDB, $id)
{
  $qry = "SELECT MAX(persdata) FROM PERSDATA
           WHERE Person = $id AND
             pdSurname IS NOT NULL";
  list($pd) = GetSQL ($handDB, $qry, "fetch key for surname");
  $qry = "SELECT pdSurname FROM PERSDATA WHERE persdata = $pd";
  list($surname) = GetSQL($handDB, $qry, "fetch surname");
  return ($surname);
}
```

10.4.4 FetchForename

In a similar fasion to **FetchSurname**, obtain a person's forename.

```
function FetchForename ($handDB, $id)
{
  $qry = "SELECT MAX(persdata) FROM PERSDATA
           WHERE Person = $id AND
             pdForename IS NOT NULL";
```

```
  list($pd) = GetSQL ($handDB, $qry, "fetch key for forename");
  $qry = "SELECT pdForename FROM PERSDATA WHERE persdata = $pd";
  list($forename) = GetSQL($handDB, $qry, "fetch forename");
  return ($forename);
}
```

FetchRole

The following can be made far simpler if we accept that in this database a person will only ever have one role.

```
function FetchRole ($handDB, $id)
{
  $qry = "SELECT MAX(persdata) FROM PERSDATA
           WHERE Person = $id AND
             PersonRole IS NOT NULL";
  list($pd) = GetSQL ($handDB, $qry, "fetch key for role");
  $qry = "SELECT PERSONROLE.rText FROM PERSDATA,PERSONROLE
            WHERE PERSDATA.PersonRole = PERSONROLE.personrole AND
             PERSDATA.persdata = $pd";
  list($PersonRole) = GetSQL($handDB, $qry, "fetch role");
  return ($PersonRole);
}
```

10.5 CSV processing

The variable $TOTALLINESREAD is declared as global in SaveCsvLine and incremented there for each successful line inserted. Note that the header array is exploded willy nilly, so you're likely to get duff SQL code if the format of this line isn't exactly correct! (Another good reason not to allow users to submit such CSV files). Here's the meaty routine to parse lines and insert them as database rows:

```
function SaveCsvLine($TABLENAME, $headarray, $thisline, $handDB)
{
  GLOBAL $DEBUGGING;
  GLOBAL $TOTALLINESREAD;

  $cols = count($headarray);
  if ($cols < 1)
     { print ( "<br>Column has no length for table $TABLENAME" );
       return 1; # fail
     };

  # first fix up format of thisline:
  #   convert \, to a temporary backtick
```

```
  #   We will convert this back to comma later(FormatOneItem)
  $thisline = preg_replace( '/\\\\,/', '', $thisline);
  # split up the line at each comma:
  $itemarray = explode (',', $thisline);
  if ( count($itemarray) != $cols)
     { $c = count($itemarray);
       print ( "<br><b>Error</b>--
 Mismatch between column count($cols) and line($thisline)[$c]" );
       print_r ( $itemarray );
       return (1); # fail.
     };
  if ($DEBUGGING)
     { print ( "<br>--line data:" );
       print_r ( $itemarray );
     };

  $left = '';
  $right = '';
  $i = 0;
  while ($i < $cols)
    { list($col, $val) = FormatOneItem($headarray[$i],
                                       $itemarray[$i]);
      $left  .= "$col,";
      $right .= "$val,";
      $i ++;
    };
  # remove final commas:
  $left  = rtrim ($left, ',');
  $right = rtrim ($right, ',');
  $qry = "INSERT INTO $TABLENAME ($left) VALUES ($right)";
  if ($DEBUGGING)
     { print ( "<br>--QUERY: $qry" );
     };
  if (! mysql_query( $qry, $handDB ) )
     { print ( "<br> ** SQL ERROR:" . mysql_error() );
       return 1; # fail
     };
  $TOTALLINESREAD ++; # bump count of successful inserts
  return 0; # success
}
```

A subsidiary routine, FormatOneItem formats a single data item:

```
function FormatOneItem ($col, $itm)
{
  $retval;  # return array of 2 elements
  $retval[0] = $col; # default.
```

```
  $itm = preg_replace ( '/\'/', ',', $itm );
     # restore commas
  $itm = preg_replace ( '/\\n/', "\n", $itm);
     # fix carriage returns
  $itm = preg_replace ( "/'/", "''", $itm);
     # duplicate single quotes

  $mtch;
  if (preg_match ( '/^\s*"(.*)"\s*$/', $itm, $mtch) )
     #encasing quotes
     { $itm = $mtch[1]; # get $1
     } else
     { $itm = rtrim($itm);
       $itm = ltrim($itm);
     };
  if (strlen($itm) < 1)
     { $itm = 'NULL';
     };
  $retval[1] = $itm; #default return value

  # next pull out column name:
  # NOTE: column name must NOT contain double quotes!
  if (! preg_match ( "/\s*(\w*)\s*'(.+)'\s*/", $col, $mtch ) )
  # [check the above line in view of the double quotes ???]
     { # if no match, must be integer, simply return defaults:
       return ($retval);
     };

  $nature = $mtch[1]; # $1, can be null
  $retval[0] = $mtch[2];  # column name is $2
  if ($retval[1] == 'NULL')
     { return ($retval);
     };

  $retval[1] = "$nature '$itm'";
  # an example is "DATE '2006-01-03'"
  # simpler than complex switch/case stmt!
  return ($retval); # return both values.
}
```

10.6 SQL-related functions

10.6.1 GetSQL

Obtain a single row from the database as an array.

```
function GetSQL ($handDB, $qry, $message)
{
  if ($DEBUGGING)
     { print ("<br>DEBUGGING: Query was &lt;$qry&gt; ==&gt; ");
     };
  $handQ = mysql_query($qry, $handDB);
  if (! is_resource($handQ))
     { print ( "<br>SQL error &lt;$qry&gt;<br> ($message):"
               . mysql_error() );
       exit(); # this is serious: FAIL!
     };
  # problem: SELECT MAX(mysession), sTrainee FROM MYSESSION
  #    WHERE sAssessor = 3 GROUP BY sTrainee
  # IF there are no entries in MYSESSION this will not fail,
  # but also will not return a valid handle in $handQ. Thus:
  if (   (! is_resource($handQ)) # is a resource!
      || (! mysql_num_rows($handQ))
     )
     { return('');  # NULL
     };
  $row = mysql_fetch_array($handQ, MYSQL_NUM);
  if ($DEBUGGING)
     {
      print (" ==&gt;");
      print_r ($row);  # DEBUGGING
     };
  return ($row); # return array!
}
```

10.6.2 DoSQL

```
function DoSQL ($handDB, $qry, $message)
{
  $ok = mysql_query($qry, $handDB);
  if (! $ok)
     { print "<p>Query error &lt;$qry&gt;";
       $e = mysql_error();
       print "<p>Error message: $e";
     };
  if ($DEBUGGING)
     { print ("<br>DEBUGGING: Stmt to execute was &lt;$qry&gt; ");
     };
```

```
  # simply return.
 }
```

10.6.3 SQLManySQL

Obtain array of arrays (all data).

```
function SQLManySQL ($handDB, $qry, $msg)
{
  $handQ = mysql_query($qry, $handDB);
  if (! is_resource($handQ))
    { # debug:
      print ( "<br> SQL QUERY error: ($msg) " . mysql_error() );
      exit(); # serious failure. [???]
    };
  $outAry = array();
  if (! is_resource($handQ))
     {
       print ( "<br> SQL QUERY error: ($msg) " . mysql_error() );
       return ($outAry);
     };
  $i = 0;
  while ($row = mysql_fetch_array($handQ, MYSQL_NUM))
    { $outAry[$i] = $row; # array of arrays
      $i ++;
    };
  return ($outAry);
}
```

10.7 FetchKey

Within *ancillary.php* we initially had a FetchKey function as follows, but there are substantial (albeit infrequent) problems with this approach:

```
function FetchKey ($handDB, $ky)
{
  # start atomic:
  list($keyval) = GetSQL($handDB,
                    "SELECT u$ky FROM UIDS WHERE uids = 1",
                    "get key value");
  $keyval ++; # the problem is here..!
  DoSQL($handDB,
         "UPDATE UIDS SET u$ky = $keyval WHERE uids = 1",
         "set new key value");
  #end atomic.

  $keyval --;
  return ($keyval);
}
```

The above doesn't cater for two competing processes, and the requirement that such key increments must be atomic. We therefore describe a generic mechanism for ensuring retrieval of sequential keys in an SQL database.

10.7.1 The problem

The problem is as follows:

1. There is no *standard* mechanism for generating auto-incrementing keys, and most solutions require either proprietary modifications to the SQL standard, or use of non-core measures such as invocation of appropriate functions written in a language acceptable for extending SQL.

2. If we try something along the lines of the above, where two SQL statements are used with an intervening increment, there are many potential points of breakdown. As the two SQL statements with the intervening increment are not atomic, another process can interrupt, retrieving the same value, and then can use this same key, also writing the key value once more. The potential errors are numerous, as even more than one process might interfere.

3. Proprietary solutions exist which should allow the user to conjoin multiple queries into a single 'atomic' statement but we wish for a more generic solution.

10.7.2 An initial solution

How about the following schema? (In all of the following we assume we have a ‘key generator’ table called UIDS that contains a single row filled with key-generating columns).

1. First, lock the key generator. We assume that we have two keys in the ‘UIDS’ generator table, uValue and uValueLock, and that (for now) both are integers:

```
$count = 0;
$ok = 0;
while (! $ok && ($count < 1000))
   { $count ++;
     $ok = TryDoSQL($handDB,
       "UPDATE UIDS
         SET uValueLock = uValue+1 WHERE uValueLock = uValue",
       'lock generator field');
   };
```

2. Next, either fail (if repeated attempts all failed), or fetch the new generated value:

```
if ($ok)
   { ($j) = GetSQL($handDB,
       "SELECT uValueLock FROM UIDS",
       'get new key');
   } else
   { return (0); # fail
   };
```

3. Finally, allow others to get the next value:

```
DoSQL($handDB,
       "UPDATE UIDS SET uValueLock = $j",
       'release lock');
```

10.7.3 A refined solution

The preceding approach seems fairly robust, but there is a major problem — if a process locks a key generator and then dies, the generator is locked forever. We therefore need a robust timeout. If all processes are accessing the same source of a timestamp, here is one solution:

1. Lock the generator as above (assuming the initial value in uValueLockTxt *is* the same as uValue), but write a timestamp (provided as $now) to uValueLock.[22] This will cause subsequent attempts to fail nastily! But anyone *failing* can now see when the value was set.

```
$count = 0;
$ok = 0;
while (! $ok && ($count < 1000))
   { $count ++;
     $ok = TryDoSQL($handDB,
       "UPDATE UIDS
         SET uValueLock = '$now'
           WHERE uValue = cast(uValueLock AS INTEGER)",
       'lock generator');
```

   Here's the catch. Within the same loop, if we fail, check that the previous person who grabbed control hasn't died disgracefully. If this isn't the case (no time-out) then we delay and try again, but if the person has timed out, we still grant them the increment (by doing it ourself), but reset uValueLock to again make it accessible:

```
if (! ok)
   { ($timeout) = GetSQL($handDB,
      "SELECT uValueLock FROM UIDS",
      'get lock time');
      $delta = Julian($now) - Julian($timeout);
      if ($delta > $MAXTIMEOUT)
         { DoSQL($handDB,
           "UPDATE UIDS
             SET uValueLock = cast(uValue+1 as varchar(32)),
                 uValue = uValue+1
             WHERE uValueLock = '$timeout'",
           'restore function');
         } else
         { # here we might insert a random delay!
```

[22] We assume that $now is an accurate timestamp, with a millisecond value and preferably a microsecond value.

```
                };
            };
        };
```

Here, Julian calculates a Julian date from a standard timestamp, which is a floating point value. Alternatively we might use modified unix timestamps without the Julian fussing. One good modification would be to use a microtime value combined with a UNIX timestamp modulo an appropriate interval! (It would then be wise to determine the maximum key value *ever* and always add this to the modified timestamp, to prevent that odd failure — See below).

2. There is a residual catch. How do we obtain our value?

```
if (! $ok)
   { return (0); # fail
   };
($j) = GetSQL($handDB,
       "SELECT uValue FROM UIDS
         WHERE uValueLock = '$now'",
       'get new key');
$j ++;
```

The above ensures that if a reset has already occurred, the key fetch will fail. It is true that if this occurs, then there will be a 'gap in the record' but this gap signals a process defect which must be identified!

The important consideration is that if the 'timeout detecting process' *now* kicks in, the number will still be updated. If it doesn't then the following will succeed, and the two are mutually exclusive:

3. Finally, allow others to get the next value:

```
DoSQL($handDB,
       "UPDATE UIDS SET uValueLock = '$j',
                    SET uValue = $j
        WHERE uValueLock = '$now'",
       'release lock');
```

This will fail if somebody has already performed the reset, but even then the increment will have occurred, and we can proceed, so there is no need to test for success!

Note that the casts and multiple accesses in the above will tend to slow things a fair bit, so if we really require vast numbers of keys generated quickly, this 'solution' is probably unwise.

10.7.4 Working code

Let's flesh out the above ideas. First we will generate a number which 'reflects' a timestamp without being a full timestamp. We will use the microtime timer to obtain a microseconds stamp, which we modify as follows:

```
function ModTimeNow ()
{ list($usec, $sec) = explode(' ',microtime());
 $sec %= 1000000; # modulo 1 million seconds
 $stamp = (int) (1000*((float)$usec + (float)$sec));
 return ($stamp);
}
```

Let's assume that we want a granularity of 1 millisecond in our counting. We wish to time out if a key locks for more than, say, a few seconds (say 2 seconds). We'll use a large 'safety factor', so that even if a process fails in the middle of key retrieval, several days must elapse before the process briefly 'appears not to have failed' again.

The number we store is thus the current UNIX timestamp modulo 1 000 000 (seconds). We add the microseconds, multiply this modified number by 1000 to get millisecond chunks, and thus create an integer value between 0 and 1 billion.

A subsequent program which is locked out calculates the current timestamp (modulo 1 million seconds as above). If the value has wrapped (the current is less than the stored value), we first add 1 million to this number, but otherwise we simply find the difference. If the difference is over the cutoff (say 2 seconds, i.e. 2000 microseconds) then timeout has occurred, and we act accordingly.

We have accordingly modified the SQL for creation of the UIDS table — in previous versions all of the aux- columns in UIDS were varchar(32), but now they are simple integers.[23] See Section 7.7.

Now we can set about fetching a key. We assume the existence of the constants MAXKEY, for example set at 1 billion, MAXFETCHTRIES (perhaps 1000), and the FETCHTIMEOUT, set at say 2000 milliseconds.

```
function FetchKey ($handDB, $ky)
{ $now = MAXKEY + ModTimeNow(); # +MAXKEY prevents collision!
##  print "<p>Debug: time modification is [$now].";
  $count = 0;
  $ok = 0;
  while (! $ok && ($count < MAXFETCHTRIES))
     { $count ++;
       $q = "UPDATE UIDS SET aux$ky = $now
```

[23] If the database used doesn't support integer values up to 9 digits, then the code might need to be modified ever so slightly.

```
            WHERE aux$ky = u$ky";
       $ok = TryDoSQL($handDB,
         $q,
         'lock generator');
##       print "<p>Debug: attempt was &lt;$q&gt; giving $ok";

        # problem is 'succeed' if no rows updated, thus:
       if ( $ok)  # if apparent success:
          {  $q = "SELECT u$ky FROM UIDS
                WHERE aux$ky = $now";
             list($j) = GetSQL($handDB,
                $q,
                'get new key');
##             print "<p>Tried: &lt;$q&gt; giving $j.";
             if (strlen($j < 1))
                { $ok = 0;
                };
          };
       if (! $ok) # if failed
          { list($locktime) = GetSQL($handDB,
             "SELECT aux$ky FROM UIDS",
             'get lock time');
            if (strlen($locktime) < 1)
              { die ("<p>KEY FETCH FAILURE. Null ($ky)");
              };
## print "<p>Debug: Key clash: ($ky)"; # debug only
             if ($locktime > $now) # if wrapped..
                { $now += 1000000000;
                };
             if ( ($now-$locktime) > FETCHTIMEOUT)
                { DoSQL($handDB,
                  "UPDATE UIDS
                    SET aux$ky = u$ky+1,
                        u$ky = u$ky+1
                    WHERE aux$ky = $locktime",
                  'unlock generator');
## print "<p>Debug: unlocking generator ($ky: $locktime)"; # debug only
                }; # next, sleep a bit..
             usleep(rand()%10000); # up to 10ms
          };
     };
  if (! $ok) # if fail after MAXFETCHTRIES, really fail!
     { die ("<p>Aagh! key generation failed ($ky)");
     };
  # set the new value, unlocking: (might simply use $j in the following)
  DoSQL($handDB,
         "UPDATE UIDS SET aux$ky = u$ky+1,
                          u$ky = u$ky+1
```

```
          WHERE aux$ky = $now",
         'release lock');
  return ($j);
}
```

We might wish to use the TryDoSQL variant and print a reassuring warning instead of the more general 'failure' of DoSQL in the last statement!

Note that in the above, we return as the new key the original key value from the database (not the incremented value as in our previous example). The next fetch will retrieve this incremented value, and so on.

Here's a variant of DoSQL which returns 1 on success, 0 on failure, *without* an error message:

```
function TryDoSQL ($handDB, $qry, $message)
{
  return (mysql_query($qry, $handDB));
 }
```

11 Hardware, software, useful files & Change Log

Hardware

[Describe]

Notes on languages and programs

At present, SAFERsleep runs under Microsoft Windows, and therefore all of our code has been developed in this environment. The code might however be modified for similar purposes on dissimilar operating systems. AutoIt is a scripting language designed to compensate for the inadequacies of Windows, but Perl is cross-platform. We assume that the user can interact in a basic fashion with the DOS console under Windows. The code runs well under Windows 2000 and Windows XP. This code has not been tested under Windows Vista, and it is likely (owing to the somewhat paranoid behaviour of this incarnation of Windows) that problems will be encountered with this platform.

At present, all files used in this suite of programs can be generated from a single source — the file *ez-XXX.tex*, where XXX is the version number (e.g. 035 for version 0.35). The documentation (this document) is generated from the same source using LaTeX. To generate the various files, use our application *Dogwagger*, version 2.1 or greater.

In summary, our code requires the following software for *development*. (All that is needed to interact with the final product is a recent web browser with Javascript and cookies enabled).

1. Driving software for the digital calipers [STATE DETAILS];
2. SaferSleep [state version number] with server access (ODBC i.e. SQL-CLI connectivity);
3. Perl[24] version 5.6 (See notes above, we used ActivePerl version 5.6.1 binary build 635 2003-02-04).
4. AutoIt version 3.2.10.0 (© 1999-2007 Jonathan Bennett & AutoIt Team)
5. Javascript (ECMAscript) within a fairly recent Web browser (e.g. FireFox version 2.0 or greater, or Internet Explorer 5 or greater)

[24] At present, owing to the removal of the Tk toolkit from later versions of Perl, Dogwagger version 2.1, which requires Tk, will choke on higher versions of Perl.

6. A web-based version of mySQL, and a server running PHP version 5 or greater. (Optionally, an SQL-3 compliant database other than mySQL might be used to test the code).

7. LaTeX for generation of documentation. We used MikTex but any LaTeX2E-compliant engine will work.

8. My small *Dogwagger* program, version 2.1, to extract files from the .TEX source.[25]

Make: extract code from LaTeX file

Update this and manually run DogWagger if update LaTeX file name.

```
echo off
cls
rem makefile
perl dogwagger21.pl ez-037.tex
```

Creating the directory structure

Here's a little DOS batch file (*makedirectories.bat*) to create this structure, if not already made:

```
echo off
cls
md \ez
cd \ez
md ez-captur
md ez-captur\images
md ez-xlate
md ez-shw
md ez-shw\html
md ez-shw\html\css
md ez-shw\html\images
md ez-shw\php
md ez-shw\php\images
md ez-shw\php\js
md ez-shw\php\css
```

[25]Note that DogWagger is a Perl program written for Perl version 5.6. It uses the Perl/Tk interface. For mysterious reasons, Tk has been excised not only from more recent versions of ActivePerl (eg. 5.8 and 5.10), but even from 'current' version 5.6, even though previously available versions included Tk! This is frustrating and irritating. You will either need to run a more recent version of Perl (installing Tk, and, perhaps modifying the code slightly) or use an older version (Drop me a line)!

```
md ez-shw\sql
md ez-replay
md ez-replay\capture
md rawdata
md iar
md xml
md csv
md log
md dat
md images
md ez-info
```

A help file

Here's *help.bat*, that simply reminds the user of what we can do from the command line.

```
echo off
cls
echo ================================================================================
echo =                              HELP WITH EZ                                      =
echo ================================================================================
echo captur                                     capture from calipers, process
echo iar2xml                                    list is in \ez\ez-xlate\iarlist.txt
echo xml2csv test.xml                           xml to csv
echo csv2dat test.csv                           csv to IDAS format
echo replay                                     replay list via IDAS
echo --------------------------------------------------------------------------------
echo test-x2c                                   demo xml to csv
echo test-c2d                                   demo csv to dat
echo test-csv2dat                               demo captured csv to dat
echo test-captur                                demo raw csv processing
echo ================================================================================
```

11.1 Change Log

Changes for versions 0.35–0.36

1. Fixed Y-calibration in *ez-captur.pl*. Previously, because we cycled the times, we overwrote Y calibration values (!) Push values to a stack,[26] rather than writing to array. Fixes the problem with overwriting at 'same time'. ie. push YCALS to store each value, and for GetYCal, sum the values, dividing by their length.

2. Amendments to the capture of raw data:

   (a) Modified the AutoIt program so that it expects three sets of data, rather than one;

   (b) Start as usual with zero and cal, but at end of first series (systolic blood pressures) terminate before repeat zero and cal i.e. enter Excel.

   (c) AutoIt saves this series as eg X0001-sbp.csv;

   (d) Likewise for X0001-dbp.csv and X0001-hr.csv, but with the final series, terminate with a zero, cal, and x-measurement (time);

   (e) Then only do we submit the name and timestamp to Perl. What Perl does is tries to open X0001.csv. If this fails, then it looks for the components, concatenates them, and does its magic. Perl uses an exit code to report failure: bits 0,1 or 2 of the exit code are set, depending on which sequence failed (sbp, dbp or hr).

   (f) AutoIt reads the Perl exit code, and offers the user the opportunity to repeat the offending series, overwriting these.

   (g) Created an AutoIt interface with three buttons. Press a button to acquire each series, and a final one to validate the combination. If a component is invalid, this is renamed to (for example) X0001-dbp-timestamp.inv (for invalid). If the user presses a button where an unchecked component exists, he/she is asked whether they wish to overwrite, and if so, that component is invalidated.

3. Have the ability to bypass actual capture, 'faking' the Excel submission etc. (for debugging purposes)!

4. Version 0.36. Get IDAS database interrogation working.

[26] As initially conceived

5. We also need the ability to review the playlist, log played items, and remove them!
6. Use MWSnap rather than Paintshop (but see next version)
7. Introduced another flag in the .au3 interface: have we saved to playlist?
8. Also store the following: DURATION and ALPHA-INTERVAL (2.5, 5, or 10min)!
9. WRITE TO PLAYLIST.
10. See GetFileList etc.

11.1.1 Changes for version 0.37

1. Finalised replay through IDAS. Complex. Enabled invocation of replay in main GUI. Insert non-displayed entries to be read by IDAS every 30s, as DEMO.DAT always plays values every 30s, regardless of time!
2. Enter the *end time* as accurately as possible. Now we store the duration as a parameter, and pass it around, using the duration and start time to display SpO_2 requests etc. with actual times rather than offsets. We can also check the duration at various points. We can also use the duration to calculate alpha.
3. Our reference time interval is always 5 min. If just 10-minutely values are charted, then NAs must *always* be inserted in between, by the capturer!
4. Note that if an anaesthetic starts at a time not divisible by e.g. 5 min (or the relevant number of minutes corresponding to alpha) then we should start recording at the first integral time *before* the actual start time, and record the first datum as zero/zero (missing). Conversely, if the anaesthetic doesn't end on a mark (non-integral time with respect to alpha) then we must record the final value but *not* put in a missing value at the end In measuring the x-duration, we should measure from the point at which we 'start recording' (even if this is before the actual start of the anaesthetic — i.e. on an integral boundary) and end at the last mark, even if the anaesthetic ends several minutes after this. Our rationale is that our x-measurement is a calibration value, and if we measure between manually-constructed marks on the chart, this is less reliable that if we measure between pre-printed lines on the chart. In addition, if we either skip or include both the start and the end, there is too much variability (a nominal ± 9 minutes, or $\pm 2\alpha$) for us to check things accurately.

5. Based on the preceding points:
   (a) Acquire end time and duration within *ez-gui.au3*, check that these match, and pass duration as a parameter to *ez-captur.pl*.
   (b) In *ez-captur.pl*, accept duration as a parameter, and check this against the measured duration, failing if the two don't match. The only role of the time passed to this program is in calculation/validation of alpha!
   (c) Similarly pass duration to the keyboard capture routine *ez-keyboard.pl*.
6. Also added (a) Operation name and (b) Brief description of the patient.
7. Added ability to view CSV, XML, DAT files from GUI!
8. QUIT button added to entry screen for GUI;
9. Confirm OVERWRITE if file exists for raw data. Also back up prior file.
10. WEB/PHP CREATION [TO UPDATE]

12 Still to Do

1. Make GPL.htm page.
2. Still need to move PNG-related code to server, and upgrade it
3. Still must write menu for viewing all anaesthetists' input!
4. We need 'sensible' values for EtCO2 i.e. if in kPa, translate to mmHg. (760.000 mmHg = 101.325 kPa);
5. For first-timer (and subsequently) permit alteration of details...
6. Rejig HTML to permit capture of times, etc.
7. ... and restart using cookies.
8. Note that in the HTML we must hide whether images originate from -m or -s. (Randomly alter names to -a and -b)?
9. FIX UP demo/test batch files, with the new formats etc.
10. Perhaps later provide ability to view raw source csv files?

11. We should fix up minor warnings in *ez-captur.pl* related to data counts etc. (These have been temporarily suppressed). We might react more emphatically to errors in DURATION in this program.

12. Give some leeway (with a warning) if a value between alpha and MINVAL is read in directly (not simply an error; assume low reading, and only die if subsequent problems, but issue a warning).

13. With 5-min changes, we have a vast amount of redundant information in the PNG files. Might compare these. A simple strategy is to change nothing if they are the same (omitted file = leave the same!) and an advanced one might be to subtract the old from the new, and only store the new (changes) as an overlay (transparent background) but this latter change will require sophisticated programming!