Detection of DNA Polymorphisms in Thermal Gradients via a Scanning Laser Confocal System.

by

Matthew Ryan Graham
S.B. Physics
Massachusetts Institute of Technology, 1998

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN MECHANICAL ENGINEERING AT THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY JUNE 2000

© 2000 Massachusetts Institute of Technology. All rights reserved.

Signature of Author: ________________________

Department of Mechanical Engineering
May 5, 2000

Certified by: ________________________

Professor of Mechanical Engineering and Professor of BioEngineering
Thesis Supervisor

Accepted by: ________________________

Professor of Mechanical Engineering
Graduate Officer, Department of Mechanical Engineering
Detection of DNA Polymorphisms in Thermal Gradients via a Scanning Laser Confocal System.

by

Matthew Ryan Graham

Submitted to the Department of Mechanical Engineering on May 5, 2000 in Partial Fulfillment of the Requirements for the Degree of Master of Science in Mechanical Engineering

ABSTRACT

Discovery and detection of single nucleotide polymorphisms (SNPs) in DNA has become an important topic of research in the past few years. However, SNPs are very often detected by the labor intensive process of sequencing lengths of DNA. A scanning laser confocal capillary gel electrophoresis (CGE) system has been designed to provide a faster and more efficient method of detecting SNPs in short lengths of DNA.

The velocities of migrating DNA segments were measured over 100 mm as they moved through a specially designed CGE system. The velocity of these DNA segments was found to have an exponential relationship to the segment length. Also, a thermal gradient was then applied to the length of the capillary. As the DNA migrated, it was heated, and as the segments denatured, the DNA bands slowed. A "GC clamp" kept the two partially denatured strands from completely separating. When the thermal gradient was applied to the capillaries, the DNA segments demonstrated a velocity shift. However, the DNA did not show the expected sudden "break" in the velocity curve, but rather a more complex melting/velocity curve.

Thesis Supervisor: Ian W. Hunter
Title: Professor of Mechanical Engineering and Professor of BioEngineering
Acknowledgments:

Thank you to everyone in the Bioinstrumentation Laboratory and in Professor Lerman's Lab in the Biology Department. My time here would not have been nearly half as fun without you. Special thanks to Professor Ian Hunter for giving me the chance to learn so much in such an exciting environment. And most importantly, thank you to my parents for always supporting me no matter where I've chosen to go.
Table of Contents

I. Background.................................................................5
   A. DNA Structure.......................................................5
   B. Single Nucleotide Polymorphisms.........................5
II. Theory..............................................................................6
   A. Motion of DNA in capillary.....................................6
      1. Electrophoresis...................................................6
      2. Electroosmotic Flow..........................................7
   B. DNA and SNPs.......................................................9
   C. Confocal Optical Systems......................................10
   D. Temperature Gradient..........................................12
III. Experimental Design & Setup....................................14
   A. Optical Subsystem..............................................14
   B. Capillary Plate and Heaters.................................16
   C. Capillary Interface..............................................17
   D. Capillaries & Gel Chemistry.................................18
   E. Electrical Subsystem...........................................21
   F. Computer Automation and System Operation...........22
   G. System Mounting and Housing............................23
IV. Results............................................................................24
   A. Simple Run..........................................................24
   B. Simple Scan........................................................25
   C. Gradient Scan.....................................................26
V. Analysis............................................................................28
   A. DNA Velocities.....................................................28
   B. Denaturant............................................................29
   C. Curve Tracking and Identification.........................30
VI. Conclusions...............................................................31
VII. References...............................................................32
Appendix A: MATLAB® Code........................................34
Appendix B: Electrical Diagrams...................................36
Appendix C: Mechanical Drawings..................................38
Appendix D: Derivation of Electroosmotic Flow...............40
Appendix E: Visual Basic Code.......................................41
I. Background

A. DNA Structure

The structure of DNA was discovered in 1953 by James Watson and Francis Crick (Watson and Crick, 1953). DNA, which is an organic polymer that contains all of the genetic code for life, consists, structurally, of the backbone and bases. The backbone of DNA is further divided into alternating sugars and phosphate groups forming a linear strand. Each unit of DNA can be segmented into a single sugar and phosphate group along with a single base. These three components compose the structure known as a nucleotide. The nucleotide is the base unit of information in DNA. Nucleotides are differentiated by their bases, of which there are four types in DNA (there is a fifth one in RNA). These bases are Adenine (A), Cytosine (C), Guanine (G), and Thymine (T).

Each single strand of DNA has a direction sense and can be represented by a sequence of these four bases. Genomic DNA is typically found in double-stranded form (dsDNA), in which two strands arrange themselves in a complementary, anti-parallel sense. The bases are arranged in complementary base pairs (A with T, and C with G) forming a series of hydrogen bonds between the two strands. The A-T bond consists of two hydrogen bonds while the G-C is made up of three. These two strands usually form into a secondary structure known as the double-helix, a form that most people now recognize as pertaining to the workings inside their bodies.

B. Single Nucleotide Polymorphisms

Single nucleotide polymorphisms, or SNPs, are normal mutations in human DNA which occur at a single base pair location. To qualify as SNPs, the least common of these mutation combinations, or alleles, must occur at a rate of 1% or greater in the population. In any SNP location, it is possible to have four different allele combinations: A-T, T-A, G-C, and C-G. The direction sense of DNA makes the difference between the G-C and C-G alleles (as well as between the A-T and T-A alleles) very important, and they must be considered as separate cases. However, in human DNA, most SNPs occur with only two different alleles, and SNPs with three or four are very rare (Brookes, 1999).
Certain SNPs are known to predispose people to certain diseases. In these cases the relationship can be simple: persons with one SNP allele are more likely to have a certain genetic disease than are those with the other allele. However, researchers are only now beginning to investigate the vast relationships and interactions between multiple SNPs and disease. This effort will require large public libraries of SNPs and accompanying population information. Already, a few major efforts are under way to establish such public libraries (dbSNP, HGBASE).

Detecting SNPs in a fast and cost effective way is an important challenge to be met in this research. Currently, SNPs are detected by sequencing a small length DNA in which they might occur. However, sequencing a single sample is still a relatively time consuming method and can be complicated by mitigating factors depending upon the location of the SNP in the genome. Recent work (Lerman, 2000) has shown that measuring melting temperatures of DNA sequences with SNPs can be used to identify whether SNPs exist and even which allele is present. This technique presents a quicker alternative to sequencing and is the basis for the current project.

II. Theory

A. Motion of DNA in capillary.

1. Electrophoresis

Electrophoresis is a process similar to chromatography in which different species of molecules are separated by physical means. Most commonly electrophoresis is used to separate molecules by size in a fluid or gel by using inherent net charges. In capillary gel electrophoresis (CGE) the separation medium is a neutral polymer gel inside of a glass capillary tube. The ends of the capillary tube are suspended in a reservoir of buffer with high voltage electrodes, which are used to apply a high electric field across the length of the capillary.
As was said above, electrophoresis requires the molecules being separated to have a non-zero net charge. DNA is negatively charged and therefore will feel a force in the opposite direction of the electric field. This force on a charged particle is of the form:

\[ F_{\text{elec}} = ZeE, \]

(1)

where \( e \) is the fundamental charge, \( Z \) is the number of charges, and \( E \) is the electric field in potential per unit length. Viscous resistance from the polymer gel acts as a function of velocity, \( v \), and at steady state (constant velocity, \( v_{ss} \)), the force balance is,

\[ ZeE = bv_{ss}, \]

(2)

where \( b \) is the viscous drag (Camilleri, 1998). Thus the steady state velocity does not depend on the mass of the molecule, and if all of the molecules are assumed to have the same charge density (as in DNA), then the steady state velocities depend only upon the drag, which is a function of the size, or length, of the molecule.

Slab gels were the most common form of gel electrophoresis until a few years ago, but CGE has since become a more accurate and reliable method, especially when automation is desired. One of the most important benefits of CGE over slab gels is its ability to dissipate heat at very consistent rates across the capillary's whole length. Due to Joule heating, the temperature of the gels will increase, but the large surface area to volume ratio of capillary tubes prevents so-called "hot spots," or local heating, which introduce sometimes unmanageable nonlinearities into the system. Capillaries also use significantly less reagents, which make up the majority of the cost when doing large projects such as sequencing the human genome. CGE also provides a sealed system which is more amenable to automation.

2. Electroosmotic Flow

The dynamics of CGE are not as simple as a few charged macromolecules in an electric field. The buffer, DNA, and gel solutions all contain salt and other small charged molecules which make a comprehensive theoretical analysis of CGE systems difficult. One problem resulting from these extra ions is a phenomenon called "electroosmosis," which creates a flow in the capillary in the direction along the applied electric field,
opposite that of the DNA motion.

This electroosmotic flow is caused by the negatively charged inside surface of the silica capillary. The wall's silanol groups become negatively charged in an aqueous solution and draw positive ions from the solution. This positive charge sets up a double layer between the negatively charged wall and the neutral buffer solution. This double layer, or sheath, is positively charged, and when a potential is applied, the charged double layer moves from anode to cathode and carries the fluid in the capillary with it. Figure 1 shows the electric potential as a function of distance from the wall of the capillary. Here, \( \zeta \) is the potential at the shear surface, and \( K \) is the Debye-Hückel parameter and \( K^{-1} \) is the effective width of the double layer (Foret, et al., 1993). The mathematics of electroosmotic flow can be seen in Appendix D.

![Figure 1: Electroosmotic potential in a capillary (taken from Foret, et al., 1993).](image)

When separating DNA in a capillary, electroosmotic flow can slow and spread the individual DNA peaks. Thus, in most DNA analysis CGE systems, the surface of the capillary is coated to eliminate this flow. Some companies, such as Perkin-Elmer, have developed proprietary polymer gels which dynamically coat the capillary surface, eliminating the need for capillary coatings.
B. DNA and SNPs

The process of the two (hydrogen) bonded strands of dsDNA separating into two completely separate (unbonded) strands is termed denaturing or "melting." DNA melting is most often associated with an increase in temperature, though it can be effected by other system parameter changes, such as pH. The melting of a population of identical dsDNA has a statistical distribution but centers around one melting temperature for a short segment. This temperature primarily depends on the nucleotide makeup of a given segment. Generally, if the segment contains more G-C bonds it will melt at a higher temperature. This is because G-C bonds are triple hydrogen bonds and require more energy to break than the double hydrogen A-T bonds. The melting temperature also depends upon the buffer solution and whether any denaturants are present.

In the CGE system, the DNA is progressively denatured along the length of the capillary using a temperature gradient (See Section II D). As the DNA melts it presents a larger area to the gel, thus increasing its drag coefficient and slowing down. In a short region, the different alleles of a SNP will change the melting temperature of that region by up to half of a degree Celsius. Two populations of the same DNA sequence with different alleles of a SNP will melt at slightly different positions in a temperature gradient and will have different velocity curves which will be representative of the melting curves. Measuring these velocity curves was the first method of melting curve detection.

In longer sections of DNA, a single strand can have several melting regions. These subsegments each have different melting temperatures based on their sequences. Thus, if a region has a high concentration G-C base pairs, then it will melt at a much higher temperature than regions with a lesser G-C percentage. These regions are typically known as "GC clamps" (Gille, et al., 1998). By selecting a section of DNA with a SNP location and a GC clamp, the melting temperature of the SNP region can be completely observed since the GC clamp will hold the two strands together even after the rest of the segment has melted. GC clamps can also be added to a segment of DNA during the process of polymerase chain reaction (PCR™).
The DNA is also tagged with fluorescent molecules which only fluoresce when bonded to dsDNA. As the DNA melts in the gradient, the fluorescence will drop off proportionately to the amount of remaining dsDNA. This provides a second orthogonal measurement technique for determining the melting curve.

### C. Confocal Optical Systems

Confocal optical systems can be separated into two classes, typically called Type 1 and Type 2 (Wilson and Sheppard, 1984). A Type 1 confocal system uses a large area detector or source or both, while a Type 2 confocal system uses a point source and a point detector (or good approximations of these). By using a Type 2 system, very fine volumetric resolution (diffraction limited) can be achieved while eliminating noise from objects which are off-axis or out of the focal plane. However, total signal level is sacrificed when using a Type 2 system, and a large gain may be needed to detect the decreased signal.

Type 2 systems are good for performing three dimensional imaging. In the current system, though, a Type 2 confocal setup is not necessary. In fact, the spot size of the focal region should not be diffraction limited. Such a spot would be several hundred times smaller than the capillary diameter and would therefore miss much of the fluorescent signal from the DNA solution. By operating the system in the Type 1 confocal regime, the focal region is larger, and thus the detector is integrating signal over a larger volume.

To determine quantitatively in which regime a confocal system is, the full-width-half-max (FWHM) of the transverse response can be calculated as a function of the pinhole radius. According to Doukoglou (1995), for a confocal system, the normalized coordinate transverse to the optical axis can be expressed as,

$$
\nu \approx \frac{2\pi}{\lambda M} \frac{\sin(\alpha)}{r}, 
$$

where $\lambda$ is the wavelength of the light, $M$ is the magnification of the system, $\sin(\alpha)$ is the numerical aperture of the objective lens, and $r$ is the distance from the center of the optical axis. The transverse response from a point source in a reflection confocal system can then
be described as (Wilson and Sheppard, 1984),

\[ I(v) = h_L^2(v) \left( h_L^2(v) \otimes d_p(v) \right), \]

where \( \otimes \) denotes the convolution operation, \( h_L(v) \) is the point spread function of the objective lens on the image plane given by,

\[ h_L(v) = \left( \frac{2J_1(v)}{v} \right)^2; \]

and \( d_p(v) \), for a circular pinhole, is,

\[ d_p(v) = \text{circ}(\frac{v}{v_p}) = \begin{cases} 1 & \text{for } v \leq v_p \\ 0 & \text{elsewhere} \end{cases}. \]

By solving these equations numerically for different pinhole sizes, a plot can be constructed of the FWHM of the transverse response versus the pinhole size. An example of such a plot can be seen in Figure 2. The regime for Type 2 confocal systems is typically considered to be \( v_p \leq 0.5 \), and in the current setup the normalized pinhole radius is approximately 6.4, well into the Type 1 confocal regime.

![Figure 2: FWHM transverse response of a confocal system.](image-url)
D. Temperature Gradient

The capillary plate in the scanning CGE system can be modeled as a thin plate (or fin) with constant flux sources at either end and allowing for convective and radiative losses across the length of the plate. The dynamic model is complex, and instead of solving the physical model, an inverse mathematical model was created for the plate so that a temperature gradient could be chosen and the appropriate voltages for each heater could be set. The final model is 3rd-order in the two temperature variables and takes the following form,

\[ V(T_1, T_2) = a_0 + a_1 T_1 + a_2 T_2 + a_3 T_1^2 + a_4 T_1 T_2 + a_5 T_2^2 + a_6 T_1^3 \\
+ a_7 T_1 T_2 + a_8 T_1^2 T_2 + a_9 T_2^3, \]  

although a 2nd-order model could be used without much more error. Temperature data was taken from the capillary plate at varying temperatures with voltages ranging from 2-14 V on the cold side to 2-18 V on the hot side, with the cold side heater having a lesser or equal voltage than the hot side heater. The above model was then fitted to these data using a Marquardt minimization routine. The Visual Basic code for this routine is given in Appendix E.

Once the 3rd-order model was fit to the data, the model was evaluated at the temperatures of the original data points and the errors between the model and data were calculated. The statistics of the errors can be seen in the Table 1.

<table>
<thead>
<tr>
<th>Error Statistics</th>
<th>Hot Side Heater</th>
<th>Cold Side Heater</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Error [V]</td>
<td>$6.759 \times 10^{-14}$</td>
<td>$3.109 \times 10^{-13}$</td>
</tr>
<tr>
<td>St. Dev. [V]</td>
<td>$9.380 \times 10^{-2}$</td>
<td>0.2531</td>
</tr>
<tr>
<td>Min [V]</td>
<td>-0.2600</td>
<td>-0.3373</td>
</tr>
<tr>
<td>Max [V]</td>
<td>0.2046</td>
<td>0.5626</td>
</tr>
</tbody>
</table>

Table 1: Temperature gradient model error statistics.
Figure 3: Cold side heater voltage as a function of temperature gradient.

\[ V_{\text{cold}} = V(T_{\text{hot}}, T_{\text{cold}}) \]

Figure 4: Hot side heater voltage as a function of temperature gradient.
The model curves and actual data are plotted in Figures 3 and 4. The model should be approximately symmetric about the $T_{\text{HOT}} = T_{\text{COLD}}$ line but it is not. However, no data was taken to on the opposite side ($T_{\text{HOT}} < T_{\text{COLD}}$) to predict this part of the curve.

### III. Experimental Design & Setup

#### A. Optical Subsystem

A Type II confocal optical system was used to detect the fluorescence emissions from the tagged DNA in the capillary tube. The system consisted of an argon ion laser at 488 nm, a series of lenses and pinholes, a beamsplitter cube, two laser "notch" filters (Omega Optical, Brattleboro, VT), an objective and a small photomultiplier tube. A diagram of the optical subsystem can be seen in Figure 5.

![Optical subsystem schematic.](image)

The laser was a refurbished Uniphase air-cooled argon ion laser (488nm) with a maximum power output of 40 mW. All optics were mounted in an Spindler-Hoyer optical framework, which provided precision and ease of use. Two plano-convex lenses and a pinhole were used as a beam expander and as a spatial filter. The lenses had focal lengths of 15 mm and 60 mm respectively, providing a beam expansion of 4×, and the pinhole was 40 µm in diameter as was specified in Section IIC.

The expanded beam passed through a beamsplitter cube and into an 10× objective (Olympus, Melville, NY). The fluorescent light from the capillary tube then
passed back through the objective and was reflected by the beamsplitter cube into the
detection portion of the system. The light first passed through two laser notch filters to
block any reflected light at 488 nm (T~1×10⁻¹⁰). The light was then focused with a plano-
convex lens (f = 50 mm) through a second pinhole with a diameter of 100 μm and onto
the Hamamatsu photomultiplier (PMT).

![Image of PMT and laser filter.](image)

The nominal bandwidth of the Hamamatsu PMT was 20 kHz. The gain for
the PMT was controlled by an applied voltage ranging from 0-1 V, and was typically
operated with a gain of 0.75-0.95 V. The output of the PMT was passed through a series
of amplifiers and filters (See Section III E) to reduce noise and for interface with the PC
electronics.
B. Capillary Plate and Heaters

An aluminum plate was designed to hold the capillary and provided a steady and linear temperature gradient across the length of the capillary. Copper blocks at each end of the plate served as heat reservoirs. Each end of the plate was heated independently by thin Joule heating elements on a Kapton™ substrate (KHLV-202/10; Omega, Stamford, CT). A 375 μm wide groove was cut into the surface of the plate using electro-discharge machining (EDM) techniques. The groove was used to hold the capillary so that it maintained a more stable temperature profile. Small pieces of rubber were used at either end of the plate to secure the capillary, which was press-fit into the groove.

Adjustable plastic holders were attached at either end of the plate so that the buffer vials were securely fastened to the frame. The whole capillary holder plate was attached to a series of translational and rotational stages. The connection between the
first stage and the plate was made using acetyl standoffs and nylon screws to minimize heat loss into the stages and to isolate the plate electrically. The stages consisted of two orthogonally mounted translational stages (000-9141-01; Parker, Irwin, PA), and two orthogonally mounted rotational stages (124-0055; OptoSigma, Santa Ana, CA). The rotational stages were used to account for any misalignment in the rest of the mounting so that the capillary could be made perpendicular to the optical axis and parallel to the ground plane of the scanning stage. One translational stage, which was parallel to the optical axis, was used to bring the system into focus, while the other stage was used for transverse motion during scanning operations. Each translational stage was driven by a microstepping motor (ZETA57-83-MO; Compumotor, Rohnert Park, CA) and controller (ZETA6104; Compumotor) which was connected to the PC by a serial cable.

A series of twenty thermocouples (5TC-TT-E-36-36; Omega, Stamford, CT) were distributed along the length of the plate to measure the temperature gradient. The thermocouples were measured using an HP34970 data acquisition system and an HP34901 acquisition card set up to read thermocouples in degrees Celsius.

C. Capillary Interface

![Figure 8: Capillary high pressure vial interface.](image)
High pressure injections were required to fill the capillaries with gel. The injections were done with two different capillary interfaces. The first interface is pictured below and uses a high pressure gas canister to push the gel into the capillary. The gas used for the gel injections was typically argon or helium. The gel was injected at a pressure of $1.379 \times 10^5$ Pa (20 psi). The valves controlling the gas for the gel injection were computer actuated to lessen introduction of contaminants into the system. Septa on the sealed vials were pierced by disposable syringe tips through which the capillary was fed to avoid coring. This method, however, did allow a significant amount of dirt to enter the system, and could not provide enough pressure to pump the more viscous gels into the capillary.

The second means of pumping gels into the capillary was by syringe injections. A piece of Teflon™ tubing was secured over each end of the capillary tube. The tip of a 100 µL syringe filled with gel was then inserted into the tubing at one end. The Teflon™/steel interface provided a suitable seal for the injections. In this way, the polymer gel was manually injected into the capillary tube.

![Figure 9: Capillary high pressure syringe interface.](image)

**D. Capillaries & Gel Chemistry**

The capillary tubes were purchased in spools from PolyMicro Technologies (Phoenix, AZ). Two sizes of capillaries were used, one with an inner diameter (ID) of 75 µm and one with a 100 µm ID. The capillaries were ordered with a polyimide coating to
provide strength and to prevent breakage. The outer diameter (OD) of the coated capillaries was approximately 375 μm. Capillaries used in the CGE experiments were typically 550 mm long, and the polyimide coating of the middle 100 mm was burned off with a butane torch so that the fluid inside could be optically interrogated. The capillaries were then filled with a viscous polymer gel which was used as the separation matrix for the DNA.

Two separate series of chemistries were used to develop this setup. The first was a proprietary gel mixture purchased from Beckman-Coulter (477628; Beckman-Coulter, Fullerton, CA). The second polymer gel chemistry was prepared in-house roughly following the procedure described by Khrapko, et al. (1996).

The first gel was adequate for separating a standard DNA "ladder" but problems arose when attempting to inject other DNA. It was decided that a gel produced in house would provide more control over the gel characteristics as well as being cheaper to produce. First the capillaries were washed with 1 M NaOH for 2 hours. Then they were washed sequentially with 1 M HCl and methanol. Afterwards, the capillaries were filled with γ-methacryloxypropyltrimethoxysilane and left overnight. Then the capillaries were washed again with methanol, and a 6% acrylamide gel solution in 1× TBE, 0.1% TEMED, and 0.025% ammonium persulfate, was injected into the capillary and left for 30 minutes to polymerize. This gel was then pumped out to prevent clogging the capillary and a 5% acrylamide gel solution in 1× TBE, 0.03% TEMED, and 0.003% ammonium sulfate was injected into the capillary for separating the DNA. This gel was replaced with new gel after every run. During the runs the capillary ends were submerged with the high voltage electrodes in a vial of the running gel.
The DNA was tagged with a fluorescent intercalator, YO-PRO-1 (or oxysol yellow) purchased from Molecular Probes (Y-3603; Molecular Probes, Eugene, OR). This molecule, as was mentioned above, fluoresced proportional to the amount of dsDNA to which it was bound. Ethidium bromide (E-1305; Molecular Probes), another fluorescent intercalator, was used in some early runs, but YO-PRO-1 was known to have a better response as the dsDNA melted (Hogan, 2000). The absorbance peak of YO-PRO-1 is at 491 nm and the emission peak is at 509 nm. The absorbance and emission spectra of YO-PRO-1 when bound to dsDNA can be seen in Figure 10.

For some runs of DNA in temperature gradients, a denaturant, nicotinamide, was added to the gel buffer before polymerization. The denaturant was used to lower the melting temperature of the DNA segment to a range more amenable to good signal-to-noise ratio in the CGE scanning system.
E. Electrical Subsystem

The electrical subsystem of the CGE system is shown in Figure 11. The electrical subsystem consists of two programmable triple output power supplies (HP3631A; Hewlett Packard, Palo Alto, CA), three programmable single output power supplies (HP3632A; Hewlett Packard), one high voltage power supply, (CZE1000R; Spellman, Hauppauge, New York), a series of intermediate electronics and a data acquisition board (Allios board, MIT Bioinstrumentation Lab) that was produced in-house. The HP power sources provided all power for intermediate electronics, for the PMT, and for heating the capillary plate. The HP sources were controlled by a PC through a IEEE 488 connection using a National Instruments IEEE 488 interface board.

The CZE1000R had a maximum output voltage of 30 kV and a maximum output current of 300 μA. Polarity could be controlled on the front panel, but this option was modified so that the polarity could be controlled by an applied voltage to the terminal block on the back of the device. High voltage and the current limit on the device were also set by applying voltages to the terminal block. Actual current and high voltage values were monitored off the terminal block. All voltage inputs and outputs to the terminal block ranged from 0 V to 10 V. To interface these inputs and outputs to the PC's data acquisition board the voltages were scaled to the proper voltage ranges. Outputs from the CZE1000R were scaled down by a factor of 2.2 using a simple voltage divider, and the outputs from the Allios board were amplified by a factor of 3.3 using a simple non-inverting amplifier (see Appendix B).

The gain of the Hamamatsu PMT was changed by a 0-1 V control voltage which was applied by one of the HP3631As under computer control. The PMT outputted a current which was first changed into a voltage by a simple current amplifier circuit. The current amplifier converted from 0-100 μA to 0-4 V with a bandwidth of 20 kHz. The voltage signal was then passed through a 2nd-order Butterworth filter to reduce high frequency electro-magnetic noise, including electrical line noise. The cutoff of the Butterworth filter was 43 Hz with a DC gain of 1.597. These values were measured with a Hewlett-Packard Dynamic Signal Analyzer (HP3562A, Hewlett-Packard). The phase and gain frequency response plots of these two components can be seen in Appendix B.
F. Computer Automation and System Operation

Operation of the setup was automated through the use of a personal computer (PC) and Visual Basic (VB) programming. The PC had a 450 MHz Pentium II processor, 384 MB of RAM and ran the Microsoft Windows NT 4.0 operating system. All Visual Basic programming was done with Microsoft Visual Basic 6.0 and compiled for increased speed. All HP devices were controlled through their IEEE 488 ports using a National Instruments IEEE 488 interface card in a PCI slot. The National Instruments board was accessed by Visual Basic through Visual Basic Class Modules which were built on top of the basic National Instruments VB interface. The Allios data acquisition board, which resided in one of the PC's PCI slots, was accessed through a Visual Basic Class Module which was built on top of a specially written WinRT™ dynamic linked library (DLL).
The CGE system was programmed to operate in two different modes, Simple Run and Simple Scan. In Simple Run mode, the plate remained stationary while the CGE run was performed and temperature was not recorded. The user was able to set the sample rate, the total number of samples, the applied high voltage, the current limit, and the PMT gain voltage. In Simple Scan mode, the plate was scanned back and forth and the user was given the option of recording the temperature of the plate at the end of each scan. The user was also able to set the following parameters (with typical running values in parentheses): length of scan (100 mm), time for scan (10 s), sample rate (10 Hz), number of scans (1000 or 1800), high voltage (5.5 kV), current limit (100 µA), and PMT gain voltage (0.75-0.9 V). At the end of each scan the plate motion paused to allow for temperature readings from the twenty thermocouples and for resynchronization with the program.

After each data collection, the data was saved in text files on the hard drive which could be imported into MATLAB or some other data analysis program. A typical run consisted of electrokinetically injecting fluorescently tagged DNA into the capillary and then switching the cathode solution back to the buffer gel and running for several hours. The DNA was injected at a field strength of 20 V/mm for 90 s while recording data at 100 Hz in the Simple Run mode with the PMT gain voltage set to 0 V. The plate was then aligned perpendicularly to the optical axis and run in Simple Scan mode.

**G. System Mounting and Housing**

The entire CGE system was mounted onto an optical bench with built-in vibration isolation (2\textsuperscript{nd}-order lowpass with a cutoff of ~1 Hz) which was then floated on air bearings (TMC, Peabody, MA). To further minimize vibration, the cooling fan from the laser was removed from the laser head. Ductwork was then installed to provide proper cooling while transmitting significantly less vibration.

Due to the sensitivity of the PMT detector, experiments were performed in darkness. An outer housing for the setup was constructed from 45 mm (1¾") square steel beams (Unistrut, Woburn, MA). This framework was covered with corrugated black
plastic (AIN Plastics, Norwood, MA) and sealed with black vinyl tape to provide a light-tight box.

IV. Results

A. Simple Run

![Graph showing fluorescent signal over time](attachment:figure12.png)

Figure 12: Simple run data.

As was mentioned above, during the Simple Run mode of operation of the experimental setup, the stage was kept stationary and temperature was not recorded. The Simple Run mode typically was used to load the capillary electrokinetically before a Simple Scan experiment. When loading the capillary, the PMT gain was set to 0 V, or no gain. The Simple Run mode was also used initially to test new mixtures of DNA and fluorescent molecules because the Simple Run mode was more easily aligned than the Simple Scan mode. Data from a Simple Run can be seen in Figure 12. The DNA in this run consists of a standard "ladder" tagged with ethidium bromide.
The peaks seen are the first seven bands of the DNA ladder. The sample in this run was electrokinetically injected for 90 seconds at 10.6 kV for a field strength of 20 V/mm. The PMT signal was sampled at 2 Hz for 3 hours and DNA was run also run at a potential of 10.6 kV. The PMT gain was set at 0.9 V. After the run, a 3-point median filter was applied to the data to eliminate single point noise spikes. The peaks in this plot correspond to the following DNA segment lengths (all lengths give in basepairs or bp): 72, 118, 194, 234, 271, 281, and 301.

B. Simple Scan

Using the Simple Scan mode, data could be taken across the length of the capillary rather than at just one point. Due to the dynamic nature of this mode, the alignment procedure was much more important. Small misalignments often resulted in wide fluctuations of the background noise and decreases in the S/N ratio. Said misalignments most often were the result of the scanning axis not being completely parallel to the capillary. Once the proper alignments were completed, though, the signal and background were quite consistent across the whole scanning length.

Figure 13 shows a Simple Scan of both standard ladder DNA and the SNP DNA sample would possibly have a SNP. The sample DNA and the DNA standard were mixed into a solution at approximately equal concentrations, but because the DNA standard had 11 bands, the amount in each of its band (and therefore the brightness) is significantly less than the SNP sample band, which is obviously brighter than the others.

In the figure the scans are taken along the vertical axis and time along the horizontal axis. Due to a constant delay caused by the finite acceleration of the linear stages, the anti-parallel scans had to be slightly shifted to align the picture. In the Simple Scan shown in Figure 13, the even and odd scans were shifted by a total of 6 data points. This value was determined by trial and error and was perfected by aligning the constant features in the image. A 3-by-3 median filter was applied to the raw data to eliminate single points of noise so that a proper range expansion could be done. This run was done without turning on the heaters and served as a temperature "standard" for later runs with a
temperature gradient.

As could be seen in the Simple Run shown above, the 5th and 6th bands are quite close together, but could still be discerned with a stationary detector. In the Simple Scan plot the 5th band is approximately twice the brightness of the other ladder bands. This suggests that the real 5th and 6th bands were unresolveable in this run.

C. Gradient Scan

After the system was properly calibrated and aligned by doing Simple Scans, the heaters at either end were powered to apply a specified temperature gradient across the length of the plate. The values for either end of the temperature gradient were chosen first and then the formula described in Section III D was applied to find the approximate voltages to be applied to the hot and cold side heaters.

The heaters were allowed to heat to steady state for approximately one hour.
Then the plate was aligned as described above and DNA was injected. Gradient Scans were run with similar parameters as Simple Scans except that the number of scans was increased (up to 2300) to account for the DNA slowing once it was melted.

![Gradient Scan](image)

**Figure 14: Gradient scan data.**

Figure 14 shows a Gradient Scan with both the DNA ladder and the DNA SNP sample. Multiple bands can be seen to change velocity during the course of the run. Also, the peaks can be seen to broaden as they slow, possibly a result of the higher temperatures contributing to faster diffusion. Because there are so many bands from the ladder and because no single band stands out as brighter than the rest, identification of the SNP DNA is difficult. In this particular run the gel buffer did not contain nicotinamide, and in general, the runs with nicotinamide (at 1.5 M) were very poor and did not show any DNA separation.
V. Analysis

A. DNA Velocities

The fact that larger DNA fragments should move through gels more slowly than small DNA fragments is the basis for all of CGE work. This can be confirmed through more in-depth analysis on some of the resultant data. The bands in Figure 13 provide a clear data set with which to work, although all of the successful Simple Scan data sets would provide similar information.

To measure the velocities of each peak, the data set was broken into regions in which there was only a single band. Then, the scan during which each position along the capillary had its maximum was calculated. This was found to be a good measure of the position of the center of each peak. A straight line was then fit to each data set with the slopes of the lines equal to the respective velocities of the peaks. It was necessary to reduce the amount of data available to the fitting routine for some of the DNA segments due to close bands corrupting the peak finding calculations. With more sophisticated peak finding algorithms, this manual step could be eliminated and the velocity estimates improved.

The velocities of DNA bands can be seen plotted in Figure 14 along with the fitted model. The model is a simple exponential with the form,

\[ v(l) = e^{a_1 l + a_2} + a_3, \]

and was fitted to the data using the Marquardt minimization routine listed in Appendix E. For this data set, the fitted parameters are listed in Table 2, and the percent-variance-accounted-for (%VAF) of the model was calculated to be 99.988%. Then, according to Equation 2, for a constant voltage the velocity is inversely proportional to the viscous drag coefficient. Therefore, the drag coefficient must have an exponential relationship to DNA segment length.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1)</td>
<td>(a_2)</td>
<td>(a_3)</td>
</tr>
<tr>
<td>-0.003437</td>
<td>-1.827866</td>
<td>0.094323</td>
</tr>
</tbody>
</table>

Table 2: Velocity parameter fit.
Figure 14: Velocity vs. DNA Length.

**B. Denaturant**

The denaturant, nicotinamide, was added to the acrylamide gel before polymerization to lower the melting temperature of the DNA segments. However, the denaturant seemed to interrupt the polymerization processes, and the resulting buffer was not suitable for electrophoresis.

A denaturant is desirable in the current setup for several reasons. First, as the aluminum capillary holder plate heats, it will expand and cause the plate to move out of alignment, and getting consistent data becomes much more difficult. Also, as the fluid inside the capillary heats, the viscosity and thus the efficacy of the polyacrylamide gel changes. This causes nonlinearities in the separation and makes velocity changes of the DNA bands more difficult to identify and quantify. By lowering the melting temperature of the DNA strands, the system can be moved away from the regime where these
nonlinear effects become significant. Nicotinamide, as yet, does not seem suitable for use in gels, and other denaturant choices should be pursued.

C. Curve Tracking and Identification

As can be seen in Figure 14, analyzing many DNA peaks in one gradient scan is complicated. The curves for each ladder peak confound the identification of the SNP DNA peak, and ideally the system would be run with the DNA ladder. However, in experiments where only the SNP DNA was run in the CGE system (Figure 15), the migrating band did not experience the expected velocity shift. The slight shift which did occur also came at a higher than expected temperature. This suggests that either the temperature sensors do not reflect the actual temperature within the capillary or that the high temperatures are affecting the interaction between the DNA and the polymer gel.

Figure 15: Gradient scan with single DNA band.
VI. Conclusions

The scanning confocal CGE system provides a new way to obtain information about CGE systems. This technology also allows CGE systems to be run in new ways that could dramatically affect the discovery, analysis and screening of SNPs. Future configurations could also take greater advantage of the CGE technique's ease of automation to produce a system for mass scale SNP screening.

Current data from the CGE system does not show the expected DNA velocity break in the thermal gradient, however the results are promising. Lowering the overall required temperature through the use of denaturants is one aspect of the project that should continue to be pursued. Not only will the gel be more predictable at lower temperatures, but as much of the data has shown, the signal-to-noise ratio seems to be better at lower temperatures.

Efforts are currently being made to use feedback control for tracking individual peaks as they travel through the capillary rather than simply raster scan across the capillary's length. This technique, once perfected, should provide additional increases in signal to noise ration by sampling more often around the peaks rather than in areas of no interest. Tracking peaks should also allow the linear stages to move more slowly and thus more smoothly, eliminating some of the noise from mechanical vibrations.

In the next iteration, much consideration should be given to the capillary plate and interface as well as the entire temperature measurement system. Capillary alignment was a critical and time consuming step which had to be performed before every experiment. A better mounting system would immediately speed setups times and improve data quality. The decision to use thermocouples for measuring the temperature gradient in the plate should also be revisited. Mounting the thermocouples so that they produced accurate and repeatable data was not an insignificant task. With a better theoretical model for the temperature gradient in the plate, fewer and more accurate temperature probes might be used to measure the temperature data.
VII. References


Hogan, C. Personal communication. Department of Biology, MIT: 2000.


Lerman, L. S. Personal communication. Department of Biology, MIT: 2000.


National Center for Biotechnology Information, dbSNP. Bethesda, MD.
Ruiz-Martinez, M. C., J. Berka, A. Belenkii, F. Foret, A. W. Miller, and B. L. Karger.
Appendix A: MATLAB® Code

function a = segment(data,scans, lag)
% segment.m
% Written by Matthew R. Graham
% Last modified on 18 April 2000.
% 2000 Copyright Matthew R. Graham and Bioinstrumentation Lab.
% Converts single column of interlaced data to 2D array of data which
% can be have every other row shifted to compensate for a constant lag
% in a standard raster scan. Inserts mean of data into empty entries
% where the lag shift takes place.

s = size(data);
scanlength = s(l) / scans;
mn = mean(data);

for m = 1 : scans
    if (lag > 0)
        if (mod(m,2) == 1)
            a(1:scanlength,m) = data((m-1)*scanlength + 1:m*scanlength);
            a(1:scanlength+1:scanlength+lag,m) = ones(lag,1)*mn;
        else
            a(1:lag,m) = ones(lag,1)*mn;
            a(lag+1:scanlength+lag,m) = data(m*scanlength:-1:(m-1)*scanlength + 1);
        end
    else
        if (mod(m,2) == 1)
            a(:,m) = data((m-1)*scanlength + 1:m*scanlength);
        else
            a(:,m) = data(m*scanlength:-1:(m-1)*scanlength + 1);
        end
    end
end

function f = platemodel(ht,ct,p)
% platemodel.m
% Written by Matthew R. Graham
% Last modified on 18 April 2000.
% 2000 Copyright Matthew R. Graham and Bioinstrumentation Lab.
% Applies 3rd order model for heating of capillary plate.
% Inputs: Hot Side Temperature, Cold Side Temperature,
% and (Hot or Cold) Model Parameters.
% Outputs: (Hot or Cold Side Heater) Voltage necessary.

f = p(1) + p(2)*ht + p(3)*ht^2 + p(4)*ct + p(5)*ct^2 + p(6)*ht*ct + p(7)*ht^3 + p(8)*ct*ht^2 + p(9)*ht*ct^2 + p(10)*ct^3;
function f = trackmax(data)
% trackmax.m
% Written by Matthew R. Graham
% Last Modified on 25 April 2000.
% 2000 Copyright Matthew R. Graham and Bioinstrumentation Lab.
% Returns position of maximum value across length of array.

s = size(data);
l = s(1);

for i = 1:l
    [x,f(i)] = max(data(i,:));
end

function f = unwind(p)
% unwind.m
% Written by Matthew R. Graham
% Last Modified on 29 April 2000.
% 2000 Copyright Matthew R. Graham and Bioinstrumentation Lab.
% Unwraps phase data from HP3562A taken through HPVEE programming.

p1 = flipud(p);
p2 = p1.*pi./90;
p3 = unwrap(p2);
p4 = p3.*90./pi;
f = flipud(p4);
Appendix B: Electrical Diagrams

Figure B1: Voltage divider.

Figure B2: 2nd order Butterworth filter.

Figure B3: Voltage amplifier.

Figure B4: Current-to-voltage amplifier.
Figure B5: Phase and magnitude plots for filter and amplifier.
Figure C1: Capillary plate mechanical schematic (taken from Angel, 2000).
Figure C2: Capillary plate and vial holders mechanical drawing (taken from Angel, 2000).
Appendix D: Derivation of Electroosmotic Flow
(taken from Foret, et al., 1993)

The charge/potential relationship for a double layer can be described by the equation,

\[ \zeta = \frac{\sigma_0 K^{-1}}{\varepsilon}, \]  

(D1)

where \( \sigma_0 \) is the charge density on the surface of the silica capillary wall, \( \varepsilon \) is the permittivity of the buffer solution, and the zeta potential, \( \zeta \), is the potential at the shear surface. \( K^{-1} \), the inverse of the Debye-Hückel parameter, is the width of the double layer and the charge necessary to balance the surface charge lies within this distance from the wall. \( K \) can be expressed as,

\[ K = \sqrt{\frac{2000 \cdot F^2}{\varepsilon_0 \kappa RT}}, \]  

(D2)

where \( \varepsilon_0 \) is the permittivity of free space, \( \kappa \) is the dielectric constant of the solution, \( R \) is the ideal gas constant, \( T \) is the temperature, and \( F \) is the Faraday constant. \( \mu \) is the ionic strength which can be calculated by,

\[ \mu = \frac{1}{2} \sum c_i z_i^2, \]  

(D3)

where \( z_i \) is the individual charge of each ion population at concentration, \( c_i \).

The applied electric field, \( E \), will exert a force on the layer of positive charge and will create a velocity profile across the capillary diameter, with the velocity at the shear plane equal to zero and then rising to a constant velocity, \( v_{eo} \), at the center of the capillary. Smoluchowski's formula gives the expression for this electroosmotic velocity which, for a large capillary, is constant across most of the diameter,

\[ v_{eo} = -\frac{\varepsilon \zeta}{\eta} E, \]  

(D4)

where \( \eta \) is the viscosity of the gel buffer.
Appendix E: Visual Basic Code

Option Explicit

'frmAllios.frm
'Visual Basic form for direct control of the Allios component.
'By Matthew R. Graham
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.
'To do:
'Add digital access.

Dim a As Allios
Dim DAVals(0 To 15) As Double
Dim ADChan As Long
Dim DAChan As Long

Private Sub cmbAD_Click()
    'Choose A/D output channel.
    ADChan = CLng(cmbAD.Text)
    txtADVolt.Text = CStr(a.VoltsIn(ADChan))
End Sub

Private Sub cmbDA_Click()
    'Choose D/A channel.
    DAChan = CLng(cmbDA.Text)
    txtDAVolt.Text = CStr(DAVals(DAChan))
End Sub

Private Sub cmdGet_Click()
    'Read A/D value from specified channel.
    txtADVolt.Text = a.VoltsIn(ADChan)
End Sub

Private Sub cmdSet_Click()
    'Set D/A value to specified channel.
    DAVals(DAChan) = CDbl(txtDAVolt.Text)
    Call a.VoltsOut(DAChan, DAVals(DAChan))
End Sub

Private Sub Form_Load()
    'Get access to Allios object from main form.
    Set a = frmMain.getComponent("Allios")
    'Sets all D/A channels to 0.
    For Counter = 0 To 15
        DAVals(Counter) = 0#
        Call a.VoltsOut(Counter, DAVals(Counter))
    Next Counter
End Sub

Private Sub Form_Unload(Cancel As Integer)
    frmMain.CurrentWindow.SetFocus
End Sub

Option Explicit

'frmCZE1000.frm
'Visual Basic form for directly controlling the CZE1000 component.
'Written by Matthew R. Graham
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.
'To do:
'Fix goofiness in enabling buttons.

Dim C As CZE1000
Dim MeasVolt As Double
Dim MeasCurr As Double
Dim CmdVolt As Double
Dim CmdCurr As Double
Dim Pol As Polarity

Private Sub cmbPolarity_Click()
    'Choose high voltage polarity.
    If (cmbPolarity.Text = "Positive") Then
        Pol = Positive
    Else
        Pol = Negative
    End If
    cmdPolarity.Enabled = True
End Sub

Private Sub cmdCmdVoltCurr_Click()
    'Set high voltage and current limit.
    CmdVolt = CDbl(txtCmdVolt.Text)
    CmdCurr = CDbl(txtCmdCurr.Text)
    C.nextVal CmdVolt, CmdCurr, MeasVolt, MeasCurr
    cmdCmdVoltCurr.Enabled = False
End Sub

Private Sub cmdMeasCurr_Click()
    'Read current value in microamperes.
    txtMeasCurr.Text = CStr(C.GetCurrent)
End Sub

Private Sub cmdCmdVoltCurr_Click()
    'Set high voltage and current limit.
    CmdVolt = CDbl(txtCmdVolt.Text)
    CmdCurr = CDbl(txtCmdCurr.Text)
    C.nextVal CmdVolt, CmdCurr, MeasVolt, MeasCurr
    cmdCmdVoltCurr.Enabled = False
End Sub

Private Sub cmdMeasCurr_Click()
    'Read current value in microamperes.
    txtMeasCurr.Text = CStr(C.GetCurrent)
End Sub
Private Sub cmdMeasVolt_Click()
    'Read voltage in kilovolts.
    txtMeasVolt.Text = CStr(C.GetVoltage)
End Sub

Private Sub cmdPolarity_Click()
    'Set high voltage polarity.
    C.SetPolarity Pol
cmdPolarity.Enabled = False
End Sub

Private Sub Form_Load()
    'Get access to CZE1000 object from main form.
    Set C = frmMain.getComponent("CZE1000")
    CmdVolt = 0
    CmdCurr = 100
    C.nextVal CmdVolt, CmdCurr, MeasVolt, MeasCurr
    txtCmdVolt.Text = CStr(CmdVolt)
    txtCmdCurr.Text = CStr(CmdCurr)
    txtMeasVolt.Text = CStr(MeasVolt)
    txtMeasCurr.Text = CStr(MeasCurr)
    Pol = frmMain.Pol
    If (Pol = Positive) Then
cmbPolarity.ListIndex = 0
    Else
        cmbPolarity.ListIndex = 1
    End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
    'Set everything back to zero.
    C.zero
    frmMain.CurrentWindow.SetFocus
End Sub

Private Sub txtCmdCurr_Validate(Cancel As Boolean)
    'Set everything back to zero.
    C.zero
    frmMain.CurrentWindow.SetFocus
End Sub

Private Sub txtCmdVolt_Validate(Cancel As Boolean)
    'Set everything back to zero.
    C.zero
    frmMain.CurrentWindow.SetFocus
End Sub

Option Explicit
'frmExp.frm
'Visual Basic form for running CGE experiments.
'Written by Matthew R. Graham
'Last Modified on 15 April 1999.
Private Sub Form Load()
'Initializes form.
    ExpStop 'Sets status to STOPPED.
    If frmMain.Exp = SimpleRun Then
        SetSimpleRun 'Sets mode to SimpleRun.
    Else
        SetSimpleScan 'Sets mode to SimpleScan.
    End If
    LoadParams 'Load params from main form.
    UpdateParams 'Make sure parameter math is okay & set
           params on form.
    'Other inits.
    Set Watcher = New DataWatcher
    Watcher.Init
    LagTime = 2.5
    StepConvert = 500
    VelConvert = 5
    'Repackage this alarm stuff:
    chkAlarm.value = 0
    AlarmGain = 4
    txtAlarmGain = CStr(AlarmGain)
    Alarm = False
End Sub

Private Sub Form_Unload(Cancel As Integer)
'Prepare to go back to main form.
    M1.FlushInput
    Cl.zero
    Pow3.SetP6V 0#
    SaveParams
    frmMain.Enabled = True
    Set frmMain.CurrentWindow = frmMain
    frmMain.SetFocus
End Sub

Private Sub tabExp_Click(PreviousTab As Integer)
'Change between experiment modes.
    Select Case tabExp.Tab
    Case ExpType.SimpleRun
        SetSimpleRun UpdateParams Exit Sub
    Case ExpType.SimpleScan
        SetSimpleScan UpdateParams Exit Sub
    Case Else
        MsgBox "Tab number not implemented!", , "Bad tab."
        Exit Sub
    End Select
End Sub

Private Sub cmdPause_Click()
'Pauses experiment.
    ExpPause
End Sub

Private Sub cmdStart_Click()
'Starts experiment.
    ExpStart
End Sub

Private Sub cmdStop_Click()
'Stops experiment. -
    ExpStop
End Sub

Private Sub ExpStop()
'Stops experiment. Successive PLAYs will delete current data
    Arrays.
    cmdStop.Enabled = False
    cmdPause.Enabled = False
    cmdStart.Enabled = True
    Status = Stopped
End Sub

Private Sub ExpStart()
'Starts experiment.
    cmdStart.Enabled = False
    cmdStop.Enabled = True
    cmdPause.Enabled = True
    Status = Running
    RunExperiment
End Sub

Private Sub ExpPause()
'Pauses experiment.
    If Status = Running Then
        cmdPause.Enabled = False
        Status = Paused
    Else
        cmdPause.Enabled = True
        Status = Running
    End If
End Sub

'Reides alarm flag.
    Alarm = IIf(chkAlarm.value = 1, True, False)
txtAlarmGain.Enabled = Alarm
End Sub

Private Sub chkRecordTemp_Click()
'Changes record temperature flag to correct state.
    RecordTemp = IIf(chkRecordTemp.value = 1, True, False)
End Sub
Private Sub DesignSignals()
'Initialize data tensors.
Select Case CurrentExp
Case ExpType.SimpleScan
ReDim thisRun(O To 7) 'Data, Time, CmdCurrent, MeasCurrent, CmdVoltage, MeasVoltage, Motion, Temperature
'Data tensor
thisRun(0) = GenConstTensor(0, 0, RecordLength - 1, 1, "Fluorescent Signal", "V", "", "")
With thisRun(0)
.Class = "CGE Run"
.Type = "Data"
End With
'Time
thisRun(1) = GenRamp(0, CDbL(RecordLength - 1), 1, 0, (RecordLength - 1) / SampleRate)
With thisRun(1)
.Class = "CGE Run"
.Type = "Time"
End With
'CmdVoltage
thisRun(2) = GenConstTensor(voltage, 0, RecordLength - 1, 1, "Commanded Voltage", "kV", "", "")
With thisRun(2)
.Class = "CGE Run"
.Type = "CmdVoltage"
End With
'MeasVoltage
thisRun(3) = GenConstTensor(0, 0, RecordLength - 1, 1, "Measured Voltage", "kV", "", "")
With thisRun(3)
.Class = "CGE Run"
.Type = "MeasVoltage"
End With
'CmdCurrent
thisRun(4) = GenConstTensor(current, 0, RecordLength - 1, 1, "Commanded Current", "mA", "", "")
With thisRun(4)
.Class = "CGE Run"
.Type = "CmdCurrent"
End With
'MeasCurrent
thisRun(5) = GenConstTensor(0, 0, RecordLength - 1, 1, "Measured Current", "mA", "", "")
With thisRun(5)
.Class = "CGE Run"
.Type = "MeasCurrent"
End With
'Motion
thisRun(6) = GenConstTensor(0, 0, NumScans - 1, 1, "Commanded Motion", "steps", "", "")
With thisRun(6)
.Class = "CGE Run"
.Type = "Motion"
End With
'Temperature
thisRun(7) = GenConstTensor(0, 0, NumScans + 20 - 1, 1, "Temperature", "°C", "", "")
With thisRun(7)
.Class = "CGE Run"
.Type = "Temperature"
End With
Case ExpType.SimpleRun
ReDim thisRun(0 To 5) 'Data, Time, CmdCurrent, MeasCurrent, CmdVoltage, MeasVoltage, Temperature
'Data tensor
thisRun(0) = GenConstTensor(0, 0, RecordLength - 1, 1, "Fluorescent Signal", "V", "", "")
With thisRun(0)
.Class = "CGE Run"
.Type = "Data"
End With
'Time
thisRun(1) = GenRamp(0, CDbL(RecordLength - 1), 1, 0, (RecordLength - 1) / SampleRate)
With thisRun(1)
.Class = "CGE Run"
.Type = "Time"
End With
'CmdVoltage
thisRun(2) = GenConstTensor(voltage, 0, RecordLength - 1, 1, "Commanded Voltage", "kV", "", "")
With thisRun(2)
.Class = "CGE Run"
.Type = "CmdVoltage"
End With
'MeasVoltage
thisRun(3) = GenConstTensor(0, 0, RecordLength - 1, 1, "Measured Voltage", "kV", "", "")
With thisRun(3)
.Class = "CGE Run"
.Type = "MeasVoltage"
End With
'CmdCurrent
thisRun(4) = GenConstTensor(current, 0, RecordLength - 1, 1, "Commanded Current", "mA", "", "")
With thisRun(4)
.Class = "CGE Run"
.Type = "CmdCurrent"
End With
'MeasCurrent
thisRun(5) = GenConstTensor(0, 0, RecordLength - 1, 1, "Measured Current", "mA", "", "")
End With
End Select
End Private
With thisRun(4)
  .Class = "CGE Run"
  .Type = "CmdCurrent"
End With

' MeasCurrent
thisRun(5) = GenConstTensor(0, 0, RecordLength - 1, 1,
  "Measured Current", "μA", "", "")
With thisRun(5)
  .Class = "CGE Run"
  .Type = "MeasCurrent"
End With
End Select
End Sub

Private Sub LoadParams()
  ' Load parameters form main form.
  With frmMain
    ' Load local component implementations.
    Set Al = .getComponent("Allios")
    Set Ml = .getComponent("Motor")
    Set PMT = .getComponent("DataPort")
    Set temp = .getComponent("Temp")
    Set Pow3 = .getComponent("Pow", 3)
    ' Load parameter variables.
    SampleRate = .SampleRate
    ScanTime = .ScanTime
    ScanLength = .ScanLength
    NumScans = .NumScans
    voltage = .voltage
    current = .current
    RecordLength = .RecordLength
  End With
  PMTGain = 0# ' Set to zero everytime for safety.
End Sub

Private Sub RunExperiment()
  ' All code to run current experiment.
  DesignSignals ' Create signal arrays.
  ' Select running method according to experiment type.
  Select Case CurrentExp
    Case ExpType.SimpleRun
      Dim Counter1, Counter2 As Long ' Generic counter variable.
      Dim m, s As Double ' Mean and StDev for alarm.
      Dim AlarmData As Tensor

      ' Allow user to align apparatus.
      Ml.Command "DRIVE0"
      Pow3.SetP6V PMTGain
      MsgBox "Align objective to capillary." , , "Running CGE6"
      Ml.Command "DRIVE1"
      ' Initialize alarm, if necessary.
      If Alarm Then
        ReDim AlarmData.Val(l To 1000)
        Ml.Pause 0.001
        For Counter1 = 1 To 1000
          PMT.nextVal AlarmData.Val(Counter1)
        Next Counter1
        m = mean(AlarmData)
        s = sd(AlarmData)
        Watcher.SetWatcher "LevelWatcher", m, (m +
        AlarmGain * s) / m
      End If

      StatusBar1.Panels(2).Text = ParseTime(RunTime)
      A1.Pause CDbl(l / SampleRate)
      ' Begin main loop.
      For Counter1 = 0 To RecordLength - 1
        ' Get data.
        Cl.nextVal thisRun(2).Val(Counter1), _
        thisRun(4).Val(Counter1), _
        thisRun(3).Val(Counter1), _
        thisRun(5).Val(Counter1) _
        PMT.nextVal thisRun(0).Val(Counter1)
        ' Check alarm.
        If Alarm Then Watcher.Check
        thisRun(0).Val(Counter1) _
        If (Counter1 Mod SampleRate) = 0 Then
          StatusBar1.Panels(2).Text = ParseTime(Clng((RecordLength -
            Counter1) / SampleRate))
          ' Let other processes work.
          DoEvents
          If (Status = Running) _
            Else
            Exit Sub
          End If
        End If
        ' Check for stoppage of play.
        Do While Not (Status = Running)
          Cl.zero
          If Status = Paused Then
            Do While (Status = Paused)
              DoEvents
              Loop
            Else
              Exit Sub
            End If
          End If
        End Do
      Next Counter1
      ' End main loop.
      StatusBar1.Panels(2).Text = ParseTime(RunTime)
      A1.Pause CDbl(l / SampleRate)
      ' End experiment.
      For Counter1 = 0 To RecordLength - 1
        PMT.nextVal thisRun(0).Val(Counter1)
      Next Counter1
      StatusBar1.Panels(2).Text = ParseTime(RunTime)
      A1.Pause CDbl(l / SampleRate)
      ' Reset apparatus.
      Monitor("Resting")
      Ml.Command "DRIVE0"
      Pow3.SetP6V PMTGain

      ' End experiment.
      For Counter1 = 0 To RecordLength - 1
        PMT.nextVal thisRun(0).Val(Counter1)
      Next Counter1
      StatusBar1.Panels(2).Text = ParseTime(RunTime)
      A1.Pause CDbl(l / SampleRate)
      ' Reset apparatus.
      Monitor("Resting")
      Ml.Command "DRIVE0"
      Pow3.SetP6V PMTGain
  Case ExpType.SimpleRun
  Case Else
    ' Handle other cases.
End Select
End Sub
thisRun(4).Val(Counter1), _
thisRun(3).Val(Counter1), _
thisRun(5).Val(Counter1)
End If
Loop

'Set loop timing.
Al.Pause
Next Counter1

StatusBar1.Panels(2).Text = ParseTime(0)
Cl.zero
Pow3.SetP6V 0#
ExpStop
Exit Sub

'SIMPLE SCAN EXPERIMENT
Case ExpType.SimpleScan

'Generic counter variable.
Dim Element As Long

M1.Command "DRIVE0"
Pow3.SetP6V PMTGain
MsgBox "Align objective at center of capillary."

"Running CGE6."
M1.Command "DRIVE1"
M1.setAccel 1000#
M1.setVel 10#
M1.setStep (CLng(-ScanLength * StepConvert / 2))
M1.GO
Al.Pause 5#
Al.Pause
M1.setVel (ScanLength / ScanTime / VelConvert)
StatusBar1.Panels(2).Text = ParseTime(CLng(NumScans * (ScanTime + LagTime)))

For Counter1 = 0 To NumScans - 1

'Alternate scanning directions.
If (Counter1 Mod 2) = 0 Then
   Call M1.setStep(ScanLength * StepConvert)
   thisRun(6).Val(Counter1) = ScanLength
Else
   Call M1.setStep(-ScanLength * StepConvert)
   thisRun(6).Val(Counter1) = -ScanLength
End If
M1.GO
Al.Pause CDbl(1 / SampleRate)

For Counter2 = 0 To CLng(SampleRate * ScanTime) - 1

'Each loop is one data point.
   Element = Counter1 * ScanTime * SampleRate +
   thisRun(2).Val(Counter1)
   thisRun(4).Val(Counter1), _
   thisRun(3).Val(Counter1), _
   thisRun(5).Val(Counter1)

   PMT.nextVal thisRun(0).Val(Counter1)
   DoEvents

   'Check for stoppage of play.
   Do While Not (Status = Running)
      Cl.zero
      If Status = Paused Then
         Do While (Status = Paused)
            DoEvents
            Loop
      Else
         Exit Sub
      End If
      If Status = Running Then
         Cl.nextVal thisRun(2).Val(Counter1), _
         thisRun(4).Val(Counter1), _
         thisRun(3).Val(Counter1), _
         thisRun(5).Val(Counter1)
      End If
   Loop

End If

'Set internal loop timing.
Al.Pause
Next Counter2

Al.Pause (LagTime)
StatusBar1.Panels(2).Text = ParseTime(CLng(NumScans * (ScanTime + LagTime) - (Element + 1) * (ScanTime + LagTime) / ScanTime / SampleRate))
M1.FlushInput

'Record temperatures.
If RecordTemp Then
   For Counter2 = 0 To 19
      thisRun(7).Val(Counter2 + 1) = temp.GetVal(Counter2 + 1)
      Next Counter2
   End If

'Set external loop timing.
Al.Pause
Next Counter1

Cl.zero
Pow3.SetP6V 0#
ExpStop
Exit Sub
Exit Sub

Case Else
   MsgBox "Whoa! You took a wrong turn there, buddy."
   Exit Sub
End Select
End Sub

Private Sub SetParams()
'Apply current parameters.
    txtSampleRate.Text = Format(SampleRate, "0.##")
    txtRecordLength.Text = Format(RecordLength)
    txtPMTGain.Text = Format(PMTGain, "0.##")
    txtCurrent.Text = Format(current, "0.##")
    txtAlarmGain.Text = Format(AlarmGain, "0.##")
    txtNumScans.Text = Format(NumScans)
    txtScanLength.Text = Format(ScanLength, "0.##")
    txtScanTime.Text = Format(ScanTime)
    txtRunTime.Text = ParseTime(RunTime)
End Sub

Private Sub SetSimpleRun()
'Change to a simple run experiment.
    CurrentExp = SimpleRun
    tabExp.Tab = CurrentExp
    txtRecordLength.Locked = False
End Sub

Private Sub SetSimpleScan()
'Change to a simple scan experiment.
    CurrentExp = SimpleScan
    tabExp.Tab = CurrentExp
    txtRecordLength.Locked = True
End Sub

Private Sub UpdateParams()
'Apply parameter math dependent upon experiment mode.
    Select Case CurrentExp
        Case ExpType.SimpleRun
            RunTime = CLng(RecordLength / SampleRate)
        Case ExpType.SimpleScan
            RecordLength = CLng(NumScans * ScanTime * SampleRate)
            RunTime = CLng((ScanTime + LagTime) * NumScans)
        Case Else
            MsgBox "Tab number is invalid.", , "Bad tab."
    End Select
    SetParams
End Sub

Private Sub txtAlarmGain_LostFocus()
    AlarmGain = CDbl(txtAlarmGain.Text)
End Sub

Private Sub txtCurrent_LostFocus()
    current = CDbl(txtCurrent.Text)
End Sub

Private Sub txtNumScans_LostFocus()
    NumScans = CLng(txtNumScans.Text)
    UpdateParams
End Sub

Private Sub txtPMTGain_LostFocus()
    PMTGain = CDbl(txtPMTGain.Text)
End Sub

Private Sub txtRecordLength_LostFocus()
    RecordLength = CLng(txtRecordLength.Text)
    UpdateParams
End Sub

Private Sub txtScanTime_LostFocus()
    ScanTime = CLng(txtScanTime.Text)
    UpdateParams
End Sub

Private Sub txtScanTime_LostFocus()
    ScanTime = CLng(txtScanTime.Text)
    UpdateParams
End Sub

Private Sub txtVm Voltage_LostFocus()
    voltage = CDbl(txtVoltage.Text)
End Sub

Private Function ParseTime(Sees As Long) As String
    'Parses time in the seconds to the following form: hh:mm:ss
    Dim Hours As Long
    Dim Minutes As Long
    Dim Seconds As Long
    Dim temp As Long
    Hours = CLng(Fix(Sees / 3600))
    temp = Sees Mod 3600
    Minutes = CLng(Fix(temp / 60))
    Seconds = temp Mod 60
    ParseTime = Format(Hours, "00") & ":" & Format(Minutes, "00") & ":" & Format(Seconds, "00")
End Function

Private Sub SaveParams()
'Saves parameters that have been changed during this experiment.
With frmMain
    .SampleRate = SampleRate
    .ScanTime = ScanTime
    .ScanLength = ScanLength
End With
End Sub
Option Explicit

' frmMain.frm
' Visual Basic main form for CGE6 application.
' Written by Matthew R. Graham
' Last Modified on 15 April 1999.
' Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.

' CGE6
' This is meant to be a comprehensive program for controlling the capillary gel electrophoresis apparatus. Power supply and temperature reading control has been implemented in this version.

'To do:
' Implement Reset menu item.

' Components.
Dim A1 As Allios ' Only implementation of Allios interface.
Dim C1 As CZE1000 ' Only implementation of CZE1000 interface.
Dim M1 As Motor ' Implementation on single Motor interface.
Dim s1 As Switch ' Implementation of single Switch interface.
Dim V1 As Valve ' Implementation of single Valve interface.
Dim V2 As Valve ' Implementation of single Valve interface.
Dim PMT As DataPort ' Implementation of single DataPort interface for a photomultiplier tube.
Dim Pow1 As NHPE3631 ' Implementation on single HPE3631 interface.
Dim Pow2 As NHPE3632 ' Implementation on single HPE3632 interface.
Dim Pow3 As NHPE3631 ' Implementation on single HPE3631 interface.
Dim Dat1 As NHPE3632 ' Implementation on single HPE3632 interface.
Dim temp1 As NHPE3631 ' Implementation on single HPE3631 interface.
Dim Hot As NHPE3632 ' Implementation on single HPE3632 interface.
Dim Cold As NHPE3632 ' Implementation on single HPE3632 interface.

' Basic Parameters.
Dim CCmdChan As Long ' Current limit command D/A channel.
Dim CMesChan As Long ' Current measurement A/D channel.
Dim VCmdChan As Long ' HV command D/A channel.
Dim VMesChan As Long ' HV measurement A/D channel.
Dim PolChan As Long ' HV polarity control D/A channel.
Dim S1PCan As Long ' Switch 1's polarity D/A channel.
Dim S1CCan As Long ' Switch 1's control D/A channel.
Dim S2PCan As Long ' Switch 2's polarity D/A channel.
Dim S2CCan As Long ' Switch 2's control D/A channel.
Dim V1Chan As Long ' Valve 1's control D/A channel.
Dim V2Chan As Long ' Valve 2's control D/A channel.
Dim PMTChan As Long ' A/D channel for PMT data.
Public SampleRate As Double ' Sampling rate for PMT data.
Public ScanTime As Long ' Time to do single one directional scan.
Public ScanLength As Double ' Length of each one directional scan.
Public NumScans As Long ' Number of scans in each scan run.
Public voltage As Double ' High voltage value in kilovolts.
Public current As Double ' Current limit in microamperes.
Public RecordLength As Long ' Length of PMT data record.
Public Exp As ExpType ' Current type of experiment.

' Working variables.
Public CurrentWindow As Object ' Keeps track of currently active form.
Public ChanBipolar(0 To 15) As Boolean ' Channel array for Allios initialization.
Public Pol As Polarity ' HV Polarity variable.
Public CurrentRun() As Tensor ' Includes all data necessary to reconstruct run.

Private Sub Form_Load()
    ' Variable initialization.
    SampleRate = 100
    ScanTime = 10
    ScanLength = 100
    NumScans = 500
    voltage = 5
    current = 100
    RecordLength = CLng(ScanTime * NumScans * SampleRate)
    CCmdChan = 3
    VCmdChan = 2
    CMesChan = 3
    VMesChan = 2
    PolChan = 0
    Pol = Positive
    S1PCan = 4
    S1CCan = 5
    S2PCan = 6
    S2CCan = 7
    V1Chan = 8
V2Chan = 9
PMTChan = 0
ChanBipolar(0) = False 'These settings are hardwired on the Allios board
ChanBipolar(1) = False 'and should not be changed unless the hardware
ChanBipolar(2) = False 'settings are altered.
ChanBipolar(3) = False
ChanBipolar(4) = True
ChanBipolar(5) = True
ChanBipolar(6) = True
ChanBipolar(7) = True
ChanBipolar(8) = False
ChanBipolar(9) = False
ChanBipolar(10) = False
ChanBipolar(11) = False
ChanBipolar(12) = True
ChanBipolar(13) = True
ChanBipolar(14) = True
ChanBipolar(15) = True
Exp = SimpleRun
Set CurrentWindow = Me

'Create new components.
Set Al = New Allios
Set Cl = New CZE1000
Set M1 = New Motor
Set s1 = New Switch
Set s2 = New Switch
Set V1 = New Valve
Set V2 = New Valve
Set PMT = New DataPort
Set Powl = New NIHPE3631
Set Pow2 = New NIHPE3632
Set Pow3 = New NIHPE3631
Set Datal = New NIHPE3632
Set templ = New NIHPE34970
Set Hot = New NIHPE3632
Set Cold = New NIHPE3632

'Initialize components.
Al.Init ChanBipolar 'Must initialize this one first!
Cl.Init Al, VCmdChan, CCmdChan, VMeasChan, CMeasChan,
Powl.Init Al, PMTC Chan
Powl.OpenDev, 1
Pow2.OpenDev, 2
Pow3.OpenDev, 3

Private Sub Form_Load(Cancel As Integer)
'Zero all outputs.
Cl.zero
Hot.SetVoltage 0#
Hot.SetCurrent 4#
Cold.SetVoltage 0#
Cold.SetCurrent 4#
Powl.SetP6V 0#
Powl.SetP6C 0#
Powl.SetP25V 0#
Powl.SetP25C 0#
End Sub

Private Sub Form_Unload(Cancel As Integer)
'Zero all outputs.
Cl.zero
Hot.SetVoltage 0#
Cold.SetVoltage 0#
Hot.SetCurrent 0#
Cold.SetCurrent 0#
Powl.SetP6V 0#
Powl.SetP6C 0#
End Sub
Private Sub mnuAbout_Click()
    'Displays About dialog form.
    Me.Enabled = False
    frmAbout.Show
    Set CurrentWindow = frmAbout
    frmAbout.SetFocus
End Sub

Private Sub mnuAllios_Click()
    'Display Allios toolbar.
    frmAllios.Show
    frmAllios.SetFocus
End Sub

Private Sub mnuCZE1000_Click()
    'Display CZE1000 toolbar.
    frmCZE1000.Show
    frmCZE1000.SetFocus
End Sub

Private Sub mnuExit_Click()
    'Quit program.
    Unload Me
End Sub

Private Sub mnuExperiment_Click()
    'Display experiment form.
    Me.Enabled = False
    frmExp.Show
End Sub
Set CurrentWindow = frmExp
frmExp.SetFocus
End Sub

Private Sub mnuExport_Click()
'Brings up menu for saving current data.
dlgCGE.Filter = "All Files (*.*)|*.*"
dlgCGE.InitDir = "d:\users\matt\data"
dlgCGE.filename = ""   
dlgCGE.CancelError = True
On Error GoTo ErrHandler
dlgCGE.Flags = cdLOFNHideReadOnly
dlgCGEDialogTitle = "Export Data..."
dlgCGE.ShowSave
Dim name As String
name = Split(dlgCGE.filename, ".")
Call SaveTensorData(CurrentRun(0), name & CurrentRun(0).Type & ".txt")
Call SaveTensorData(CurrentRun(1), name & CurrentRun(1).Type & ".txt")
Call SaveTensorData(CurrentRun(2), name & CurrentRun(2).Type & ".txt")
Call SaveTensorData(CurrentRun(3), name & CurrentRun(3).Type & ".txt")
Call SaveTensorData(CurrentRun(4), name & CurrentRun(4).Type & ".txt")
Call SaveTensorData(CurrentRun(5), name & CurrentRun(5).Type & ".txt")
Call SaveTensorData(CurrentRun(6), name & CurrentRun(6).Type & ".txt")
Call SaveTensorData(CurrentRun(7), name & CurrentRun(7).Type & ".txt")
If (Exp = SimpleScan) Then
    Call SaveTensorData(CurrentRun(6), name & CurrentRun(6).Type & ".txt")
    Call SaveTensorData(CurrentRun(7), name & CurrentRun(7).Type & ".txt")
End If
ErrHandler:
Exit Sub
End Sub

Private Sub mnuMotor_Click()
'Display Motor toolbar.
frmMotor.Show
frmMotor.SetFocus
End Sub

Private Sub mnuPMT_Click()
'Display PMT toolbar.
frmPMT.Show
frmPMT.SetFocus
End Sub

Private Sub mnuSwitch_Click()
'Display Switch toolbar.
frmSwitch.Show
frmSwitch.SetFocus
End Sub

Private Sub mnuTemperature_Click()
'Display Temperature toolbar.
frmTemperature.Show
frmTemperature.SetFocus
End Sub

Private Sub mnuValve_Click()
'Display Valve toolbar.
frmValve.Show
frmValve.SetFocus
End Sub

Public Sub CopyData(newData As Tensor)
'Allows import of data back into main form.
CurrentRun = newData
End Sub

Public Sub MoveScan(M1, Al)
M1.GO
Al.Pause 5#
M1.setVel (ScanLength / ScanTime / VelConvert)
StatusBarI.Panels(2).Text = ParseTime(CLng(NumScans * ScanTime + LagTime))
For Counter1 = 0 To NumScans - 1
    If (Counter1 Mod 2) = 0 Then
        Call M1.setStep(ScanLength * StepConvert)
        thisRun(6).Val(Counter1) = ScanLength
    Else
        Call M1.setStep(-ScanLength * StepConvert)
        thisRun(6).Val(Counter1) = -ScanLength
    End If
    M1.GO
    Al.Pause CDbl(1 / SampleRate)
End Sub

For Counter2 = 0 To CLng(SampleRate * ScanTime) - 1
    Element = Counter1 * ScanTime * SampleRate + Counter2
    thisRun(2).Val(Element) = thisRun(4).Val(Element) = thisRun(3).Val(Element) = thisRun(5).Val(Element) = thisRun(0).Val(Element)
    PMT.nextVal Element
    DoEvents
End Sub

'Check for stoppage of play.
Do While Not (Status = Running)
If Status = Paused Then
    Do While (Status = Paused)
        DoEvents
    Loop
Else
    Exit Sub
End If
If Status = Running Then
    Cl.nextVal thisRun(2).Val(Counter1), thisRun(4).Val(Counter1),
    thisRun(3).Val(Counter1), thisRun(5).Val(Counter1)
End If
Loop
' Set internal loop timing.
Al.Pause
Next Counter2
Al.Pause (LagTime)
StatusBar1.Panels(2).Text = ParseTime(CLng(NumScans * (ScanTime + LagTime) - (Element + 1) * (ScanTime + LagTime) / ScanTime / SampleRate))
M1.FlushInput
' Record temperatures.
If RecordTemp Then
    For Counter2 = 0 To 19
        thisRun(7).Val(Counter1 * 20 + Counter2) = temp.GetVal(Counter2 + 1)
    Next Counter2
End If
' Set external loop timing.
Al.Pause
Next Counter1
Cl.zero
Pow3.SetP6V 0#
Exp$Stop
Exit Sub
Case Else
    MsgBox "Whoa! You took a wrong turn there, buddy."
Exit Sub
End Select
End Sub
Private Sub SetParams()
    ' Apply parameter math dependent upon experiment mode.
    Select Case CurrentExp
        Case ExpType.SimpleRun
            RunTime = CLng(RecordLength / SampleRate)
        Case ExpType.SimpleScan
            RecordLength = CLng(NumScans * ScanTime * SampleRate)
            RunTime = CLng((ScanTime + LagTime) * NumScans)
        Case Else
            MsgBox "Tab number is invalid.", , "Bad tab."
    End Select
    SetParams
End Sub
Private Sub SetSimpleRun()
    ' Change to a simple run experiment.
    CurrentExp = SimpleRun
tabExp.Tab = CurrentExp
txtRecordLength.Locked = False
End Sub
Private Sub SetSimpleScan()
    ' Change to a simple scan experiment.
    CurrentExp = SimpleScan
tabExp.Tab = CurrentExp
txtRecordLength.Locked = True
End Sub
Private Sub UpdateParams()
    ' Apply parameter math dependent upon experiment mode.
    Select Case CurrentExp
        Case ExpType.SimpleRun
            RunTime = CLng(RecordLength / SampleRate)
        Case ExpType.SimpleScan
            RecordLength = CLng(NumScans * ScanTime * SampleRate)
            RunTime = CLng((ScanTime + LagTime) * NumScans)
        Case Else
            MsgBox "Tab number is invalid.", , "Bad tab."
    End Select
    SetParams
End Sub
Private Sub txtAlarmGain_LostFocus()
    AlarmGain = CDbl(txtAlarmGain.Text)
End Sub
Private Sub txtCurrent_LostFocus()
    current = CDbl(txtCurrent.Text)
End Sub
Private Sub txtNumScans_LostFocus()
    NumScans = CLng(txtNumScans.Text)
    UpdateParams
End Sub
Private Sub txtPMTGain_LostFocus()
    PMTGain = CDbl(txtPMTGain.Text)
End Sub
Private Sub txtSampleRate_LostFocus()
    SampleRate = Format(SampleRate, "0.##")
End Sub
Private Sub txtRecordLength_LostFocus()
    RecordLength = Format(RecordLength)
End Sub
Private Sub txtVoltage_LostFocus()
    voltage = Format(voltage, "0.##")
End Sub
Private Sub txtAlarma
    AlarmGain = CDbl(txtAlarmGain.Text)
End Sub
Private Sub txtCurrent_LostFocus()
    current = CDbl(txtCurrent.Text)
End Sub
Private Sub txtNumScans_LostFocus()
    NumScans = CLng(txtNumScans.Text)
    UpdateParams
End Sub
Private Sub txtPMTGain_LostFocus()
    PMTGain = CDbl(txtPMTGain.Text)
End Sub
Private Sub txtSampleRate_LostFocus()
    SampleRate = Format(SampleRate, "0.##")
End Sub
Private Sub txtRecordLength_LostFocus()
    RecordLength = Format(RecordLength)
End Sub
Private Sub txtVoltage_LostFocus()
    voltage = Format(voltage, "0.##")
End Sub
Private Sub txtRecordLength_LostFocus()
    RecordLength = CLng(txtRecordLength.Text)
    UpdateParams
End Sub

Private Sub txtSampleRate_LostFocus()
    SampleRate = CDbl(txtSampleRate.Text)
    UpdateParams
End Sub

Private Sub txtScanLength_LostFocus()
    ScanLength = CDbl(txtScanLength.Text)
End Sub

Private Sub txtScanTime_LostFocus()
    ScanTime = CLng(txtScanTime.Text)
    UpdateParams
End Sub

Private Sub txtVoltage_LostFocus()
    Voltage = CDbl(txtVoltage.Text)
End Sub

Private Function ParseTime(Secs As Long) As String
'Parses time in the seconds to the following form: hh:mm:ss
    Dim Hours As Long
    Dim Minutes As Long
    Dim Seconds As Long
    Dim temp As Long
    Hours = CLng(Fix(Secs / 3600))
    temp = Secs Mod 3600
    Minutes = CLng(Fix(temp / 60))
    Seconds = temp Mod 60
    ParseTime = Format(Hours, "00") & ":" & Format(Minutes, "00") & ":" & Format(Seconds, "00")
End Function

Private Sub SaveParams()
' Saves parameters that have been changed during this experiment.
    With frmMain
        .SampleRate = SampleRate
        .ScanTime = ScanTime
        .ScanLength = ScanLength
        .NumScans = NumScans
        .voltage = Voltage
        .current = current
        .RecordLength = RecordLength
        .Exp = CurrentExp
    End With
End Sub

Private Sub SaveParams()
' Saves parameters that have been changed during this experiment.
    With frmMain
        .SampleRate = SampleRate
        .ScanTime = ScanTime
        .ScanLength = ScanLength
        .NumScans = NumScans
        .voltage = Voltage
        .current = current
        .RecordLength = RecordLength
        .Exp = CurrentExp
    End With
End Sub

Option Explicit
' frmMotor.frm
' Visual Basic form for direct access to Motor components.
' Written by Matthew R. Graham
' Last Modified on 15 April 1999.
' Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.
' To do:
Dim m As Motor
Dim Command As String
Dim Step As Long
Dim Velocity As Double
Dim Acceleration As Double

Private Sub cmdChoice_Click()
' Chooses which command to give: Distance, Velocity, or Acceleration.
    Select Case cmdChoice.Text
    Case "Step"
        cmdStep (CLng(txtValue.Text))
    Case "Velocity"
        cmdVel (CDbl(txtValue.Text))
    Case "Acceleration"
        cmdAccel (CDbl(txtValue.Text))
    End Select
End Sub

Private Sub cmdCommand_Click()
' Sends command.
    m.Command (txtCommand.Text)
End Sub

Private Sub cmdSetValue_Click()
' Set value for chosen command.
    Select Case cmdChoice.Text
    Case "Step"
        m.setStep (CLng(txtValue.Text))
        m.GO
    Case "Velocity"
        m.setVel (CDbl(txtValue.Text))
    Case "Acceleration"
        m.setAccel (CDbl(txtValue.Text))
    End Select
End Sub

Private Sub cmdHome_Click()
' Brings motor to home position.
' Doesn't work very accurately.
    m.Home
End Sub
End Select
End Sub

Private Sub Form_Load()
'Get access to motor object from main form.
Set m = frmMain.getComponent("Motor")
cmbChoice.ListIndex = 0
Step = m.getStep
Velocity = m.getVel
Acceleration = m.getAccel
txtValue.Text = CStr(Step)
End Sub

Private Sub Form_Unload(Cancel As Integer)
frmMain.CurrentWindow.SetFocus
End Sub

Option Explicit

'frmPMT.frm
'Visual Basic form for direct control of Photomultiplier tube.
'Uses HPIB interface and Allios interface.
'Written by Matthew R. Graham
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.

Dim p As DataPort
Dim Pow3 As NiHPE3631

Private Sub cmdRead_Click()
'Reads value from PMT.
Dim v As Double
p.nextVal v
txtRead.Text = Format(v, "0.00")
End Sub

Private Sub Command1_Click()
'Check and Set gain voltage.
Dim v As Double
v = CDbI(txtGain.Text)
If (v > 1) Then
    MsgBox "Gain must be in the range 0-1 V.", , "Bad PMT"
Exit Sub
End If
Pow3.SetP6V v
End Sub

Private Sub Form_Load()
'Get access to component objects from main form.
Set p = frmMain.getComponent("DataPort")
Set Pow3 = frmMain.getComponent("Pow", 3)
End Sub

Private Sub Form_Load(Cancel As Integer)
frmMain.CurrentWindow.SetFocus
End Sub

Option Explicit

'frmSwitch.frm
'Visual Basic form for direct control of Switch component.
'Written by Matthew R. Graham
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.

'To do:
Dim s1 As Switch
Dim s2 As Switch

Private Sub Form_Load()
'Get Switches from Main form.
Set s1 = frmMain.getComponent("Switch", 1)
Set s2 = frmMain.getComponent("Switch", 2)
'Get current states of Switches.
opSwitch1Pull.value = (s1.getState = Pulled)
opSwitch2Pull.value = (s2.getState = Pulled)
End Sub

Private Sub Form_Unload(Cancel As Integer)
frmMain.CurrentWindow.SetFocus
End Sub

Private Sub opSwitch1Pull_Click()
s1.Pull
End Sub

Private Sub opSwitch1Push_Click()
s1.Push
End Sub

Private Sub opSwitch2Pull_Click()
s2.Pull
End Sub

Private Sub opSwitch2Push_Click()
s2.Push
End Sub

Private Sub Form_Load()
'Get access to component objects from main form.

Option Explicit

'frmTemperature.frm
'Visual Basic form for direct control of setup temperature.
'Uses HPIB interface.
'Written by Matthew R. Graham
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.

Dim Hot As NiHPE3632
Dim Cold As NiHPE3632
Dim temp As NiHP34901

Private Sub cmdRead_Click()
    'Read temperature of specified channel.
    txtTemp.Text = Format(temp.GetVal(CLng(txtChannel.Text)), 
        "0.00")
End Sub

Private Sub cmdUpdate_Click()
    'Set heater voltages.
    Hot.SetVoltage CDbl(txtHot.Text)
    Cold.SetVoltage CDbl(txtCold.Text)
End Sub

Private Sub Form_Load() 
    'Get access to objects from main form.
    Set Hot = frmMain.getComponent("Hot")
    Set Cold = frmMain.getComponent("Cold")
    Set temp = frmMain.getComponent("Temp")
End Sub

Private Sub Form_Unload(Cancel As Integer)
    'Set V2 = frmMain.getComponent("Valve", 2)
    'Get current states of Valves.
    opValve1Closed.value = (V1.getState = Closed)
    opValve2Closed.value = (V2.getState = Closed)
End Sub

Private Sub opValve1Closed_Click()
    V1.CloseValve
End Sub

Private Sub opValve1Open_Click()
    V1.OpenValve
End Sub

Private Sub opValve2Closed_Click()
    V2.CloseValve
End Sub

Private Sub opValve2Open_Click()
    V2.OpenValve
End Sub

Option Explicit

'MorePlexusFunctions.bas
'This Visual Basic module extends Professor Ian W. Hunter's
PlexusFunctions which implements
'a variety of functions for the Plexus Tensor representation of
data. In some cases functions
'have been corrected from Prof. Hunter's code, while some
'others are modifications which expand
'the capabilities of the Tensor representation. Others are
'wholly new functions which were
'necessary for specific purposes.

'Written by Matthew R. Graham
'Some functions base on code written by Ian W. Hunter.
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab
and Ian W. Hunter where appropriate.

Function GenConstTensor(Val As Double = 1#, imin As Long = 0, 
    imax As Long = 1000, DomainIncrement As Double = 1#, 
    ...)
Generates a Tensor of specified length with a constant value in all entries.

```vba
Dim i As Long
Dim temp() As Double
ReDim temp(imin To imax)
For i = imin To imax
temp(i) = Val
Next i
GenConstTensor = GenTensor(temp, DomainIncrement, RangeName, RangeUnit, DomainName, DomainUnit)
End Function
```

Loads unformatted data from a text file into a Tensor. Data in text file should be in a single column.

```vba
Dim tempstr As String
Dim StrArray() As String
Dim Counter As Long
Dim tempdata() As Double
Open filename For Input As #1
Counter = 1
tempstr = Input(LOF(1), 1)
StrArray = Split(tempstr, Chr$(13) + Chr$(10))
Close #1
ReDim tempdata(1 To UBound(StrArray) + 1)
For Counter = 0 To UBound(StrArray)
tempdata(Counter + 1) = CDbl(StrArray(Counter))
Next Counter
LoadTensorData = GenTensor(tempdata, DomainIncrement, RangeName, RangeUnit, DomainName, DomainUnit)
End Function
```

Shifts Tensor entries to a zero-based array.

```vba
Function ShiftToZero(
    Tens As Tensor
) As Tensor
    Dim Counter As Long
    Dim Sum As Long
    BuildTensor = ValTensor
    For Counter1 = 1 To UBound(ParamTens.Val)
        Sum = Sum + ParamTens.Val(Counter1)
    Next Counter1
    ReDim BuildTensor.Val(1 To Sum)
    For Counter1 = 1 To UBound(ParamTens.Val)
        BuildTensor.Val(1 To Sum) = BuildTensor.Val(Counter1) + Counter1
    Next Counter1
    Sum = Sum + ParamTens.Val(Counter1)
    Next Counter1
End Function
```

Saves two-column data into a two-column text file.

```vba
Sub SaveTensorData2(
    Data As Tensor,
    Optional filename As String = "c:\temp\tempdata.txt"
) As Tensor
    Dim Counter1 As Long
    Dim Counter2 As Long
    Open filename For Output As #1
    For Counter1 = LBound(Data.Val, 1) To UBound(Data.Val, 1)
        For Counter2 = LBound(Data.Val, 2) To UBound(Data.Val, 2)
            Write #1, Data.Val(Counter1, Counter2)
        Next Counter2
    Next Counter1
    Close #1
End Sub
```

Saves data in a Tensor into a single column text file.

```vba
Sub SaveTensorData(
    Data As Tensor,
    Optional filename As String = "c:\temp\tempdata.txt"
) As Tensor
    Dim Counter As Long
    Open filename For Output As #1
    For Counter = LBound(Data.Val) To UBound(Data.Val)
        Write #1, Data.Val(Counter)
    Next Counter
    Close #1
End Sub
```

Generates a Tensor of specified length with a constant value in all entries.

```vba
Optional RangeName As String = "Amplitude",
Optional RangeUnit As String = "",
Optional DomainName As String = "Time",
Optional DomainUnit As String = "s"
) As Tensor
```

Saves two-column data into a two-column text file.

```vba
Sub SaveTensorData2(
    Data As Tensor,
    Optional filename As String = "c:\temp\tempdata.txt"
) As Tensor
    Dim Counter1 As Long
    Dim Counter2 As Long
    Open filename For Output As #1
    For Counter1 = LBound(Data.Val, 1) To UBound(Data.Val, 1)
        For Counter2 = LBound(Data.Val, 2) To UBound(Data.Val, 2)
            Write #1, Data.Val(Counter1, Counter2)
        Next Counter2
    Next Counter1
    Close #1
End Sub
```

Saves data in a Tensor into a single column text file.

```vba
Sub SaveTensorData(
    Data As Tensor,
    Optional filename As String = "c:\temp\tempdata.txt"
) As Tensor
    Dim Counter As Long
    Open filename For Output As #1
    For Counter = LBound(Data.Val) To UBound(Data.Val)
        Write #1, Data.Val(Counter)
    Next Counter
    Close #1
End Sub
```
Dim Min As Long
Dim Max As Long
Dim Counter As Long

Min = LBound(Tens.Val)
Max = UBound(Tens.Val)
ShiftToZero = Tens
ReDim ShiftToZero.Val(0 To Max - Min)
For Counter = Min To Max
    ShiftToZero.Val(Counter - Min) = Tens.Val(Counter)
Next Counter

End Function

Function WindowAverage(Tens As Tensor, Optional WindowSize As Long = 10)
    As Tensor
    'Returns a Tensor filled with moving window averages from original data.
    Dim Windows As Long
    Dim ArraySize As Long
    Dim Counter As Long
    Dim Counter2 As Long
    Dim Sum As Double
    WindowAverage = Tens
    ArraySize = UBound(Tens.Val) - LBound(Tens.Val) + 1
    Windows = Round((ArraySize / WindowSize) - 0.5, 0)
    ReDim WindowAverage.Val(1 To Windows)
    For Counter1 = 1 To Windows
        Sum = 0
        For Counter2 = 1 To WindowSize
            Sum = Sum + Tens.Val(WindowSize * (Counter1 - 1) + Counter2 - LBound(Tens.Val) - 1)
        Next Counter2
        WindowAverage.Val(Counter1) = Sum / WindowSize
    Next Counter1
    End Function

Function GetColumn(Data() As Double, ColumnIndex As Long) As Double()
    'Extracts single columns of data from a 2D array.
    Dim LoIndex As Long
    Dim HiIndex As Long
    Dim Counter As Long
    Dim temp() As Double
    LoIndex = LBound(Data, 2)
    HiIndex = UBound(Data, 2)
    ReDim temp(LoIndex To HiIndex)
    For Counter = LoIndex To HiIndex
        temp(Counter) = Data(ColumnIndex, Counter)
    Next Counter
    GetColumn = temp
End Function

Function PhaseUnwrap(Phase As Tensor, Optional MinVal As Double = 0, Optional MaxVal As Double = 360)
    As Tensor
    'Phase is a tensor with one column of phase data, where the phase is bounded by MinVal and MaxVal. PhaseUnwrap returns a tensor with phase data unwrapped beyond these min and max values.
    'Written by Matthew Graham
    'May 4, 1999
    Dim Counter As Long
    Dim MinIndex As Long
    Dim MaxIndex As Long
    Dim Range As Double
    Dim Last As Double
    Dim phasediff As Double
    Dim PhaseOffset As Double
    PhaseUnwrap = Phase
    MinIndex = LBound(Phase.Val)
    MaxIndex = UBound(Phase.Val)
    Range = MaxVal - MinVal
    Last = Phase.Val(MinIndex)
    PhaseOffset = 0
    For Counter = MinIndex To MaxIndex
        phasediff = Phase.Val(Counter) - Last
        If (Abs(phasediff) > (Range / 2)) Then
            If (phasediff > MinVal) Then
                PhaseOffset = PhaseOffset - Range
            Else
                PhaseOffset = PhaseOffset + Range
            End If
        End If
        PhaseUnwrap.Val(Counter) = Phase.Val(Counter) + PhaseOffset
        Last = Phase.Val(Counter)
    Next Counter
    End Function

Public Function TFPhase2(H As Tensor, freq As Tensor) As Tensor
    'Not edited yet...
    'Returns the gain part of the Fourier transform of h.
    'If h is a finite impulse response function then the gain part of the transfer function is returned.
Subroutine TFPhase2

\[
\begin{align*}
\text{TFPhase2}(H) = \text{Phase} \\
\text{TFPhase2}(H) \text{Inc}(l) = 1#
\end{align*}
\]

\[
\begin{align*}
imin = \text{LBound}(H.\text{Val}) \\
imax = \text{UBound}(H.\text{Val}) \\
FreqMin = \text{LBound}(freq.\text{Val}) \\
n = imax - imin + 1
\end{align*}
\]

\[
\text{ReDim TFP}_{\text{phase2}}.\text{Val}(j_{\text{min}} \text{To } j_{\text{max}})
\]

\[
\begin{align*}
\text{For } j = j_{\text{min}} \text{ To } j_{\text{max}} \\
re = 0# \\
im = 0# \\
\text{For } i = 0 \text{ To } n - 1 \\
re = re + H.\text{Val}(i + imin) \times \cos(twopi \times freq.\text{Val}(FreqMin + j) \times i / (n - 1)) \\
im = im + H.\text{Val}(i + imin) \times \sin(twopi \times freq.\text{Val}(FreqMin + j) \times i / (n - 1))
\end{align*}
\]

\[
\text{Next } i
\]

\[
\text{TFPhase2}.\text{Rname}(l) = "\text{Phase}" \\
\text{TFPhase2}.\text{Runit}(l) = "\text{Deg}" \\
\text{TFPhase2}.\text{Dname}(l) = "\text{Frequency}" \\
\text{TFPhase2}.\text{Dunit}(l) = "\text{Hz}"
\]

End Function

Function TFGain2(H As Tensor, freq As Tensor) As Tensor

'This subroutine calculates the gain part of the Fourier transform of H.
If H is a finite impulse response function then the gain part of the transfer function is returned.

\[
\begin{align*}
\text{TFGain2}(H) = H \\
\text{TFGain2}.\text{Inc}(l) = 1#
\end{align*}
\]

\[
\begin{align*}
imin = \text{LBound}(H.\text{Val}) \\
imax = \text{UBound}(H.\text{Val}) \\
FreqMin = \text{LBound}(freq.\text{Val}) \\
n = imax - imin + 1
\end{align*}
\]

\[
\text{ReDim TFG}_{\text{gain2}}.\text{Val}(j_{\text{min}} \text{To } j_{\text{max}})
\]

\[
\begin{align*}
\text{For } j = j_{\text{min}} \text{ To } j_{\text{max}} \\
re = 0# \\
im = 0# \\
\text{For } i = 0 \text{ To } n - 1 \\
re = re + H.\text{Val}(i + imin) \times \cos(twopi \times freq.\text{Val}(FreqMin + j) \times i / (n - 1)) \\
im = im + H.\text{Val}(i + imin) \times \sin(twopi \times freq.\text{Val}(FreqMin + j) \times i / (n - 1))
\end{align*}
\]

\[
\text{Next } i
\]

\[
\text{TFGain2}.\text{Rname}(l) = "\text{Frequency}" \\
\text{TFGain2}.\text{Runit}(l) = "\text{Hz}"
\]

End Function

Function LogSamples(StartFreq As Double, StopFreq As Double, Samples As Long) As Tensor

'Returns a Tensor of values evenly spaced in the log

\[
\begin{align*}
\text{LogSamples}(\text{StartFreq}, \text{StopFreq}, \text{Samples})
\end{align*}
\]
Dim Freqs() As Double
ReDim Freqs(0 To Samples - 1)
Dim Counter As Long
Dim LogFactor
LogFactor = (Log(StopFreq) / Log(10#) - Log(StartFreq) / Log(10#)) / (Samples - 1)
For Counter = 0 To Samples - 1
Freqs(Counter) = (10 ^ (LogFactor * Counter + Log(StartFreq) / Log(10#)))
Next Counter

Dim MaxFreq As Double
pi = 4# * Atn(1#)
imin = 0
imax = UBound(Cxx.Val)
ReDim wCxx(0 To imax)
MaxFreq = 1 / 2 / Cxx.Val(1)
jmin = LBound(freq.Val)
jmax = UBound(freq.Val)
AutoSpectrum2 = Cxx
ReDim AutoSpectrum2.Val(jmin To jmax)
If WindowFunction = Tukey Then 'default
For i = 0 To imax
wCxx(i) = 0.5 * (1# + Cos(pi * i / imax)) * Cxx.Val(1)
Next i
ElseIf WindowFunction = Triangular Then '(i.e. Bartlett)
For i = 0 To imax
wCxx(i) = (1# - i / imax) * Cxx.Val(1)
Next i
Else 'If WindowFunction = None (i.e. rectangular)
For i = 0 To imax
wCxx(i) = Cxx.Val(1)
Next i
End If

'Perform Fourier transform
For j = jmin To jmax
cosf = Cos(freq.Val(j) * pi / MaxFreq)
cos0 = 0#
cos1 = 0# For i = imax To 1 Step -1 'recursive calculation of cos
cos2 = 2# * cosf * cos1 - cos0 + wCxx(i)
cos0 = cos1
cos1 = cos2
Next i
AutoSpectrum2.Val(j) = 2# * Cxx.Inc(1) * (wCxx(0) + 2# * (cos1 * cosf - cos0))
Next j

AutoSpectrum2.Rname(1) = "Power"
AutoSpectrum2.Dname(1) = "Frequency"
AutoSpectrum2.Dunit(1) = "Hz"
AutoSpectrum2.Inc(1) = 1

End Function

Function RMS(X As Tensor) As Double
'Takes Root-Mean-Square of a Tensor of data.
RMS = (SS(X) / (UBound(X.Val) - LBound(X.Val) + 1)) ^ 0.5
End Function

Public Function Integrate2( _
X As Tensor, _
Optional Y _
) As Tensor
'Returns the integral of X(i) over Y(i), i=imin,...,imax
'Uses simple trapezoidal numerical integration
'The integration starts at 0
'
'This function has not been copied or adapted from any software.
'It was written by Ian Hunter
'Last update Jan 10 1999
'Modified by Matthew R. Graham

Dim i As Long
Dim Sum As Double
Dim imin As Long
Dimimax As Long
imin = LBound(X.Val)
imax = UBound(X.Val)

Integrate2 = X
Sum = 0#
If IsMissing(Y) Then
  Integrate2.Val(imin) = Sum
  For i = imin + 1 To imax
    Sum = Sum + X.Inc(1) * (X.Val(i) + X.Val(i - 1)) / 2
    Integrate2.Val(i) = Sum
  Next i
Else
  Integrate2.Val(imin) = Sum
  For i = imin + 1 To imax
    Sum = Sum + (Y.Val(i) - Y.Val(i - 1)) * (X.Val(i) + X.Val(i - 1)) / 2
    Integrate2.Val(i) = Sum
  Next i
End If
End Function

Public Function MSeq(n As Long) As Tensor
'Returns M-Sequence as a Tensor.
Dim X() As Double
Dim Length As Long
Length = 2 ^ n - 1
ReDim X(0 To Length - 1)
Dim Tap1 As Long
Dim Tap2 As Long
Dim Counter

Select Case n
Case 2
  Tap1 = 1
  Tap2 = 2
Case 3
  Tap1 = 2
  Tap2 = 3
Case 4
  Tap1 = 3
  Tap2 = 4
Case 5
  Tap1 = 3
  Tap2 = 5
Case 6
  Tap1 = 5
  Tap2 = 6
Case 7
  Tap1 = 6
  Tap2 = 7
Case 8
  Tap1 = 4
  Tap2 = 5
Case 9
  Tap1 = 5
  Tap2 = 9
Case 10
  Tap1 = 7
  Tap2 = 10
Case 11
  Tap1 = 9
  Tap2 = 11
Case 12
  Tap1 = 6
  Tap2 = 8
Case 13
  Tap1 = 9
  Tap2 = 10
Case 14
  Tap1 = 4
  Tap2 = 8
Case 15
  Tap1 = 14
  Tap2 = 15
End Select

For Counter = 0 To n - 1
  X(Counter) = 1
Next Counter

For Counter = n To Length - 1
  If X(Counter - Tap1) = X(Counter - Tap2) Then
    X(Counter) = 0
  Else

End If

End Function
Dim m As Long
Dim LUP() As Double
m = UBound(p)
Redim alpha(1 To m, 1 To m)
Redim Beta(1 To m, 1 To 1)
Redim delta(1 To m, 1 To 1)
Redim Pnew(1 To m)
Redim Pbest(1 To m)
Redim LUP(1 To m, 1 To m + 1)
For j = 1 To m
    Pbest(j) = p(j)
Next j
SS = CalcSS2(X, Y, Pbest)
SSold = SS
lambda = 0.001
For iter = 1 To IterMax
    alpha = CalcAlpha2(X, Y, Pbest)
    Beta = CalcBeta2(X, Y, Pbest)
    'add 1+lambda to main diagonal
    For j = 1 To m
        alpha(j, j) = alpha(j, j) * (1 + lambda)
    Next j
    'solve for the parameter increments, delta
    LUP = MatLUPDecompose(alpha)
    delta = MatLUPSolve(LUP, Beta)
    For j = 1 To m
        Pnew(j) = Pbest(j) + delta(j, 1)
    Next j
    SSnew = CalcSS2(X, Y, Pnew)
    If SSnew < SS Then 'new parameters are better
        lambda = 0.1 * lambda
        SS = SSnew
        For j = 1 To m
            Pbest(j) = Pnew(j)
        Next j
    Else 'keep original parameters
        lambda = 10 * lambda
        If SSnew = SS And iter > 1 Then
            Exit For
        End If
    End If
End If
'alterations by matt
frmMain.ProgressBar1.value = iter / IterMax * 100
Next iter

MarquardtMin2 = Pbest

End Function

Public Function CalcAlpha2( _
    X As Tensor, _
    Y As Tensor, _
    p() As Double _
    ) As Double()

    'Used by Marquardt
    'Returns the m\*m alpha matrix (0.5 times the Hessian matrix) of partial
    'derivatives of the function to be fitted with respect to the m parameters
    'The data pairs are Y(i) and X(i) where i = 1 to n
    'P(j) j = 1 to m are the function parameters
    'It is a little inefficient to separate the calculation of alpha, beta and SS
    'but it makes for much more readable code
    'This function was written by Ian Hunter and has not been copied or adapted
    'from any existing software.
    'Last update Jan 10 1999
    'Modified on 28 January by Matthew R Graham.

    Dim i As Long
    Dim j As Long
    Dim k As Long
    Dim m As Long
    Dim imin As Long
    Dimimax As Long
    Dim FunDerivative() As Double
    Dim alpha() As Double

    m = UBound(p)
imin = LBound(Y.Val)
imax = UBound(Y.Val)
ReDim FunDerivative(1 To m)
ReDim alpha(1 To m, 1 To m)

    For i = imin To imax
        FunDerivative = FunD(X.Val(i), p)
        For j = 1 To m
            For k = 1 To j 'only compute upper diagonal as alpha is symmetric
                alpha(j, k) = alpha(j, k) + FunDerivative(j) * FunDerivative(k)
            Next k
        Next j
    Next i

    'populate lower diagonal
    For j = 2 To m
        For k = 1 To j - 1
            alpha(k, j) = alpha(j, k)
        Next k
    Next j

    CalcAlpha2 = alpha

End Function

Public Function CalcBeta2( _
    X As Tensor, _
    Y As Tensor, _
    p() As Double _
    ) As Double()

    'Used by Marquardt
    'Returns the m length beta vector
    'The data pairs are X(i) and Y(i) where i = 1 to n
    'P(j) j = 1 to m are the function parameters
    'It is a little inefficient to separate the calculation of alpha, beta and SS
    'but it makes for much more readable code
    'This function was written by Ian Hunter and has not been copied or adapted
    'from any existing software.
    'Last update Jan 10 1999
    'Modified on 28 January 2000 by Matthew R Graham.

    Dim i As Long
    Dim j As Long
    Dim imin As Long
    Dimimax As Long
    Dim m As Long
    Dim error As Double
    Dim FunDerivative() As Double
    Dim Beta() As Double

    imin = LBound(Y.Val)
imax = UBound(Y.Val)
m = UBound(p)
ReDim FunDerivative(1 To m)
ReDim Beta(1 To m, 1 To 1)

    For i = imin To imax
        error = Y.Val(i) - FunD(X.Val(i), p)
        FunDerivative = FunD(X.Val(i), p)
        For j = 1 To m
            Beta(j, 1) = Beta(j, 1) + FunDerivative(j) * error
        Next j
    Next i
Next i
CalcBeta2 = Beta
End Function

Public Function CalcSS2( _
X As Tensor, _
Y As Tensor, _
p() As Double _
) As Double

'Used by Marquardt
'Returns the sum of squared error between the Y and
' the parametric function, Fun, being fitted.
'The data pairs are X(i) and Y(i) where i = 1 to n
'P(j) j = 1 to m are the function parameters
'It is a little inefficient to separate the calculation of
alpha, beta and SS
' but it makes for much more readable code
'This function was written by Ian Hunter and has not been
copied or adapted
'from any existing software.
'Last update Jan 10 1999
'Modified on 28 January 2000 by Matthew R Graham.
Dim i As Long
Dim imin As Long
Dim imax As Long
imin = LBound(Y.Val)
imax = UBound(Y.Val)
CalcSS2 = 0#
For i = imin To imax
CalcSS2 = CalcSS2 + (Y.Val(i) - Fun(X.Val(i), p)) ^ 2
Next i
End Function

Public Function StochasticMinl( _
Y As Tensor, _
Pest() As Double, _
Psd() As Double, _
Optional NumIterations As Long = 1000, _
Optional NumRandomParameterSets As Long = 100 _
) As Double()

'Uses stochastic minimization to find the parameters which
"best fit" some externally defined objective function
(called ObjFunName in ObjFunEnv eg Forml) whose only argument is the vector of parameters
'P(j) j = 1 to m
'On entry initial estimates of the parameters must be supplied together
'with initial estimates of the range (specified as a standard deviation)
'likely to contain the "best" parameter.
'On exit Pest(j) and Psd(j) j = 1 to m contain the updated parameter
'estimates and standard deviations. These are then ready to be used
'in a subsequent call.
'The value of the objective function is returned
'The function randomly (stochastically) chooses m parameter values called
Ptest from a Gaussian distributions centered on Pest and
with standard deviations of Psd. For each iteration it stochastically chooses
NumRandomParameterSets sets of m parameter values.
NumRandomParameterSets (default 100) should be roughly 10^NumParameters.
'This function uses the supplied objective function to calculate
the value of the objective function for the NumRandomParameterSets.
'If the smallest objective function is smaller than the current value (ie.
the one from the previous iteration, then the Pest is updated to Ptest
and the Psd is reduced by 10%.
'This process is repeated NumIterations times.
'Stochastic minimization is slower than the usual
deterministic methods
'but will find an objective function minimum even when the function
'to be fitted is highly nonlinear and the parameters are'not orthogonal
'to each other.
'This function may be called once by setting NumIterations to
a suitably high value (default -1000)
Alternatively it may be set to a smaller value and the minimization
function may then be called repeatedly (perhaps plotting the results
after each call.)
This function differs from most Plexus functions in that its arguments: (Pest and Psd) are changed on each call.

This function was written by Ian Hunter and has not been copied or adapted from any existing software.

Last update Jan 10 1999 Modified by Matthew R. Graham

Dim i As Long
Dim j As Long
Dim k As Long
Dim n As Long
Dim OFn As Double
Dim ObjFn As Double
Dim ObjFnMin As Double
Dim gwn As Double
Dim Ptest() As Double 'trial parameter estimates
Dim NumParameters As Long 'number of parameters to estimate

NumParameters = UBound(Pest) 'which must equal UBound(Psd)
ReDim Ptest(1 To NumParameters)

ObjFnMin = CalcSS(Y, Pest)

'Main iteration loop
For i = 1 To NumIterations
    ObjFn = ObjFnMin
    For n = 1 To NumRandomParameterSets 'should be about 10^NumParameters
        'randomly choose next set of parameters
        For j = 1 To NumParameters
            gwn = -6#
            For k = 1 To 12
                gwn = gwn + Rnd()
            Next k
            Ptest(j) = Pest(j) + gwn * Psd(j) 'centers Gaussian PDF around Pest
        Next j
        OFn = CalcSS(Y, Ptest)
        If OFn < ObjFn Then
            ObjFn = OFn
            For j = 1 To NumParameters
                Pest(j) = Ptest(j)
            Next j
        End If
    Next n
    If ObjFn < ObjFnMin Then
        ObjFnMin = ObjFn
    End If
    Next i
End Function

Public Function StochasticMin2( _
    X As Tensor, _
    Y As Tensor, _
    Pest) As Double,
    _
    Psd() As Double, _
    Optional NumIterations As Long = 1000, _
    Optional NumRandomParameterSets As Long = 100 _
) As Double()

'Uses stochastic minimization to find the parameters which minimize some externally defined objective function (called ObjFunName) in ObjFunEnv eg Form1) whose only argument is the vector of parameters P(j) j = 1 to m

'On entry initial estimates of the parameters must be supplied together with initial estimates of the range (specified as a standard deviation) likely to contain the "best" parameter.

'On exit Pest(j) and Psd(j) j = 1 to m contain the updated parameter estimates and standard deviations. These are then ready to be used in a subsequent call.

'The value of the objective function is returned.

'The function randomly (stochastically) chooses m parameter values called Ptest from a Gaussian distributions centered on Pest and with standard deviations of Psd. For each iteration it stochastically chooses NumRandomParameterSets sets of m parameter values.

'NumRandomParameterSets (default 100) should be roughly 10^NumParameters.
This function uses the supplied objective function to calculate the value of the objective function for the NumRandomParameterSets.

If the smallest objective function is smaller than the current value (ie. the one from the previous iteration, then the Pest is updated to Ptest and the Psd is reduced by 10%.

This process is repeated NumIterations times.

'Stochastic minimization is slower than the usual deterministic methods but will find an objective function minimum even when the function to be fitted is highly nonlinear and the parameters are not orthogonal to each other.

This function may be called once by setting NumIterations to a suitably high value (default =1000) Alternatively it may be set to a smaller value and the minimization function may then be called repeatedly (perhaps plotting the results after each call.

This function differs from most Plexus functions in that its arguments (Pest and Psd) are changed on each call.

This function was written by Ian Hunter and has not been copied or adapted from any existing software.

'Last update Jan 10 1999
'Modified by Matthew R. Graham

Dim i As Long
Dim j As Long
Dim k As Long
Dim n As Long
Dim gwn As Double
Dim ObjFn As Double
Dim Ptest() As Double 'trial parameter estimates
Dim NumParameters As Long 'number of parameters to estimate
Dim ObjFnMin As Double
Dim ObjFn As Double
Dim ObjFnMin As Double
Dim gwn As Double
Dim Ptest() As Double 'trial parameter estimates
Dim NumParameters As Long 'number of parameters to estimate
Dim ObjFnMin As Double
Dim ObjFn As Double
Dim gwn As Double
Dim Ptest() As Double 'trial parameter estimates
Dim NumParameters As Long 'number of parameters to estimate
Dim ObjFnMin As Double
Dim ObjFn As Double
Dim gwn As Double
Dim Ptest() As Double 'trial parameter estimates
Dim NumParameters As Long 'number of parameters to estimate

For i = 1 To NumIterations
    ObjFn = ObjFnMin
    For n = 1 To NumRandomParameterSets 'should be about 10^NumParameters
        ObjFn = CalcSS2(X, Y, Pest)
    Next n
    If ObjFn < ObjFnMin Then
        ObjFnMin = ObjFn
        For j = 1 To NumParameters
            Pest(j) = Ptest(j) + gwn * Psd(j) 'centers Gaussian PDF around Pest
        Next j
    End If
Next i

StochasticMin2 = Pest
End Function

Public Function MarquardtMinN( _
    X() As Tensor, _
    Y As Tensor, _
    P() As Double, _
    Optional IterMax As Long = 1000 _
    ) As Double()

    'Returns the set of parameters which "best" fit the data pairs given by Y and X using a sum of squares objective function.
The data pairs are \( Y(i) \) and \( X(i) \) where \( i = 1 \) to \( n \)

\( P(j) \) \( j = 1 \) to \( m \) are the initial parameter estimates.

The minimization stops when there is no change in the sum of squares.

This function calls CalcAlpha, CalcBeta and CalcSS who in turn call

Fun\((X, P)\) and FunD\((X, P)\) see examples below.

Once the minimization has been completed Call CalcAlpha to get the parameter covariances and call CalcBeta to get the parameter variances.

This function is very fast but does require the partial derivatives of the function to be evaluated. Use the much slower StochasticMin function if the partial derivatives are not available.

This function was written by Ian Hunter and has not been copied or adapted from any existing software.

Last update Nov 7 1999

Modified by Matthew R Graham.

```vba
Dim j As Long
Dim k As Long
Dim iter As Long
Dim alpha() As Double
Dim Beta() As Double
Dim delta() As Double
Dim Pnew() As Double
Dim Pbest() As Double
Dim lambda As Double
Dim SS As Double
Dim SSnew As Double
Dim SSold As Double
Dim m As Long
Dim LUP() As Double
m = UBound(P)
ReDim alpha(1 To m, 1 To m)
ReDim Beta(1 To m, 1 To 1)
ReDim delta(1 To m, 1 To 1)
ReDim Pnew(1 To m)
ReDim Pbest(1 To m)
ReDim LUP(1 To m, 1 To m + 1)

For j = 1 To m
    Pbest(j) = P(j)
Next j

SS = CalcSSN(X, Y, Pbest)
SSold = SS

lambda = 0.001

For iter = 1 To IterMax
    alpha = CalcAlphaN(X, Y, Pbest)
    Beta = CalcBetaN(X, Y, Pbest)

    'add 1+lambda to main diagonal
    For j = 1 To m
        alpha(j, j) = alpha(j, j) * (1# + lambda)
    Next j

    'solve for the parameter increments, delta
    LUP = MatLUPDecompose(alpha)
    delta = MatLUPSolve(LUP, Beta)

    For j = 1 To m
        Pnew(j) = Pbest(j) + delta(j, 1)
    Next j

    SSnew = CalcSSN(X, Y, Pnew)
    Debug.Print iter; SSnew

    If SSnew < SS Then 'new parameters are better
        lambda = 0.1 * lambda
        SS = SSnew
        For j = 1 To m
            Pbest(j) = Pnew(j)
        Next j
    Else 'keep original parameters
        lambda = 10# * lambda
        If SSnew = SS And iter > 1 Then
            Exit For
        End If
    End If

Next iter

MarquardtMinN Pbest
End Function

Public Function CalcAlphaN( _
    X() As Tensor, _
    Y As Tensor, _
    P() As Double _
) As Double()

'Used by Marquardt
'Returns the \( m \times m \) alpha matrix (0.5 times the Hessian matrix) of partial deriviatives of the function to be fitted with respect to
```
the m parameters
'The data pairs are Y(i) and X(i) where i = 1 to n
P(j) j = 1 to m are the function parameters
'
'It is a little inefficient to separate the calculation of alpha, beta and SS
' but it makes for much more readable code
'
'This function was written by Ian Hunter and has not been copied or adapted
'from any existing software.
'Last update Jan 10 1999
'Modified by Matthew R Graham.

Dim i As Long
Dim j As Long
Dim k As Long
Dim m As Long
Dim imin As Long
Dim imax As Long
Dim FunDerivative() As Double
Dim alpha() As Double

m = UBound(P)
imin = LBound(Y.Val)
imax = UBound(Y.Val)
ReDim FunDerivative(1 To m)
ReDim alpha(1 To m, 1 To m)

For i = imin To imax
    FunDerivative = FunD(X, i, P)
    For j = 1 To m
        For k = 1 To j 'only compute upper diagonal as alpha is symmetric
            alpha(j, k) = alpha(j, k) + FunDerivative(j) * FunDerivative(k)
        Next k
    Next j
Next i

'populate lower diagonal
For j = 2 To m
    For k = 1 To j - 1
        alpha(k, j) = alpha(j, k)
    Next k
Next j

CalcAlphaN = alpha
End Function

Public Function CalcBetaN( _
    X() As Tensor, _
    Y As Tensor, _
    P() As Double _
) As Double()

'Used by Marquardt
'Returns the m length beta vector
'The data pairs are X(i) and Y(i) where i = 1 to n
'P(j) j = 1 to m are the function parameters
'
'It is a little inefficient to separate the calculation of alpha, beta and SS
' but it makes for much more readable code
'
'This function was written by Ian Hunter and has not been copied or adapted
'from any existing software.
'Last update Jan 10 1999
'Modified by Matthew R Graham.

Dim i As Long
Dim j As Long
Dim k As Long
Dim imin As Long
Dim imax As Long
Dim error As Double
Dim FunDerivative() As Double
Dim Beta() As Double

imin = LBound(Y.Val)
imax = UBound(Y.Val)
m = UBound(P)
ReDim FunDerivative(1 To m)
ReDim Beta(1 To m, 1 To 1)

For i = imin To imax
    error = Y.Val(i) - Fun(X, i, P)
    FunDerivative = FunD(X, i, P)
    For j = 1 To m
        Beta(j, 1) = Beta(j, 1) + FunDerivative(j) * error
    Next j
Next i

CalcBetaN = Beta
End Function

Public Function CalcSSN( _
    X() As Tensor, _
    Y As Tensor, _
    P() As Double _
) As Double()

'Used by Marquardt
'Returns the sum of squared error between the Y and
'the parametric function, Fun, being fitted.
'The data pairs are X(i) and Y(i) where i = 1 to n
'P(j) j = 1 to m are the function parameters
'It is a little inefficient to separate the calculation of
alpha, beta and SS
' but it makes for much more readable code
' This function was written by Ian Hunter and has not been
copied or adapted
' from any existing software.
' Last update Jan 10 1999
' Modified by Matthew R. Graham.

Dim i As Long
Dim imin As Long
Dim imax As Long

imin = LBound(Y.Val)
imax = UBound(Y.Val)

CalcSSN = 0#
For i = imin To imax
   CalcSSN = CalcSSN + (Y.Val(i) - Fun(X, i, P)) ^ 2
Next i
End Function

Public Function StochasticMinN(_
   X() As Tensor, _
   Y As Tensor, _
   Pest() As Double, _
   PSD() As Double, _
   Optional NumIterations As Long = 1000, _
   Optional NumRandomParameterSets As Long = 100 _
) As Double()

' Uses stochastic minimization to find the parameters which
"best fit"
' minimize some externally defined objective function
{called ObjFunName
' in ObjFunEnv eg Form1) whose only argument is the vector
of parameters
' P[j], j = 1 to m
'
' On entry initial estimates of the parameters must be
supplied together
' with initial estimates of the range (specified as a
standard deviation)
' likely to contain the "best" parameter.
' On exit Pest[j] and Psd[j], j = 1 to m contain the updated
parameter
' estimates and standard deviations. These are then ready to
be used
' in a subsequent call.
'
' The value of the objective function is returned
'
' The function randomly (stochastically) chooses m parameter
values called
' Pest from a Gaussian distributions centered on Pest and
with standard
' deviations of PSD. For each iteration it stochastically
choses
' NumRandomParameterSets sets of m parameter values.
' NumRandomParameterSets (default 100) should be roughly
10^NumParameters.
'
' This function uses the supplied objective function to
calculate
' the value of the objective function for the
NumRandomParameterSets.
'
' If the smallest objective function is smaller than the
current value (ie.
' the one from the previous iteration, then the Pest is
updated to Pest
' and the Psd is reduced by 10%.
' This process is repeated NumIterations times.
'
' Stochastic minimization is slower than the usual
deterministic methods
' but will find an objective function minimum even when the
function
' to be fitted is highly nonlinear and the parameters are
not orthogonal
' to each other.
'
' This function may be called once by setting NumIterations to
a
' suitably high value (default =1000)
' Alternatively it may be set to a smaller value and the
minimization
' function may then be called repeatedly (perhaps plotting
the results
' after each call.
'
' This function differs from most Plexus functions in that its
arguments
' (Pest and Psd) are changed on each call.
'
' This function was written by Ian Hunter and has not been
copied or adapted
' from any existing software.
' Last update Jan 10 1999
' Modified by Matthew R. Graham

Dim i As Long
Dim j As Long
Dim k As Long
Dim n As Long
Dim Ofn As Double
Dim ObjFn As Double
Dim ObjFnMin As Double
Dim gwn As Double
Dim Ptest() As Double 'trial parameter estimates
Dim NumParameters As Long 'number of parameters to estimate
NumParameters = UBound(Pest) 'which must equal UBound(Psd)
ReDim Ptest(1 To NumParameters)

ObjFnMin = CalcSSN(X, Y, Pest)
'Main iteration loop
For i = 1 To NumIterations
    ObjFn = ObjFnMin
    For n = 1 To NumRandomParameterSets 'should be about 10^NumParameters
        'randomly choose next set of parameters
        For j = 1 To NumParameters
            gwn = -6#
            For k = 1 To 12
                gwn = gwn + Rnd()
                Next k
            Ptest(j) = Pest(j) + gwn * PSD(j) 'centers Gaussian PDF around Pest
        Next j
    Next n
    OFn = CalcSSN(X, Y, Ptest)
    If OFn < ObjFn Then
        ObjFn = OFn
        For j = 1 To NumParameters
            Pest(j) = Ptest(j)
        Next j
    End If
    Debug.Print i; ObjFn
    If ObjFn < ObjFnMin Then
        ObjFnMin = ObjFn
        For j = 1 To NumParameters
            PSD(j) = 0.5 * PSD(j) 'reduce SD by 10%
        Next j
    End If
Next i
StochasticMinN = Pest
End Function

Option Explicit
ByVal value As Long)
Private Declare Function RTsint Lib "MirApi.dll" (ByVal addr As Long) As Long
Private Declare Sub RTsout Lib "MirApi.dll" (ByVal addr As Long, ByVal value As Long)
Private Declare Function RTusint Lib "MirApi.dll" (ByVal addr As Long) As Long
Private Declare Sub RTusout Lib "MirApi.dll" (ByVal addr As Long, ByVal value As Long)
Private Declare Function WinTimer Lib "MirApi.dll" (ByVal Inc As Double) As Double
Private Declare Function WinSecond Lib "MirApi.dll" () As Double
Private Declare Function Timer Lib "MirApi.dll" (ByVal dt As Double) As Double
Private Declare Sub RTSetPriority Lib "MirApi.dll" (ByVal Priority As Long)
Dim ChannelPolarity() As Boolean 'Record of channel polarities.
Dim Initialized As Boolean 'Allios initialization flag.
Public Sub Init(ChanBipolar() As Boolean)'Must be called first.
    RTOpen
    ChannelPolarity = ChanBipolar
    Initialized = True
End Sub
Public Sub shut() 'Always call when finished with Allios.
    RTClose
    Initialized = False
End Sub
Public Function VoltsIn(Channel As Long) As Double 'Returns voltage as applied to specified A/D channel.
    If Bipolar(Channel) Then
        VoltsIn = BiADVolts(RTsint(Channel))
    Else
        VoltsIn = UniADVolts(RTsint(Channel))
    End If
End Function
Public Sub VoltsOut(Channel As Long, value As Double) 'Applies voltage to specified D/A channel.
    Call RTout(Channel, DABits(value))
End Sub
Function UniADVolts(Bits As Long) As Double 'Converts bits to volts on a unipolar A/D channel.
    UniADVolts = CDbl(Bits + 2 ^ 15) * 4.5 / (2 ^ 16 - 1)
End Function
Function UniADBits(Volts As Double) As Long 'Converts volts to bits on a unipolar A/D channel.
    UniADBits = (Volts) / 4.5 * (2 ^ 16 - 1) - 2 ^ 15
End Function
Function BiADVolts(Bits As Long) As Double 'Converts bits to volts on a bipolar A/D channel.
    BiADVolts = CDbl(Bits) * 4.5 / (2 ^ 15 - 1)
End Function
Function BiADBits(Volts As Double) As Long 'Converts volts to bits on a bipolar A/D channel.
    BiADBits = Volts / 4.5 * (2 ^ 15 - 1)
End Function
Function DAVolts(Bits As Long) As Double 'Converts bits to volts for D/A channel.
    DAVolts = CDbl(Bits) * 3# / (2 ^ 17 - 1)
End Function
Function DABits(Volts As Double) As Long 'Converts volts to bits for D/A channel.
    DABits = (Volts / 3) * (2 ^ 17 - 1)
End Function
Public Function bipolar(Channel As Long) As Boolean 'Returns polarity on a specified channel.
    Bipolar = ChannelPolarity(Channel)
End Function
Private Sub Class_Initialize() 'Always call when finished with Allios.
    Initialized = False
End Sub
Public Function Pause(Optional dt As Double = 0#) As Double 'Mid-level implementation of timer function.
    Pause = Timer(dt)
End Function
Public Function DigIn() As Integer 'DigIn Outputs: Digital bit pattern in binary.
    DigIn = RTsint(19)
End Function
Public Sub DigOut(iPattern As Long) 'DigOut Inputs: Digital bit pattern in binary.
    Call RTout(18, iPattern)
End Sub
Option Explicit

'CZE1000.cls
'Visual Basic Class Module interface for a Spellman power
supply.
'Written by Matthew R. Graham
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.
'Implements Allios interface.
'To do:
'Implement DataPort

Public Enum Polarity
  Negative = 0
  Positive = 3
End Enum

Dim Alll As Allios
'D/Local Allios reference.
Dim VCmdChan As Long
'D/A Voltage command D/A channel.
Dim CCmdChan As Long
'D/A Current command D/A channel.
Dim VMeasChan As Long
'D/A Voltage measure A/D channel.
Dim CMeasChan As Long
'D/A Current measure A/D channel.
Dim PolChan As Long
'D/A Polarity command D/A channel.

Public Sub Init(newAllios, 
  VOutChan As Long, 
  COutChan As Long, 
  VInChan As Long, 
  CInChan As Long, 
  PolarityChan As Long)
  'Must be called first.
  Set Alll = newAllios
  VCmdChan = VOutChan
  CCmdChan = COutChan
  VMeasChan = VInChan
  CMeasChan = CInChan
  PolChan = PolarityChan
End Sub

Private Sub ClassInitialize()
  'Initializes basic parameters.
  Set Alll = New Allios
End Sub

Public Sub SetPolarity(p As Polarity)
  'Set HV polarity.
  Call Alll.VoltsOut(PolChan, CDbl(p))
End Sub

Public Sub nextVal(CmdVolt As Double, 
  CmdCurrent As Double, 
  RcdVolt As Double, 
  RcdCurrent As Double)

End Sub

Public Sub zero()
  'Sets voltage and current to zero - for safety.
  Call Alll.VoltsOut(VCmdChan, 0)
  Call Alll.VoltsOut(CCmdChan, 0)
End Sub

Public Sub shut()
  'Should call this when done using CZE1000.
  Set zero
End Sub

Public Function GetVoltage() As Double
  'Returns current HV value.
  GetVoltage = Alll.VoltsIn(VMeasChan) * 12
End Function

Public Function GetCurrent() As Double
  'Returns current current value.
  GetCurrent = Alll.VoltsIn(CMeasChan) * 120
End Function

Option Explicit

'DataAlert.cls
'Written by Matthew R. Graham.
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.
'Visual Basic Class module which implements a generic alert as
specified at runtime.
'To Do:
'Implement other Alerts.
'Implement argument passing to Alerts.

Dim Alert As String

Public Sub SetAlert(WhichAlert As String)
  'Choose which alert to use.
  Alert = WhichAlert
End Sub

Public Function GetAlert() As String
  'Return current alert.
  GetAlert = Alert
End Function
Option Explicit

' DataPort.cls
' Visual Basic Class Module top-level interface for A/D use.
' Written by Matthew R. Graham
' Last Modified on 15 April 1999.
' Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.
' Implements Allos interface.

Dim Alll As Allos 'Local Allos reference.
Dim Chan As Long 'DataPort channel.
Dim Mult As Long 'Value multiplier.

Public Sub Init(newAllios As Allos, _,
    Channel As Long, _,
    Optional Multiplier As Long = 1)
    'Must call this first.
    Set Alll = newAllios
    Chan = Channel
    Mult = Multiplier
End Sub

Private Sub Class_Initialize()
    'Initializing basic parameters.
    Set Alll = New Allos
End Sub

Public Sub nextVal(value As Double)
    'Getting current A/D value.
    value = Alll.VoltsIn(Chan) * Mult
End Sub

Option Explicit

'DataWatcher.cls
' Written by Matthew R. Graham.
Private VelMin As Double 'Default minimum velocity value.
Private VelDef As Double 'Default default velocity value.
Private VelMax As Double 'Default maximum velocity value.
Private Velocity As Double 'Current velocity value.
Private AccelMin As Double 'Default minimum acceleration value.
Private AccelDef As Double 'Default default acceleration value.
Private AccelMax As Double 'Default maximum acceleration value.
Private Acceleration As Double 'Default acceleration value.

Private Sub Class_Initialize()
'Initial values for basic parameters.
'User should not change.
StepMin = -1000000
StepDef = 0
StepMax = 1000000
VelMin = 0#
VelDef = 1#
VelMax = 50#
AccelMin = 0#
AccelDef = -1#
AccelMax = 1000#
Initialized = False
Prefix = ""
Suffix = Chr$(13)
End Sub

Public Sub Init(NewComm As MSComm, _
Optional IsDaisyChain As Boolean = False, _
Optional NewAddress As Integer = 0, _
Optional NewStep As Long = -1, _
Optional NewVel As Double = -1#, _
Optional NewAccel As Double = -1#)
'Must call this first.
If NewStep < 0 Then
NewStep = StepDef
End If
If NewVel < 0 Then
NewVel = VelDef
End If
If NewAccel < 0 Then
NewAccel = AccelDef
End If
Set Comm = NewComm
DaisyChain = IsDaisyChain
If DaisyChain Then
Prefix = CStr(address) & "_
Else
Prefix = ""
End If

Static Factor As Double
Dim Val As Double

Select Case Action
Case Initialize
Norm = CDbl(Args(0)(0))
Factor = CDbl(Args(0)(1))
LevelWatcher = False
End Function
Case Operate
Val = CDbl(Args(0))
If (Val > Factor * Norm) Then
LevelWatcher = True
Else
LevelWatcher = False
End If
End Function
Case Else
MsgBox "Error in LevelWatcher", _
"Invalid Watcher Action"
LevelWatcher = True
End Function
End Select
End Function

Option Explicit

'Motor.cls
'Visual Basic Class Module interface for Parker Zeta motor drivers.
'Written by Matthew R. Graham
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.
'Implements MSComm interface.

'To do:
Private address As Integer 'Used for daisy-chaining.
Private DaisyChain As Boolean 'For multiple drives on same COM port.
Private Comm As MSComm 'Local reference to COM port.
Private Initialized As Boolean 'Initialization flag.
Private Prefix As String 'ASCII characters required to address driver.
Private Suffix As String 'End of command characters.
Private StepMin As Long 'Default minimum step value.
Private StepDef As Long 'Default default step value.
Private StepMax As Long 'Default maximum step value.
Private Step As Long 'Current step value.
End If
If (Not Comm.PortOpen) Then
Comm.PortOpen = True
End If
Step = NewStep
Velocity = NewVel
Acceleration = NewAccel
'These commands set up the driver for "standard" operation.
Command ("HOMLVL1")
Command ("LHLVL1")
Command ("ENCI")
Command ("ERES5000")
Command ("DRIVE1")
Initialized = True
End Sub

Public Sub shut()
'Disengages stepper motor.
Command ("DRIVE0")
End Sub

Public Sub setAddress(NewAddress As Integer)
'Set motor address.
address = NewAddress
End Sub

Public Function getAddress() As Integer
'Return motor address.
getAddress = address
End Function

Private Function inStepRange(thisStep As Long) As Boolean
'Is this value in the allowable range.
inStepRange = ((thisStep >= StepMin) And (thisStep <= StepMax))
End Function

Private Function inVelRange(thisVel As Long) As Boolean
'Is this value in the allowable range.
inVelRange = ((thisVel >= VelMin) And (thisVel <= VelMax))
End Function

Private Function inAccelRange(thisAccel As Long) As Boolean
'Is this value in the allowable range.
inAccelRange = ((thisAccel >= AccelMin) And (thisAccel <= AccelMax))
End Function

Public Sub setStep(thisStep As Long)
'Set the current step value.
If Initialized Then
If inStepRange(thisStep) Then
Step = thisStep
Else
Step = StepDef
Else
MsgBox "Motor not initialized!", , "Virgin Motor"
End If
End Sub

Public Sub setVel(thisVel As Double)
'Set the current velocity value.
If Initialized Then
If inVelRange(CDbl(thisVel)) Then
Velocity = thisVel
Else
Velocity = VelDef
End If
Command ("V" & CStr(Velocity))
Else
MsgBox "Motor not initialized!", , "Virgin Motor"
End If
End Sub

Public Sub setAccel(thisAccel As Double)
'Set the current acceleration value.
If Initialized Then
If inAccelRange(CDbl(thisAccel)) Then
Acceleration = thisAccel
Else
thisAccel = AccelDef
End If
Command ("A" & CStr(Acceleration))
Else
MsgBox "Motor not initialized!", , "Virgin Motor"
End If
End Sub

Public Sub GO()
'Move the motor.
If Initialized Then
Command ("GO")
Else
MsgBox "Motor not initialized!", , "Virgin Motor"
End If
End Sub

Public Sub Home()
'Send the motor to the home position.
If Initialized Then
Command ("HOM0")
Else
MsgBox "Motor not initialized!", , "Virgin Motor"
End If
End Sub

74
Public Sub Command(CmdName As String)
    'Build and send a standard command.
    Comm.Output = Prefix & CmdName & Suffix
End Sub

Public Sub FlushInput()
    'Clear input buffer.
    'Important to do this occasionally during long runs.
    Dim junk
    junk = Comm.Input
End Sub

Public Function getStep()
    'Returns latest step value.
    getStep = Step
End Function

Public Function getVel()
    'Returns latest velocity value.
    getVel = Velocity
End Function

Public Function getAccel()
    'Returns latest acceleration value.
    getAccel = Acceleration
End Function

Option Explicit

'Switch.cls
'Visual Basic Class Module interface for bistable solenoid switch.
'Written by Matthew R. Graham
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.
'Implements Allios interface.
'To do:
'Move control to digital ports of Allios board.

Enum SwitchState
    Pushed = 0
    Pulled = 1
End Enum

Private chanPolarity As Long 'Polarity control channel.
Private chanControl As Long 'Activation control channel.
Private PushVolt As Double 'Voltage required to "push" switch.
Private PullVolt As Double 'Voltage required to "pull" switch.
Private OnVolt As Double 'Voltage required to activate switch.
Private OffVolt As Double 'Voltage required to deactivate switch.
Private PauseTime As Double 'Imposed bandwidth limit for switch.
Private State As SwitchState 'Current switch state flag.
Private Alll As Allios 'Implementation of Allios interface.

Public Sub Init(newAllios As Allios, _
    PolarityChannel As Long, _
    ControlChannel As Long, _
    firstState As SwitchState)
    'Must be called first.
    Set Alll = newAllios
    chanPolarity = PolarityChannel
    chanControl = ControlChannel
    State = firstState
    If (State = Pulled) Then
        Pull
    Else
        Push
    End If
End Sub

Public Sub Push()
    'All commands required to "push" switch.
    Call Alll.VoltsOut(chanPolarity, PushVolt)
    Call Alll.VoltsOut(chanControl, OnVolt)
    Call Alll.Pause(PauseTime)
    Call Alll.VoltsOut(chanControl, OffVolt)
    State = Pushed
End Sub

Public Sub Pull()
    'All commands required to "pull" switch.
    Call Alll.VoltsOut(chanPolarity, PullVolt)
    Call Alll.VoltsOut(chanControl, OnVolt)
    Call Alll.Pause(PauseTime)
    Call Alll.VoltsOut(chanControl, OffVolt)
    State = Pulled
End Sub

Private Sub Class Initialize()
    'Sets initial values for basic parameters.
    'User should get no control of these.
    PushVolt = 3#
    PullVolt = 0#
    OnVolt = 0#
End Sub
OffVolt = 3#
PauseTime = 0.1
End Sub

Public Function getState() As SwitchState
'Returns current state.
gateState = State
End Function

Option Explicit

'Valve.cls
'Visual Basic Class Module interface for solenoid valve.
'Written by Matthew R. Graham
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.
'Implements Allios interface.
'Uses single D/A channel to control valve behavior.
'To do:
'Switch control to digital ports on Allios board.

Enum ValveState
Opened = 0
Closed = 1
End Enum

Private Channel As Long
Private OpenVolt As Double
Private CloseVolt As Double
Private State As ValveState
Private Alll As Allios
'Control channel
'Required voltage to open valve.
'Required voltage to close valve.
'Current valve state.
'Implementation of Allios interface.

Public Sub Init(newAllios As Allios, 
    newChannel As Long, firstState As ValveState)
'Must call this first.
Set Alll = newAllios
Channel = newChannel
State = firstState
If (State = Closed) Then
    CloseValve
Else
    OpenValve
End If
End Sub

Public Sub OpenValve()
'Opens valve.
Call Alll.VoltsOut(Channel, OpenVolt)

Public Sub CloseValve()
'Closes valve.
Call Alll.VoltsOut(Channel, CloseVolt)
State = Closed
End Sub

Private Sub Class_Initialize()
'Sets initial values for basic parameters.
'User should not change these.
OpenVolt = 3#
CloseVolt = 0#
End Sub

Public Function getState() As ValveState
'Returns current state.
getState = State
End Function

Option Explicit

'NiHP34901.cls
'Visual Basic Class Module for controlling an HP34901 through a gpib interface.
'Written by Peter Madden.
'Modified and converted to class module by Matthew R. Graham.
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham and Bioinstrumentation Lab.
'To do:
'Complete SetupChannels with all options implemented.

Public Enum ModeType
voltage
Temperature
current
End Enum

Dim Slot As Long
Dim Device As NiHP34970

Private Enabled(l To 22) As Boolean
Private Mode(l To 22) As ModeType
Private Gain(l To 22) As Long
Private Names(l To 22) As String * 30
Private Units(l To 22) As String * 30

Public Sub Enable(Chan As Long)
'Sets enabled flag.
Enabled(Chan) = True
End Sub

Public Sub Disable(Chan As Long)
' Sets disabled flag.
   Enabled(Chan) = False
End Sub

Public Sub SetGain(Chan As Long, Val As Long)
' Sets gain.
   Gain(Chan) = Val
End Sub

Public Sub SetMode(Chan As Long, Val As ModeType)
' Sets mode.
   Mode(Chan) = Val
End Sub

Public Sub SetName(Chan As Long, Val As String)
' Sets name.
   Names(Chan) = Val
End Sub

Public Sub SetUnits(Chan As Long, Val As String)
' Sets units.
   Units(Chan) = Val
End Sub

Public Function IsEnabled(Chan As Long) As Boolean
' Checks enabled flag.
   IsEnabled = Enabled(Chan)
End Function

Public Function GetName(Chan As Long) As String
' Returns name.
   GetName = Names(Chan)
End Function

Public Function GetUnits(Chan As Long) As String
' Returns units.
   GetUnits = Units(Chan)
End Function

Public Function SetSlot(SlotNum As Long) As Long
' Sets slot number.
   If (SlotNum >= 1) And (SlotNum <= 3) Then
      Slot = SlotNum * 100
      SetSlot = 0
   Else
      SetSlot = -1
   End If
End Function

Public Function SetSlotAuto() As Long
' Sets slot choice to auto.
   Dim i As Long
   Dim tmpStr As String
   For i = 1 To 3
      tmpStr = Device.GetCardType(i * 100)
      If (InStr(tmpStr, "34901") > 0) Then
         SetSlot i
         SetSlotAuto = i
         Exit Function
      End If
   Next i
   SetSlotAuto = -1
   Exit Function
End Function

Public Function GetVal(Chan As Long) As Double
' Returns value from channel.
   On Error GoTo ErrorHandler
   Select Case Mode(Chan)
      Case ModeType.Temperature
         If (Chan >= 1 And Chan <= 20) Then
            GetVal = Device.GetTemperature(Slot + Chan)
         Else
            Call ErrHand("Channel out of range for temperature reading on HP34901.")
            GetVal = -9999999#
         End If
      Exit Function
      Case ModeType.voltage
         If (Chan >= 1 And Chan <= 20) Then
            GetVal = Device.GetVoltage(Slot + Chan)
         Else
            Call ErrHand("Channel out of range for voltage reading on HP34901.")
            GetVal = -9999999#
         End If
      Exit Function
      Case ModeType.current
         If (Chan >= 21 And Chan <= 22) Then
            GetVal = Device.GetVoltage(Slot + Chan)
         Else
            Call ErrHand("Channel out of range for current reading on HP34901.")
            GetVal = -9999999#
         End If
      Exit Function
   End Select
ErrorHandler:
   Call ErrHand("Error in GetVal of HP34901.")
   GetVal = -9999999#
   Exit Function
End Function

Public Function GetTempSeries() As String
Shortcut to get data from all 20 channels.

Using E-type thermocouples.


Public Function SetupChannels() As Double
    'Quick fix to set all the parameters for temperature reading.
    Dim i As Long
    For i = 1 To 20
        SetMode i, Temperature
    Next i
End Function

Private Sub ErrHand(errMsg As String)
    MsgBox errMsg, vbOKOnly, "HP34901"
End Sub

Public Function OpenDev(Dev As NiHP34970) As Long
    'Opens and initializes device.
    Dim brd As Long
    brd = GetBoardAddress(devname)
    If brd = -1 Then
        Call ErrHand("Illegal device name in call to OpenDev.")
        OpenDev = -1
        Pause
        Exit Function
    End If
    On Error GoTo ErrorHandler
    brd = GetBoardAddress(devname)
    If (brd = -1) Then
        Call ErrHand("Error opening gpib connection.")
        OpenDev = -1
        Pause
        Exit Function
    End If
    OpenDev = ID
    Pause
    Exit Function
ErrorHandler:
    ErrHand "OpenDev error."
    If OpenDev >= 0 Then
        OpenDev = -1
        Pause
        Exit Function
    End If
    CloseDev = -1
    Pause
    Exit Function
End Function

Public Function CloseDev() As Long
    'Close down device.
    On Error GoTo ErrorHandler
    If (ID >= 0) Then
        Call ibonl(ID, 0)
        ID = -1
        CloseDev = 0
    Else
        Call ErrHand("Invalid ID in CloseDev.")
        CloseDev = -1
        Pause
        Exit Function
    End If
ErrorHandler:
    ErrHand "CloseDev error."
    CloseDev = -1
    Pause
    Exit Function
End Function

Public Function ClearDev() As Long
    'Clears error from device.
    ClearDev = Send("*CLS")
End Function

Public Function ResetDev() As Long
    'Resets device.
    ResetDev = Send("*RST")
End Function

Public Function SetDisplay(strDisp As String) As Long
'Set display to specified text.
SetDisplay = Send("DISP:TEXT " & strDisp & ") 
End Function

Public Function GetCardType(Slot As Long) As String
'Return card type as string.
GetCardType = Receive("SYST:CTYPE? " & CStr(Slot))
End Function

Public Function GetIDN() As String
'Return identification number.
GetIDN = Receive("IDN?"
End Function

Public Function GetVoltage(Location As Long) As Double
'Read voltage from specified channel.
GetVoltage = CDbl(Receive("MEAS:VOLT? (@" & CStr(Location) & ")"))
End Function

Public Function GetCurrent(Location As Long) As Double
'Read current from specified channel.
GetCurrent = CDbl(Receive("MEAS:CURR? (@" & CStr(Location) & ")"))
End Function

Public Function GetTemperature(Location As Long) As Double
'Return temperature from specified channel.
'Uses type-E thermocouple.
GetTemperature = CDbl(Receive("MEAS:TEMP? TC, E, 1, DEF, (@" & CStr(Location) & ")"))
End Function

Private Function GetBoardAddress(devname As String) As Long
'Returns board address.
If CharIsaNum(Right$(devname, 1)) Then
    If CharIsaNum(Left$(Right$(devname, 2), 1)) Then
        GetBoardAddress = CLng(Right$(devname, 2))
    Else
        GetBoardAddress = CLng(Right$(devname, 1))
    End If
Else
    GetBoardAddress = -1
End If
End Function

Private Function CharIsaNum(C As String) As Boolean
If Len(C) <= 1 Then
    CharIsaNum = False
Else
    CharIsaNum = (Left$(C, 1) >= "0") And (Left$(C, 1) <= "9")
End Function

Private Sub ErrHand(errMsg As String)
MsgBox errMsg, vbOKOnly, "HP34970, Device#: " & CStr(ID)
End Sub

Public Function Send Estr(Str As String) As String
'Generic send command.
On Error GoTo ErrorHandler
Call iwr(ID, Str)
If (ibsta And EERR) Then
    ErrHand("Error sending " & Str)
    Send = -1
Else
    Send = 0
End If
Pause Exit Function
ErrorHandler:
ErrHand("Error sending " & Str)
Send = -1
Pause Exit Function
End Function

Public Function Receive Estr(Str As String) As String
'Generic receive command.
Dim strRead As String
On Error GoTo ErrorHandler
If (Send Estr(Str) < 0) Then
    Receive Estr = "-9999999"
Else
    Call ibrd(ID, strRead)
    If (ibsta And EERR) Then
        Call ErrHand("Error getting reading with " & Str)
        Receive Estr = "-9999999"
    Else
        Receive Estr = strRead
    End If
End If
Pause Exit Function
ErrorHandler:
ErrHand("Error receiving with " & Str)
Receive Estr = "-9999999"
Pause Exit Function
End Function
Private Sub Pause()
    'Pause required for assured GPIB communications.
    Dim Start As Double
    Do While Timer < Start + 0.1
        Loop
End Sub

Option Explicit

'NHPE3631.cls
'Visual Basic Class Module for controlling an HPE3631 through a gpib interface.
'Written by Peter Madden.
'Modified and converted to class module by Matthew R. Graham.
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham, Peter Madden, and Bioinstrumentation Lab.
'To do:

Private Const P6VMaxV = 6.18
Private Const P25VMaxV = 25.75
Private Const N25VMinV = -25.75
Private Const P6VMaxI = 5.15
Private Const P25VMaxI = 1.03
Private Const N25VMaxI = 1.03

Private ID As Integer

Public Function OpenDev(Optional devname As String = "gpib0", _
    Optional address As Long = 1) As Long
    'Open and initialize device.
    Dim brd As Long
    brd = GetBoardAddress(devname)
    If brd = -1 Then
        Call ErrHand("Illegal device name in call to OpenDev.")
        OpenDev = -1
        Pause
        Exit Function
    End If

    On Error GoTo ErrorHandler
    Call ibdev(brd, address, 0, Tls, 1, 0, ID)
    If (ID < 0) Then
        Call ErrHand("Error opening gpib connection.")
    End If
    OpenDev = ID
    Pause
    Exit Function
End Function

ErrorHandler:
    Call ErrHand "OpenDev error."
    If OpenDev >= 0 Then
        OpenDev = -1
    End If
    Pause
    Exit Function
End Function

Public Function CloseDev() As Long
    'Close device.
    On Error GoTo ErrorHandler
    If (ID >= 0) Then
        Call ibonl(ID, 0)
        ID = -1
        CloseDev = 0
    Else
        Call ErrHand("Invalid ID in CloseDev.")
        CloseDev = -1
    End If
    Pause
    Exit Function
End Function

ErrorHandler:
    Call ErrHand "CloseDev error."
    CloseDev = -1
    Exit Function
End Function

Public Function ClearDev() As Long
    'Clear device errors.
    ClearDev = Send("*CLS")
End Function

Public Function SetDisplay(strDisp As String) As Long
    'Set display to specified text.
    SetDisplay = Send("DISP:TEXT ", strDisp)
End Function

Public Function SetP6V(voltage As Double) As Long
    'Set voltage on +6 volt terminal.
    If (voltage >= 0# And voltage <= P6VMaxV) Then
        If (Send("INST P6V") >= 0) Then
            SetP6V = Send("VOLT ", CStr(voltage))
        Else
            SetP6V = -1
        End If
    Else
        Call ErrHand("Voltage out of range.")
        SetP6V = -1
    End If
End Function

80
Public Function SetP25V(voltage As Double) As Long
'Set voltage on +25 volt terminal.
If (voltage >= 0# And voltage <= P25VMaxV) Then
    If (Send("INST P25V") >= 0) Then
        SetP25V = Send("VOLT " & CStr(voltage))
    Else
        SetP25V = -1
    End If
Else
    Call ErrHand("Voltage out of range.")
    SetP25V = -1
End If
End Function

Public Function SetN25V(voltage As Double) As Long
'Set voltage on -25 volt terminal.
If (voltage >= N25VMinV And voltage <= 0#) Then
    If (Send("INST N25V") >= 0) Then
        SetN25V = Send("VOLT " & CStr(voltage))
    Else
        SetN25V = -1
    End If
Else
    Call ErrHand("Voltage out of range.")
    SetN25V = -1
End If
End Function

Public Function GetP6V() As Double
'Return voltage on +6 volt terminal.
If (Send("INST P6V") >= 0) Then
    GetP6V = CDbl(Receive("VOLT"))
Else
    GetP6V = -9999999#
End If
End Function

Public Function GetP25V() As Double
'Return voltage on +25 volt terminal.
If (Send("INST P25V") >= 0) Then
    GetP25V = CDbl(Receive("VOLT"))
Else
    GetP25V = -9999999#
End If
End Function

Public Function GetN25V() As Double
'Return voltage on -25 volt terminal.
If (Send("INST N25V") >= 0) Then
    GetN25V = CDbl(Receive("VOLT"))
Else
    GetN25V = -9999999#
End If
End Function

Public Function SetP6C(current As Double) As Long
'Set current limit on +6 volt terminal.
If (current >= 0# And current <= P6VMaxI) Then
    If (Send("INST P6V") >= 0) Then
        SetP6C = Send("CURR " & CStr(current))
    Else
        SetP6C = -1
    End If
Else
    Call ErrHand("Current out of range.")
    SetP6C = -1
End If
End Function

Public Function SetP25C(current As Double) As Long
'Set current limit on +25V terminal.
If (current >= 0# And current <= P25VMaxI) Then
    If (Send("INST P25V") >= 0) Then
        SetP25C = Send("CURR " & CStr(current))
    Else
        SetP25C = -1
    End If
Else
    Call ErrHand("Current out of range.")
    SetP25C = -1
End If
End Function

Public Function SetN25C(current As Double) As Long
'Set current limit on -25V terminal.
If (current >= 0# And current <= N25VMaxI) Then
    If (Send("INST N25V") >= 0) Then
        SetN25C = Send("CURR " & CStr(current))
    Else
        SetN25C = -1
    End If
Else
    Call ErrHand("Current out of range.")
    SetN25C = -1
End If
End Function

Public Function GetP6C() As Double
'Return current on +6 volt terminal.
If (Send("INST P6V") >= 0) Then
    GetP6C = CDbl(Receive("CURR"))
Else
    GetP6C = -9999999#
End If
End Function

Public Function GetP25C() As Double
'Return current on +25 volt terminal.
If (Send("INST P25V") >= 0) Then
    GetP25C = CDbl(Receive("CURR"))
Else
    GetP25C = -9999999#
End If
End Function
'Return current on +25 volt terminal.
If (Send("INST P25V") >= 0) Then
    GetP25C = CDbl(Receive("CURR"))
Else
    GetP25C = -9999999#
End If
End Function

Public Function GetN25C() As Double
'Return current on -25 volt terminal.
If (Send("INST N25V") >= 0) Then
    GetN25C = CDbl(Receive("CURR"))
Else
    GetN25C = -9999999#
End If
End Function

Public Function OutputOn() As Long
'Turn on outputs.
    OutputOn = Send("OUTPUT ON")
End Function

Public Function OutputOff() As Long
'Turn off outputs.
    OutputOff = Send("OUTPUT OFF")
End Function

Private Function GetBoardAddress(devname As String) As Long
'Return board address.
    If CharIsaNum(Right$(devname, 1)) Then
        If CharIsaNum(Left$(Right$(devname, 2), 1)) Then
            GetBoardAddress = CLng(Right$(devname, 2))
        Else
            GetBoardAddress = CLng(Right$(devname, 1))
        End If
    Else
        GetBoardAddress = -1
    End If
End Function

Private Function CharIsaNum(C As String) As Boolean
'If len(C) <= 1 Then
    CharIsaNum = False
Else
    CharIsaNum = (Left$(C, 1) >= "0") And (Left$(C, 1) <= "9")
End If
End Function

Private Sub ErrHand(errMsg As String)
    MsgBox errMsg, vbOKOnly, "HPE3631, Device#: " & CStr(ID)
End Sub

Public Function Send(Str As String) As Long
'Generic send command.
On Error GoTo ErrorHandler
    Call ibwrt(ID, Str)
    If (ibsta And EERR) Then
        ErrHand("Error sending " & Str)
        Send = -1
    Else
        Send = 0
    End If
    Pause
Exit Function
End Function

ErrorHandler:
    ErrHand("Error sending " & Str)
    Send = -1
    Pause
    Exit Function
End Function

Public Function Receive(Str As String) As String
'Generic receive command.
    Dim strRead As String
    On Error GoTo ErrorHandler
    If (Send(Str) < 0) Then
        Receive = "-9999999"
    Else
        Call ibrd(ID, strRead)
        If (ibsta And EERR) Then
            Call ErrHand("Error getting reading with " & Str)
            Receive = "-9999999"
        Else
            Receive = strRead
        End If
    End If
    Pause
Exit Function
End Function

Private Sub Pause()
'Pause required for assured GPIB communications.
    Dim Start As Double
    Start = Timer
    Do While Timer < Start + 0.1
        Loop
End Sub
Option Explicit

'NiHPE3632.cls
'Visual Basic Class Module for controlling an HPE3632 through a gpib interface.
'Written by Peter Madden.
'Modified and converted to class module by Matthew R. Graham.
'Last Modified on 15 April 1999.
'Copyright 2000, Matthew R. Graham, Peter Madden, and Bioinstrumentation Lab.

'To do:
Private Const MaxV = 30.9
Private Const MaxI = 4.12
Private ID As Integer

Public Function OpenDev(Optional devname As String = "gpib0", Optional address As Long = 1) As Long
'Open and initialize device.
Dim brd As Long
brd = GetBoardAddress(devname)
If brd = -1 Then
    Call ErrHand("Illegal device name in call to OpenDev.")
    OpenDev = -1
    Pause
    Exit Function
End If
On Error GoTo ErrorHandler
Call ibdev(brd, address, 0, Tls, 1, 0, ID)
If (ID < 0) Then
    Call ErrHand("Error opening gpib connection.")
End If
OpenDev = ID
Pause
Exit Function

ErrorHandler:
ErrHand "OpenDev error."
If OpenDev = -1 Then
    CloseDev = 0
Else
    CloseDev = -1
End If
Pause
Exit Function

Public Function CloseDev() As Long
'Close device.
On Error GoTo ErrorHandler
If (ID >= 0) Then
    Call ibonl(ID, 0)
    ID = -1
    CloseDev = 0
Else
    Call ErrHand("Invalid ID in CloseDev.")
    CloseDev = -1
End If
Pause
Exit Function

Public Function ClearDev() As Long
'Clear device of errors.
ClearDev = Send("*CLS")
End Function

Public Function SetDisplay(strDisp As String) As Long
'Set device display to specified text.
SetDisplay = Send("DISP:TEXT " & strDisp & ")
End Function

Public Function SetVoltage(voltage As Double) As Long
'Set voltage.
If (voltage >= 0# And voltage <= MaxV) Then
    SetVoltage = Send("VOLT " & CStr(voltage))
Else
    Call ErrHand("Voltage out of range.")
    SetVoltage = -1
End If
End Function

Public Function GetVoltage() As Double
'Return voltage.
GetVoltage = CDbl(Receive("MEAS:VOLT?"))
End Function

Public Function SetCurrent(current As Double) As Long
'Set current limit.
If (current >= 0# And current <= MaxI) Then
    SetCurrent = Send("CURR " & CStr(current))
Else
    Call ErrHand("Current out of range.")
    SetCurrent = -1
End If
End Function

Public FunctionGetCurrent() As Double
End Function

Public Function GetGpibVersion() As Double
End Function
'Return current.
GetCurrent = CDbl(Receive("MEAS:CURR?"))
End Function

Public Function OutputOn() As Long
'Turn on output.
OutputOn = Send("OUTPUT ON")
End Function

Public Function OutputOff() As Long
'Turn off output.
OutputOff = Send("OUTPUT OFF")
End Function

Private Function GetBoardAddress(devname As String) As Long
'Return board address.
If CharIsaNum(Right$(devname, 1)) Then
  If CharIsaNum(Left$(Right$(devname, 2), 1)) Then
    GetBoardAddress = CLng(Right$(devname, 2))
  Else
    GetBoardAddress = CLng(Right$(devname, 1))
  End If
Else
  GetBoardAddress = -1
End If
End Function

Private Function CharIsaNum(C As String) As Boolean
If Len(C) <> 1 Then
  CharIsaNum = False
Else
  CharIsaNum = (Left$(C, 1) >= "0") And (Left$(C, 1) <= "9")
End If
End Function

Private Sub ErrHand(errMsg As String)
MsgBox errMsg, vbOKOnly, "HPE3632, Device#: " & CStr(ID)
End Sub

Public Function Send(Str As String) As Long
'Generic send command.
On Error GoTo ErrorHandler
  Call ibwrt(ID, Str)
  If (ibsta And EERR) Then
    Call ErrHand("Error sending " & Str)
    Send = -1
  Else
    Send = 0
  End If
Pause
Exit Function
ErrorHandler:
  ErrHand("Error sending " & Str)
  Send = -1
Pause
Exit Function

Public Function Receive(Str As String) As String
'Generic receive command.
Dim strRead As String * 128
On Error GoTo ErrorHandler
  If (Send(Str) < 0) Then
    Receive = "-9999999"
  Else
    Call ibrd(ID, strRead)
    If (ibsta And EERR) Then
      Call ErrHand("Error getting reading with " & Str)
      Receive = "-9999999"
    Else
      Receive = strRead
    End If
  End If
Pause
Exit Function
ErrorHandler:
  ErrHand("Error receiving with " & Str)
  Receive = "-9999999"
Pause
Exit Function

Private Sub Pause()
'Pause required for assured GPIB communications.
  Dim Start As Double
  Start = Timer
  Do While Timer < Start + 0.1
    Loop
End Sub