Spectral Analysis of Wave Propagation Through a Polymeric Hopkinson Bar

by

Christopher Salisbury

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of

Master of Applied Science

in

Mechanical Engineering

Waterloo, Ontario, Canada, 2001

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The importance of understanding non-metallic material behaviour at high strain rates is becoming ever more important as new materials are being developed and used in shock loading applications. Applying conventional methods for high strain rate testing to non-metallic materials proved ineffective due to impedance mismatch between the specimen and apparatus and so a new test method was developed.

A polymeric Hopkinson bar was developed enabling non-metallic materials, such as polycarbonate and rubber, to be tested at strain rates from $500\,s^{-1}$ to $4000\ s^{-1}$.

Conventional Hopkinson bar analysis is invalid due to the viscoelastic nature of the polymeric bar material. As waves propagate along the bar length, the inherent material behaviour causes the waves to undergo a degree of attenuation and dispersion. Through the use of spectral analysis, a comparison of the dispersive relationships for 6061 T-6 aluminium, extruded acrylic and low density polyethylene is presented.

The application of spectral methods to viscoelastic wave analysis was validated by three separate methods. A comparison of predicted and measured waves along the bar length allowed the dispersive relationship to be substantiated. The use of an enhanced laser velocity system further verified the predicted wave magnitude. A comparison of results for polycarbonate and ballistic gelatin to published results showed good agreement.

Acknowledgements

I am grateful to Dr. Roy Pick for his patience and guidance in preparation of this dissertation and throughout the duration of my academic studies.

Sincere thanks to Dr. Duane Cronin and Dr. Mike Worswick who kept me from straying from the correct path, enabling me to complete this work. I am also indebted to Mr. Christophe Bacon of the University of Bordeaux without whom I would still be pondering the perplexities of spectral analysis.

To Steve Truttmann, Ernst Huber and Andy Barber I extend my appreciation for their technical support and advice.

I would also like to extend my gratitude to my wife Karen for her continued support throughout my scholastic career. Without her understanding, patience and encouragement none of this would be possible.

Table of Contents

Chapter 4

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The study of material behaviour at high strain rates is becoming ever more important with the desire to fabricate structures that are capable of withstanding high velocity impacts. For computer simulation of such impacts, the need for accurate properties for a variety of materials is required. As the design of such structures becomes more complex to withstand higher energy ballistic and blast impacts, knowledge of the behaviour of a greater variety of materials with vastly different properties is required. The object of the research undertaken by the author was to develop a test fixture that was capable of testing soft materials such as bone and rubber at high strain rates to provide material models for the simulation of both personnel protection devices and the parts of the human body being protected. Experimental determination of the high strain rate constitutive behaviour of soft materials is particularly challenging because there has been little development of experiments for such low impedance materials. The following sections describe, in general, high strain rate experiments developed for high impedance materials such as metals and their extension to low impedance materials.

1.2 General Methods for High Strain Rate Testing of Materials

This section deals with testing methods which achieve strain rates from 10 to 10^8 s^{-1}. No single test can achieve the full range of rates and therefore it is convenient to divide the methods into three categories: low strain rate, high strain rate and ultrahigh velocity impacts. Table 1.1 illustrates the different types of tests that exist for various strain rates.

Low strain rates are the easiest to achieve. These rates range from 10^1 to 10^2 s^{-1} and are marginally higher than quasi –static tests. Simple hydraulic or pneumatic actuators can achieve these rates, Meyers [1994]. A different method for achieving these rates is the cam plastomer. Figure 1.1 shows a diagram of a cam plastomer testing machine. This machine uses a cam rotating at a specific velocity to deform the specimen. Once the cam is rotating at the desired velocity, the follower is moved into place and the specimen is deformed.

For high strain rates up to 10^4 s^{-1} three tests are commonly used to determine the mechanical properties of the sample material.

1.2.1 The Taylor Anvil

This method propels a cylindrical projectile into a target which is rigid or symmetric. The process of deformation results from a sequence of elastic and plastic waves propagating in the cylinder. Figure 1.2 shows the sequence of events that occurs when the projectile impacts. From initial and final measurements, the initial velocity of the projectile and the velocity of the target, the material behaviour can be deduced through the application of conservation equations. More recently, computer simulation of the impact allows the selection of an appropriate material model by comparing the deformed shape to the simulation.

1.2.2 Expanding Ring

The expanding ring technique uses a hollow cylinder with an explosive core as a method of initiating a shock wave. Figure 1.3 shows the different components of this system. The material to be tested is placed in a ring around the hollow cylinder. As the explosive is detonated, the shock wave expands radially, which in turn causes the cylinder and specimen to expand. In order to determine the stress strain curve for the material, the velocity history

of the ring is recorded. The velocity history can be related to stress and strain through a set of simple equations. This technique uses varying amounts of explosives to select the strain rate desired.

1.2.3 Split Hopkinson Bar

The most commonly used method for testing material between strain rates of 10^2 and 10^4 s^{-1} is the Hopkinson bar. The Compressive Split Hopkinson Pressure Bar (CSHB) is comprised of three separate bars known as the striker, the incident and the transmitter bar. Figure 1.4 shows a schematic of this apparatus. The striker bar is propelled towards the incident bar at a desired velocity. The striker bar can be accelerated either by mechanical means (such as a spring) or by using compressed gas (with a gas gun). A compressive wave is imparted into the incident bar as the striker impacts. The elastic wave propagates uninterrupted along the incident bar until it reaches the sample. The velocity of the striker controls the strain rate achieved, while the length of the striker determines the duration of the test. Some of the wave in the incident bar is transmitted through the sample into the transmitter bar and the rest of the wave is reflected in the incident bar. In order to determine the strain within the sample, strain gauges are mounted on both the incident and transmitter bars. The strain gauges are usually placed in a location where no superposition between the reflected wave and incident wave will occur over the duration of the test. The wavelength of the pulse is approximately equal to two times the length of the striker. That means that the incident bar must be greater than two times the length of the striker and the gauges must be mounted at least the length of the striker from the specimen end in order to avoid superposition. Lundberg and Henchoz [1977] describe a method using two strain gauge stations per bar allowing separation of superimposed pulses.

1.2.4 Ultrahigh Strain Rate Testing

In order to achieve ultrahigh strain rates a very rapid application of energy at the surface of the sample material is required. This can be accomplished by a variety of methods as indicated in Table 1.1. The easiest way to apply a large amount of energy to the sample surface is to detonate explosives that are in contact with the surface. This would generate the required plastic wave in the sample. Similarly, a pulsed laser of very high power would develop high pressures in the sample through radiation.

Single and dual stage gas guns are capable of launching projectiles towards a sample material at high velocities. Single stage gas guns are capable of propelling a projectile at velocities of up to 1200 m/s while two-stage gas guns are capable of velocities up to 8000 m/s.

Flyer plates can be used in achieving ultrahigh strain rates as well. A plate is driven by an explosive charge or gas gun to impact a stationary plate. The reader is referred to Meyers [1994] for a more detailed discussion of ultrahigh strain rate testing methods.

1.2.5 Current Application

The strain rates desired for the current application are in the range of 10^1 s^{-1} to 10^4 s^{-1} which suggests the use of the high strain rate test techniques. The Taylor impact test requires that the specimen's dimensions be measured after the test is conducted. Although acceptable for metals, this is impractical when testing soft materials. A significant portion of the deformation process can be elastic allowing the sample material to recover in time, leading to incorrect results. In addition, the specimens must be propelled into the rigid mass. Since soft materials are easily deformed, the shock of propelling the material could cause deformation and possibly failure.

Material limitations make the expanding ring technique impractical as well. This method requires that the material be formed into a ring. This is would be impossible for many organic materials such as bone. Additionally, the requirement of an explosive core makes use of such a technique costly and potentially hazardous.

The nature of the Hopkinson bar is ideal for the present task. This type of test fixture is not limited to specific sample configurations. The strain rates that can be achieved by the Hopkinson bar correspond to the desired strain rates since it is unlikely that strain rates above 10^4 s^{-1} will be required. It should be mentioned, however, that strain rates approaching the 10^4 s^{-1} represent the upper limit of Hopkinson bar testing and can only be achieved using a miniature Hopkinson bar apparatus.

1.3 Hopkinson Bar History

The idea of using a pressure bar to determine material properties was first used by John Hopkinson [1872] in 1872. He was able to determine properties of iron wire by transferring the energy of a dropping weight into the wire and measuring how much it deformed before breaking. His son Bertram Hopkinson [1914] continued his work in 1914 by using a bar to determine the pressures developed by the impact of a bullet or from a blast. A schematic of his device is shown in Figure 1.5. He used a bar (B) suspended by two sets of wires that was inline with a box (D) which was also suspended. A short section of rod (C) was placed at the end of the main bar and held in place by a small magnetic force. A bullet was then shot at the end (A) of the long bar causing a pressure wave to be imparted into the rod. The wave travelled down the long rod and into the short rod causing it to fly off and be caught by the box. The displacement of the box and the rod were measured with a simple displacement device, allowing the calculation of momentum. The crude measuring devices available at that time limited the accuracy of the results of the experiments.

Little was done in this area of research until 1948 when Davies [1948] performed a critical study on the Hopkinson pressure bar. Davies used a condenser unit to more accurately measure the displacement of the bar's end. Kolsky [1949] presented a more conventional form of the Hopkinson bar in 1949. Kolsky used a three bar system which comprised of the striker bar, the incident bar and the transmitter bar described earlier. Kolsky used condenser units attached to the incident and transmitter bar to derive the properties of the specimen. This three bar system is sometimes referred to as a Kolsky pressure bar, however, in keeping with the conventions used at the University of Waterloo and elsewhere, the apparatus will be termed the Compressive Split Hopkinson Bar (CSHB) apparatus.

Chapter 2

Conventional Split Hopkinson Bar

2.1 Theory

In the conventional Hopkinson bar, the material behaviour is determined from the difference in interface velocities (V_1, V_2, Figure 1.4). As the elastic pulse deforms the sample, of length L, the distance between the incident and transmitter bars, decrease since $V_1 > V_2$.

This deformation occurs over a period of time that leads to the strain rate being calculated as:

$$\frac{d\varepsilon_s}{dt} = \frac{V_1 - V_2}{L_s} \tag{2.1}$$

The measurement of the velocity at the end of each bar is difficult. Therefore, a different approach using elastic wave propagation in the incident and transmitter bars is often adopted.

The velocity of sound in the material is given as:

$$C_o = \sqrt{\frac{E}{\rho}} \tag{2.2}$$

where C_o is the sound velocity, E is Young's Modulus and ρ is the density of the bar material. Longitudinal waves propagate through elastic media at this speed.

In order to determine the stress, strain and strain rate history of the sample, the incident strain $\varepsilon_I(t)$, the reflected strain $\varepsilon_R(t)$ and the transmitted strain $\varepsilon_T(t)$ can be used. The velocities at the interface can then be related to the strain by:

$$V_1 = C_o \varepsilon_I \text{ at (t=0)}$$
$$V_2 = C_o \varepsilon_T \tag{2.3}$$

At t>0 the incident and reflected waves are superimposed so that the velocity is reduced and V_1 becomes:

$$V_1 = C_o(\varepsilon_I - \varepsilon_R) \tag{2.4}$$

This results in the strain rate becoming:

$$\dot{\varepsilon}_s(t) = \frac{C_o}{L}\left(\varepsilon_I - \varepsilon_R - \varepsilon_T\right) \tag{2.5}$$

if the bars are made from the same material.

The stress in the sample is determined by:

$$\sigma_s = \frac{F_1(t) + F_2(t)}{2A_s} \tag{2.6}$$

where F_1 and F_2 are the forces applied at the specimen faces by the bars and A_s is the area of the sample. The forces in the bar can be related to the strains in the bar by:

$$F_1 = A_b E_b \left(\varepsilon_I + \varepsilon_R\right)$$
$$F_2 = A_b E_b \left(\varepsilon_T\right) \tag{2.7}$$

The stress is then:

$$\sigma_s = \frac{A_b E_b}{2A_s}\left(\varepsilon_I + \varepsilon_R + \varepsilon_T\right) \tag{2.8}$$

where A_b and E_b are respectively the area and Young's Modulus of the bar.

For equilibrium to exist, $F_1=F_2$ and $\varepsilon_{I+}\ \varepsilon_R = \varepsilon_T$ simplifying the equations to:

$$\sigma_s = E_b \frac{A_b}{A_s}\varepsilon_T \tag{2.9}$$

$$\dot{\varepsilon}_s = -2\frac{C_o}{L_s}\varepsilon_R \tag{2.10}$$

$$\varepsilon_s = -2\frac{C_o}{L_s}\int_0^t \varepsilon_R dt \tag{2.11}$$

The so called Hopkinson bar equations are based on certain assumptions as described by Davies [1948] that must be satisfied in order for this analysis to be valid.

- The bars must remain elastic through out the test.
- No attenuation or dispersion occurs.
- The pulse is uniform over the cross section of the bar.
- The specimen remains in equilibrium through out the test.

Since the Hopkinson equations are based on elastic waves equations, the first assumption must be enforced. The second assumption states that no attenuation or dispersion occurs. This property must exist so that the strain measured at the strain gauge location is similar to the actual strain at the interface. If attenuation or dispersion occurs, then the equations that relate the specimen properties to the strain gauge output become invalid. As will be seen later, attenuation and dispersion become a significant problem when developing CSHB equipment for testing soft materials.

The third assumption indicates that for the standard Hopkinson bar analysis to be valid, the waves can be described by the one dimensional wave equation. If the distribution of pressure and displacement across any section of the bar is non-uniform then distortion of the wave can result. It is suggested that the wave is fully developed in four (Davies [1948]) to ten (Follansbee [1985]) bar diameters from the interface.

The fourth assumption refers to sample considerations and is discussed in detail in the following section.

2.2 Sample Considerations

The fourth assumption states that equilibrium must exist in the sample during the test. If equilibrium is not reached then non-homogeneous deformation can occur. To achieve equilibrium the specimen should be sufficiently short to ensure that pressure throughout the specimen is constant. Kolsky [1949] suggests that the specimen length should be small compared to the wavelength of the shortest operative wave in the Fourier spectrum of the pulse. Davies and Hunter [1962] established that equilibrium is achieved when the loading pulse travels back and forth inside the specimen more than π times. Dioh et al. [1993] conducted tests through a range of strain rates to determine the strain rate sensitivity due to material dimensions. They suggest that it is critical to choose appropriate specimen

dimensions when testing at higher strain rates in order to correctly determine the material's behaviour. Dioh et al. [1994] further conclude, through numerical simulation, that at high striker velocities, plastic wave fronts are developed in the sample, increasing the strain rates and flow stress, which causes inaccuracies in representing the material's inherent behaviour. By reducing the specimen length, lower velocities can be used, resulting in lower stress and strain gradients throughout the specimen. Further research was conducted by Dioh et al. [1995] in this area using a different type of numerical simulation resulting in similar conclusions.

Another consideration when attempting to achieve equilibrium in the specimen is lubrication. The interface between the bar ends and sample must be lubricated to allow the sample to expand radially. The use of petroleum jelly has been shown as a good lubricant for testing polymers at high strains; Dioh et al. [1993].

2.3 Hopkinson Bar for Testing Low Impedance Material

One of the requirements for the Hopkinson bar analysis is that a transmitted pulse be measured. This means that the acoustical impedance of the specimen should be similar to that of the bars. The acoustic impedance is given by:

$$I = C_o \rho \tag{2.12}$$

When testing low impedance materials such as foam and rubber, the transmitted pulse is significantly reduced causing the signal to noise ratio to decrease. In order to boost the signal transferred to the transmitter bar, the cross sectional area of the transmitter bar can be reduced, the signal can be measured with a more sensitive measurement technique or the impedance of the bar material can be reduced. Various researchers have made use of these techniques. Chen et al. [1999] used a hollow metallic transmitter bar with an end cap. By reducing the cross-sectional area, the transmitted strain is increased for the same force level. This method avoids any dispersion and attenuation problems that are encountered when viscoelastic bars with lower impedance are used. However, the problem with this method is that the wave mechanics at the interface of the hollow tube with the end cap become complex. There can be a variety of reflections off the various interfaces since the end cap is press fitted into the tube and secured with a pin. The transmitted strain seen at the gauge location may therefore not be indicative of the true strain being transmitted at the interface.

Chen et al. [2000] suggest another method of increasing the signal in the transmitter bar. This method requires that a small quartz crystal be imbedded into the transmitter bar. The quartz crystal gave a much larger signal than surface mounted strain gauges and had a similar impedance to the aluminium bars used. The quartz crystal is approximately 3 times more sensitive than a conventional strain gauge. The similar impedance and shape of the crystal help to prevent any ill effects of placing this gauge in the middle of the bar. Problems with this method occur when the wave hits the crystal causing complex reflections and refractions. These reflections and refractions are inevitable due to discontinuities at the crystal interface.

A method used to reduce the impedance of the bar material is to switch from the conventional steel bars to aluminium bars. Aluminium bars have a reduced modulus and density that makes them applicable for testing stiffer polymers and they do not suffer from viscoelastic effects of non-metallic bars. Most strain gauge technology is applicable to aluminium bars and few changes need to be made to existing experimental techniques. Magnesium bars can be used in a similar fashion although this method will only be applicable to stiffer polymers. Since aluminium and magnesium bars are linearly elastic, there is little dispersion or attenuation and the conventional Hopkinson equations can be used.

Once the material being tested enters into the realm of foams and rubbers, the bars must be made out of lower modulus non-metallic materials such as acrylic.

2.4 Summary

Since weight is an important factor in personnel protection devices, it is expected that new protection systems will make maximum use of low impedance material such as foam. Therefore, to study these materials, the author has selected a CSHB with acrylic bars for the test apparatus.

The conventional means for analysing forces and velocities from strains in a Hopkinson bar apparatus has been presented. The equations used are based on basic wave propagation principles that work only for metallic bars. In order to fully comprehend the dynamics of wave propagation through softer media, spectral methods must be used. Spectral methods, discussed in the next chapter, require the transformation of a wave into the frequency domain where its components can be further analysed. This method of analysis will allow the inclusion of a dispersion and attenuation adjustment that would be impractical in the time domain.

Chapter 3

Spectral Wave Theory

3.1 Introduction

The analysis of how waves disperse and attenuate is generally performed using spectral methods. A detailed explanation of how waves are changed into the frequency domain and how they propagate is given here. This understanding is fundamental when analysing wave propagation through any media.

3.1.1 Fourier Transform and the FFT

The basis of spectral analysis is the idea that a complex periodic wave can be represented by the superposition of a series of sinusoids of harmonically related frequencies, Chapra and Canale [1988]. The general equation for a simple harmonic sinusoid is

$$f(t) = a_o + r_1 \sin(\omega_o t + \theta_1) \tag{3.1}$$

where f is the amplitude, a_o is the offset or mean value, r_1 the amplitude, ω_0 is angular frequency which describes the periodic nature, and θ_1 is the phase angle or shift of the wave. The phase angle describes the amount of shift along the time axis of the wave. By applying the trigonometric identity:

$$r_1 \cos(\omega_o t + \theta_1) = r_1 \left[\cos(\omega_o t)\cos(\theta_1) - \sin(\omega_o t)\sin(\theta_1)\right]$$

to equation (3.1), an alternate form of the general wave can be written as:

$$y(t) = a_o + a_1 \cos(\omega_o t) + b_1 \sin(\omega_o t) \tag{3.2}$$

where

$$a_1 = r_1 \cos(\theta_1) \text{ and } b_1 = -r_1 \sin(\theta_1) \tag{3.3}$$

Therefore, a signal can be represented by a continuous Fourier series written as:

$$f(t) = a_o + \sum_{k=1}^{\infty} [a_k \cos(k\omega_o t) + b_k \sin(k\omega_o t)] \tag{3.4}$$

since

$$\omega_o = \frac{2\pi}{T} \tag{3.5}$$

and is called the base or fundamental frequency and k is an integer. The frequency multiples, $k\omega_o$, are known as harmonics. A function with a period T in the time domain can therefore be related to a spectrum of components (a_k and b_k) in the frequency domain. Figures 3.1a, 3.1b and 3.1c illustrate how a square wave can be broken into a summation of a series of cosine waves. If enough terms are included, then the superposition of the all the components would result in a wave identical to the square wave.

In addition to the amplitude of each of the components of the Fourier series, a corresponding phase angle must also exist. Both the amplitude and the phase spectra are needed to reconstruct the wave in the time domain. For the example of the square wave, the amplitude and phase spectra are shown in Figures 3.2a and 3.2b respectively. By analysing the amplitude and phase spectra a greater insight into the properties of the wave can be achieved.

The above analysis was performed for a periodic or repeating signal. This is, however, impractical for analysing wave propagation since a stress wave is aperiodic. To analysis aperiodic signals, an alternative to the Fourier series was developed. A Fourier transform pair allows the transformation of an aperiodic signal into the frequency domain and back. The basis of the Fourier transform is the Fourier integral and is given by:

$$\tilde{F}(\omega) = \frac{1}{T} \int_{-\infty}^{\infty} f(t) e^{-i\omega_o t} dt \tag{3.6}$$

where $\tilde{F}$ is the continuous Fourier transform (CFT), ω_o is as defined before and i is a complex number $\left(\sqrt{-1}\right)$. The Fourier integral is derived from the Fourier series in its exponential form by applying Euler's identities. The application of infinite limits allow for the definition of an aperiodic signal. In other words, as the period becomes infinite, the

signal never repeats itself, becoming aperiodic. The second part of the Fourier transform pair is the inverse transform and is given by:

$$f(t) = \int_{-\infty}^{\infty} \tilde{F}(\omega) e^{+i\omega_o t} d\omega \tag{3.7}$$

The ~ symbol indicates the frequency domain of any function. $\tilde{F}$ has both real and imaginary components which are related to the a_k and b_k terms of the Fourier series respectively.

For most situations, however, the function f(t) is not known analytically. Normally the signal is known in terms of a discretized signal measured through a data acquisition system. For this situation, the discrete Fourier transform (DFT) was developed. The CFT can be written in terms of samples (n) resulting in the DFT transform pair given by:

$$\tilde{F}_k = \sum_{n=0}^{N-1} f_n e^{-ik\omega_o n} \quad \text{for k=0 to N-1} \tag{3.8}$$

and

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} \tilde{F}_k e^{ik\omega_o n} \quad \text{for n=0 to N-1} \tag{3.9}$$

where N is the number of samples.

Although this is a more practical form of the Fourier transform, it requires N^2 complex operations making it impractical to execute by hand and is still computationally intimidating when using computers. To alleviate some of the computational burden, fast Fourier transform (FFT) routines have been developed. Most FFT routines reduce N^2 operations to Nlog(N) operations allowing for more efficient calculation of the Fourier coefficients. The reader is referred to Press et al. [1992] for a detailed description of FFT routines.

Since $\tilde{F}_k$ has both real and imaginary parts, it can be expressed in rectangular and polar formats. In rectangular form, $\tilde{F}_k$ represents the coefficients of the Fourier series. The polar form of $\tilde{F}_k$ is related to the rectangular form by:

$$\tilde{F}_k = a_k + b_k i = r_k e^{i\theta_k} \tag{3.10}$$

where from equation (3.3):

$$r_k = \sqrt{\left(a_k^{\,2} + b_k^{\,2}\right)} \tag{3.11}$$

$$\theta_k = \tan^{-1}\left(\frac{b_k}{a_k}\right) \tag{3.12}$$

r_k refers to the modulus or amplitude and θ_k is the phase angle. Although most FFT routines return results in rectangular form, a greater understanding of the properties of a wave is attained through the polar form. Some special considerations must be taken into account when changing from rectangular to polar form. When determining the phase angle by the arctan function, one must be careful that the angle has been adjusted for its quadrant. Most mathematical programs assume that the angle lies within the first quadrant and so some adjustment is necessary. The phase angle should lie in the interval $-\pi \le \theta \le \pi$. As well, phase angles must be subjected to a numerical procedure known as unwrapping. In unwrapping the phase spectra, a continuous function is attained by adding or subtracting multiples of 2π when absolute jumps between consecutive phase spectra are greater than π radians (Figure 3.3). This procedure accounts for the phase shift relative to the first component or DC term. The DC term occurs when n=0 and represents the area under the time function.

3.1.2 Wave Propagation in the Frequency Domain

One of the most useful aspects of the Fourier transform is the ability to analyse and predict how waves will propagate. When a wave propagates along a rod, in essence it is being shifted in time. If a simple square wave is symmetric about time zero, as in the case of Figure 3.1, then it can be seen that the imaginary part of the transform is zero and that there is no phase shift. If the wave is displaced along the time axis, the transform will have both real and imaginary parts. The real part is an even function while the imaginary part is an odd function. Figure 3.4 shows these relations for a square pulse using the CFT. In terms of polar coordinates, the amplitude of the original and shifted pulses is identical; the only difference is that of phase. This indicates that a shift in time in the time domain corresponds to a change in phase in the frequency domain. This leads to the following relation:

$$f(t-t_o) = \tilde{F}_n(\omega)e^{-i\omega_n t_o} = r_n e^{(\theta_n - \omega_n t_o)i} \tag{3.13}$$

where t_o is the amount of shift in time.

Of prime importance in wave analysis, is how a wave will propagate through different types of materials. As a wave propagates through some types of media, dispersion and

attenuation of the wave can result. Dispersion relates to the lengthening of a wave as it propagates through a media while attenuation relates to a reduction in amplitude. Dispersion and attenuation are interrelated actions that are generally coupled. In other words, if you have dispersion, you generally have attenuation. Figure 3.5 shows these effects. Attenuation and dispersion can be caused by a variety of factors such as, material properties and geometric constraints.

The ability to separate the components of a wave is key to analysing dispersion and attenuation relations. Figure 3.6 illustrates the components of a wave at different times. The wave train on the left illustrates a non-dispersive system. As the wave propagates, the individual wave components have the same velocity and therefore remain in the same relative position to each other. This means that at any given time the summation of the individual wave components will result in the same wave. For the dispersive system, shown to the right of Figure 3.6, the individual wave trains have different velocities changing their relative positions. This means that as the wave train propagates, the resultant wave will change shape with time. The speed at which each component moves is called the phase speed and is given by:

$$c = \frac{x}{t} = \frac{\omega}{k} \tag{3.14}$$

where c is the phase speed, t is the time, x is the distance and k is known as the wavenumber. By relating the speed of each phase to the frequency a dispersive relationship can be developed (Doyle [1989], Follansbee and Frantz [1983], Gorham and Wu [1996]). The relationship between the wavenumber and the frequency is called the spectrum relationship. The speed at which the superimposed wave moves is called the group speed (c_g). This is the wave that is actually observed.

If the wave at a point is measured, then it can theoretically be predicted at another point by applying a transfer function to the original wave. In other words, if one knows the dispersive relationship, one can predict how a wave will propagate through a material.

3.1.3 Wave Equations

In order to predict how a wave will propagate through some media a model describing the wave motion must be formulated. The development of the Pochhammer [1876] and Chree [1889] frequency equations form the basis for analysing longitudinal wave

propagation. These equations relate the phase velocity to frequency for one-dimensional wave propagation. Follansbee [1985] suggests that a one dimensional analysis is sufficient since the majority of the energy is contained in wavelengths that exceed ten times the radius of the bar. This also means that the surface measurement of strain is a valid indicator of axial displacement. The one dimensional equation of wave motion is:

$$T\frac{\partial^2 u}{\partial x^2} = \rho\frac{\partial^2 u}{\partial t^2} \tag{3.15}$$

where T is the axial tensile force in the material and ρ is the density. Changing into the frequency domain and solving results in:

$$u(x,t) = \tilde{u}(x,\omega) = \sum F_n G(k_{mn}x)e^{i\omega_n t} \tag{3.16}$$

where F_n is the amplitude spectrum and G is the system transfer function Doyle [1989]. The subscript m refers to the mode of the solution and generally, only the first mode is considered, Cheng et al. [1998]. The transfer function determines the amount of phase shift and attenuation with respect to space. As it will be seen later, the transfer function will be related to γ, the propagation coefficient.

Historically, when linearly elastic material is being analysed, dispersion is ignored if the ratio of the wavelength (λ) to the radius (R) is much less than one (Davies [1948]). Follansbee and Frantz [1983] determined that dispersion is an important consideration even when $\lambda/R<<1$ for linearly elastic bars.

Viscoelastic effects create problems when using polymeric bars. Using bars made of these materials requires a greater understanding of the bar's material properties. Attenuation and dispersion have large effects on the incident, reflected and transmitted waves. Simply stated, the problem is that the strain gauge measurement in the middle of the bar does not correspond to the conditions at the sample interface. Therefore, some data reduction is required. A variety of methods have been suggested to combat this problem.

3.2 Corrections for Dispersion and Attenuation

The different methods for correcting dispersion and attenuation fall into either analytical or experimental categories.

3.2.1 Analytical

The theoretical approach to solving this problem requires that a viscoelastic model of the material be formulated. The model is then used to simulate the behaviour of the material so that the wave can be predicted at some point from a known measurement. Kolsky [1963] illustrates the three different models that are commonly used to simulate viscoelastic response. The three models, shown in Figure 3.7, indicate that each of the material models can be represented by a system of dashpot and spring elements. The different configurations of the Voigt, Maxwell and general solid model different types of dynamic behaviour. The Voigt model is based on the assumption that the stress components in a solid are proportional to the sum of the strain and the strain rate. The Maxwell solid, on the other hand, relates the stress rate as being proportional to the strain rate and stress. This indicates that the Maxwell and Voigt solids react in opposite ways resulting in the logarithmic decrement in amplitude of vibration being inversely proportional to frequency for Maxwell solids and directly proportional for Voigt solids. The more general model is a combination of the Maxwell and Voigt elements resulting in a model which is more useful in describing the qualitative nature of viscoelastic material. However, even the more general model does not correspond well to quantitative results except over a small frequency range.

Wang et al. [1994] suggest that the viscoelastic behaviour of polymeric materials can be simulated by the nonlinear Zhu-Wang-Tang (ZWT) model. The ZWT model is a compilation of two Maxwell solids in parallel with an elastic spring element. Through numerical simulation, they are able to accurately predict the viscoelastic response with known material properties.

Tyas and Watson [2000] use numerical simulation to determine the viscoelastic behaviour of the material. They simulate a known input force time history applied to the bar end while recording the dispersed signal at some distance from the end. From the known input and recorded output, the dispersive relationship can be determined.

Sawas et al. [1998] used bars made of cast acrylic to test samples made of polycarbonate, polyurethane foam and styrofoam with some success. Their method of data reduction requires an *a priori* knowledge of the material properties of the acrylic bars. These properties are used to solve a form of the viscoelastic wave equation allowing the propagation of the wave to be predicted.

Zhao and Gary [1995a] developed a three dimensional wave equation based on Pochhammer and Chree's longitudinal wave propagation equation. Through comparison with empirical results, they show that the application of the three-dimensional model more accurately predicts how waves will propagate through viscoelastic media. Zhao and Gary [1995b] also extended this work to include an inverse method for calculating the material parameters by measuring speed on the bar ends and then estimating the model parameters through multiple iterations. Sogabe et al. [1995] use a similar approach to define a propagation coefficient allowing for the correction of attenuation and dispersion.

3.2.2 Experimental

The advantages of determining the material properties of the incident and transmitted bars experimentally are that no prior knowledge of the material properties is required and it is not necessary to solve the Pochammer and Cree frequency equations. As well, corrections based on analytical techniques seem limited to correcting only small amounts of dispersive distortion.

Gorham and Wu [1996] suggested a method for experimentally determining the phase corrections. Their method requires that a series of tests using different size projectiles be conducted. The phase spectra for each pulse is analysed and with the knowledge of how an ideal pulse propagates, the smooth underlying phase variation common to all curves is determined. With the knowledge of how the phase velocities vary over the range of frequencies, the dispersion of the wave can therefore be predicted.

Bacon [1998] suggests an experimental method for considering attenuation and dispersion in viscoelastic bars. This method involves performing a test on each bar in order to determine the viscoelastic material behaviour. This method, outlined subsequently, determines the dispersive relationship experimentally. Bacon and Brun [2000] extend this method for determining the dispersive relationship to include bars that are non-uniform along

their length. This methodology would be useful if the ends of the bars are heated or if the bars are of non-uniform impedance to match a sample. This work is an extension of Lundberg et al. [1990] where the viscoelastic properties of the material were determined using a two point measurement technique.

Cheng et al. [1998] suggest a similar method of determining the propagation coefficient. Instead of unwrapping the phase spectra to determine the amount of phase shift between the two pulses, the wave number is estimated to give a reasonable phase velocity and frequency relationship.

3.2.2.1 Theory Behind Experimental Method

By applying spectral wave analysis to a Hopkinson bar configuration, equations relating velocity and force at the bar interfaces can be derived. The one-dimension wave equation can be written in terms of stress as:

$$\frac{\partial \sigma(x,t)}{\partial x} = \rho \frac{\partial^2 u(x,t)}{\partial t^2} \tag{3.17}$$

Strain is related to displacement by:

$$\varepsilon(x,t) = \frac{\partial u(x,t)}{\partial x} \tag{3.18}$$

Writing these basic wave equations in the Fourier domain:

$$\frac{\partial^2}{\partial x^2}\tilde{\sigma}\left(x,\omega\right) = -\rho\omega^2\tilde{\varepsilon}\left(x,\omega\right) \tag{3.19}$$

where $\tilde{\sigma}(x,\omega)$ and $\tilde{\varepsilon}(x,\omega)$ are the Fourier transforms of stress and strain respectively. The angular frequency, ω, is related to the frequency, f, by $\omega = 2\pi \mathrm{f}$. For a linearly viscoelastic media, stress is therefore related to strain by:

$$\tilde{\sigma}(x,\omega) = E^*(\omega)\tilde{\varepsilon}(x,\omega) \tag{3.20}$$

where E^* is the complex modulus of the material. The propagation coefficient, $\gamma = \gamma(\omega)$, is defined by:

$$\gamma^2 = -\frac{\rho \cdot \omega^2}{E^*} \tag{3.21}$$

Using equations (3.19), (3.20) and (3.21) the one-dimensional equation for axial motion becomes:

$$\frac{\partial^2 \tilde{\varepsilon}(x,\omega)}{\partial x^2} - \gamma^2 \tilde{\varepsilon}(x,\omega) = 0 \tag{3.22}$$

The general solution to this equation is given as:

$$\tilde{\varepsilon}(x,\omega) = \tilde{P}(\omega)e^{-\gamma x} + \tilde{N}(\omega)e^{\gamma x} \tag{3.23}$$

where $\tilde{P}(\omega)$ and $\tilde{N}(\omega)$ are the Fourier transforms of the strains at $x = 0$ due to the waves travelling in the directions of increasing and decreasing x respectively.

The velocity, $\tilde{v}(x,\omega)$, and normal force, $\tilde{F}(x,\omega)$, are then:

$$\tilde{v}(x,\omega) = -\frac{i\omega}{\gamma}\left[\tilde{P}(\omega)e^{-\gamma x} - \tilde{N}(\omega)e^{\gamma x}\right] \tag{3.24}$$

$$\tilde{F}(x,\omega) = -\frac{\rho A\omega^2}{\gamma^2}\left[\tilde{P}(\omega)e^{-\gamma x} + \tilde{N}(\omega)e^{\gamma x}\right] \tag{3.25}$$

The modulus and the phase angle of the complex exponential functions $e^{-\gamma x}$ and $e^{\gamma x}$ are related to attenuation and propagation respectively. The propagation coefficient $\gamma(\omega)$ is related to the attenuation coefficient, $\alpha(\omega)$, and phase velocity, $c(\omega)$, by:

$$\gamma(\omega) = \alpha(\omega) + ik(\omega) = \alpha(\omega) + i\frac{\omega}{c(\omega)} \tag{3.26}$$

where $k(\omega)$ is the wave number and is an odd function and $\alpha(\omega)$ is a positive even function with $\alpha(0) = 0$.

3.2.2.2 Experimental Determination of the Propagation Coefficient

The following method is based on the work by Bacon [1998]. The basis of determining $\gamma(\omega)$ experimentally is equation (3.25). By allowing one end of the bar to have a free end condition; the force must become zero (or at least much smaller than the force at the strain gauge location). When the force at the end is zero, equation (3.25) becomes:

$$\tilde{P}e^{-\gamma d} + \tilde{N}e^{\gamma d} = 0 \tag{3.27}$$

where d is the distance from the strain gauge location to the free end and $\tilde{P}$ and $\tilde{N}$ are related to the incident and reflected strains by:

$$\tilde{\varepsilon}_I = \tilde{P} \text{ and } \tilde{\varepsilon}_R = \tilde{N} \tag{3.28}$$

The transfer function $G(\omega)$ can then be defined as:

$$G(\omega) = -\frac{\tilde{\varepsilon}_R(\omega)}{\tilde{\varepsilon}_I(\omega)} = e^{-\gamma 2d} \tag{3.29}$$

The negative sign in front of the ratio is to compensate for the fact that the reflected wave is inverted. It should be applied to the Fourier transform of the reflected strain while in rectangular form. The complex ratio then describes how the wave changed, due to both attenuation and dispersion, over the distance 2d. After a negative sign is applied to the reflected strain, the complex ratio becomes:

$$G = \frac{r_R}{r_I} e^{i(\theta_R - \theta_I)} = e^{-\gamma 2d} = e^{-(\alpha + ki)2d} \tag{3.30}$$

Equating the real and imaginary parts gives:

$$\begin{aligned} \alpha &= -\frac{\ln\left(\frac{r_R}{r_I}\right)}{2d} \\ k &= -\frac{(\theta_R - \theta_I)}{2d} \end{aligned} \tag{3.31}$$

This is done for every frequency. Therefore, the dispersive relationship between the frequency and k is determined.

The determination of the propagation coefficient allows the velocity and force at the interface of both the incident and transmitter bars to be determined. This, in turn, allows a direct calculation of the strain rate from equation (2.1). The stress can be calculated from:

$$\sigma_s = \frac{F_T(t)}{A_s} \tag{3.32}$$

where the subscripts S and T refer to the sample and transmitter bar respectively. The strain can be determined by integrating equation (2.1) with respect to time and is given by:

$$\varepsilon_s = \int \dot{\varepsilon}_s dt \tag{3.33}$$

The division of the sample length to get the strain rate and the sample area to get the stress should be done in the time domain order to maintain its physical representation.

3.3 Summary

This chapter outlined the basis of spectral wave theory as it applies to the Hopkinson bar apparatus. The understanding of this method of wave propagation analysis enables one to include a dispersive relationship which allows the analysis of the viscoelastic behaviour inherent to most polymers. In addition to a detailed analysis of an experimental method, various analytical methods of determining the dispersive relationship were also discussed.

Chapter 4

Experimental Method

4.1 Introduction

As mentioned earlier, a Compressive Split Hopkinson Bar was chosen as the most applicable technique for the measurement of the dynamic properties of softer materials. In order to give a better impedance match between the bars and the specimen, acrylic was chosen as the bar material. Because of the use of acrylic bars there are a number of design constraints that differ from a conventional Compressive Split Hopkinson Bar apparatus due to the greater Poisson's ratio and lower Young's Modulus of the bar material. These produce significant radial expansion of the bars during the passage of the stress wave. Air bearings were selected to support the bars to allow radial growth of the bars and to minimize friction.

A gas gun was chosen as the method of accelerating the striker bar. Since the amplitude of the stress pulse applied to the incident bar is a function of the striker's velocity, this propulsion method was selected to attain the various desired velocities with consistency.

4.2 Experimental Design

The Hopkinson bar apparatus was designed to allow different rods with different tolerances and material properties to be used. The use of modular assemblies allows one or more assemblies to be modified independently as required. Complete engineering drawings of the apparatus are given in appendix A. The apparatus was designed using I-DEAS and Figure 4.1 shows an isometric view of the apparatus.

The apparatus is made up from several assemblies. The air bearings, interface supports and gas gun assemblies will be discussed in detail in following sections since they comprise the critical components of the apparatus.

4.2.1 Air Bearings

The dimensional tolerances on the acrylic rods, and most polymeric rods, can be quite large (typically 1.000 ± 0.050 inches) in comparison to metallic bars (typically 1.000 ± 0.005 inches). In addition, the stiffness of the rods varies from essentially rigid (steel, E = 209 GPa) to very flexible (low density polyethylene, E = 0.7 GPa). As a result, polymeric rods may not be very straight. Because of the required length of the rods, even aluminium rods can be somewhat bowed. The properties of the bars indicate that the support system must be flexible enough to accept bars that vary in size, rigidity and straightness.

When a stress pulse is induced in the incident bar by the striker bar, a wave propagates along the length of the bar causing it to expand radially. The amount of radial expansion is a function of the stress, the Elastic Modulus and the Poisson's ratio of the material. The apparatus was designed to produce a maximum stress in the incident bar of approximately 120 MPa (compressive yield strength of acrylic). Using the following material relationships

$$\varepsilon_{longitudinal} = \frac{\sigma}{E} \tag{4.1}$$

$$\varepsilon_{radial} = -\varepsilon_{longitudinal}\nu \tag{4.2}$$

with an Elastic Modulus of 3 GPa and a Poisson's ratio of 0.5, the maximum radial expansion of the 25.4 mm diameter bar is 0.51mm. This means that the supports of the rod must allow for a 1.02mm expansion in the diameter of the bar. Aluminium bars (ν=0.33, E=69GPa) would result in a diametral expansion of only 0.029mm for the same induced stress. These values indicated that conventional methods for supporting the bars (e.g. bushings) would not work effectively.

Another factor that is important in the design of the support system is friction. If friction is too great between the bars and their supports, the stress pulse will be attenuated. Friction also has an adverse effect on the repeatability of the experiments since the friction is

expected to change with time. Both of these factors indicate that the design of the bar supports is critical to the operation of the apparatus.

The solution to the problem of radial expansion and friction was an air bearing. Figures 4.2a and 4.2b show the lower part of the air bearing. The lower half of the air bearing is comprised of two concentric annuli joined together through two end caps and two sidepieces. The inner annulus has a series of holes drilled down the centre as well as matching holes drilled at 45^{o} to the centre line. The outer annulus acts as a manifold containing the air that is supplied through the air plug shown in Figure 4.2b. The holes in the inner annulus act as small jets that levitate the bar from the surface. This allows the bar to ride on a layer of air greatly reducing the friction. The holes drilled at 45^{o} to the centre line act as a lateral stabilizer preventing the bar from contacting the side. The force applied by the jets must be slightly greater than the force applied by gravity for the bar to lift off the surface.

The air bearings have a self-regulating effect eliminating the requirement for a regulating valve for each bearing. The equilibrium between the upward force applied to the bar, and the downward force of gravity, is maintained by allowing air to escape from the ends of the bearing and around the bar. As the bar approaches the bearing surface the upward force applied the bar is increased since less air is allowed to escape.

The lower part of the air bearing was made from aluminium to prevent any rust from forming. Corrosion is a significant factor since the air being supplied to the air bearing contains moisture. The upper half of the air bearing, shown in Figure 4.2c, is made from nylon. The upward normal force of the bar on the upper air bearing is small and nylon was selected to minimize friction between the bars and the upper half of the bearing.

This system of air bearings has many benefits. The diameter of the inner annulus is 26.7mm which leaves sufficient room for radial expansion and allows bars with different diametral tolerances to be used. The layer of air that the bars ride on greatly reduces the friction resulting in a signal that is less attenuated. The air bearing has the added benefit of being more tolerant of misalignment compared to conventional methods.

4.2.2 Rod Interface Supports

Special supports were needed at the interface between the incident and transmitter bars. These supports were used for two reasons. First, the ends of the bars must be aligned to ensure an accurate test. Since the bars ride on a layer of air, supplied by the air bearings, they are free to move slightly in the radial direction. This could misalign the interface between the two bars as well as make it difficult to hold specimens. Secondly, the interface supports were required to eliminate any vibration induced in the bars by the air bearings.

In keeping with the desire to minimize friction and allow for radial expansion, the interface support system shown in Figure 4.3 was designed. The support system is comprised of three Teflon pads attached to support rods that are free to move within brass bushings. The rods and pads are supported by springs that can be adjusted to allow the bar faces to be lined up. The springs between the pads and the adjustment bushings apply the necessary reaction force. The vertical support rod applies the necessary normal force to keep the bar stationary. By adjusting the bushing on the vertical support rod, the normal force imparted on the bar can be controlled. The combination of the Teflon pads and adjustment of the normal force helps to reduce friction while allowing the bar to expand radially.

4.2.3 Gas Gun

4.2.3.1 Basic Principles

Reducing friction in the gas gun is essential to being able to control the repeatability of the tests. The reservoir, the fast acting high flow valve, the cylinder assembly and the piston comprise the four main components of the gas gun. An exploded view of the gas gun is shown in Figure 4.4. The gas gun functions like a pneumatic actuator with the exception that the striker is launched from the piston at the end of the stroke. The reservoir is set to a desired pressure through a metering valve. In order to actuate the piston, the fast acting high flow valve is opened allowing the air from the reservoir to flow into the cylinder causing the piston and striker to accelerate forward. As the piston nears the end of the cylinder it slows and then comes to an abrupt halt, launching the striker bar down the barrel.

4.2.3.2 Design

The gas gun was designed to maximize velocity and sensitivity. The two design constraints contradict each other since achieving high velocities generally means large pistons, which reduces the capability of providing a fine adjustment on the velocity. Since the velocity is controlled purely by the pressure in the reservoir, a slight increase in pressure for a larger piston will result in a much higher increase in velocity. To predict the acceleration of the piston, a simple force analysis can be applied. Figure 4.5a is a free body diagram of the system. By taking the sum of the forces, the acceleration of the piston is given by:

$$\sum F = m_{system} a_{system} = F_{applied} - F_{friction} \tag{4.3}$$

where:

$$m_{system} = m_{piston} + m_{rod} \quad \text{and} \quad F_{applied} = P_{reservior} A_{piston}$$
$$a_{system} = a_{rod} = a_{piston}$$

and A_{piston} is the area of the piston. This equation is greatly simplified since it ignores losses through the valve and, as the piston moves, the reduction of pressure inside the reservoir. If linear acceleration is assumed, friction is ignored and the pressure from the reservoir is assumed constant, the velocity of the striker bar when it leaves the piston is given by:

$$V_{striker} = 2 a_{system} d \tag{4.4}$$

where d is the length of the cylinder minus the length of the piston. This simplified analysis does not represent the true dynamic behaviour of the system, but does indicate the critical design parameters. The reader is referred to McCloy [1980] for a more detailed analysis of this type of system.

A drawing of the piston is shown in Figure 4.5b. The unique shape of the piston was selected to reduce weight. The seal between the piston and cylinder is achieved through the use of two Teflon rings. Teflon rings were used instead of the conventional o-rings to reduce friction and increase the durability. The protrusion on the end of the piston fits into the end cap of the cylinder. This reduces the input force by compressing the air between the protrusion and the end cap allowing for a more gradual deceleration.

4.3 Sensors and Peripheral Equipment

The main principle of the Hopkinson bar relies on the measurement of the strain time history in the bars. For this purpose, electric resistance strain gauges are mounted on the bars.

4.3.1 Strain Gauges

Strain gauges are commonly made from either foil or semiconductor material. Each type of gauge has inherent advantages and disadvantages. Common to both types of strain gauges is the gauge factor. The gauge factor relates strain (ε) to a change in resistance (R) and is given by:

$$\frac{\Delta R}{R} = GF\varepsilon \qquad \text{(Dally and Riley [1991])} \tag{4.5}$$

where R is the resistance, GF the gauge factor and ε is the strain. In this relationship the gauge factor is the constant of proportionality between the change in resistance and the strain. The manufacturer normally calibrates and supplies the gauge factor with the gauge.

4.3.1.1 Wheatstone Bridge

A Wheatstone bridge is typically used to determine the change in resistance that occurs when a gauge is subjected to axial strain. The Wheatstone bridge configuration is shown in Figure 4.6a and is comprised of four arms. Each arm can be either a gauge or a dummy resistor depending on the desired configuration. In the case of the Hopkinson bar, any bending strain in the bar should be eliminated requiring that two opposite arms become active. In Figure 4.6a, R_1 and R_3 would be active gauges (denoted by the boxes) and R_2 and R_4 are resistors of equal resistance to the gauges. Voltage (V) is supplied to the bridge at points A and C while the output voltage (E), measured across points B and D, can be related to the strain. The voltage, E, is then equal to:

$$E = V_{BD} = V_{AB} - V_{AD} \tag{4.6}$$

From Ohms law, V_{AB} and V_{AD} can be related to the resistances by:

$$V_{AB} = \frac{R_1}{R_1 + R_2}V \text{ and } V_{AD} = \frac{R_4}{R_3 + R_4}V$$

This leads to:

$$E = \frac{R_1 R_3 - R_2 R_4}{(R_1 + R_2)(R_3 + R_4)} V \tag{4.7}$$

From equation (4.5), the resistance can be related to the strain by $R = R_o + GF\varepsilon$ where R_o is the original resistance, R the current resistance and GF is the gauge factor. Since R_1 and R_3 are the only active gauges, the output from the bridge can be related to the strain by:

$$E = \frac{(R_{1o} + GF_1\varepsilon)(R_{3o} + GF_3\varepsilon) - R_2 R_4}{\left[(R_{1o} + GF_1\varepsilon) + R_2\right]\left[(R_{3o} + GF_3\varepsilon) + R_4\right]} V \tag{4.8}$$

Therefore, strain can be derived from the bridge output with knowledge of the initial resistances and gauge factors.

In examining equation (4.7) it can be seen that in order for the bridge to be initially in balance (E=0) at zero strain, $R_1R_3=R_2R_4$. Because of the tolerances on the resistances, it is unlikely that this relation will occur naturally and, therefore, a balancing circuit is required. The bridge can also be out of balance due to differential temperatures and the resistance in the lead wires. A temperature compensated bridge is not necessary in this case since the duration of the test is quite short, Doyle [1989]. The balancing circuit must be able to compensate for the difference in resistances due to lead resistance and variability of the gauge resistance. There are a variety of circuits that can be used to balance the Wheatstone bridge. The circuit of Figure 4.6b was chosen for balancing. This circuit has some advantages when used with semiconductor gauges. By adjusting R_{Trim}, the numerator in equation (4.7) can be adjusted to equal zero.

4.3.1.2 Foil and Semiconductor Strain Gauges

For normal applications, both foil and semiconductor strain gauges are mounted to the bars using glue. The foil and semiconductor gauges have Elastic Moduli that are low and so provide no reinforcement but deform to follow the strain in the part.

Foil gauges, which are commonly used, are available in a wide variety of configurations, are easy to apply and are relatively inexpensive. Unfortunately, they have a low gauge factor and are relatively insensitive compared to semiconductor gauges. The low gauge factor, generally round 2.0, requires amplification of the bridge output signal. In amplifying the bridge output, the electronic noise is amplified as well.

Semiconductor gauges also come in a variety of configurations but are relatively difficult to apply and are more expensive than foil gauges. Unlike foil gauges, semiconductor gauges have no backing and special care must be taken when adhering the gauge to material that conducts electricity. The main advantage in using semiconductor gauges is their high gauge factor of approximately 150. This higher gauge factor allows for recording of the unamplified bridge output voltage. The high gauge factor increases the resolution of the sensor allowing for a much greater signal to noise ratio. The tolerances on the resistance of semiconductor gauges require extra consideration when choosing a balancing bridge circuit. For this application, the semiconductor gauge resistance is $1000\Omega \pm 8\%$ compared to $120\Omega \pm 1\%$ with foil gauges. This is the reason that the balancing circuit mentioned in the previous section is used. Despite all of the disadvantages of semiconductor gauges, the inherent increase in sensor resolution made them the preferred choice in this application.

4.3.1.3 Bridges and Non-linearity

Equation (4.8) indicates that the relationship between the bridge output and the strain is non-linear. A plot of strain versus bridge output is shown in Figure 4.7. This assumes, however, that the gauge factor is constant. In order to validate the bridge equations for their application to semiconductor gauges, a series of compression tests on small cylindrical specimens instrumented with both foil and semiconductor gauges were undertaken. Specimens made from 6061-T4 aluminium and PMMA (acrylic) were tested. Figure 4.8a shows the strain gauges applied to the acrylic specimen. The specimens were placed in an Instron compression test fixture and compressed at a quasi-static rate while recording the applied load and the bridge output. Figure 4.8b shows this arrangement. Contradicting the theoretical bridge equation, a relatively linear relationship was found between the force and bridge output for both the acrylic and aluminium samples over the load range. Figures 4.9a and 4.9b show the results of these tests with the theoretical results being obtained from equation (4.8). Since the relationship is linear for both the acrylic and aluminium samples, viscoelastic effects can be neglected during the test. The reader is referred to Dally and Riley [1991] for a more detailed discussion of strain gauge and bridge circuit technology.

4.3.2 Simple Velocity Detection

To accommodate the variability of the striker velocity due to friction within the gas gun, a low cost velocity detection system was adopted. The ability to measure the striker velocity prior to impact provides an additional input condition when analyzing the results. A schematic of the velocity detection system is shown in Figure 4.10a. The system is comprised of four lasers, four detectors and associated circuitry. As the striker bar breaks the first laser beam, the circuitry starts a counter which counts until the second beam is broken. Similarly, other counters time the passage of the striker between each pair of lasers. For the configuration shown, three time measurements are made. In combination with the known distances between the lasers, the times can be translated into an average velocity of the striker. The housing developed to hold the lasers and detectors is shown in Figure 4.10b.

4.3.3 Data Acquisition

A means of recording the time history of the bridge output voltages is necessary and a Nicolet Pro30 digital oscilloscope was used. It is capable of sampling at 10 MHz with two differential input channels (one for the incident bar, the other for the transmitter bar) with 12 bit accuracy. The differential mode is utilized due to the presence of electronic noise in the test area. The Nicolet Pro30 is capable of applying a high frequency rejection filter to further reduce the effect of electronic noise. Data can be transferred from the oscilloscope to a floppy disk which can then be used by a data reduction program.

4.4 Numerical Implementation

This section describes the data reduction program that was developed to analyse the data acquired from the Hopkinson bar apparatus. The data reduction program, called CSHB, was created with three distinct parts which are described below. The first part pre-processes the data, the second part contains the numerical implementation of the analytical equations described earlier and the third section is a simple post processor which allows for quick viewing of the processed data. In addition to the three main sections of the program, some useful utilities have also been created.

The program was developed using Visual C++ and has an object orientated programming structure (OOPS). A multiple document interface is supplied to allow

maximum flexibility for the user. For brevity, the details of the Visual C++ code will not be described but is included in appendix B.

4.4.1 Pre –Processing

The main objective of pre-processing is to manipulate the raw data into a form which can be analysed numerically. The pre-processor makes use of the data read from the oscilloscope, the data concerning the sample specifications and the calibration files for each bar. Additionally, it performs some initial data manipulation.

4.4.1.1 Raw Data

The data reduction program is capable of using data in two different file formats. The default file format corresponds to data acquire by the Nicolet oscilloscope. The Nicolet brand of oscilloscopes output binary files that contains the data in a low byte, high byte arrangement. The reader is directed to reference Nicolet [1991] for further details on the structure of the raw data files. The second file format uses data in an ASCII text format delimited by tabs, commas or spaces. This functionality was implemented to maximize the versatility of the program. The program reads the voltage and time data for the incident and transmitter bar from either format into two sets of dynamically created arrays. There is an option to shift the amplitude of the data, subtracting the average of a desired number of initial data points, to remove any unbalance in the bridge output voltage that was not compensated for manually. The default number of points initial points used is set to twenty which is usually acceptable.

4.4.1.2 Sample and Test Data

The “Sample Data” dialog window, shown in Figure 4.11, provides an interface allowing information about the sample and test conditions to be entered. The only data required to calculate strain, stress and strain rate are the sample’s initial dimensions. The remainder of the data is stored and exported as part of a header file after processing. Note that the dialog window provides the date of the raw data files (if already obtained) so that the user can ensure that the correct files are being used. If the files have not been read in, or if the dates are invalid, the sample date is set to the current date.

4.4.1.3 Calibration Files

In order to analyse the raw data, information about the incident and transmitter bar is required. For each strain gauge station on the bar, a calibration file must be created. The method outlining the physical calibration procedure is given in section 4.5.1. The calibration file has a series of control statements that identify the appropriate data. A typical calibration file is shown in Figure 4.12.

The first input statement shown is the name of the bar. Each bar has a unique name and the appropriate calibration information should follow the name. The second statement, following the identifier, is the geometry of the bar. The data is in the format of diameter, length and distance from the gauge station to the bar end interface. All dimensions are in meters. The next statement shown, following the identifier, contains the mechanical properties of the bar. The values have the format of density, Elastic Modulus and Poisson's ratio. The density is in units of kg/m^3, the Elastic Modulus is in units of N/m^2 and the Poisson's ratio is unitless. The data following the "calibration" identifier contains the output from the bridge, in Volts, in the first column and the corresponding strain values in the second column. These values can be determined experimentally as described in section 4.4.1 or can be calculated using the theoretical equations and known gauge factors of section 4.2.1. The data following the "filter" identifier indicates the value of the frequency exclusion filter, discussed later, used when propagation coefficient data is included. The data following the "propagation" identifier indicates that propagation coefficient data has been appended to the calibration file. The data is contained in three columns of the format: frequency, in Hz, attenuation coefficient, in 1/m and wave number.

These data sections need not be in the above order as the program identifies the relevant data through the identifier statement. A dialog window (Figure 4.13) was created to acquire the calibration files and present a summary of the data. This window allows verification of the calibration data as well as displaying the bridge output to strain conversion factor. A check box indicates whether propagation coefficient data is available.

4.4.1.4 Wave Separation

Since the incident and reflected waves are contained in the same raw data file acquired from the incident bar, the signal must be separated into the incident and reflected components. Additionally, depending on the length of time during which test data is acquired, subsequent reflections off the impacted and interface ends might be present in the acquired signal that are not relevant in calculating the sample's properties. Similarly, the transmitted wave must be separated from irrelevant data contained in the data acquired. Figure 4.14 shows raw data for the incident and transmitter bar indicating the starting and end points for the relevant data waves.

Separating the waves from the rest of the acquired data is accomplished with the "Separate Waves Values" dialog window shown in Figure 4.15. This window allows the values that will be used in the algorithm that separates the waves to be specified. Two different methods of intercept or peak location are available to separate the waves.

Common to both methods are the use of thresholds. A threshold is a datum level which initiates an event when crossed. Figure 4.16 indicates the components of the wave and where the thresholds are applied (shown on the incident wave only). The primary threshold is determined by multiplying the value indicated in Figure 4.15 by the maximum value of the entire data set. Similarly, the secondary edge threshold is found by multiplying its respective value by the maximum value of the primary edge. The remainder of this section outlines how these thresholds are used to find the starting and end points of the waves.

The intercept method attempts to separate the waves by locating the points where the voltage passes through the zero point. The routine scans the data, starting at time zero, up to the point where it crosses the primary edge threshold. The data is then scanned backwards, towards zero time, until the voltage reaches zero. This is the starting point of the wave. The end of the wave is found by scanning the voltage values from the start of the wave until it passes through zero. This is the end point of the wave. The starting and end indices of the reflected wave are found in a similar manner by using the negative value of the primary edge threshold.

The peak location method is primarily used when performing a conventional Hopkinson bar analysis. The starting points of the waves are found in a similar manner but,

the end points are determined by an algorithm that identifies all of the peaks and valleys in the data. Typically, a wave can be characterized by a starting point, followed by a peak value and then a minimum. The algorithm scans the peaks and valleys until a voltage crosses the secondary edge threshold. The valley that occurs after this threshold is the end of the wave. A filter value is supplied so that any Pochammer Chree oscillations present in the signal will be skipped when finding the end index.

If either of the two separating methods described above fail, the ability to pick the starting and end points is available through the view indices dialog window shown in Figure 4.17. The starting and end points for each wave are displayed in terms of the sample index (or sample number). By clicking on the button beside the index, the index can be changed by selecting the desired point on the graph of the raw data using the mouse. When choosing the end point of wave, a check to ensure that it is greater than the starting point is performed.

The program references each data point by an index. Since the points are sampled at a constant rate, the time at each point can be found by multiplying the index by the sample period (time between points). This reduces the amount of memory required by storing only the sample period instead of a time value for every point. The starting and end times for the waves can be viewed by selecting the "View Time" button. Selecting this button (now labelled "View Index") again will change the display back to show the sample index.

4.4.2 Analysis

The major component of the data reduction program is the analysis algorithm which utilizes a series of functions to implement the analytical method of Chapters 2 and 3. Detailed descriptions of the algorithms used are outlined below.

To initialize the analysis routine, a dialog window, shown in Figure 4.18, summarizes the input data. This Figure lists the names of the raw data and calibration files, the sample dimensions and sample name. The fields on the bottom of the "Analyse Data" window allow the choice of a conventional Hopkinson bar analysis or the Fourier method of analysis. The conventional method uses the equations of Chapter 2. If the Fourier method is chosen (the default case), the options of applying a frequency filter or using the Nyquist frequency become available. These values will be explained in detail later.

Figure 4.19 shows the structure of the analysis algorithm. The main structure is the same for both the Fourier and conventional methods with the exception of the wave manipulation method and how the sample properties are calculated. The first conditional statement indicates that the incident bar data set must have been pre-processed. For this condition to be valid, the raw data must be separated into its respective waves and the bar calibration file must have been entered. If these conditions are met, the wave is manipulated according to the method chosen. The specifics of these methods will be outlined in subsequent sections. Similarly, the next conditional statement tests for the validity of the transmitter bar data set and manipulates it accordingly.

If the sample data, in addition to the strain, velocity and force data for both the incident bar and transmitter bar data sets, is valid, the material properties for the specimen are calculated. If the sample data is omitted, only the velocity and force data will be calculated. Depending on the method chosen, the material properties will be calculated either from the strain history and bar properties or from the velocities and forces. The different methods for calculating the sample values will be outlined below.

4.4.2.1 Conventional Analysis

Figure 4.20 shows a flow chart which outlines the sequence of algorithms for processing the data with the conventional analysis. The raw voltage values are passed to a routine that requires the starting and end points of the wave. The voltage to strain conversion factor from the bar calibration file is applied to the raw data. The starting points for the waves are shifted so that they are aligned. By aligning the start of the waves (and subtracting the reflected wave from the incident), the strain time history at the end of the bar is identified. In physical terms, at the end of the bars, the starting points for the waves occur at the same instant in time. Therefore, by aligning the starting points, the waves now represent the behaviour at the end of the bar instead of at the gauge station.

Once the bar strain data is calculated, the force and velocity data for the bar ends can be determined from equations (2.7) and (2.3) respectively. This data, however, is only calculated for information and display and is not used in any further calculations.

As outlined previously if the incident bar, transmitter bar and sample data are all valid, the material properties will be calculated through the application of equations (2.9)-

(2.11). The integration that is required when calculating the strain in the sample (equation (2.11)) makes use of trapezoidal integration. This method of numerical integration is valid since the sample rate is high (corresponding to a very small sample period) making the assumption of a linear function between the points accurate. The length of the strain data set dictates the length of the data set containing material properties. For stress strain curves there can be a length mismatch since the stress uses the transmitted strain set, while the strain in the sample is a function of the reflected data set.

4.4.2.2 Fourier Analysis

Figure 4.21 illustrates the flow of the Fourier analysis algorithm for manipulating the waves. In comparison to Figure 4.20, the degree of complexity is considerably greater. Additional routines were required to correct for dispersion and allow propagation of the waves. Additional compatibility checking and manipulation routines were also implemented to allow a variety of data acquisition configurations.

The first conditional statement checks if the propagation coefficient data is included in the bar calibration file. If propagation coefficient data is not included, theoretical values for the wave number are calculated using equations (2.2) and (3.14). The combination of equations (2.2) and (3.14) indicate that the group speed is equal to the individual wave speed resulting in the absence of a dispersive relationship as outlined in Chapter 3. The attenuation coefficient is assumed to be zero. The discrete frequencies at which to calculate these values are determined from the sample period and the number of points of raw data.

If the bar calibration file includes the propagation coefficient factors, additional compatibility checking and possible manipulation are required. The reason for these additional procedures stems from the requirement that the propagation coefficient values and raw data have equal base frequencies. Additionally, both the raw data and propagation coefficient data must have the same number of terms in the frequency domain. Since it is not always valid to interpolate values in the frequency domain, any manipulation of the raw data must be done in the time domain. The data manipulations are done on the raw data instead of the propagation coefficient data since the raw data already exists, and has physical meaning, in the time domain.

In order to satisfy the compatibility condition, both the sample period and number of points acquired for the raw data and bar propagation coefficient data must be equal. Since the number of points and sample rate for the raw data can be equal to, greater than or less than that of the propagation coefficient data, nine mutually exclusive possibilities occur. Table 4.1 illustrates the nine possibilities and the required actions. The sample period for the propagation coefficient data is calculated from equation (3.5) and the number of points.

As can be seen from Table 4.1, a difference in sample period requires the raw data to be re-sampled. An algorithm was developed to replace the current data set sampled at one rate, with an equivalent data set sampled at a different rate. This is achieved through linear interpolation. This allows the desired sample period to be a value other than an integer multiple of the current sample period. The addition of points, if required, is done prior to re-sampling the data, while the deletion of excess points is done after re-sampling. The actions performed on the raw data are recorded and are included in the header file when the data in exported (discussed in post processing). The starting and end indices of the waves are then adjusted to the new sample rate.

Before the FFT is applied to the data, zeros are added to the end of the data (padding) until the number of points is an integer multiple of two. This is a requirement of the FFT routine and does not affect the data in the frequency or time domain. If dispersion data is included in the calibration file, this should not be necessary since the number of points already will be forced to an integer power of two, whereas when dispersion data is not included and theoretical values are used, this correction is necessary.

The voltage data is then converted to strain. Unlike the conventional method, the strain data sets will be identical in size and will be equal to the number of points in the raw data. The values that are outside the range between the starting and end indices are set to zero. This is required since the waves will be propagated (instead of merely shifted) and the data must remain an integer power of two. Once all of the above conditions are met, the strain data waves are transformed in the frequency domain using the FFT routine in preparation for application of the propagation algorithm.

The FFT algorithm has been previously discussed and no further details of the numerical implementation will be given here. The reader is referred to Press et al. [1992] for a complete discussion of FFT algorithms. Since the raw data is real only, the negative

frequency components are not computed. Once the strain data has been transformed into the frequency domain, the routine "change form" is called. The "change form' algorithm replaces a data set in rectangular form with a data set in polar form. The algorithm compensates for the quadrant angle as well as unwraps the phase spectra as discussed in Chapter 3.

With the waves in polar form, the propagate wave algorithm can be initiated. Each wave is propagated the distance indicated by the gauge to interface distance specified after the "geometry" identifier in the bar calibration file. The incident wave is propagated in a positive direction while the reflected and transmitted waves are propagated in a negative direction. By propagating the waves, the strain time history at the bars ends can be identified. The propagate wave algorithm is the numerical implementation of equation (3.23). The additional parameter of a frequency filter, specified in Hertz in the "Analyse Data" window (Figure 4.18), is used by the propagate wave algorithm. Calculations are performed at frequencies up the to value of the filter frequency with the remainder of the terms being set to zero. This filter is required when using supplied propagation coefficient data due numerical errors in the attenuation coefficient values discussed in the next section. If the values are calculated theoretically, then the filter can be set to the Nyquist frequency. The Nyquist frequency is given by:

$$f_{Nyquist} = \frac{1}{2 \bullet SamplePeriod} \tag{4.9}$$

and represents the highest frequency sinusoid that is detectable at the given sample rate.

Once the waves have been propagated, the "change form" algorithm is implemented again to change the data back into rectangular form. The velocities and forces at the bar ends can then be calculated from equations (3.24) and (3.25).

With the velocities and forces processed, the strain rate and stress in the sample is calculated if the sample data is valid. Before the application of equations (2.1) and (3.32), the difference in velocities and transmitted force are transformed into the time domain with the IFFT algorithm. The reason for this is discussed in Chapter 3. Equations (2.1) and (3.32) are then applied to the force and velocity data. The strain is calculated by integrating the strain rate with respect to time, as indicated in equation (3.33), using trapezoidal integration.

4.4.2.3 Propagation Coefficient Calculation

The algorithm, which calculates the propagation coefficient, requires the same input data and pre-processing that was required for the previous section. The data recorded from the free end test described in Chapter 3 is entered into the program as an incident data set. This is done for both the incident and transmitter bar. This allows the waves to be separated into the reflected and incident components.

Once the input conditions are valid, the ability to calculate the propagation coefficient becomes available. A dialog window "Propagation Coefficient", shown in Figure 4.22, was created to allow the user to view the data files and select various output options. The propagation coefficient data can be directly appended to the calibration files if the data does not already exist. If the propagation coefficient data does exist, an error will be issued and the calculations will not proceed. In this case the data must be removed manually and the bar calibration file re-entered before the calculation will continue and the file can be appended. The options of exporting the propagation coefficient data to a separate file, along with the complex modulus of the material, calculated from equation (3.21), are available. Appending the data to the calibration file results in three columns as explained in section 4.3.1 while the data exported to a file includes the phase speed calculated from equation (3.14). The complex modulus values are at the end of this file. A filter, equivalent to that explained in the previous section, is supplied.

The calculation of the propagation coefficient proceeds in the manner outlined in section 3.2.2.2. The voltages are converted to strains as outlined in the previous section and then changed into the frequency domain. The reflected strain wave is then inverted. The strains are changed to polar form and the calculations of equation (3.31) are performed. A degree of numerical error is unavoidable at the higher frequencies when calculating the attenuation coefficient. This is due to the small components of the amplitude and phase spectrum at these values. After approximately 20 Hertz the strain signals become very small causing the denominator of equation (3.31) to approach zero, which results in unrealistically high attenuation coefficients. It is for this reason that the frequency filter is required. The entire propagation coefficient data is exported to the calibration file to allow the user to

choose the range of frequencies and to indicate the number of points to be used in the calculation of the propagation coefficient data.

If a filter frequency that it too high is selected, a large high frequency component will be superimposed on the waves when they are propagated. If a filter frequency that is too low is selected, the resulting propagated waves will not adequately represent the true data and low frequency components will be unrealistically large. The choice of the frequency filter can be determined by selecting the frequency where the amplitude spectra fall to zero. This can be done by looking at the strains in the frequency domain using the FFT utility described in the utilities section.

Once all the calculations are completed, the strain, velocity and force data are transformed into the time domain to allow post processing.

4.4.3 Post Processing

A simple post processor was created to allow a quick inspection of the processed data. The processor displays the relevant calculated data. When sample dimensions are included, and all input data is valid, engineering stress, strain and strain rate curves will be displayed. In addition, the original voltage versus time and the propagated waveforms are also displayed. This helps to identify any problems in the wave propagation and allows changes to the parameters if necessary. The plot of the strain waveforms can be changed to display the velocity, force or displacement time history of the bar ends.

The post processor also has an export utility that allows any of the calculated data to be exported to an ASCII text file (tab delimited). The “Export” window shown in Figure 4.23 allows the exportation of the data. A header can be included which summarizes all of the input conditions (raw data files, sample dimensions, processing date etc.), lists any manipulations that were done on the input data and peak calculated values. The data is exported in the order show in Figure 4.23 with the first column being time values. The tab delimited form allows for the data to be quickly imported into any spreadsheet or other programs for further manipulation.

4.4.4 Utilities

A series of three utilities were created to allow manipulation of the data. The "FFT Conversion" utility allows data, in ASCII format, to be converted from the time domain to the frequency domain or vice versa depending on the parameters chosen. The time data must be in the first column and the amplitude (voltage, strain etc.) data in the second column. The FFT utility outputs the frequency representation in both rectangular and polar form for the positive frequencies up to, and including, the Nyquist frequency. If the IFFT is required, the data must have the frequency, in Hertz, in the first column, the real data in the second column and the imaginary components in the third column. The data must be in rectangular form and contain positive frequency values up to, and including, the Nyquist frequency.

The "Propagate Waves" utility allows for the propagation of waves. This utility uses algorithms, described in section 4.3.2, to propagate waves over a desired distance. This utility is useful if the behaviour of the wave is required at distances other than the gauge to interface distance specified in the calibration file. This allows the wave profile at one location to be compared to a wave measured at another location or time. For example, if on the free end test, the sample duration is long enough to capture the reflection of the wave off the impacted end, the wave initial incident wave can be propagated four times the gauge to interface distance to align it with the wave. In this way, the difference between the predicted wave and the actual wave (assuming no losses at the ends) is easily identified. This can help authenticate the dispersive relationship model that was used. If the wave is propagated beyond the duration of the recorded data, it will appear at the beginning of the sample time. This is due to the cyclic nature of the Fourier transform. If a zero distance is supplied, the frequency filter can be adjusted until the form of the propagated wave resembles the original waveform. Although this is a quick method of determining the appropriate filter frequency, it is not as accurate as the method described previously. The final waves can be exported using the export utility.

The "Nicolet File Conversion" utility is a program that converts Nicolet binary data into ASCII text format. This is useful if any unusual manipulation of the raw voltage data is necessary. The output is in tab delimited form and can be read into the data reduction program, after manipulation, provided it has a similar format.

4.4.5 Summary

A detailed explanation of the numerical implementation of the analytical equations, described previously, has been given. The Fourier method of analysis, which utilizes wave propagation algorithms, has many added benefits compared to the conventional analysis. Since the waves are propagated instead of being shifted, there is less variability when it comes to selecting the starting points of the waves. The Fourier analysis allows zeros to precede and follow the actual data without consequence to the processed data. If the conventional method of analysis is used, and incorrect points are chosen, the stress, strain and strain rate calculated will not be indicative of the specimen behaviour. Fourier analysis allows the application of a dispersive relationship which is required when using polymeric bars.

4.5 Test Procedure

The test procedure of a typical test is outlined below.

4.5.1 Calibration

Calibration of the bars is necessary so that the recorded voltage can be converted into strain. From the discussion in section 4.2.1.4, it was decided that a force versus bridge output relation was required. Since tests show that a linear fit to this data is acceptable, the bars can be calibrated by applying a known force and recording the bridge output. A calibration fixture was constructed to apply a force to the incident bar through a hydraulic jack for the aluminium bars and a turnbuckle arrangement for the acrylic bars. A load cell is placed between the incident and transmitter bars. A load was then applied to the incident bar and the load cell voltage and bridge output for the each rod was recorded. The transmitter bar was supported to react the applied load by a bracket at its end. Recording the bridge output for varying levels of force results in a force versus bridge output curve. The force can then be converted to strain using geometric and mechanical material properties resulting in a strain versus bridge output curve which the data reduction program uses. By calibrating the bars in this manner, any non-linear factors within the bridge circuitry are compensated for. It is recommended that the calibration test be repeated on a regular basis to detect any deterioration of the strain gauges.

4.5.2 Propagation Coefficient Test

The use of acrylic bars requires that the propagation coefficient parameter be calculated prior to undertaking tests on sample materials. These calculations need only be done once for each bar and are independent of the sensors used. In other words, unlike the conversion parameters mentioned previously, the dispersive relation is a material property and does not change with the strain gauges.

The propagation coefficient can be determined by allowing the striker to impact the bar with the other end of the bar being allowed to move freely. This is called a “free end” test. In order to be consistent, both the incident and transmitter bars should be tested in this manner. The recorded strain information is processed with the data reduction program (described previously) and the calibration files for the bars have the propagation coefficient information appended.

4.5.3 Sample Preparation

The sample size must be selected to ensure equilibrium is reached in the specimen while providing the desired test information. The shorter the sample length the greater the strain rate. The smaller the diameter of the sample, the greater the induced stress. The sample must be large enough that a portion of the wave propagates into the transmitter bar. In addition, the ratio of the sample length to diameter can affect how the sample deforms. If the sample is too short in comparison to the diameter, then frictional end effects can prevent the sample from deforming correctly.

Generally, samples of hard materials have a length to diameter ratio of approximately one or less. This allows for homogenous deformation of the sample while preserving the equilibrium constraint. For softer materials, a length to diameter ratio of approximately 0.2 proved to be acceptable. This reduction in ratio is to accommodate the reduced wave speed in the softer materials.

Unlike metals, great difficulty can be encountered when fabricating samples from softer materials. Materials such as synthetic and real bone can be machined in a conventional manner. The use of special coring tools, similar to a hole saw, increases the productivity of generating the samples while reducing waste. Fabricating samples from softer materials such a ballistic gelatin or RTV silicon can be achieved with the use of coring tools or a punch.

Cores can be taken from prefabricated silicon sheets and then cut to length with a razor blade.

4.5.4 Test Execution

Once the samples have been fabricated and the bars calibrated, the material's high strain rate properties can be determined. The bridge output for each bar is initially balanced and the gas gun is loaded with the desired striker. The sample is placed between the incident and transmitter bar with a thin layer of petroleum grease on each of the bar faces. The sample should be placed in the center of the bars to minimize any distortion. This can be achieved through the use of an alignment collar. The lubrication helps to hold the sample in place. Once the data acquisition system is set to capture the test data, the striker can be fired down the barrel and the resulting voltage time history for the strain gauge stations recorded. The recorded data can then be processed using the data reduction program described previously.

There are special considerations when testing soft, easily deformed, materials. With most rigid materials, such as polycarbonate, the issue of preloading the specimen doesn't exist. This is not true, however, for soft materials such as rubbers. Figure 4.24 shows the difference in the stress strain curves for two identical samples, under similar input conditions, when a sample is initially compressed. As illustrated, the strain in the compressed specimen is greatly reduced compared to the uncompressed specimen. This is expected since the change in length of the specimen was already reduced when the specimen was placed between the incident and transmitter bars. The bar ends must therefore be adjusted to only lightly touch the faces of the sample.

Chapter 5

Validation and Results

5.1 Introduction

Three separate procedures were undertaken to validate the test apparatus. The first method involves comparing predicted waves with measured waves. The second method uses an Enhanced Laser Velocity System (ELVS) to measure the velocities of the bar ends. The third method made use of the characterization of materials with known properties. The results of these procedures will be discussed in sections 5.3, 5.4 and 5.5 respectively.

As discussed in Chapter three, the rheological properties of polymeric materials must be compensated for when analysing wave propagation. Section 5.2 describes a series of tests conducted on aluminium, acrylic and low density polyethylene (LDPE) to determine the amount of dispersion and attenuation inherent in each material. Since aluminium has little to no dispersion or attenuation, it can be used as a baseline against which the viscoelastic behaviour of the other two materials can be identified. Table 5.1 contains the various mechanical properties of the three materials. As can be seen from the table, the wave speed for the LDPE is significantly lower than acrylic and aluminium. The greatly reduced wave speed and inherent rheological properties, which result in significant amounts of dispersion and attenuation, made LDPE the ideal choice for testing for a dispersive relationship.

The reader will note that the figures in the following sections, which identify either the strain or voltage time history, are inverted compared to the generally accepted convention. Normally, compressive strains are negative (with a corresponding negative voltage output from the bridge) and tensile strains are positive (with corresponding positive voltages). This

convention has been reversed to maintain consistency with previous results generated from the University of Waterloo and other authors.

5.2 Dispersion and Attenuation Coefficients

The propagation coefficient for each of the materials was identified using the equations of Chapter Three and the methodology of Chapter Four. The tests performed on the aluminium bars used a striker length of 605mm; the acrylic 235mm and the LDPE 20mm. The striker lengths were selected so the strain in the bar would return to zero and remain there for a discernable length of time before superposition of the incident and reflected wave. In addition, by using shorter striker lengths, the higher frequency components of the wave can be identified. The bars used were 25.4mm in diameter and 2.4m long. After the dispersive nature of propagating waves has been identified, longer strikers can be used to maximize the test duration.

This section is comprised of three parts. The determination of the dispersive relationship for acrylic will be discussed in detail initially. A comparison of the dispersive relationships for the acrylic, aluminium and LDPE will follow.

5.2.1 Acrylic Results

This section outlines, in detail, the determination of the dispersive relationship for acrylic. The dispersive relationships for aluminium and LDPE are determined in the same manner. Figure 5.1 shows the raw data acquired from the free end test performed on one of the acrylic bars. The raw data signal is comprised of three parts. The first pulse indicated is the passage of the incident wave by the strain gauge. This wave is generated by the impact of the striker and travels along the length of the bar towards the interface end. The inverted second pulse indicated is the return passage (reflection) of the incident wave off the free end. The third pulse indicated in the figure is from the subsequent reflection of the wave off the impact end. The incident and reflected waves were separated using the data reduction program. These waves were then transformed into the frequency domain. Figures 5.2a and 5.2b show the real and imaginary coefficients of the Fourier series for both the incident and reflected waves. These graphs indicate the frequency at which the values approach zero. The zero value frequency is important since the numerical error in calculating the propagation coefficient will increase as the magnitude of these coefficients approach zero.

Figures 5.2c and 5.2d show the amplitude and phase spectra for the incident and reflected waves. Here the phase spectra have been unwrapped to obtain a monotonically increasing function of frequency. Unlike figures 5.2a and 5.2b, this representation of the data allows qualitative observations that relate to the material behaviour to be formed. Recall from equation (3.31) that the attenuation coefficient is a function of the ratio of the amplitude spectra of the incident and reflected pulses. In other words, attenuation is represented by a decrease in the magnitude of the amplitude spectra. Comparing the incident and reflected amplitude spectra of Figure 5.2c, it can be clearly seen that there is a reduction in magnitude over the entire frequency spectrum. Recalling that the D.C. or zero frequency term represents the area under the time function curve, an inspection of the first terms clearly indicate an overall decrease in area. Similarly, a difference in phase spectra (Figure 5.2d) for the incident and reflected waves can be related to the wave number (through equation (3.31)) and corresponding phase velocity.

Figures 5.3a, 5.3b and 5.3c are respectively, the attenuation coefficient, wave number and phase velocity obtained from equations (3.31) and (3.26) for the acrylic bar. The attenuation coefficient data (Figure 5.3a) shows greater attenuation at 4.6 kHz and 9.7 kHz. If Figure 5.2c is inspected at these frequencies, it can be seen that the amplitude of the incident and reflected waves are similar and nearly zero. This indicates that these frequency components do not have a large effect on the wave. As the frequency increases, the amplitude decreases as seen in Figure 5.2c. This results in greater error being present in the higher frequency components. A limiting frequency, above which all terms are ignored, must be chosen as discussed in the previous chapter. In this case, an upper limit of 14 kHz appeared appropriate. Through inspection of figures 5.2a, 5.2b and 5.2c, the exclusion of the higher frequency terms will not have an adverse effect since the majority of components are contained in the lower frequencies.

Figure5.3c shows the phase velocities for the different frequencies. Noticeable peaks occur at the same frequencies as the attenuation data (Figure 5.3a). Inspection of Figure 5.2d reveals that these peaks correspond to plateaus in the phase spectra. If figures 5.2a and 5.2b are inspected at these frequencies, it can be seen that the magnitude of both the real and imaginary components are nearly zero. This is also indicated in Figure 5.2c. Therefore, any abnormalities, which might be expected from these peaks, will be minimized due to the small

magnitude of these components. In other words, these frequency components have little effect on the overall wave and, therefore, an increase in their phase velocity and decrease in attenuation coefficient will have little consequence.

5.2.2 Dispersive Relationship Comparison

Figure 5.4a shows the amplitude spectra for the incident wave for the aluminium, acrylic and LDPE bars. This Figure shows that the upper frequencies where the amplitudes approach zero vary for the different materials. Since the striker length for the aluminium is the longest, one would expect to see the amplitude spectrum being comprised of predominantly low frequency components with little in the high range. Through similar reasoning, the use of the shorter striker on the LDPE would result in larger high frequency components. If the amplitude spectrum for the LDPE is compared to that of the aluminium, the valid frequency band is much higher for the aluminium bars even though the striker is much longer. This indicates that the predominant factor that limits the upper value of the appropriate frequency band is the material behaviour rather than the striker length.

Figure 5.4b illustrates the difference in attenuation coefficients for the three materials. It can be seen that the attenuation coefficient values for the aluminium are quite low in comparison to the other two materials. The data for the LDPE indicates a high degree of attenuation. The data for the acrylic bars indicates that, although significant, the attenuation is less than that seen in the LDPE bars. This means that for the same initial wave, the amplitude of the wave at some distance along the bar length will be: nearly identical for the aluminium bars, somewhat reduced for the acrylics bars and very reduced for the LDPE bars.

Figure 5.4c shows the normalized phase speeds for the different materials. The phase velocities were divided by their initial value to allow the comparison of phase speeds for the different materials. The phase speeds for the aluminium bars are relatively constant throughout the 30 kHz frequency band. The normalized phase speed corresponds to an actual phase speed of approximately 5000m/s. This indicates that all the components of the wave are moving at the same speed, and therefore, little dispersion occurs. The acrylic normalized phase speeds differ over the 20 kHz frequency band. The initial normalized phase speed corresponds to an actual phase speed of 2115 m/s. Figure 5.4c indicates that the phase speed increases with the frequency. This results in different wave components

traveling at different speeds causing the wave to disperse as it propagates along the rod. The greatest amount of dispersion occurs with the LDPE rods. Figure 5.4c indicates that there is a wide variability among normalized phase speeds, from 1.0 to 1.1, over the 4.5 kHz range. This corresponds to a range of phase speeds from 843 m/s to 930 m/s. It should be pointed out that even though values are shown up to 10 kHz, the valid range of data for the LDPE rods has an upper bound of 4.5 kHz with numerical errors increasing at frequencies beyond that value. This is the cause of the instabilities seen after 4.5 kHz in Figure 5.4c.

5.3 Measured and Predicted Wave Comparison

To confirm the validity of the wave analysis in the bar, an investigation of the differences between the measured wave and predictions using the wave analysis were undertaken. For this study, LDPE rods were instrumented at two different locations along the rod. With this arrangement, the propagation of the wave measured at one point can be predicted at a second point by applying the propagation coefficient previously determined. At this second point, the predicted and measured waves can then be compared to identify any differences. The gauge stations were located at 1.219m and 0.660m from the bar end. A 20mm striker was used to avoid any superposition of the incident and reflected waves at the 0.660m location. Unfortunately, due to the slow wave speed and increased dispersive effects, a small amount of superposition occurred. The 20 mm striker is the smallest projectile that can be safely used and so could not be reduced in length any further. The superposition of the waves occurs at the end of the signal after the majority of the wave has passed.

Figure 5.5a shows the predicted and measured waves at the second strain gauge station. The predicted wave has been calculated by propagating the original wave measured at the first gauge using an elastic wave analysis. It is obvious that without the application of the attenuation coefficient, the magnitudes of the waves are quite different. This is illustrated in the difference curve which takes the original wave and subtracts the measured wave. The dispersive effect on the wave is also illustrated in the figure. It can be seen that by not using a dispersive relationship in propagating the wave, the shape of the predicted and measured waves are quite different. Figure 5.5b shows the wave predicted with the dispersive relationship, identified previously, and the measured wave. In comparison to Figure 5.5a, the

predicted wave more accurately represents the actual wave at that point. The attenuation and dispersion of the wave appears to have been correctly modelled to allow an accurate prediction of the measured wave.

The validity of the dispersive relationship can be analysed further by comparing multiple reflections of the wave. Figure 5.6a shows the raw data for a free end test for the LDPE rods. The passage of the incident wave and reflections have been identified. If the incident wave, reflected off the free end, is propagated four times the bar length (assuming the gauge station is in the middle of the bar), it can then be compared to the second reflection off the free end. This assumes that there are no losses, and that a total reflection occurs at both the free end and impacted end. In this case, comparing these two waves will then check the dispersive relationship over 4.877m instead of 0.559m. As seen in Figure 5.6a, the amount of dispersion of the wave makes it difficult to determine if any superposition of waves occur. Figure 5.6b illustrates the predicted wave (the first reflected wave propagated 4.877m) and the measured wave (second reflection off the free end). As can be seen, the dispersive relationship over predicts the attenuation of the wave. The difference in the two waves is somewhat consistent over the duration of the wave indicating that dispersive relationship accurately predicts the wave shape. In other words, the predicted pulse has a similar shape, but smaller amplitude, compared to the measured pulse.

In general, there is excellent correspondence of the predicted and measured waves for both cases. This clearly validates the use of the viscoelastic wave analysis. In addition, the results confirm the validity of the experimental method chosen to determine the dispersive relationship.

5.4 ELVS

An enhanced laser velocity system (ELVS) was recently acquired by the University of Waterloo. This system has the ability to measure the rapid movements of the rod ends. Figure 5.7 is a schematic which illustrates the five components which comprise the ELVS system. The laser generates a sheet of light that is diverging at approximately 60 degrees . The lens collimates the laser sheet so that the light rays are parallel. The aperture is used to block out the part of the beam which is not required and is also used to collimate the beam. The focusing lens focuses the laser sheet to a point which is aimed at the detector. The

detector is a power meter that measures the amount of light projected onto the its surface and outputs a measurable voltage accordingly. As an object passes through the laser sheet, varying amounts of light are blocked out. The displacement of the object causes a change in the light intensity and, therefore, voltage. If the output of the detector is recorded with an oscilloscope, a time versus voltage (and hence displacement) curve can be obtained. Through differentiation of this curve, the velocity and acceleration history of the projectile can be derived.

As a means of validation, this system was setup to measure the displacement of the bar ends during the free end test. The velocity of the bar end can be measured and compared to the predicted velocity obtained through the data reduction program. As a baseline, the tests were initially carried out on the aluminium bars. Since there is little attenuation or dispersion of the wave, the velocity at the bar end can be accurately predicted and validated with the measured velocity at the bar ends.

Figure 5.8a shows the raw data obtained from a test using the ELVS and strain gauge systems. The slope of the output from the detector can be related to the velocity of the bar end. Multiple reflections are shown to illustrate the displacement of the bar end. It can be seen that as the incident and reflected waves are superimposed on the bar end, it moves at some velocity dictated by equation (3.24). As the pulse is reflected from the bar end, it stops moving until the next wave (the result from the original wave being reflected off the impacted end) reaches the bar end.

Figure 5.8b shows the results for the tests using aluminium bars for two different striker velocities. The first observation is that the rapid rise times of the predicted results are not indicated by the ELVS results. This unfortunately represents a deficiency in the resolution of the ELVS system. The ELVS system was designed to measure the displacement of an object over a much larger distance than the bar moves during the rise time of the pulse. Normally, the ELVS would be measuring displacement between 20mm and 25 mm. If during the rise time the acceleration of the bar is assumed constant (a best case scenario), the end of the bar will displace less than 0.25mm. This value is lower than the minimum distance that the ELVS can resolve in such a small time. Therefore, only the peak values obtained from the ELVS can be compared to the predicted results. As indicated in Figure 5.8b, the peak velocity measured compares well with the peak velocity predicted for

both cases. The predicted and measured velocity magnitudes are within 0.25 m/s of each other.

Figure 5.8c shows the values for the acrylic bars measured at three different velocities. Once again the ELVS system could not correctly capture the displacement of the bar during the rise time. Comparing the peak values of the measurements, the results between the two methods differ by approximately 0.2 m/s.

Figure 5.8d shows the values for the LDPE bars measured at two different velocities. Only two velocities were attainable due to limitations with the gas gun and bearing setup. Comparison between the ELVS results and the predicted results lead to a difference in velocities of 0.15 m/s.

The excellent correspondence between the ELVS and predicted velocity magnitudes further validate the CSHB. Although limited by sensor resolution, the differences in velocities for all three materials are well within experimental tolerances.

5.5 Material Test Results

Three different materials were tested. Polycarbonate samples were tested with both the acrylic and aluminium bars and the results compared to published data. Ballistic gelatin samples were fabricated and tested. These results were also compared to data from Sawas et al. [1998]. RTV silicon was the third material tested. An analysis of the RTV silicon results identifies various limitations and special considerations for testing soft rubbers

5.5.1 Polycarbonate Results

Polycarbonate specimens are a good means of verifying the operation of the CSHB with aluminium bars since published data describing its high strain rate properties exist. Additionally, the properties of polycarbonate are more consistent over different batches than other polymers. Polycarbonate is readily available in rods and can be conventionally machined allowing quick fabrication of samples. Tests on polycarbonate specimens were also performed with acrylic bars. The samples were 6.4mm diameter by 6.4mm long for the tests using the aluminium bars and 4.0mm by 4.0mm for the tests using the acrylic bars. The results from these tests are compared to the results published by Sawas et al. [1998] generated using a similar acrylic Hopkinson bar apparatus.

Figures 5.9a, 5.9b and 5.9c show the results from the test on the polycarbonate using the aluminium bars at rates of 600, 1200, and 1440 s^{-1} respectively. The amount of pressure required to overcome the static friction within the gas gun makes it difficult to attain lower strain rates. Inspecting Figure 5.9a, it appears that the dynamic Elastic Modulus of the two tests differ. It is quite obvious that the slope of Sawas et al. result is greater than that of the test. The shapes of the curves are quite similar as the material begins to yield and plastic flow begins. An inspection of the region, where plastic flow is occurring, results in similar, yet offset results. The magnitude of the flow stress for Sawas et al. is marginally higher than the current results. An inspection of Figure 5.9b results in similar observations. The slopes of the stress strain curves are similar with the current results being less steep than the published data. As before, the shapes of the curve during the transition from the elastic to plastic regions are similar. The flow stress of the published data is higher than that of the current results. Figure 5.9c shows similar results at strain rates of 1200 s^{-1} and 1700 s^{-1} for the published results, and 1440 s^{-1} for the current results. The higher flow stress from Sawas et al. can be partially attributed to the difference in strain rates.

There are a number of reasons why the results of Sawas et al. do not exactly match those currently obtained. A significant factor is the method chosen to manipulate the waves when performing the data reduction. Sawas et al., use a time domain approach where the signals are shifted in time so that their starting times are aligned. This method of wave manipulation, in combination with a conventional type of strain, stress and strain rate analysis, can result in differing dynamic Elastic Moduli. By propagating the waves to the interfaces, the starting times of the waves are somewhat arbitrary. Since the resulting curves have similar shapes, a difference in strain in the elastic region of the curve would offset the rest of the curve accounting for the difference. Allowing for the differences in the test procedures and data reduction, there is an excellent agreement with the published data of Sawas et al.. This suggests that the CSHB is working properly.

Figures 5.10a and 5.10b show the results of the tests performed on the polycarbonate samples with the acrylic bars at strain rates of 1200 s^{-1} and 1700 s^{-1} respectively. The dynamic Elastic Moduli shown in Figure 5.10a and 5.10b for the current results are less than the published data. Once again, the transition from the elastic to plastic regions results in similar curves with the flow stress being less for the tests using the acrylic bars.

Figure 5.11 compares Sawas et al. result and the results obtain from the aluminium and acrylic bars at a strain rate of 1200 s^{-1}. The curves differ by a significant amount in the elastic region. The transition from elastic to plastic flow is higher for the acrylic bars than the aluminium bars. However, the resulting flow stress is lower for the acrylic bars. This comparison illustrates the effect that the rise time of the pulse has on polycarbonate. The rise time of the wave in the acrylic bars is much longer than in aluminium bars and this allows the specimen to deform more without an increase in stress, resulting in an apparent lower flow stress at a given strain.

5.5.2 Ballistic Gelatin

Ballistic gelatin is a gelatinous substance that is commonly used to simulate human muscle and flesh in high impact testing. In order for accurate numerical simulation, knowledge of its response to high rates of strain is necessary. The gelatin is usually formed by adding the powdered gelatin to water in an appropriate ratio. The gelatin tested currently had been prefabricated into two inch by six inch square blocks using a powder to water ratio of 1:20.

5.5.2.1 Preparation

The properties of ballistic gelatin prevent accurate coring and cutting of samples so a different method of producing CSHB specimens was developed. Ballistic gelatin was melted and cast to a sheet of approximately the required thickness. Once cooled, the sheet was placed between two plates of glass separated by slip gauges of the desired specimen length. Hot water was poured over the plates, melting the ballistic gelatin, allowing it to conform to the thickness of the slip gauge. Once the gelatin cooled and the glass plates removed, specimens were punched out using a punch of the desired diameter. Since the specimens are usually quite thin, very little deformation of the sample occurs during the punching. For the current tests, the specimens were 2mm long and 12.7 mm in diameter.

5.5.2.2 Results

Figure 5.12 shows the results from the current test as well as results obtained from Caillou et al. [1994] for ballistic gelatin. The results displayed are in terms of true stress and true strain. These curves are noticeably different than the curves for polycarbonate. The

nature of the ballistic gelatin does not allow for a great deal of plasticity and it is generally considered to be a hyper-elastic material. This means that instead of the material flowing plastically, it undergoes non-linear elastic deformation until failure. The material shows a large degree of strain rate dependency. Unfortunately the results from the Caillou et al. tests are mostly for lower strain rates. One of the current results at 1200 s^{-1} lies between the 750 s^{-1} and 1350 s^{-1} results from Caillou et al. suggesting that the current results are accurate. The series of tests continued with specimens being subjected to strain rates of 2500 s^{-1} and 4000 s^{-1}. The curves show significant strain rate dependence with the trend identified by the Caillou et al. results being continued with the current results.

5.5.3 RTV Silicon

The RTV silicon tested is a pink, rubber like, material that is also being used simulate muscle tissue in high impact testing. Similar to ballistic gelatin, RTV silicon has a high rate of dependence on strain rate.

5.5.3.1 Preparation

Although similar to ballistic gelatin, the RTV silicon cannot be melted and cast into sheets of a desired thickness. The material comes in pre-cast sheets of 12.7mm thickness. The samples were prepared using a 12.7mm coring tool and then cut to 2mm in length. The length of the specimens was relatively small compared to hard polymers to account for the slow wave speed in the material.

5.5.3.2 Results

Figure 5.13 shows the results from the tests. For clarity, only four of the twenty tests performed are shown. These curves were chosen to represent maximum strain rates from 1500 s^{-1} to 3300 s^{-1}. Similar to the ballistic gelatin, the RTV silicon shows a large degree of strain rate dependence. The curves indicate that as the maximum strain rate increases, larger stresses are developed for similar strains. As the material compresses it becomes more rigid causing a sharp increase in stress for a small increase in strain. An unfortunate problem that arises when attempting to characterize hyper-elastic materials is the change in strain rate during the test. Figure 5.14 shows typical strain rate curves for polycarbonate and RTV

silicon. It can be seen from the Figure that the polycarbonate specimens have a stable strain rate for the majority of the test. An inspection of the strain rate curve for the RTV silicon indicates that the specimen did not reach a stable strain rate. This means that over the duration of the test, the specimen was subjected to varying strain rates at different amounts of strain and stress in the sample. The strain rates quoted above are the maximum values found for each test.

The ability of the apparatus to characterize the high strain rate behaviour of low acoustical impedance is clearly identified with these tests. The excellent impedance match between the bar and specimen material increases the signal recorded in the transmitter bar allowing softer materials to be tested. The consistent and repeatable results that were achieved clearly validates the apparatus when testing soft materials.

5.5.4 Sources of Error

A variety of additional factors affect the outcome of a test. When a specimen is compressed, heat is generated within the specimen from the work being done. This rise in temperature may affect the stress strain curves if the temperature becomes elevated. In general, this is not a problem in Hopkinson Bar tests on metals but may be a problem, worth exploring, in nonmetals.

Although lubrication at the specimen interfaces help to reduce friction allowing the material to expand freely, some friction still exists. This friction can prevent the material from expanding freely causing large stress gradients to be formed within the sample. Again, this may be more of a problem with nonmetals than it is with metals.

Variability among the composition of the sample material can have a large affect on the results from the test. Careful control of the material composition should be taken where possible to ensure consistent and accurate results.

The above considerations can lead to some experimental scatter. Figure 5.15 shows two test performed on polycarbonate at the same strain rates. Although similar test conditions are used, the results differ slightly. However, for most high strain rate testing this would be considered excellent repeatability. The degree of experimental scatter shown is typical of repeated tests under the same conditions.

5.6 Summary

Three separate approaches were undertaken to validate and investigate the apparatus when using bars made from a viscoelastic material. Predicted waves were compared to waves measured at a second strain gauge location on the bar and to waves reflected off the impacted end of the bar. Comparison between the predicted and measured waves resulted in a good correspondence.

The second method of validation used the enhanced laser velocity system (ELVS) to measure the velocity at the ends of the bars. It was found that, although limited by sensor resolution, the magnitudes of the predicted and measured bar interface velocities matched within experimental tolerances.

The third method of validation compared stress strain curves for polycarbonate and ballistic gelatin tests to published results. Good correspondence was found between the published and measured data.

Chapter 6

Closure

6.1 Conclusions

The object of the current work was to develop and validate an apparatus for testing non-metallic materials at high strain rates. A Hopkinson bar apparatus that uses acrylic bars was found to be appropriate for the testing of materials of lower acoustical impedance. With the use of a gas gun, velocity range of the striker bar allowed strain rates from 500 s^{-1} to 4000 s^{-1} to be achieved. However, the attenuation and dispersion of the waves in the acrylic bars required the development of spectral analysis techniques to compensate for changes to the stress wave as it propagates along the bars. If these corrections are not made, large errors can result and the behaviour of the test material will not be identified correctly.

Different methods of determining the dispersive relationship were discussed and an experimental method was adopted. The use of the experimental method allows for the dispersive relationship of the bar material to be determined without an *a priori* knowledge of the material properties. This allows easy calculation of the propagation coefficient and compensates for any inconsistencies between the theoretical and actual material behaviour. In addition, through the application of spectral methods to the incident and transmitter bars signals, a more accurate representation of the test conditions is achieved. This method of wave manipulation allows consistent results to be achieved by removing the subjectivity inherent to conventional methods. A comparison of the dispersive relationships for aluminium (6061-T6), acrylic and low density polyethylene identified the degree of

attenuation and dispersion for viscoelastic materials. The results indicate that elastic wave propagation, although valid for aluminium bars, is invalid for acrylic and LDPE bars. Through the application of three different test methods, the approach of determining the dispersive relationship was validated.

The use of two gauge stations along the bars allows comparison between the predicted and measured waves. The application of the dispersive relationship over the distance between the gauges resulted in good correspondence between the measured and predicted results. This type of validation was extended to study the nature of the waves over multiple reflections off the ends of the bars.

The magnitudes of the predicted and measured velocities at the bar ends were within experimental tolerances. However, the limitations on the resolution of the ELVS system disallowed any comparison of wave shape between the predicted and measured results.

Testing materials with published behaviour allowed the apparatus to be further validated. Tests on polycarbonate specimens were performed using both the aluminium and acrylic bars and compared with each other and results from Sawas et al. [1998]. There is good agreement between the tests performed with the aluminium bars and those from Sawas et al.. The differences between the results obtain with the aluminium bars and acrylic bars can be attributed to the rise time of the waves applied. The acrylic bars were further validated by testing ballistic gelatin. The general trend that was identified from the results from Caillou et al. [1994] matched the present results.

A general description of the test results for RTV silicon was given. This material was tested to indicate the capability of the apparatus and help identify any special test considerations. The strain rate dependency of the material was easy to distinguish with tests conducted from 1500 to 3300 s^{-1}.

6.2 Recommendations

In order to further develop the Hopkinson bar, the following recommendations are being made:

1. Further enhancement of the gas gun would allow greater striker velocities and, as a result, greater strain rates. This enhancement could take the form of a longer cylinder or the development of a fast acting spool type valve.
2. An augmentation of the ELVS system to increase the resolution would allow a better comparison of the predicted and measured waves. Additionally, the increased sensitivity would allow the ELVS system to be applied while testing specimens. The ELVS system could be arranged to measure the difference in velocities between the bar interfaces allowing a further verification of the achieved strain rate. The sensitivity of the system can be increased with the application of different focusing lenses and mirrors.
3. Studies on soft, rubber like, material will require further investigation of strain rate stability and how it relates to the stress strain curves. The varying strain rates seen in the RTV silicon tests illustrate the degree of instability that can arise during testing.

References

Bacon and Brun [2000]

Bacon, C., Brun, A., "Methodology for a Hopkinson Test with a Non-Uniform Viscoelastic Bar", *International Journal of Impact Engineering*, Volume 24, 2000, pp. 219-230.

Bacon [1998]

Bacon, C., "An Experimental Method for Considering Dispersion and Attenuation in a Viscoelastic Hopkinson Bar", *Experimental Mechanics*, Volume 38, 1998, pp. 242-249.

Caillou et al. [1994]

Caillou, J.P., Dannawi, M., Dubar, L., Wielgosz, C., "Dynamic Behaviour of a Gelatine 20% Material Numerical Simulation", *PASS 94*, pp. 325-331.

Chapra and Canale [1988]

Chapra, S., Canal, R., *Numerical Methods for Engineers 2nd Edition*, McGraw-Hill, New York, 1988.

Cheng et al. [1998]

Cheng, Z.Q., Crandall, J.R., Pilkey, W.D., "Wave Dispersion and Attenuation in Viscoelastic Split Hopkinson Pressure Bar", *Shock and Vibration*, Volume 5, 1998, pp. 307-315.

Chen et al. [1999]

Chen, W., Zhang, B., Forrestal, M.J., "Split Hopkinson Bar Techniques for Low Impedance Materials", *Experimental Mechanics*, Volume 39, 1999, pp. 81-85.

Chen et al. [2000]

Chen,W., Lu, F., Zhou, B., "A Quartz-Crystal Embedded Split Hopkinson Pressure Bar for Soft Materials", *Experimental Mechanics*, Volume 40, 2000, pp. 1-6.

Chree [1889]

Chree, C., "The Equations of an Isotropic Elastic Solid in Polar and Cylindrical Coordinates, Their Solutions and Applications", *Cambridge Philosophical Society Transactions*, Volume 14, 1889, pp. 250-369.

Dally and Riley [1991]

Dally, J., Riley, W., *Experimental Stress Analysis 3rd Edition*, McGraw- Hill, New York, 1991.

Davies [1948]

Davies, R.M., "A Critical Study of the Hopkinson Pressure Bar", *Philosophical Transactions Royal Society London, Series A*, Volume A240, 1948, pp. 375-475.

Davies and Hunter [1962]

Davies, E.D.H., Hunter, S.C., "The Dynamic Compression Testing of Solids by the Method of the Split Hopkinson Pressure Bar", *Journal of the Mechanics Physics Solids*, Volume 11, 1963, pp. 155-179.

Dioh et al. [1993]

Dioh, N.N., Leevers, P.S., Williams, J.G., "Thickness Effects in Split Hopkinson Pressure Bar Tests", *Polymer*, Volume 34, 1993, pp. 4230-4234.

Dioh et al. [1994]

Dioh, N.N., Ivankovic, A., Leevers, P.S., Williams, J.G., "High Strain Rate Behaviour of Polymers", *Journal de Physique IV Supplement au Journal de Physique III V4*, C8, 1994, pp. 119-124.

Dioh et al. [1995]

Dioh, N.N., Ivankovic, A., Leevers, P.S., Williams, J.G., "Stress Wave Propagation Effects in Split Hopkinson Pressure bar Tests", *Proceedings of the Royal Society of London-A-Mathematical and Physical Sciences*, Volume 449, 1995, pp187-204.

Doyle [1989]

Doyle, J.F., *Wave Propagation in Structures-An FFT Based Spectral Analysis Methodology*, Springer-Verlag, New York, 1989.

Follansbee [1985]

Follansbee, P., "The Hopkinson Bar", *Metals Handbook, American Society for Metals*, Volume 8, 1985, pp. 198-203.

Follansbee and Frantz [1983]

Follansbee, P.S., Frantz, C., "Wave Propagation in the Split Hopkinson Pressure Bar", *Journal of Engineering Material Technology*, Volume 105, 1983, pp. 61-66.

Gorham and Wu [1996]

Gorham, D.A., Wu, X.J., "An Empirical Method for Correcting Dispersion in Pressure Bar Measurements of Impact Stress", *Measurement and Science Technology*, Volume 9, 1996, pp. 1227-1232.

Hopkinson [1872]

Hopkinson, J, "On the Rupture of Iron Wire by a Blow", *Proceedings of the Manchester Literary and Philosophical Society*, Vol. XI, 1872, pp. 40-45.

Hopkinson [1914]

Hopkinson, B., "A Method of Measuring the Pressure Produced in the Detonation of High Explosives or by the Impact of Bullets", *Royal Society Philosophical Transactions*, A213, 1914, pp. 437-456.

Kolsky [1949]

Kolsky, H., "A Investigation of the Mechanical Properties of Materials at High Rates of Loading", *Proceedings of the Physical Society-Section B*, Volume 62, 1949, pp. 676-700.

Kolsky [1963]

Kolsky, H., *Stress Waves in Solids*, Dover Publications, New York, 1963.

Lundberg and Henhoz [1977]

Lundberg, B., Henchoz, A., "Analysis of Elastic Waves from Two Point Strain Measurements", *Experimental Mechanics*, Volume 17, 1977, pp. 213-218.

McCloy [1980]

McCloy, D., Martin, H., *Control of Fluid Power, Analysis and Design*, Ellis Horwood Limited, West Sussex England, 1980.

Meyers [1994]

Meyers, M., *Dynamic Behaviour of Materials*, Wiley-Interscience, New York, 1994.

Nicolet [1991]

Nicolet Pro Series User's Manual, Nicolet Instrument Corporation, Madison, 1991.

Pochhammer [1876]

Pochhammer, L., "On the Propagation Velocities of Small Oscillations in an Unlimited Isotropic Circular Cylinder", *Journal Reine Angewandte Mathematic*, Volume 81, 1876, pp. 324-336.

Press et al. [1992]

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., *Numerical Recipes in C 2nd Edition*, Cambridge University Press, Cambridge, 1992.

Sawas et al. [1998]

Sawas, O., Brar, N.S., Brockman, R.A., "Dynamic Characterization of Compliant Material Using an All Polymeric Split Hopkinson Bar", *Experimental Mechanics*, Volume 38, 1998, pp. 204-210.

Sogabe et al. [1995]

Sogabe, Y., Yokoyama, Ta., Yokoyama, Te., Nakano, M., Kishida, K., "Split Hopkinson Bar Method for Testing Materials with Low Characteristic Impedance", *Dynamic Fracture, Failure and Deformation ASME PVP*, Volume 300,1995, pp. 137-143.

Tyas and Watson [2001]

Tyas, A., Watson, A, "An Investigation of Frequency Domain Dispersion Correction of Pressure Bar Signals", *International Journal of Impact Engineering*, Volume 25, 2001, pp. 87-101.

Wang et al. [1994]

Wang, L., Labibes, K., Azari, Z., Pluvinage, G., "Generalization of Split Hopkinson Bar Technique to use Viscoelastic Bars", *International Journal of Impact Engineering*, Volume 15, 1994, pp. 669-686.

Zhao and Gary [1995a]

Zhao, H., Gary, G., "A Three Dimensional Analytical Solution of the Longitudinal Wave Propagation in an Infinite Linear Viscoelastic Cylindrical Bar. Application to Experimental Techniques", *Journal of Mechanics of Physical Solids*, Volume 43, 1995, pp. 1335-1348.

Zhoa and Gary [1995b]

Zhao, H., Gary, G., "Inverse Methods for the Dynamic Study of Nonlinear Materials with a Split Hopkinson Bar", *IUTAM Symposium on Nonlinear Waves in Solids, ASME*, 1995, pp. 185-189.

Strain Rate	Testing Methods	Category
10^7 10^6 10^5	-Explosives -Gun Systems -Pulsed Laser -Plate Impact	Ultrahigh Strain Rate
10^4 10^3	-Taylor Anvil Tests -Hopkinson Bar -Expanding Ring	High Strain Rate
10^2 10^1	-High-velocity hydraulic or pneumatic machines; cam plastomer	Low Strain Rate

Table 1.1: Testing Techniques According to Strain Rate

Raw Data Sample Period = dt_{Raw}
Propagation Data Sample Period = $dt_{Propagation}$
Number of Sample in Raw Data = N_{Raw}
Number of Samples in Propagation Data = $N_{Propagation}$

Sample Period	Number of Points	Action
$dt_{Raw} = dt_{Propagation}$	$N_{Raw} = N_{Propagation}$	- Values match, no manipulation necessary
	$N_{Raw} > N_{Propagation}$	-Trim excess data
	$N_{Raw} < N_{Propagation}$	-Pad end with zeros
$dt_{Raw} > dt_{Propagation}$	$N_{Raw} = N_{Propagation}$	-Resample data decreasing sample period
	$N_{Raw} > N_{Propagation}$	-Resample data decreasing sample period, trim excess data
	$N_{Raw} < N_{Propagation}$	-Pad end with zeros, resample data decreasing sample period
$dt_{Raw} < dt_{Propagation}$	$N_{Raw} = N_{Propagation}$	-Resample data increasing sample period
	$N_{Raw} > N_{Propagation}$	-Resample data increasing sample period, trim excess data
	$N_{Raw} < N_{Propagation}$	-Pad end with zeros, resample data increasing sample period

Note: sample period =1/sample rate. As the sample period decreases, the sample rate increases.

Table 4.1: Possibilities and Required Actions for Compatibility

Material	Elastic Modulus (GPa)	Density (kg/m^3)	Estimated Speed of Sound (m/s)	Acoustical Impedance (kg/m^2s)
Aluminium	68.9	2700	5052	1.364E+07
Acrylic	3.47	1178	1716	2.022E+06
LDPE	0.8	952	917	8.727E+05

Table 5.1: Properties for Bar Materials

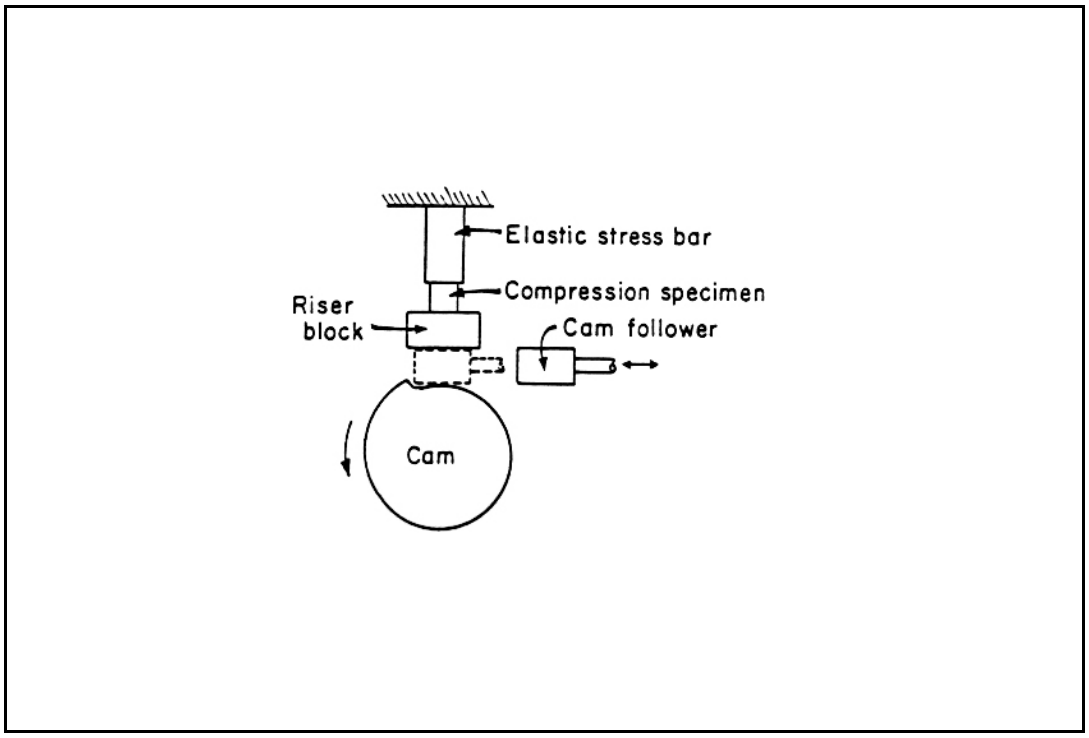


Figure 1.1: Schematic of Cam Plastomer, Meyers [1994]

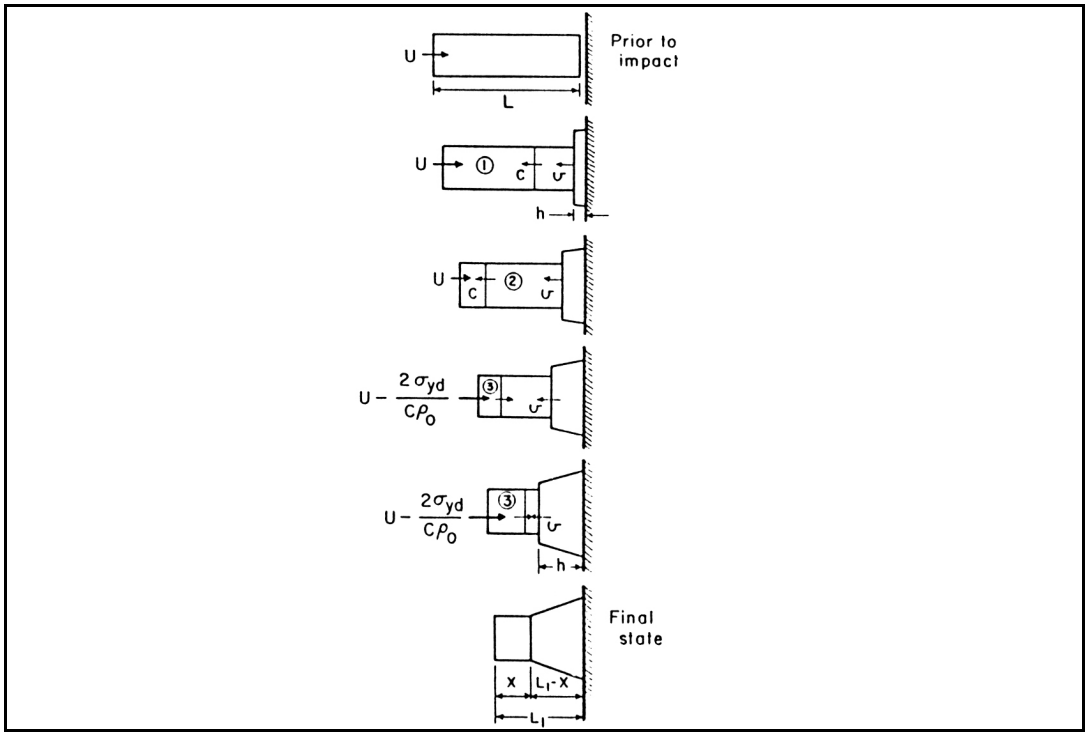


Figure 1.2: Taylor Impact Specimen, Meyers [1994]

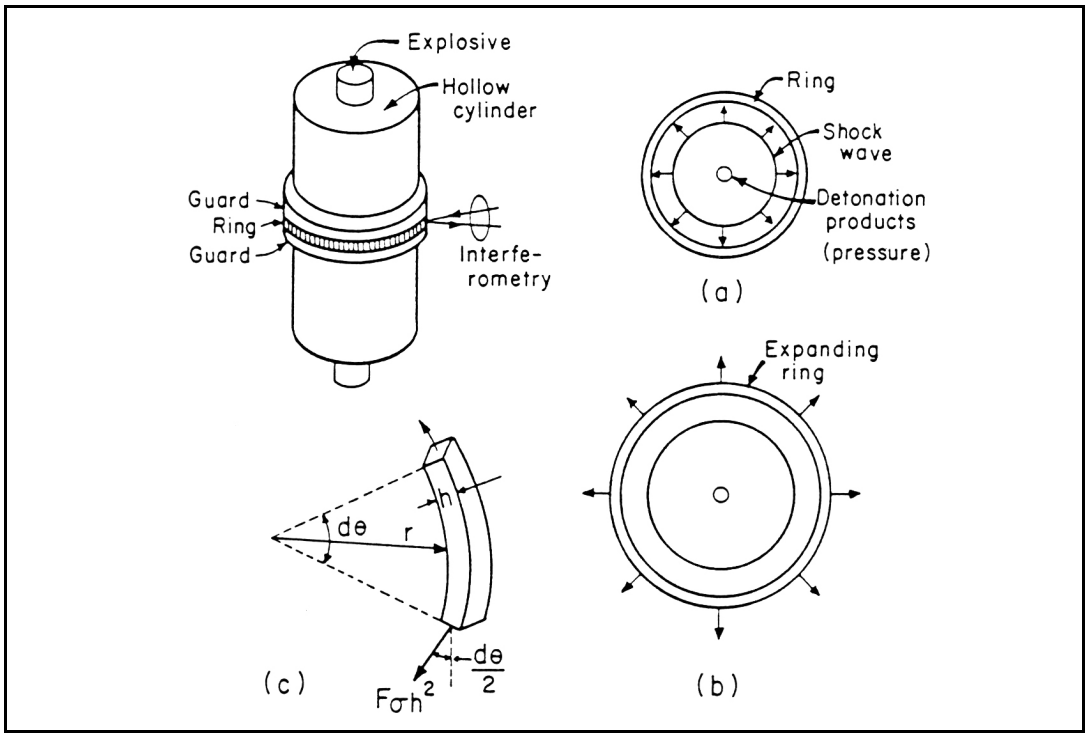


Figure 1.3:Expanding Ring Technique (a) steel block with explosive in core;(b) sections of cylinder just after detonation and during flight of ring;(c) section of ring, Meyers [1994]

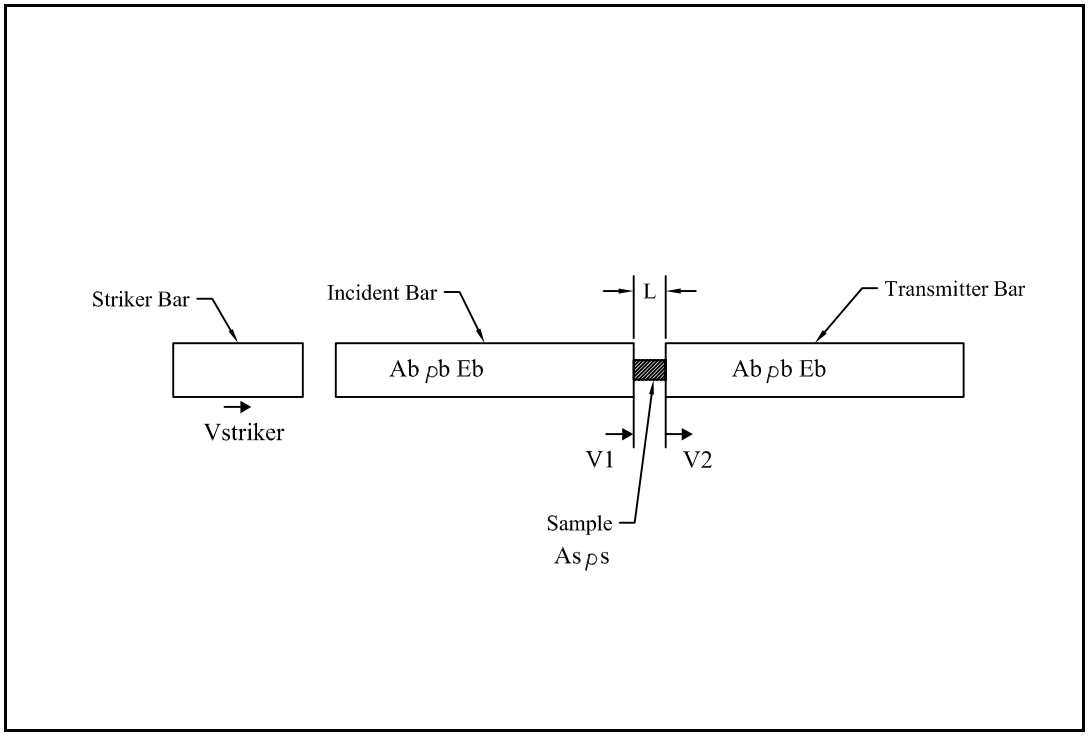


Figure 1.4: Schematic of Split Hopkinson Bar Apparatus

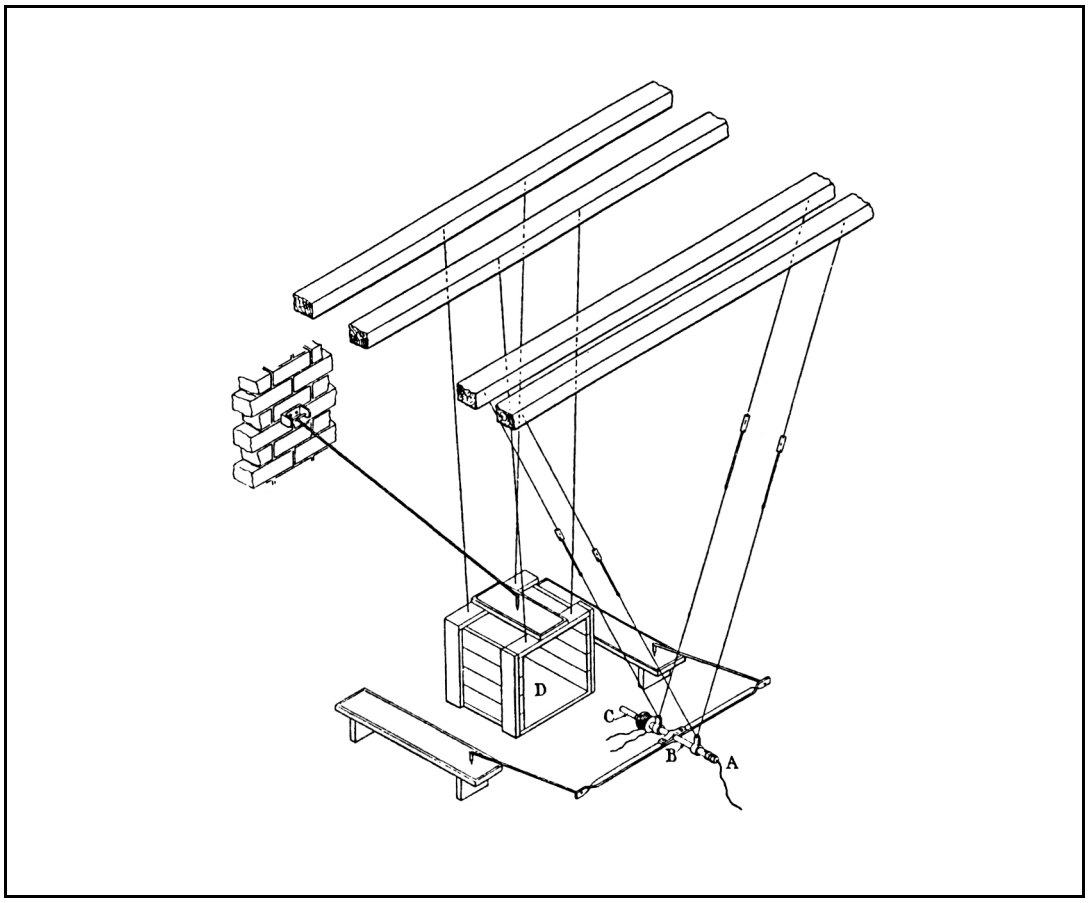


Figure 1.5: Schematic of First Pressure Bar, Hopkinson [1872]

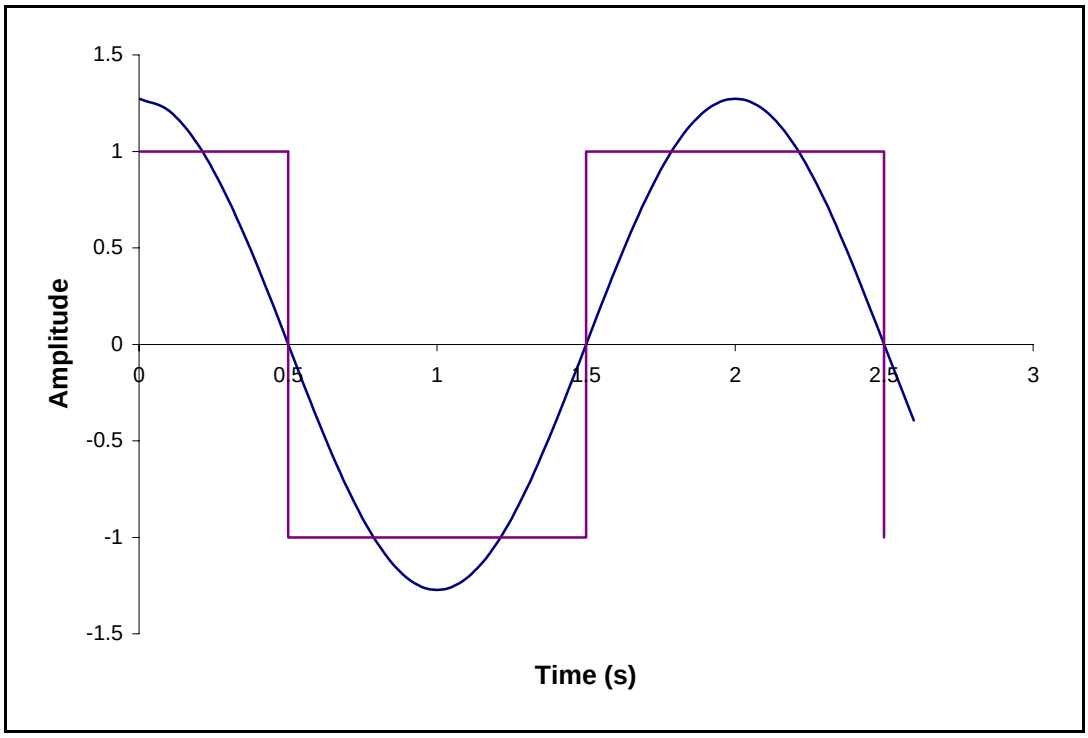


Figure 3.1a: Primary Fourier Component of a Square Wave

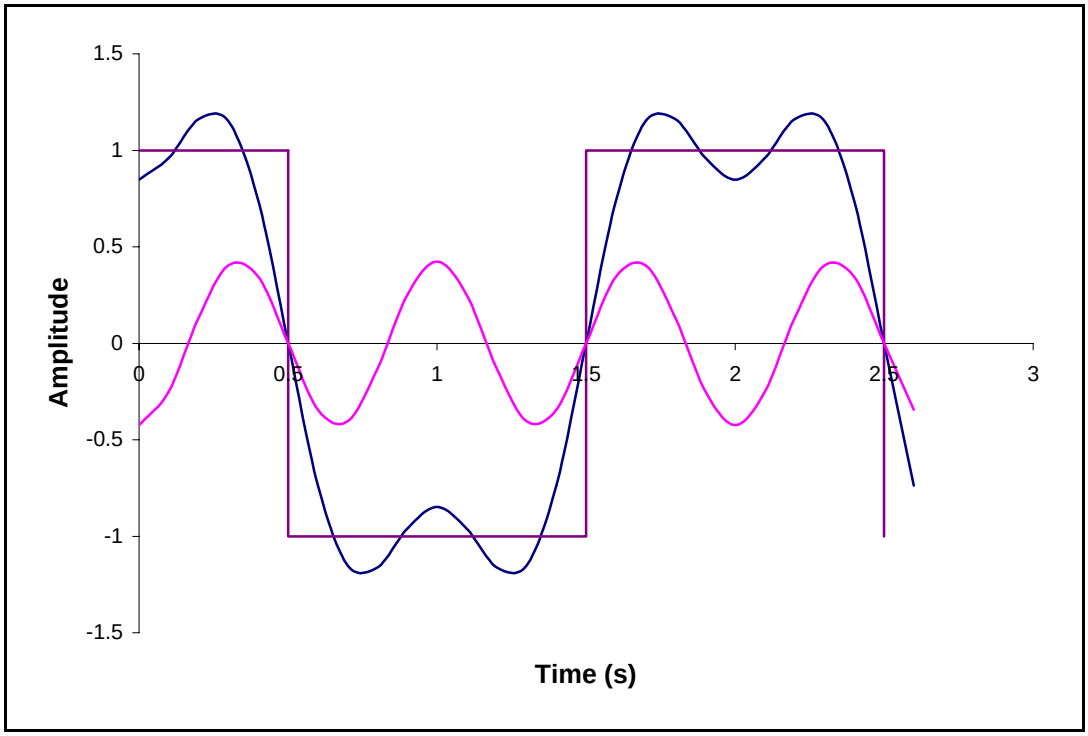


Figure 3.1b: Addition of Second Harmonic Component

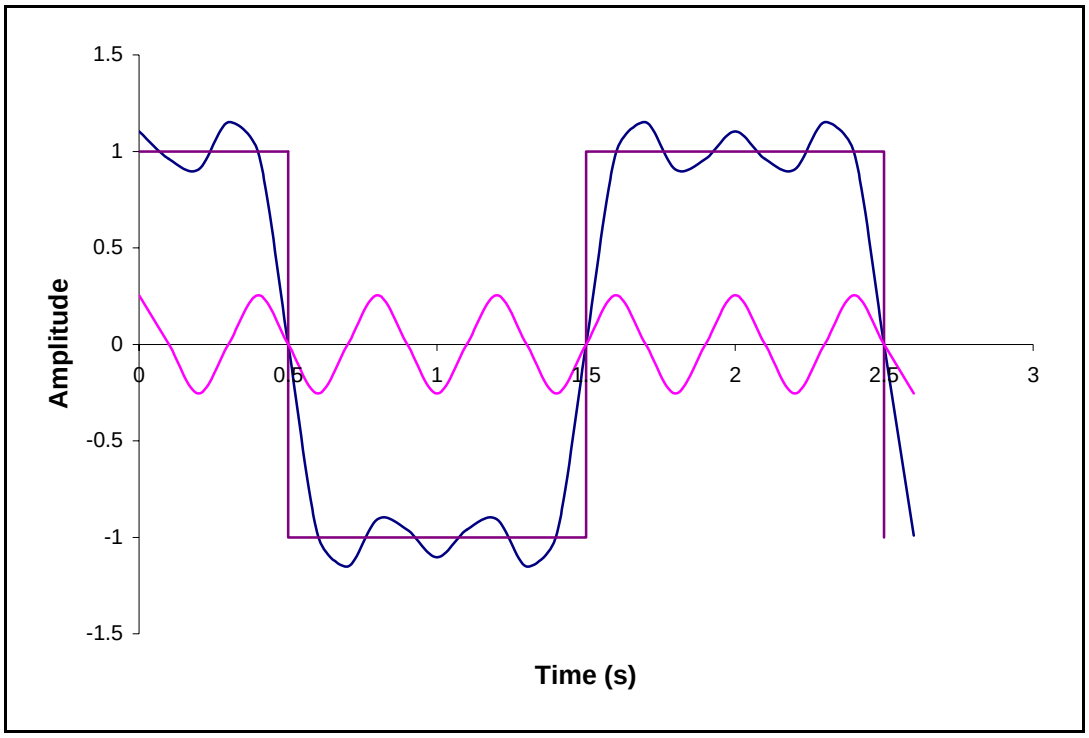


Figure 3.1c: Addition of Third Harmonic to Fourier Series

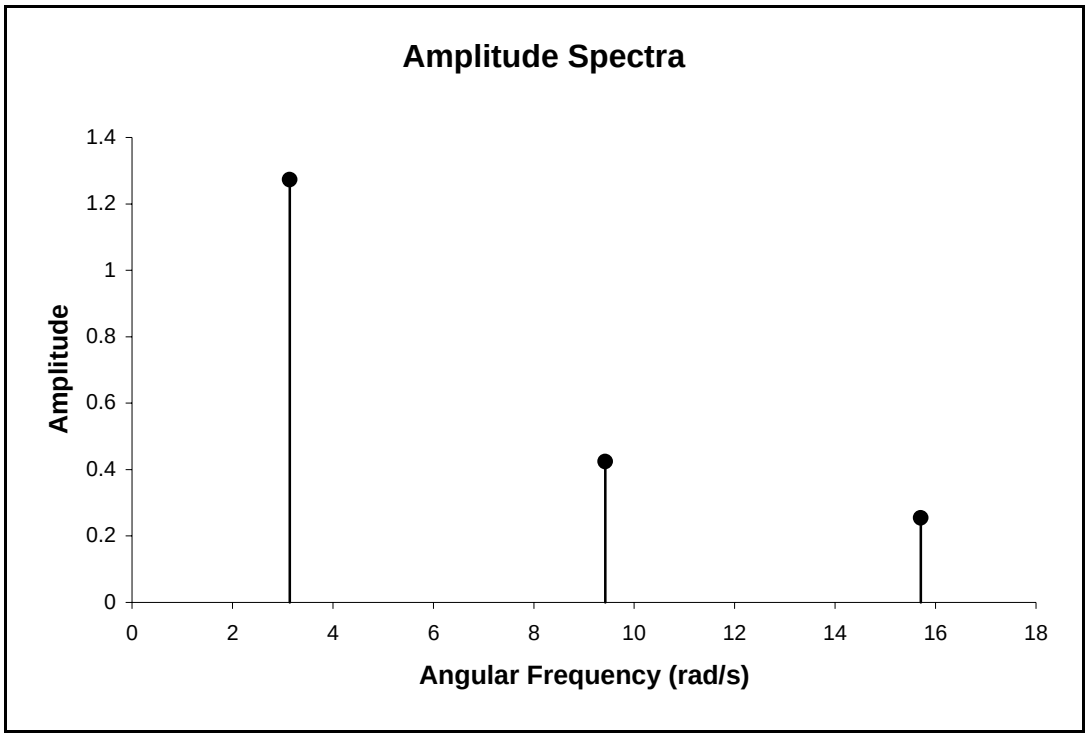


Figure 3.2a: Amplitude Spectra for First Three Terms

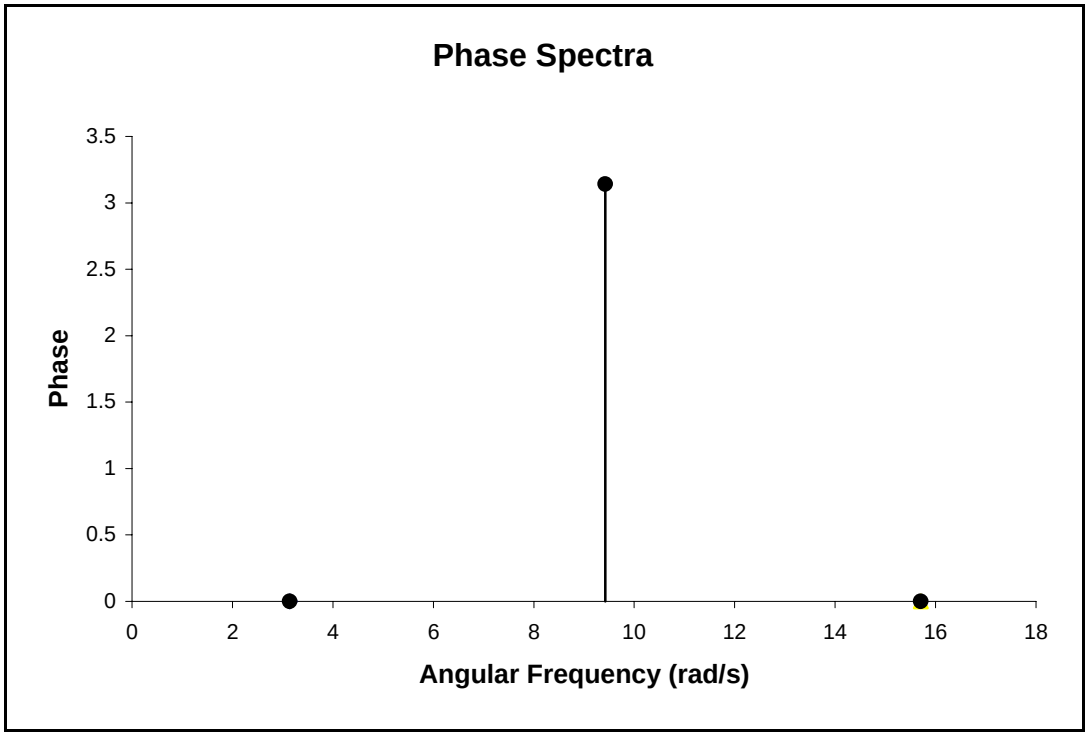


Figure 3.2b: Phase Spectra for First Three Terms

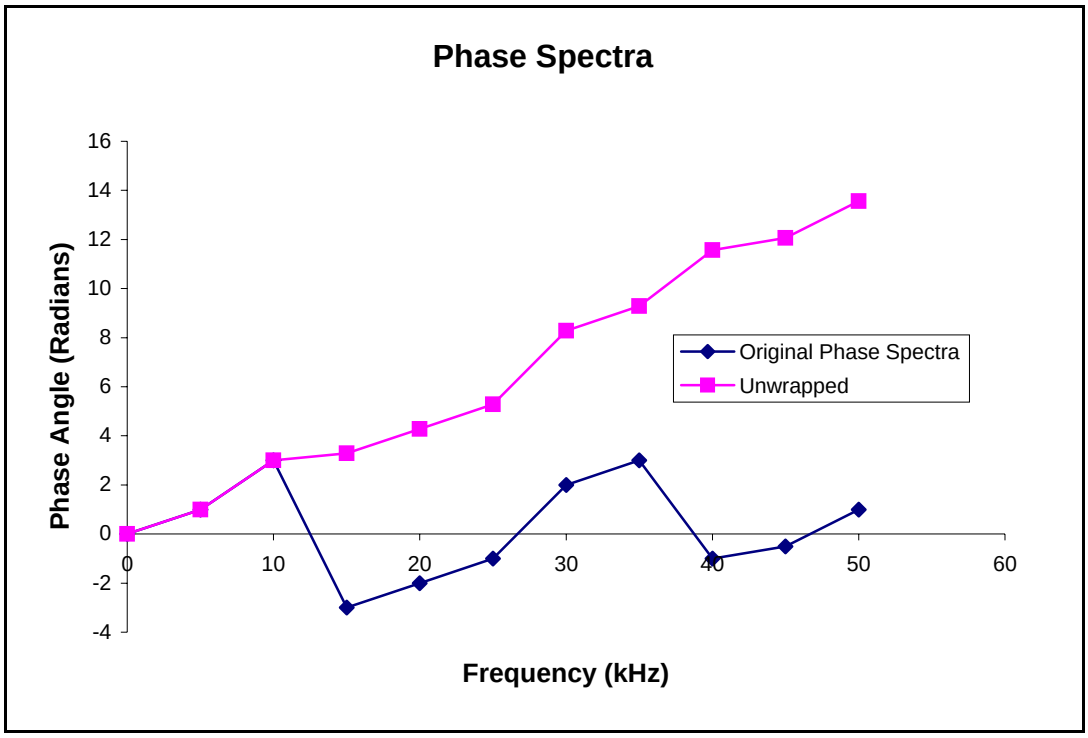


Figure 3.3: Unwrapping of Phase Spectra

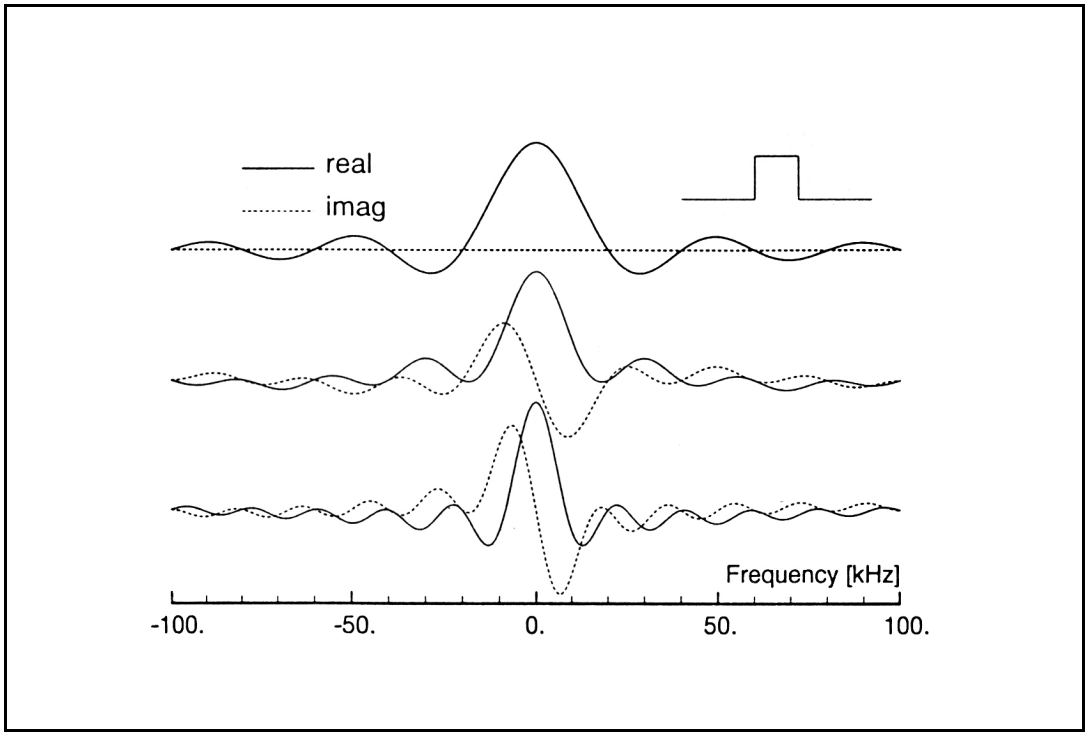


Figure 3.4: Real and Imaginary Components For a Square Pulse Subjected to Different Amount of Shift, Doyle [1989]

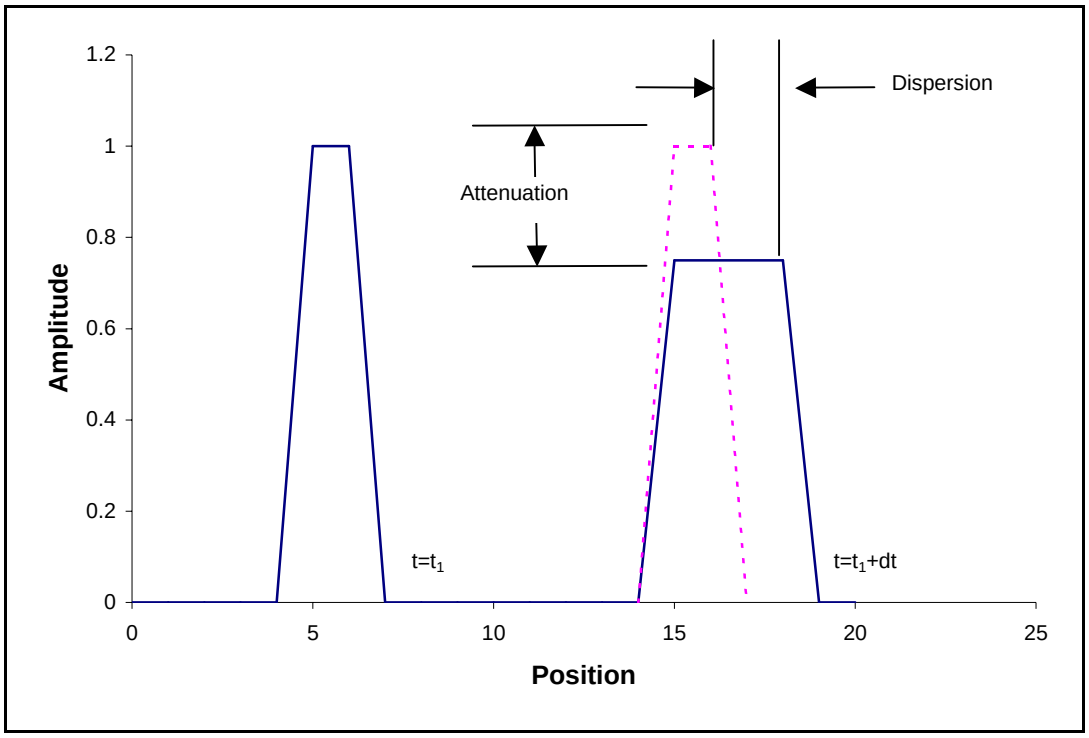


Figure 3.5: Illustration of the Effects of Dispersion and Attenuation

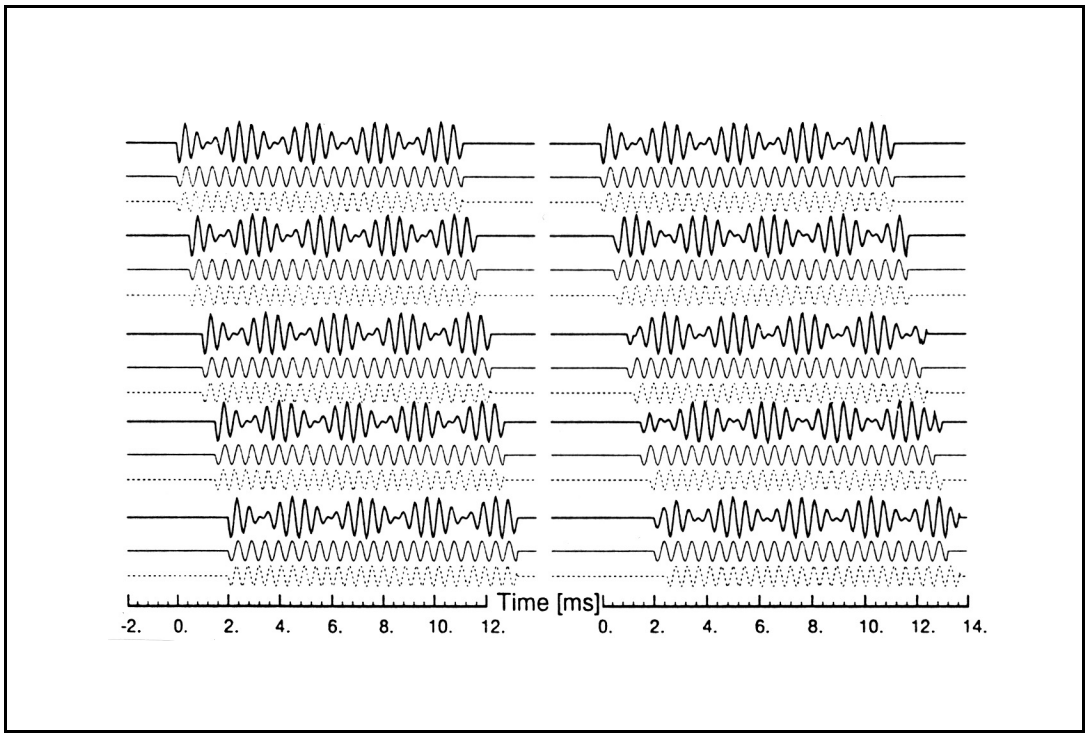


Figure 3.6: Segments of an Infinite Wave Train at Different Positions. Left: nondispersive system. Right: dispersive system, Doyle [1989]

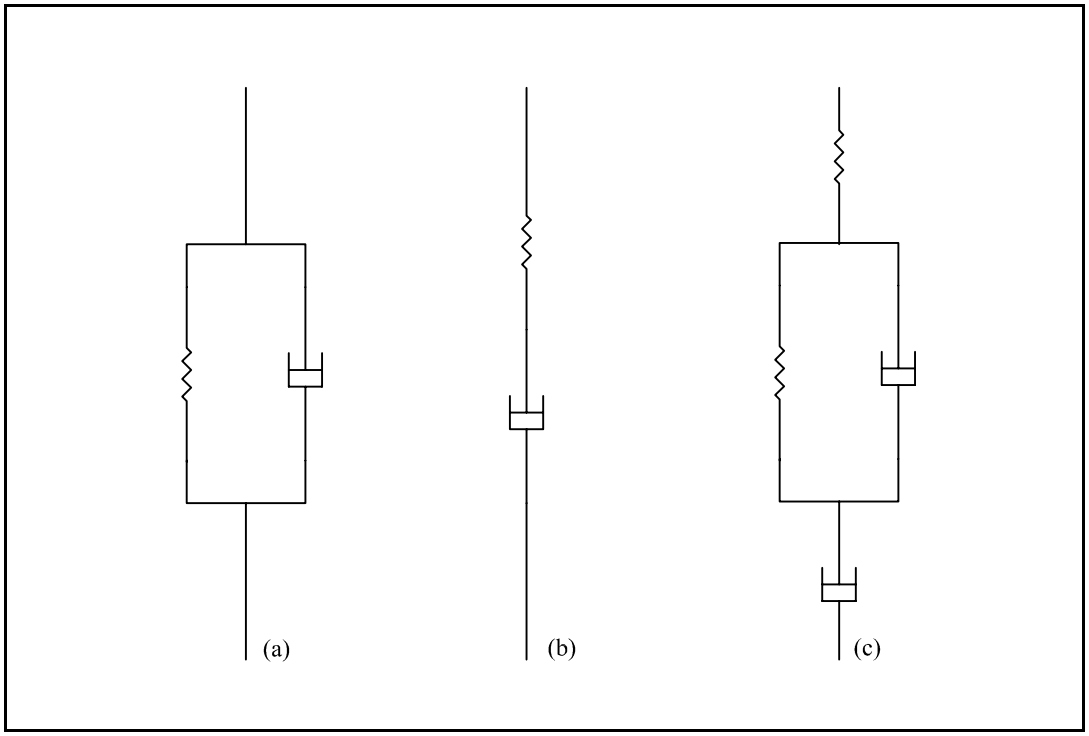
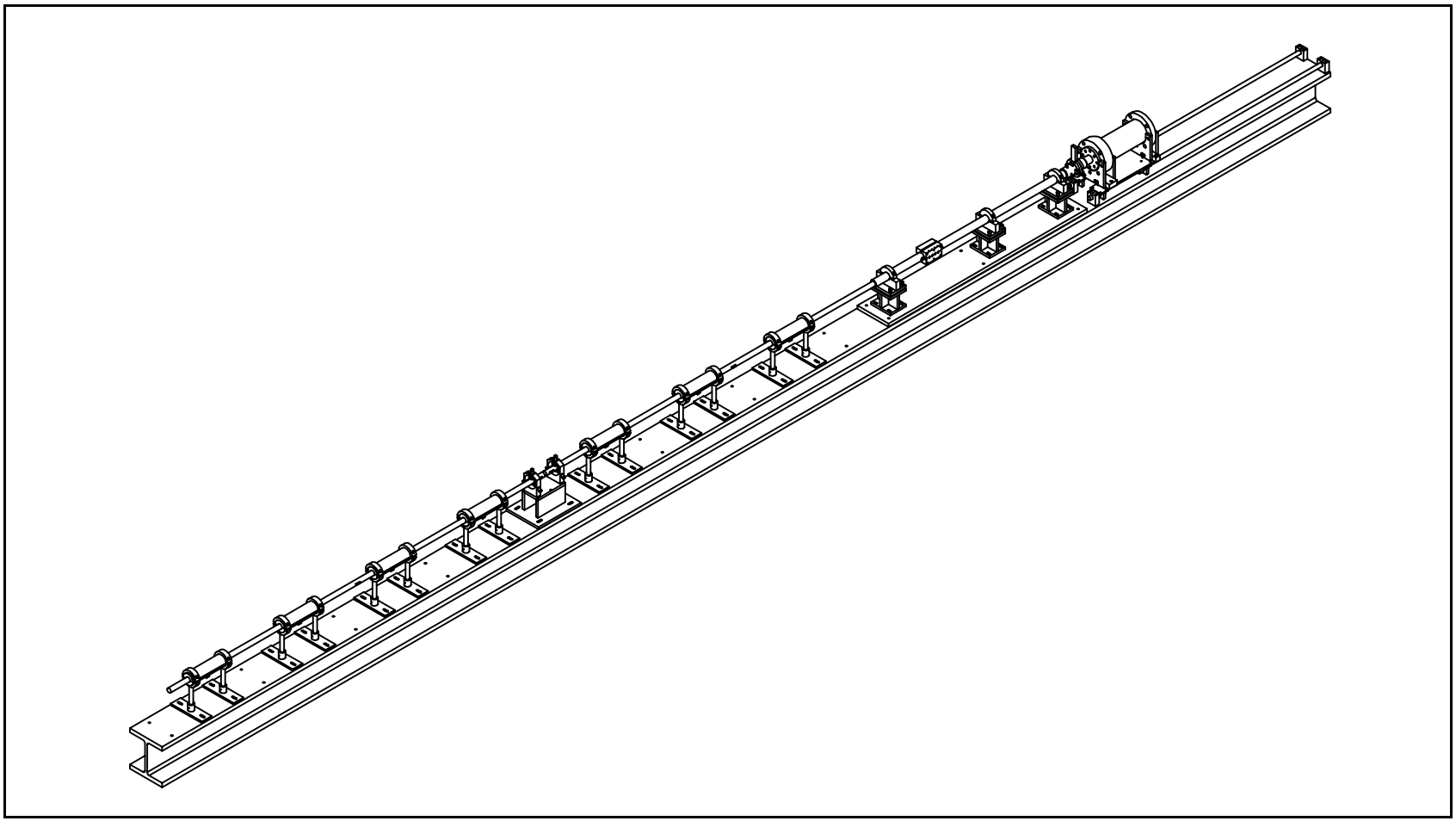


Figure 3.7: Models of Viscoelastic Solids. (a) Voigt Solid;(b) Maxwell Solid;(c) General Solid, Kolsky [1963]

Figure 4.1: Hopkinson Apparatus



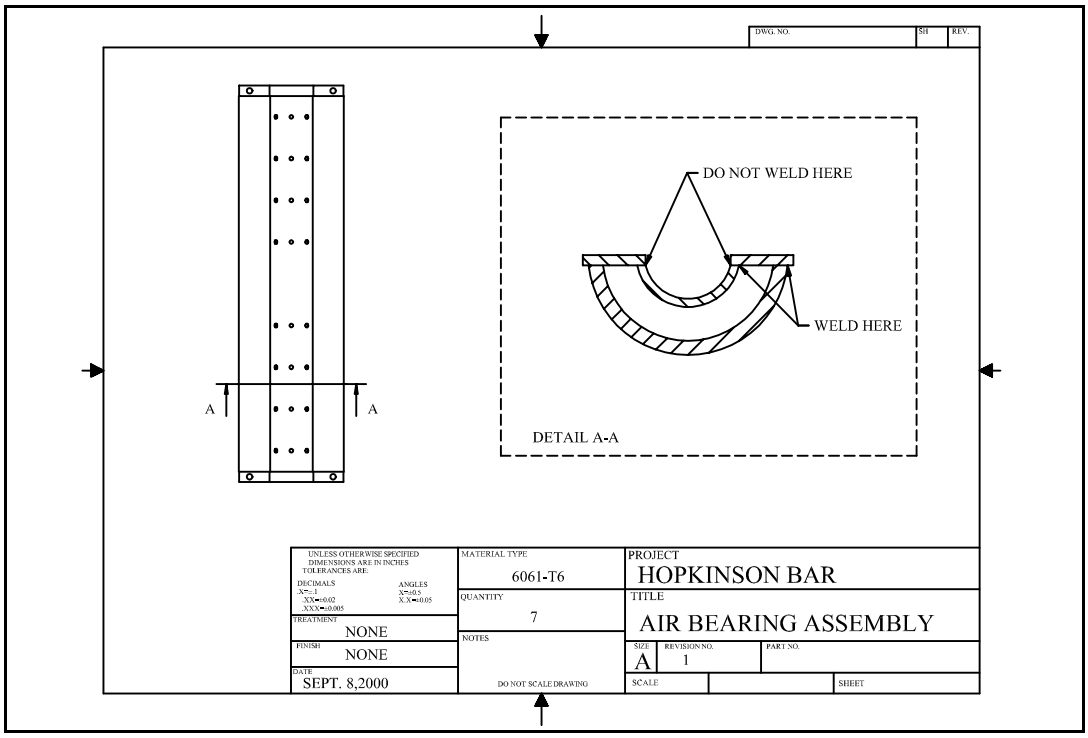


Figure 4.2a: Lower Air Bearing Assembly

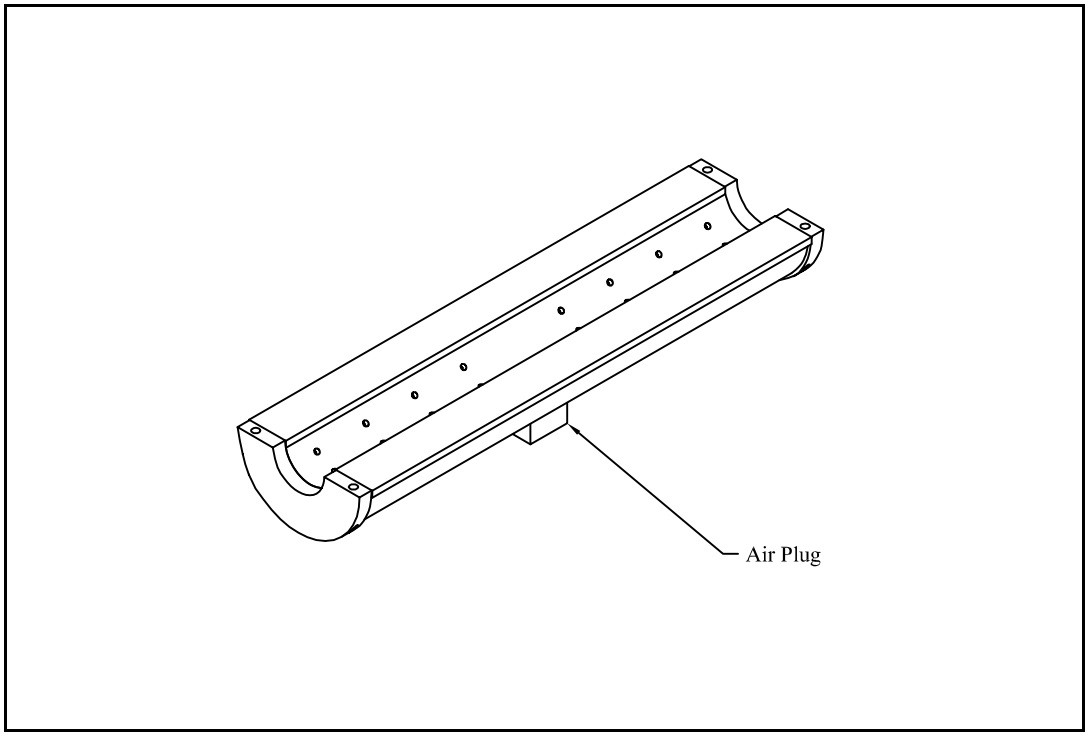


Figure 4.2b: Isometric View

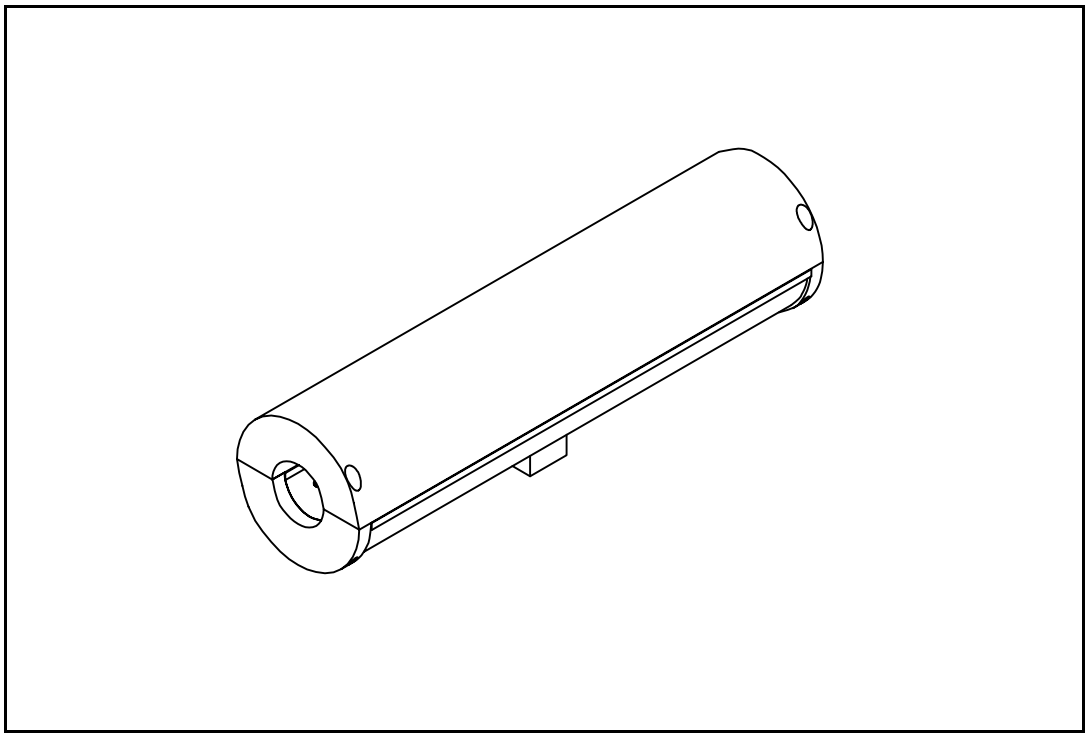


Figure 4.2c: Complete Assembly

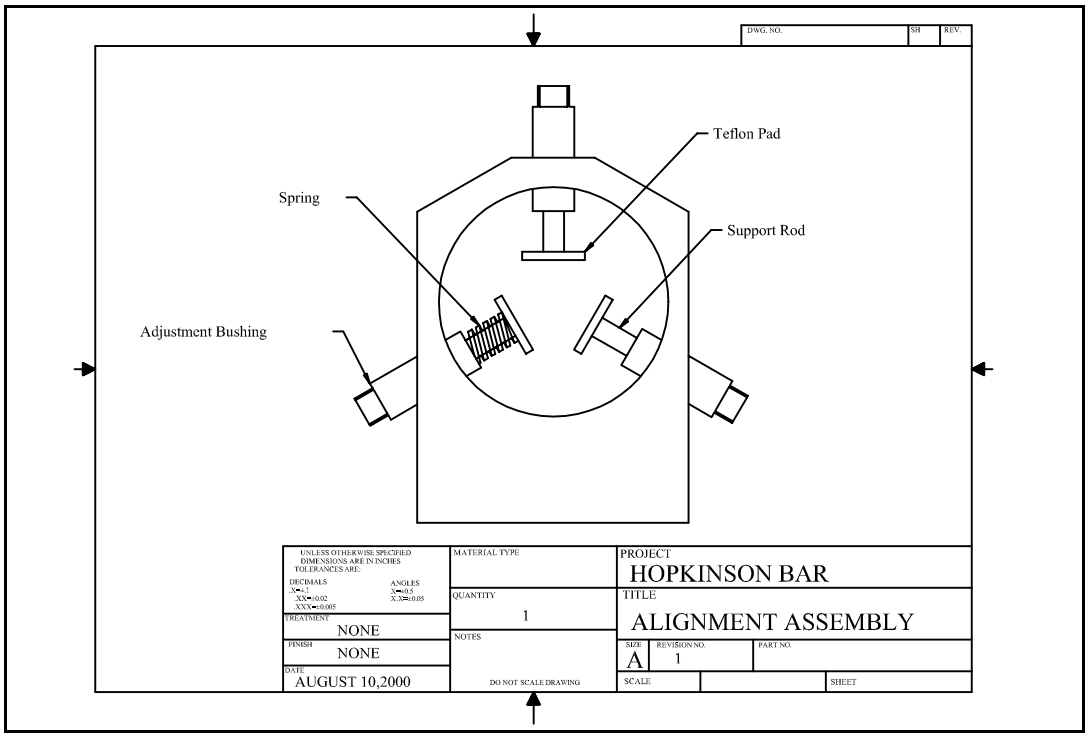


Figure 4.3: Interface Support

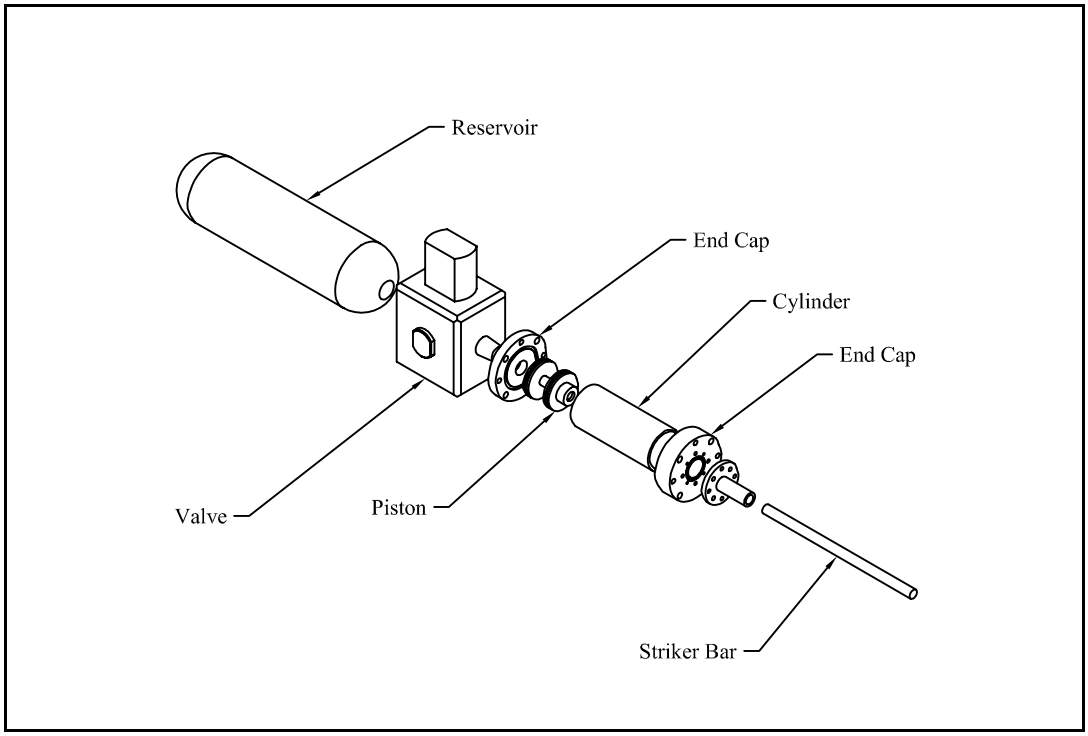


Figure 4.4: Exploded View of Gas Gun

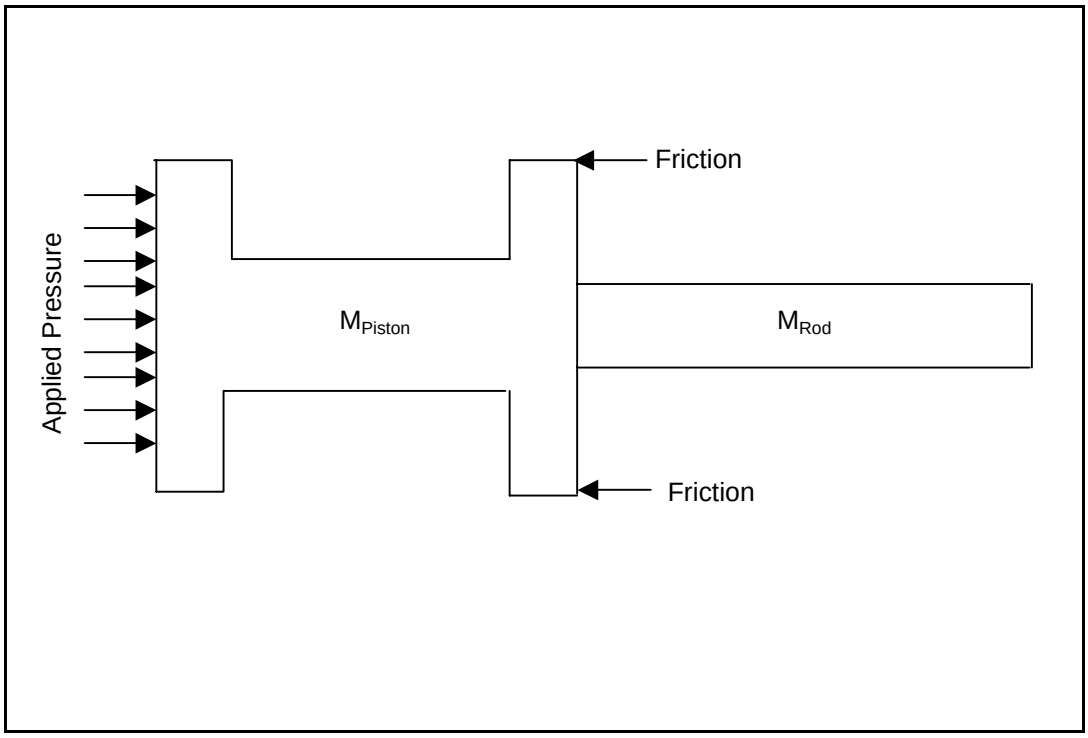


Figure 4.5a: Free Body Diagram of Piston

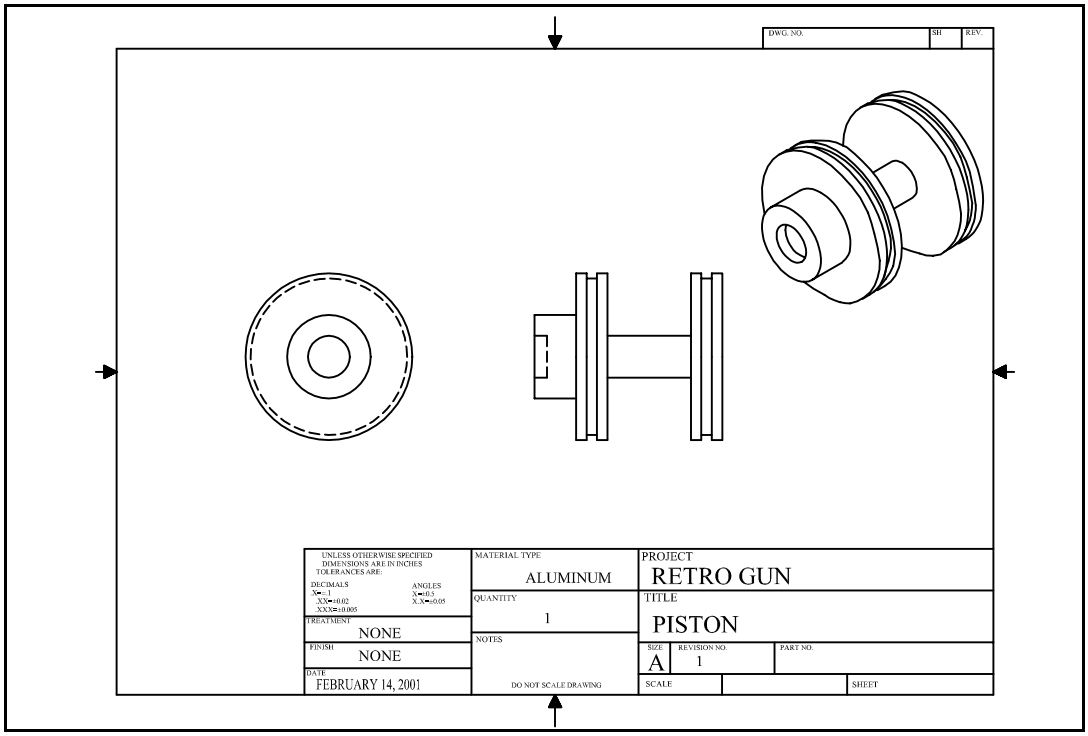


Figure 4.5b: Piston Geometry

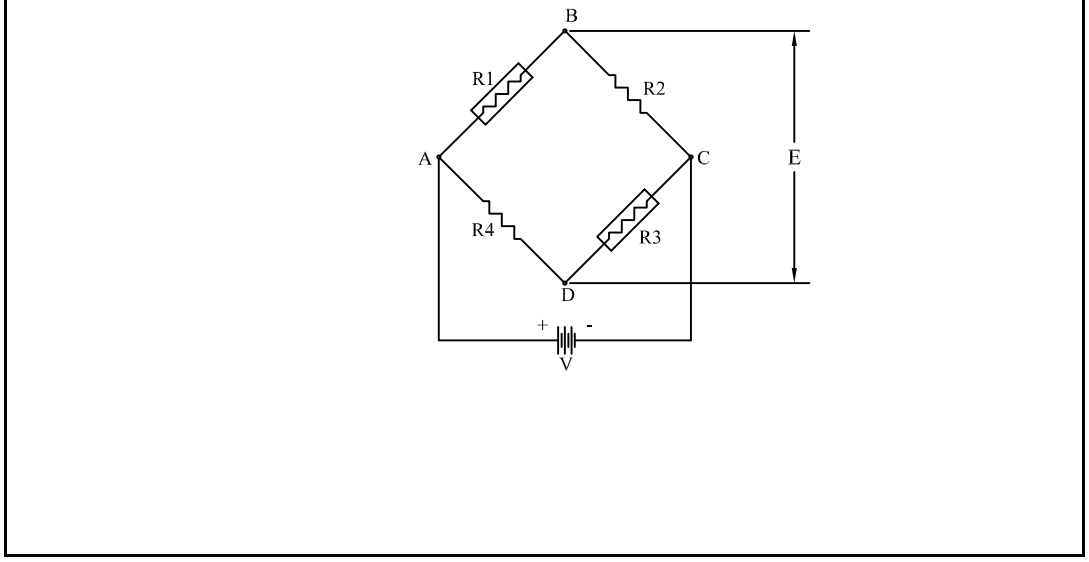


Figure 4.6a: Wheatstone Bridge

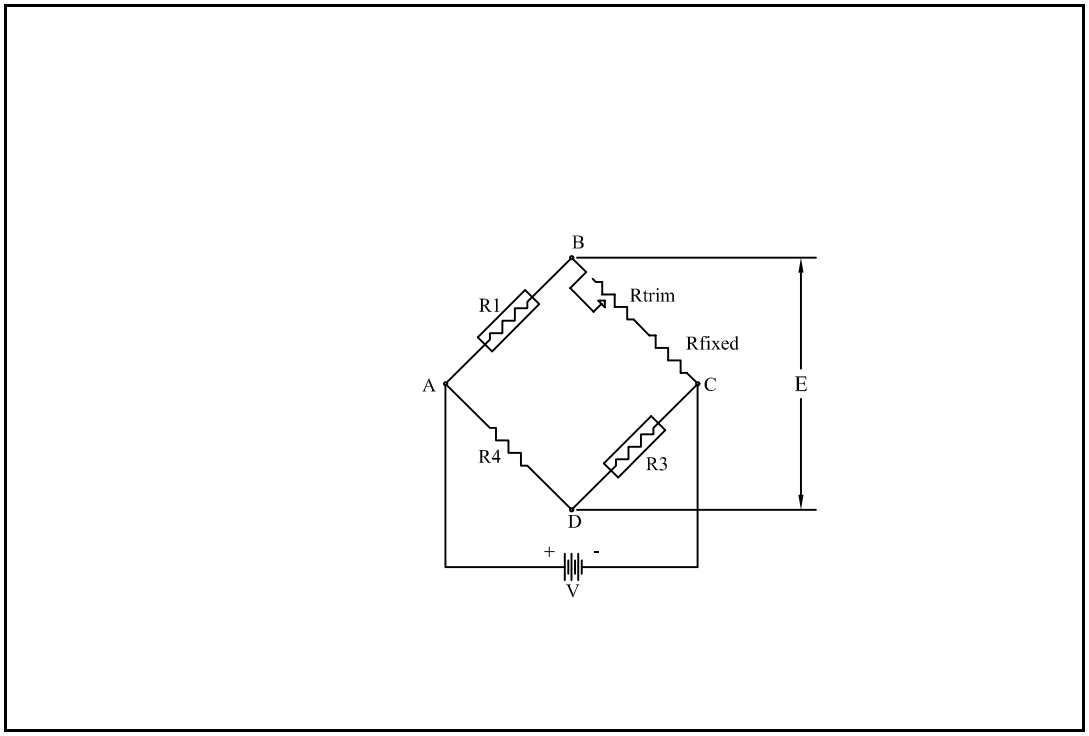


Figure 4.6b: Wheatstone Bridge with Balancing

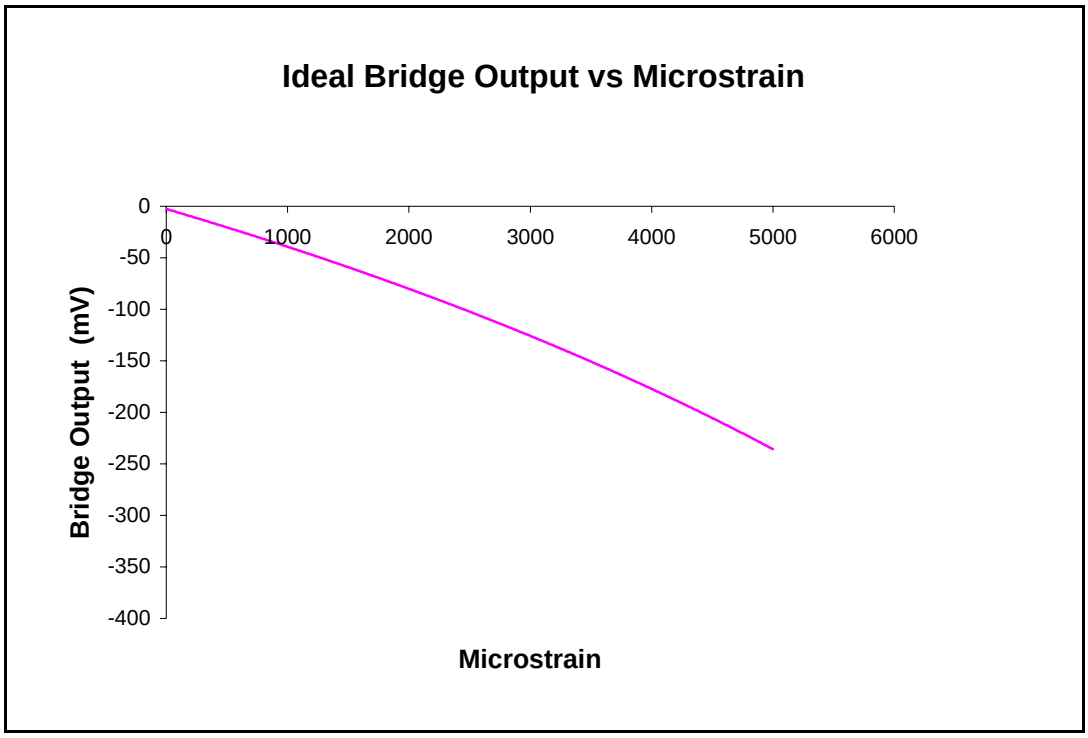


Figure 4.7: Ideal Strain versus Bridge Output

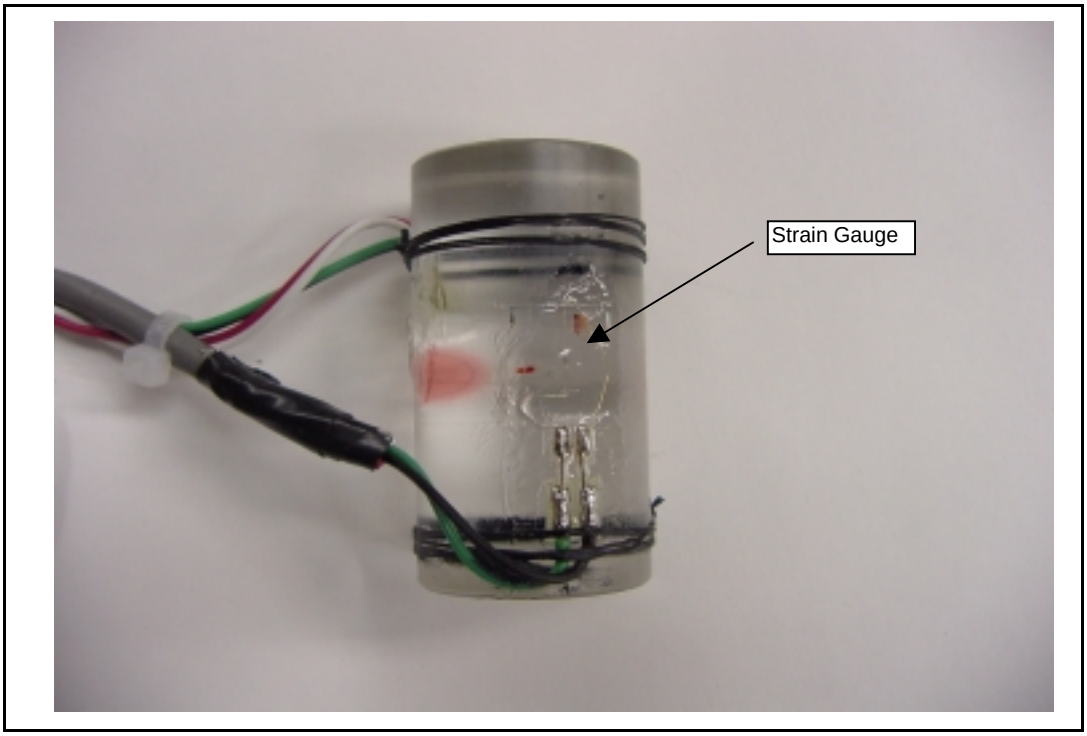


Figure 4.8a: Strain Gauge Mounted on Acrylic Specimen

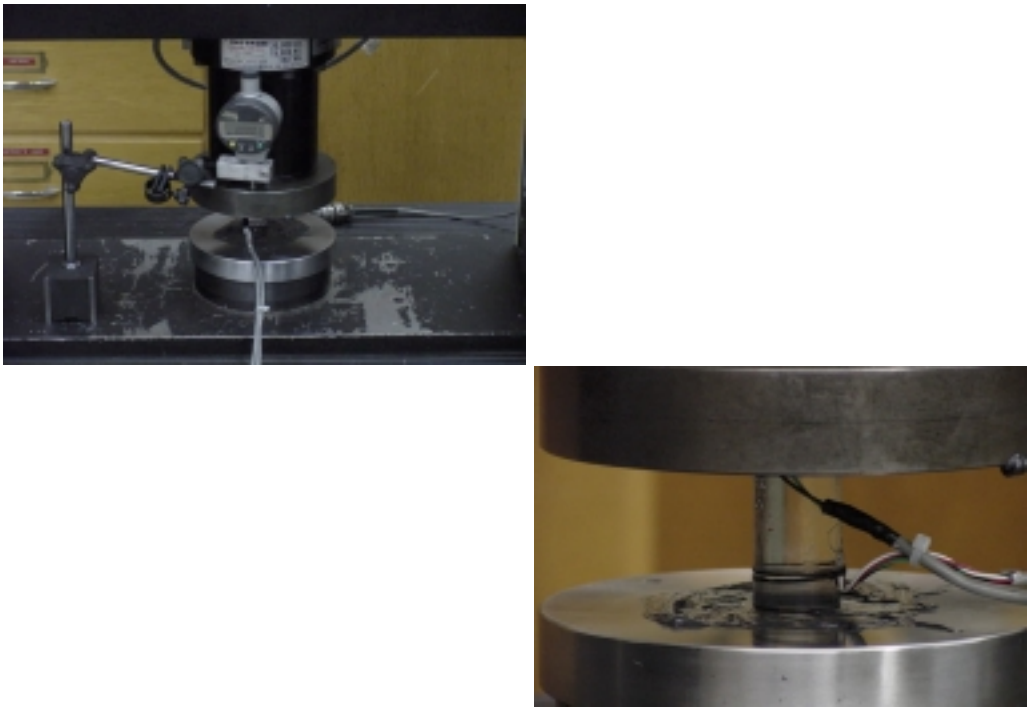


Figure 4.8b: Instron Testing Machine

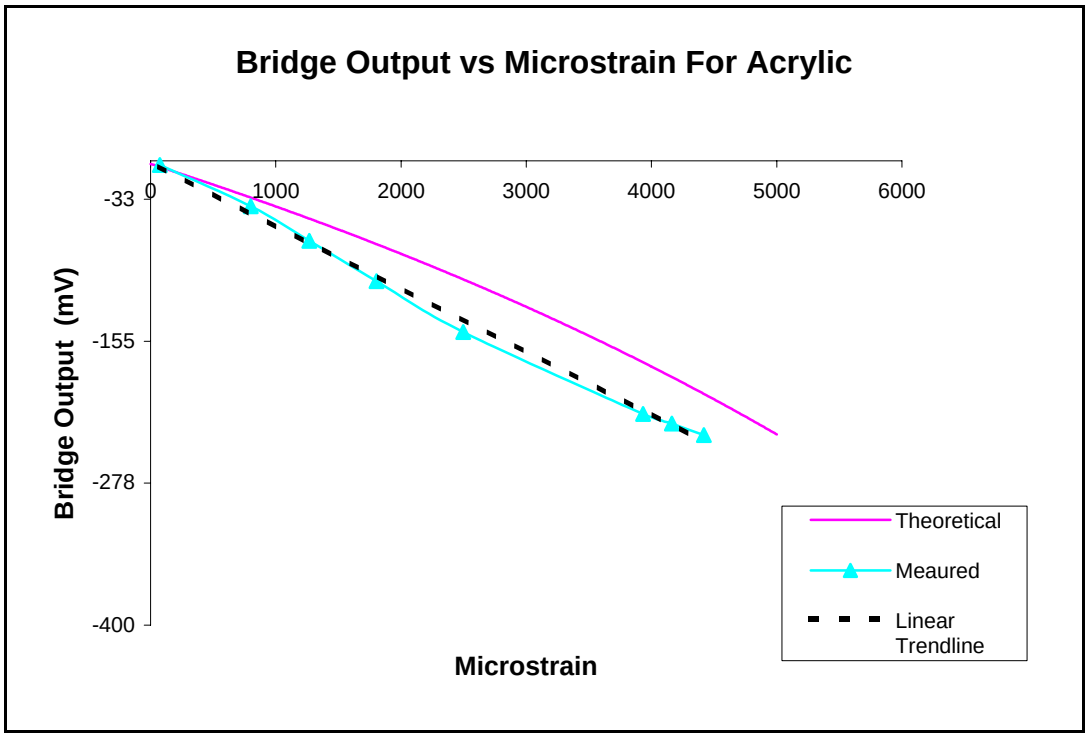


Figure 4.9a: Bridge Ouput versus Microstrain for Acrylic

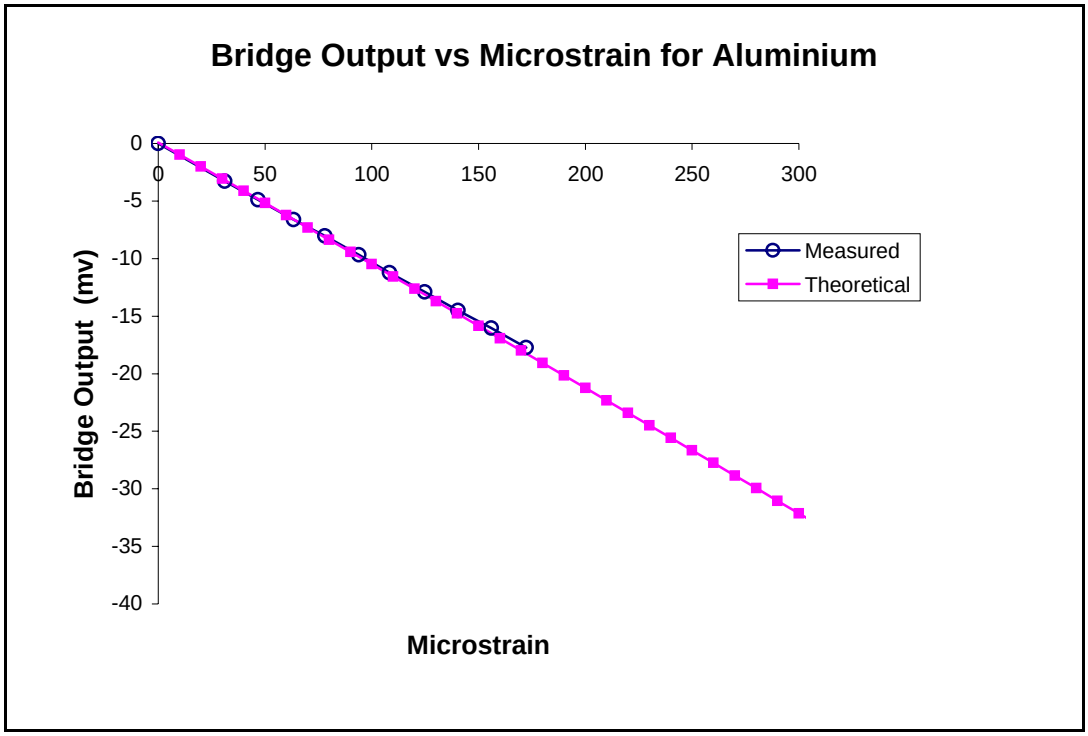


Figure 4.9b: Bridge Output versus Microstrain for Aluminium

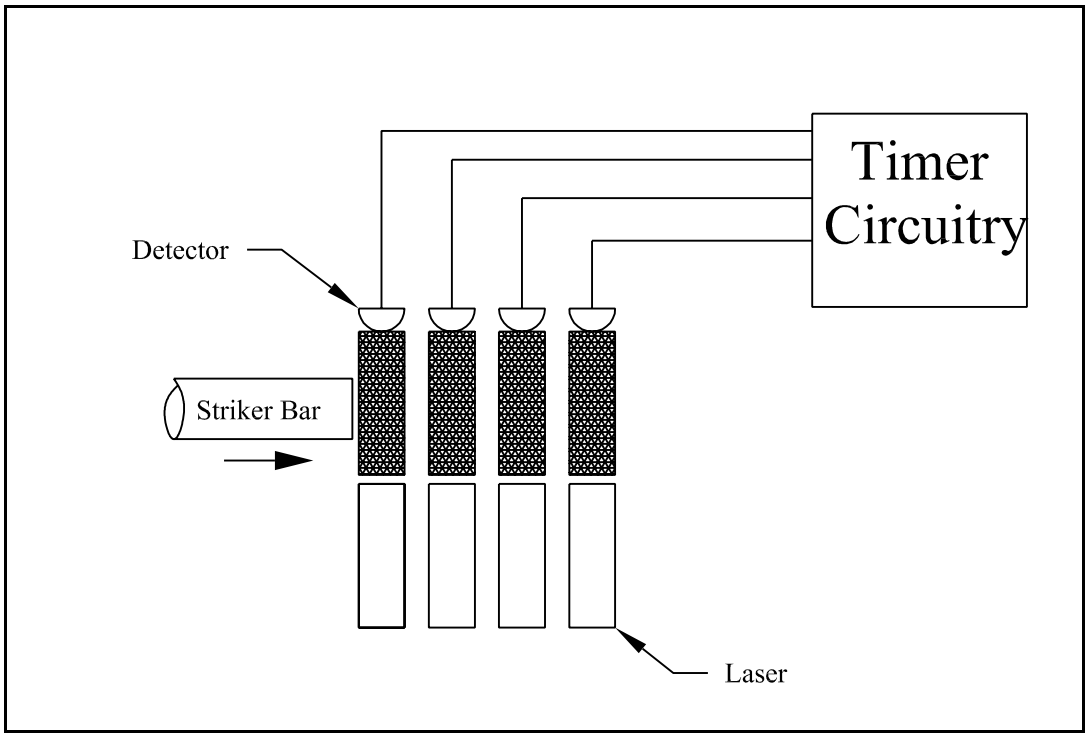


Figure 4.10a: Velocity Detection Schematic

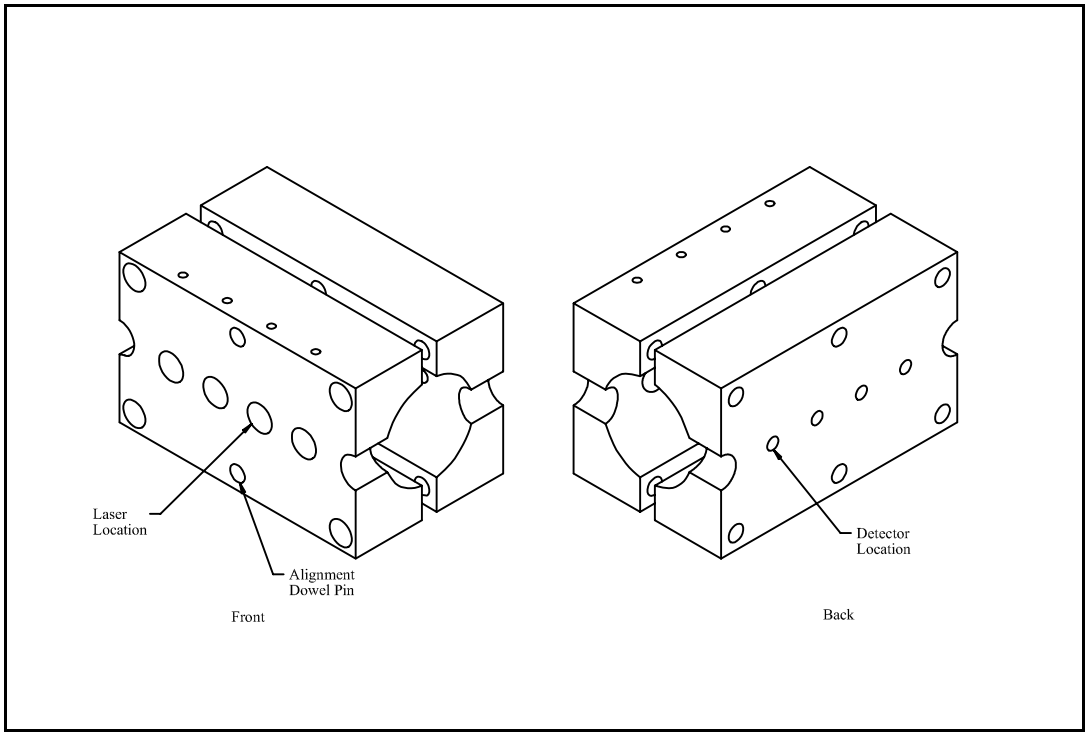


Figure 4.10b: Laser and Detector Housing

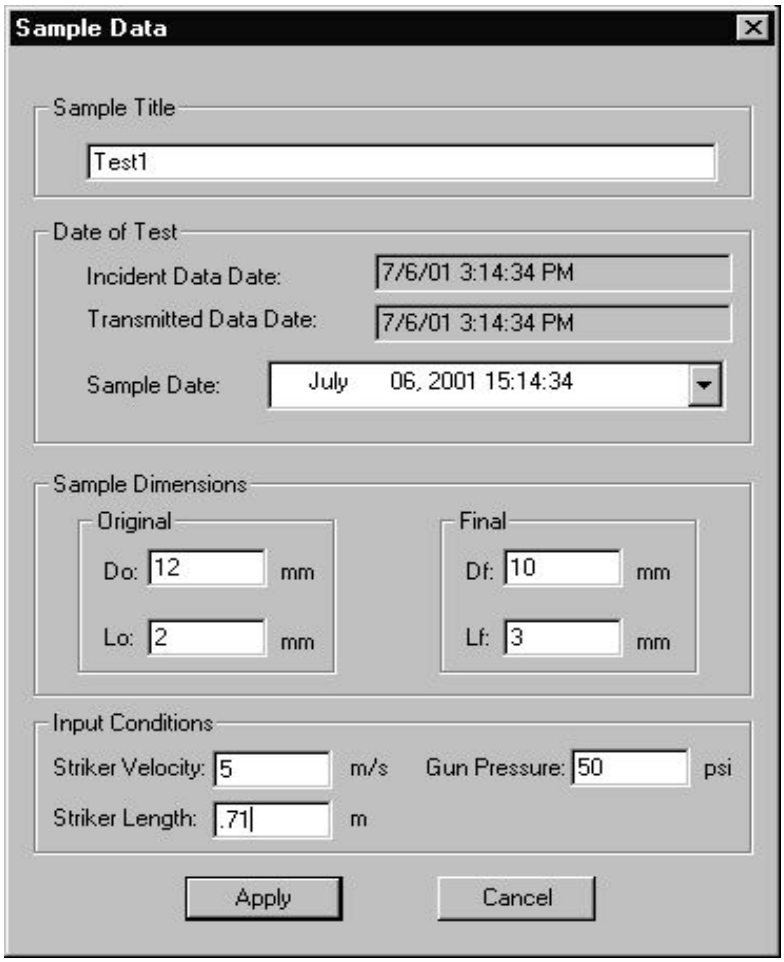


Figure 4.11: Sample Data Dialog Box

```
*NAME
Acry1 Incident 96 inches 2nd Calibration
*GEOMETRY
     0.0254       2.4384       1.2192
*PROPERTIES
       1178   3.47E+09          0.38
*CALIBRATION
            0   0.00E+00
    0.02192    2.93E-04
          .            .
          .            .
*FILTER
      20000
*PROPAGATION
            0   0.065527   -3.89E-06
    244.141   0.074876    0.740581
    488.281   0.097277     1.46891
    732.422   0.119215     2.18213
          .            .            .
```

Figure 4.12: Typical Calibration File

Figure 4.13: Bar Data Dialog Box

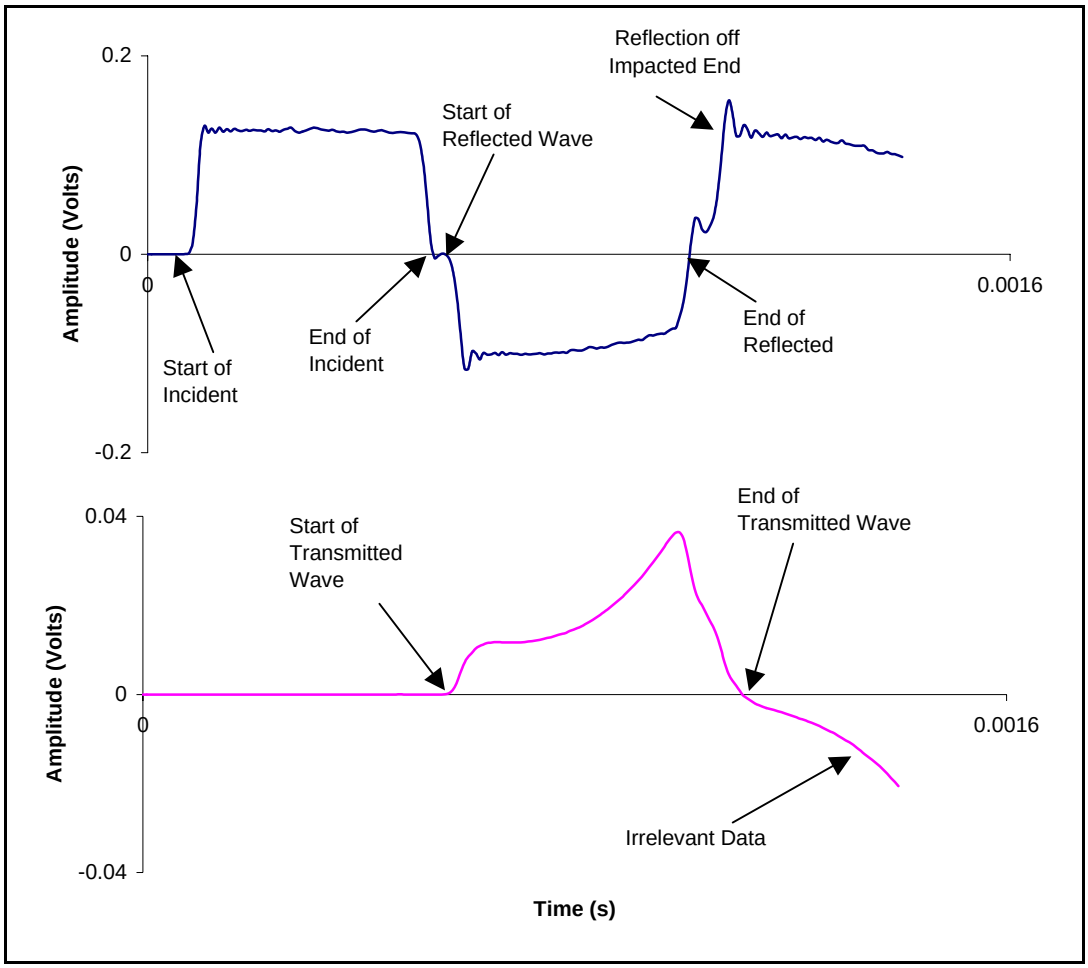


Figure 4.14: Starting and End Points for Incident, Reflected and Transmitted Waves

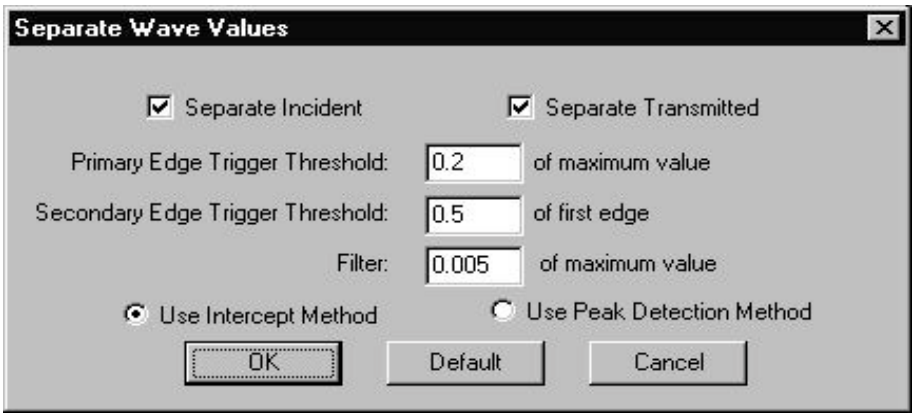


Figure 4.15: Separate Wave Data Dialog Box

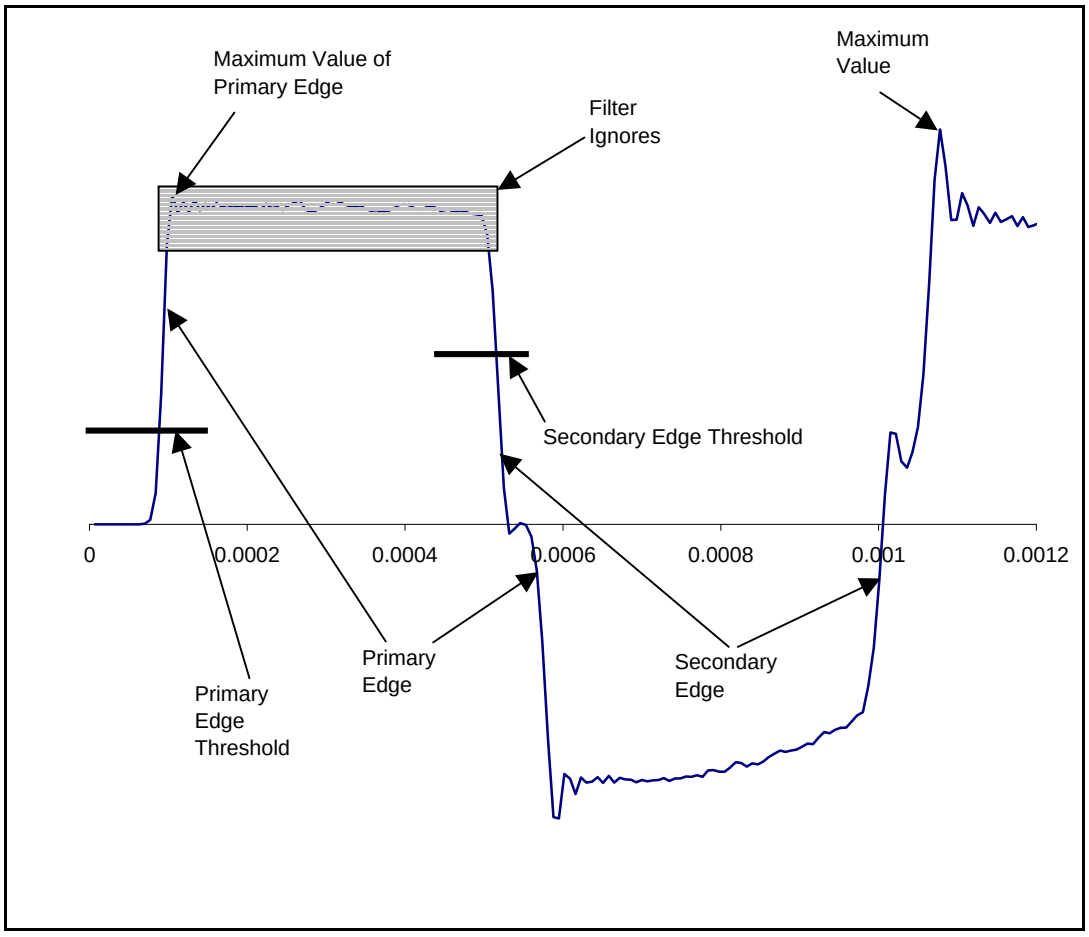


Figure 4.16: Threshold and Filter Identification

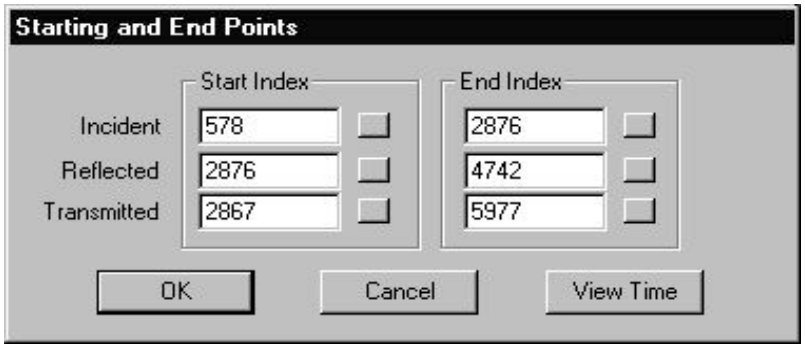


Figure 4.17: View and Choose Indices Dialog Box

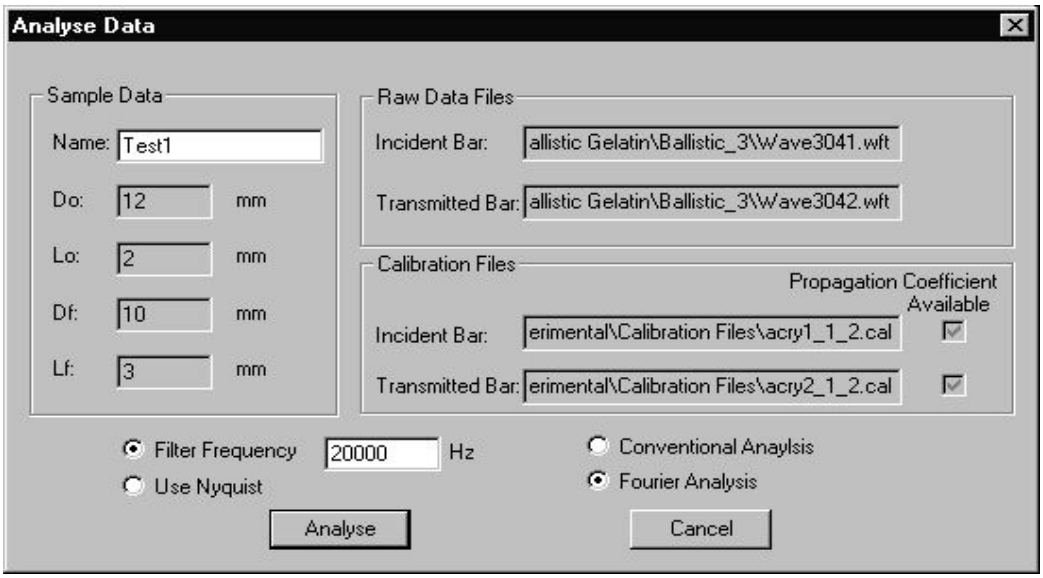


Figure 4.18: Analyse Data Dialog Box

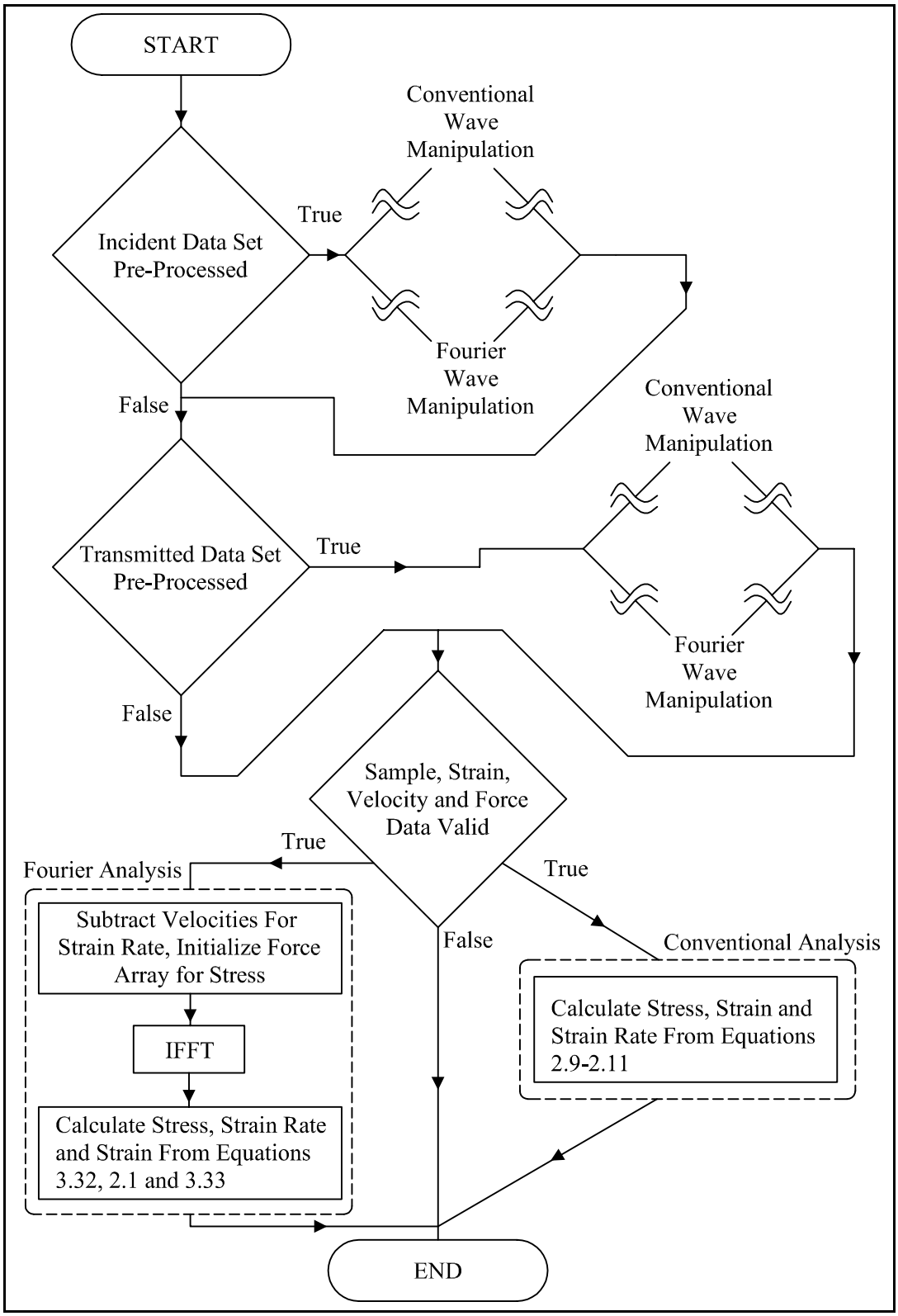


Figure 4.19: Flow Chart of Analyse Algorithm

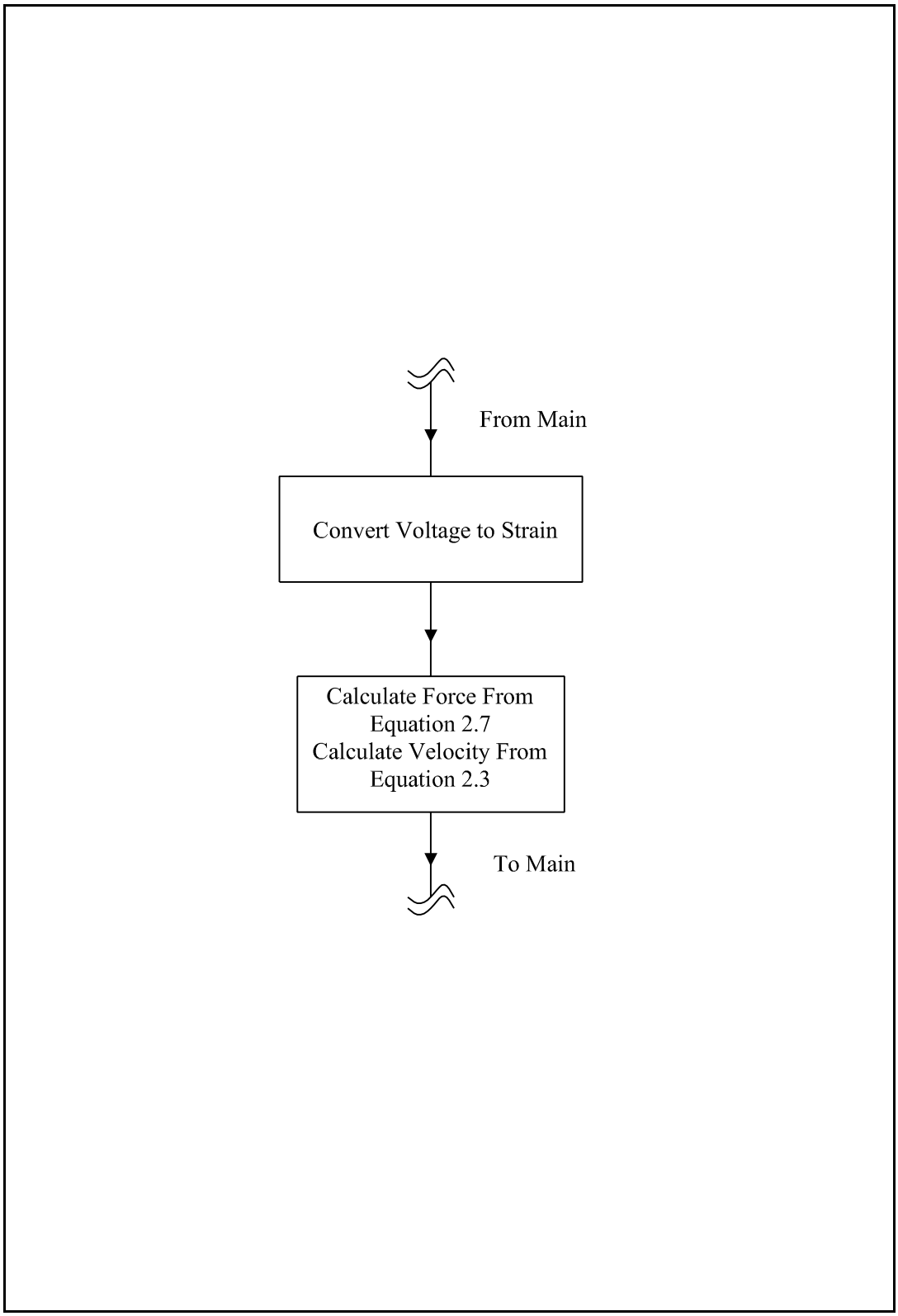


Figure 4.20: Flow Chart of Conventional Manipulation Algorithm

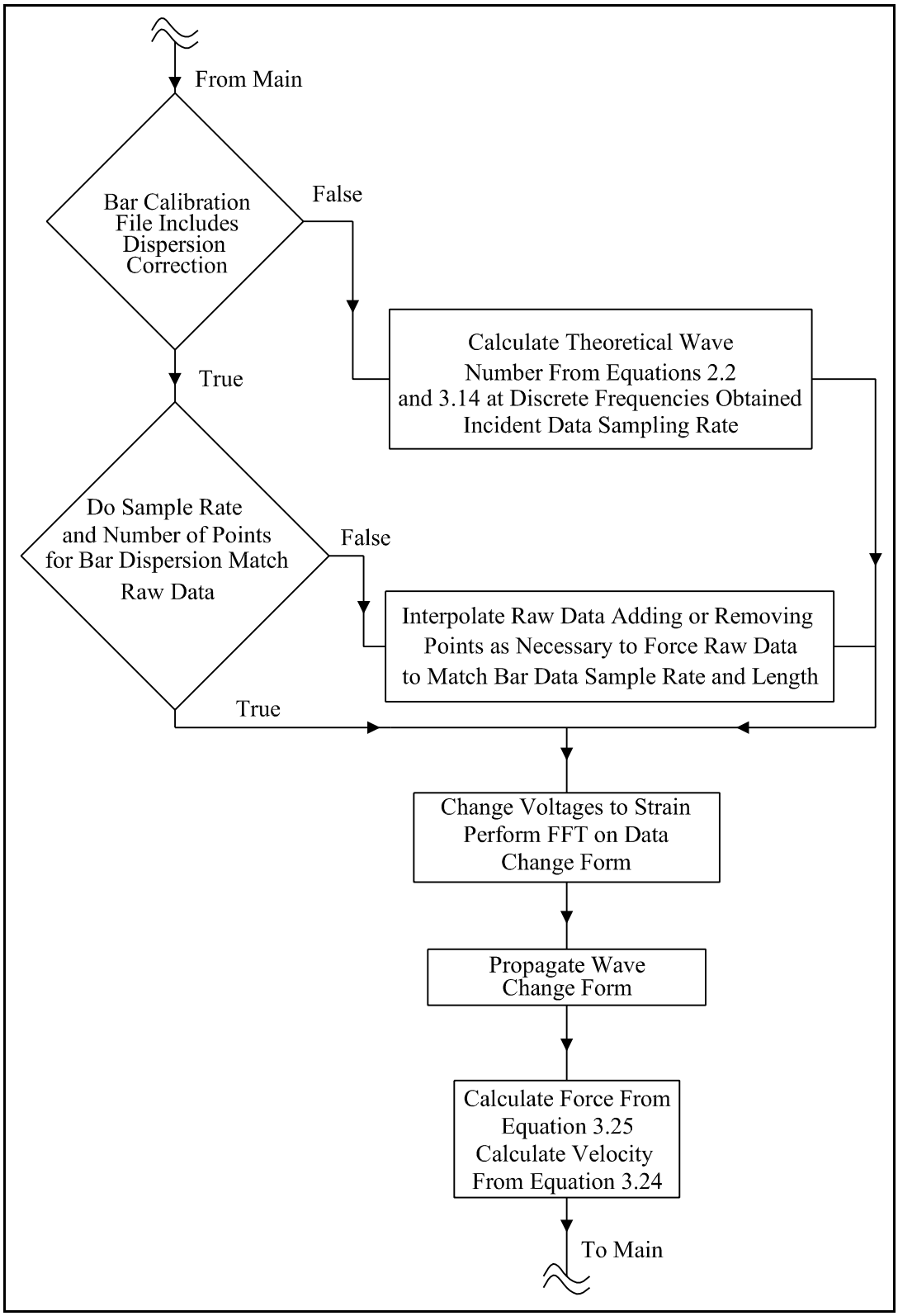


Figure 4.21: Flow Chart of Fourier Manipulation Algorithm

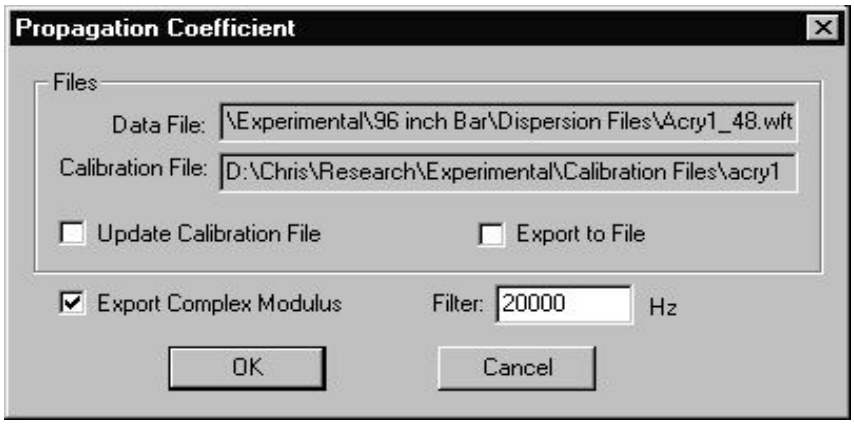


Figure 4.22: Propagation Coefficient Data Dialog Box

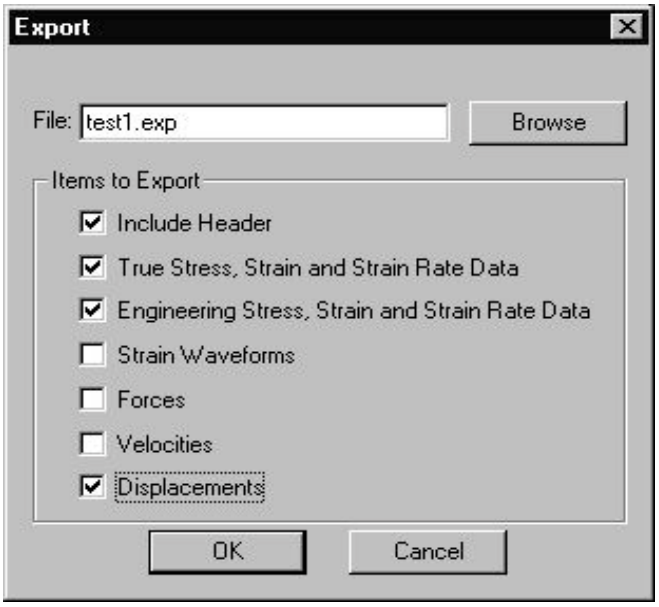


Figure 4.23: Export Data Dialog Box

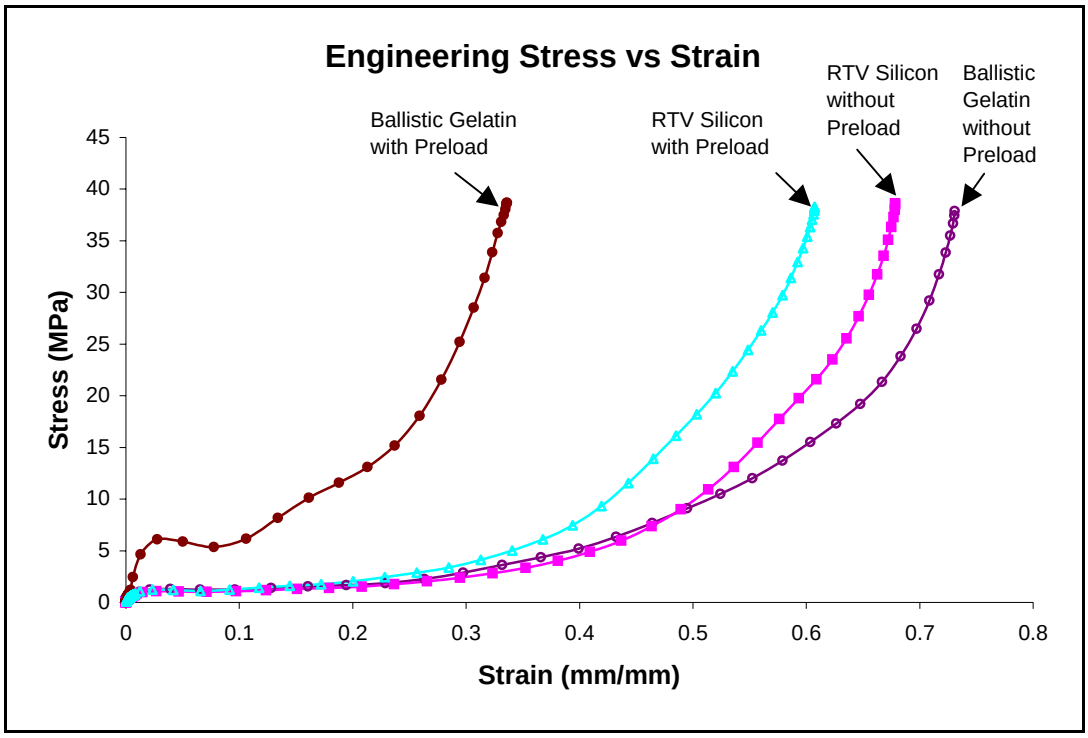


Figure 4.24: Stress Strain Curves for Preloaded Specimens

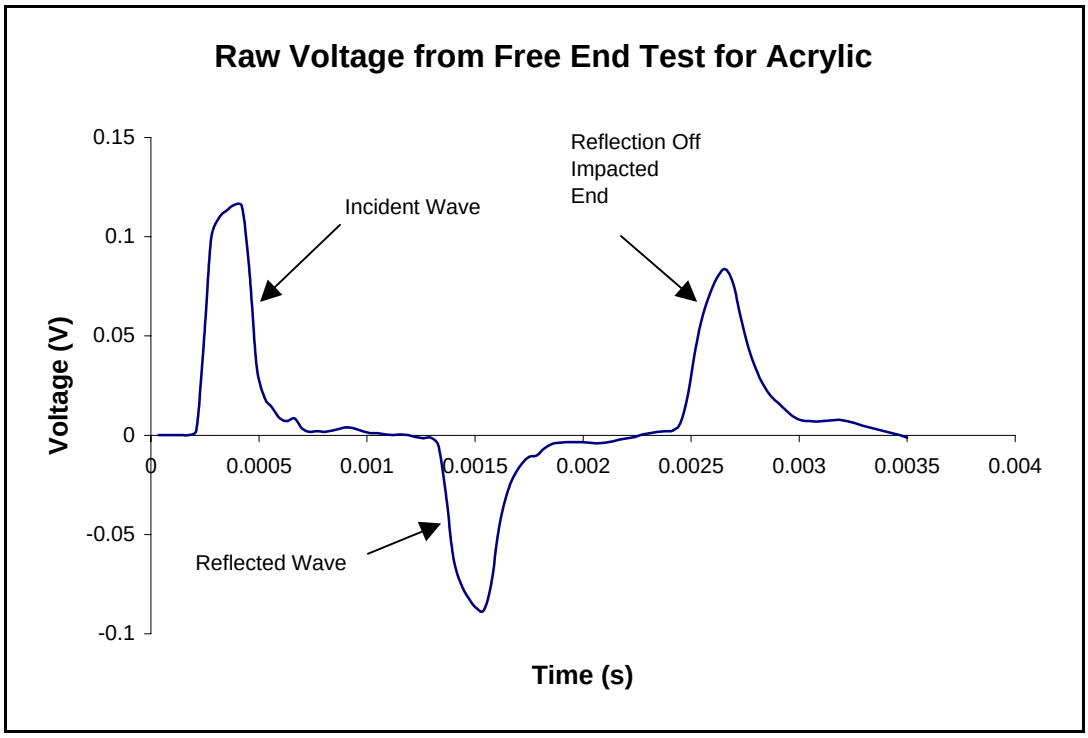


Figure 5.1: Raw Data Acquired from Free End Test

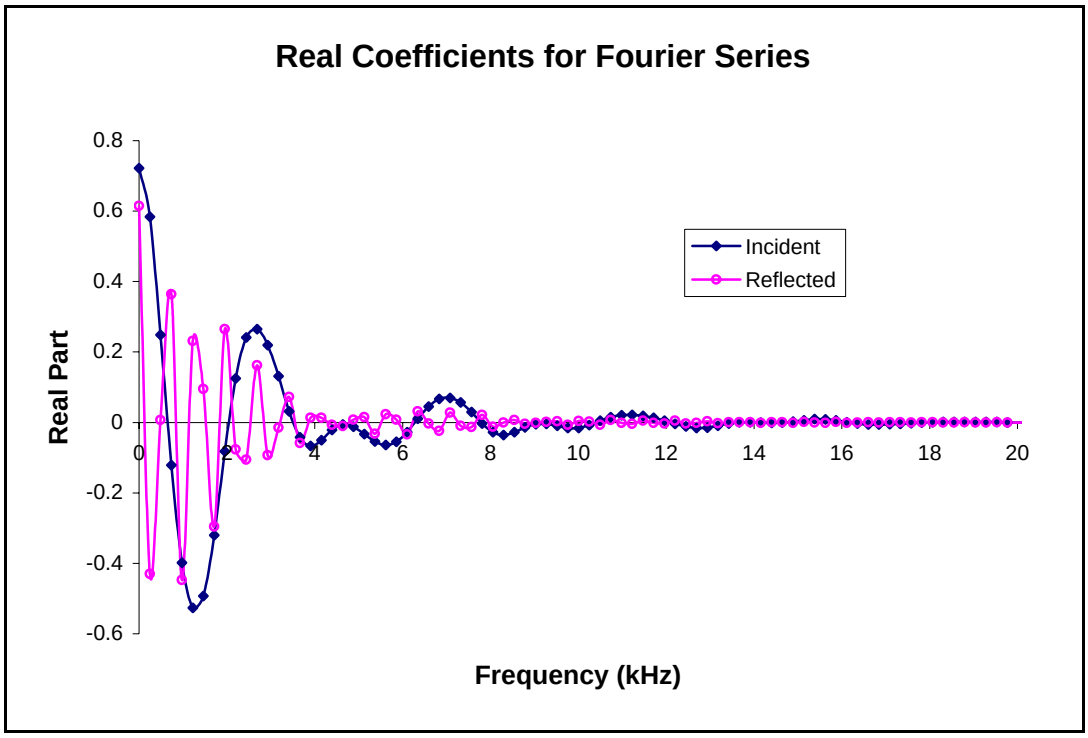


Figure 5.2a: Real Coefficients of Fourier Series

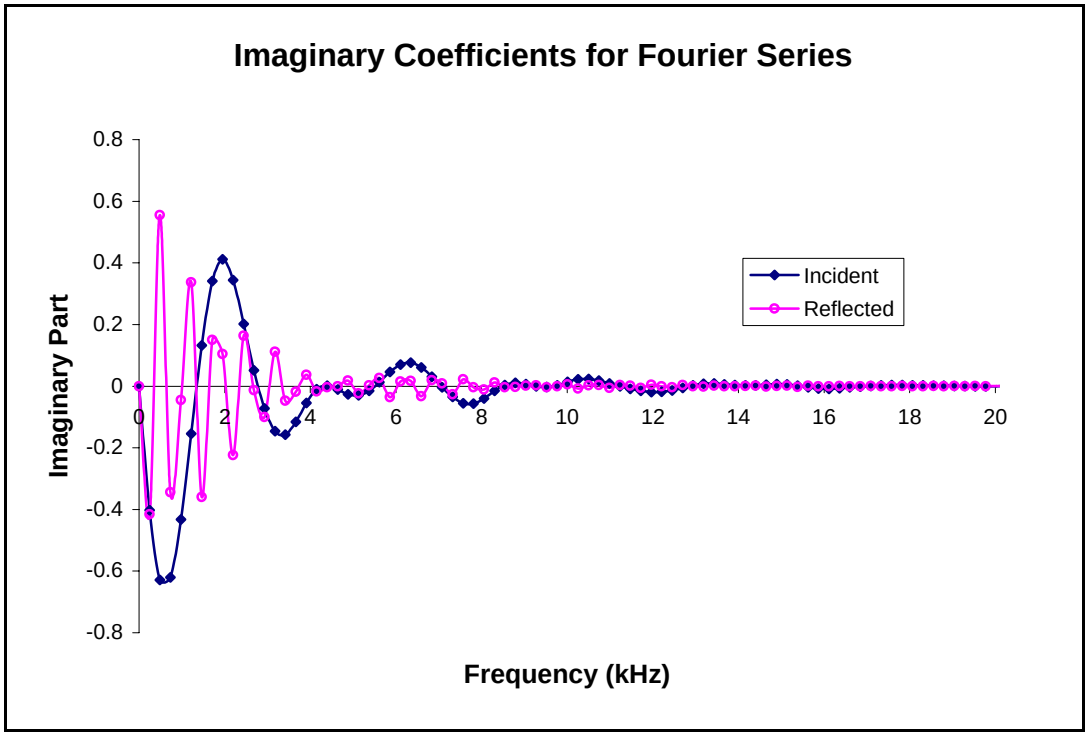


Figure 5.2b: Imaginary Coefficients of Fourier Series

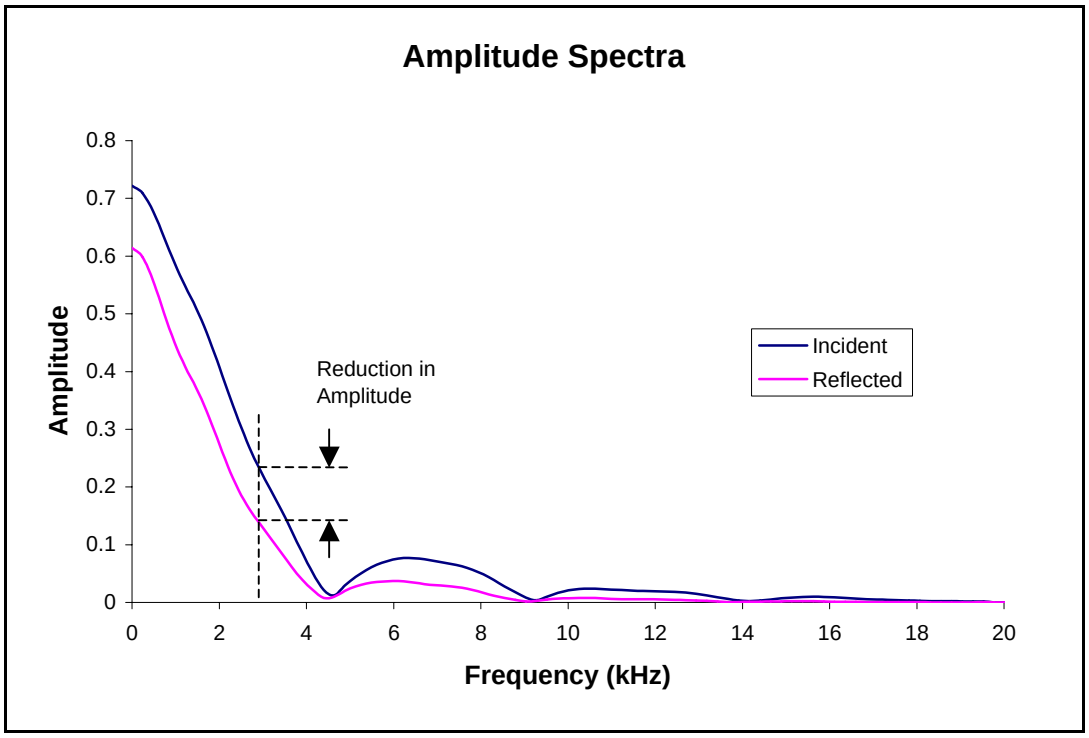


Figure 5.2c: Amplitude Spectra for Incident and Reflected Waves

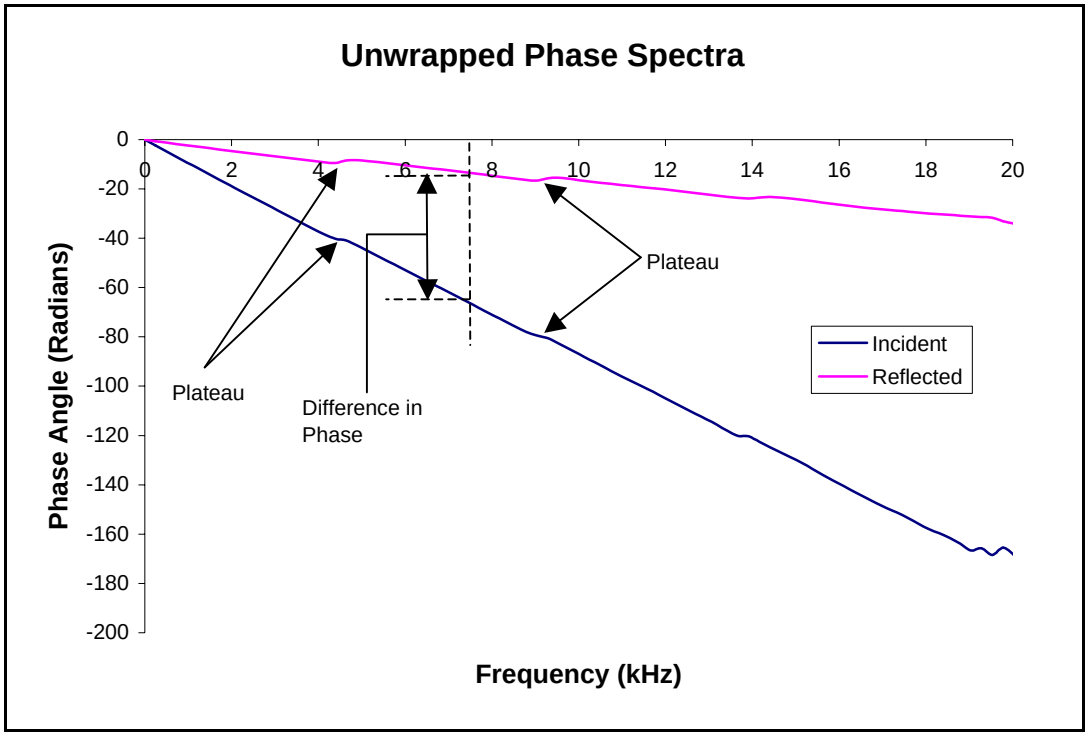


Figure 5.2d: Unwrapped Phase Spectra for Incident and Reflected Waves

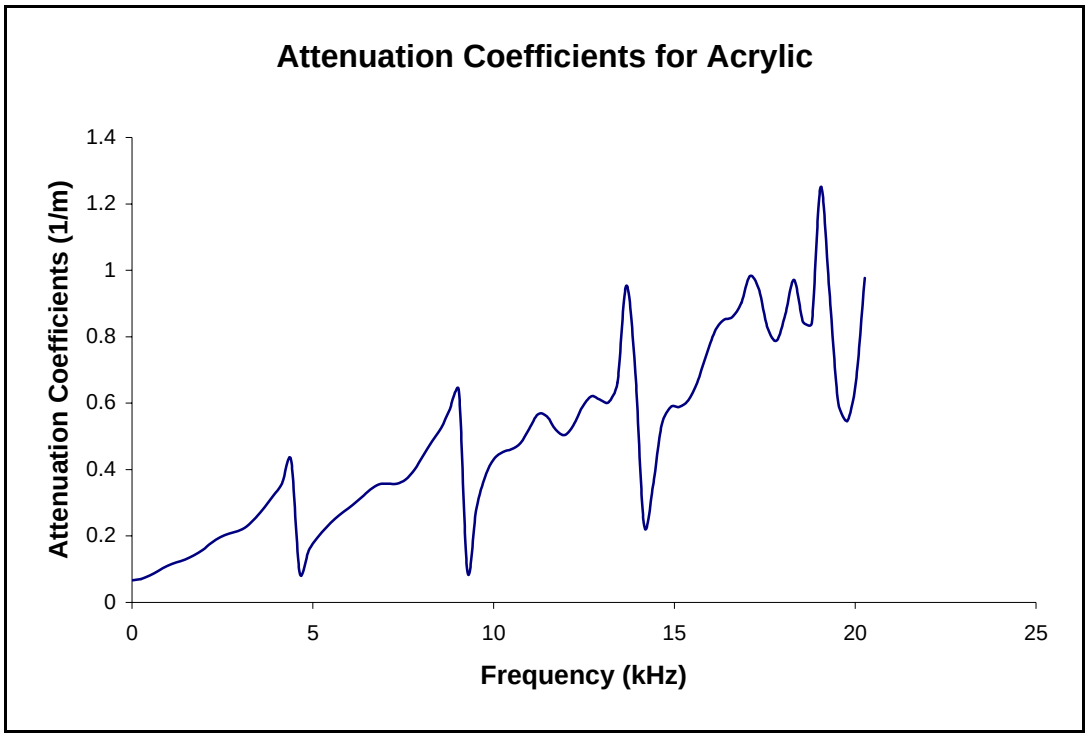


Figure 5.3a: Attenuation Coefficients for Acrylic

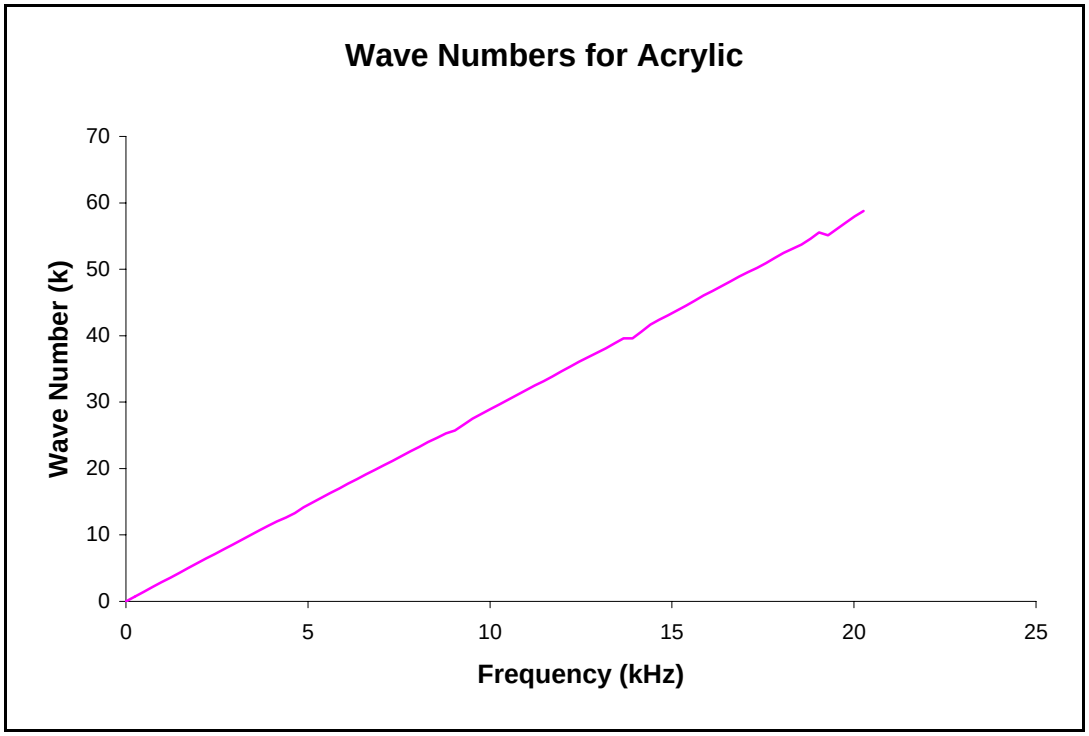


Figure 5.3b: Wave Numbers for Acrylic

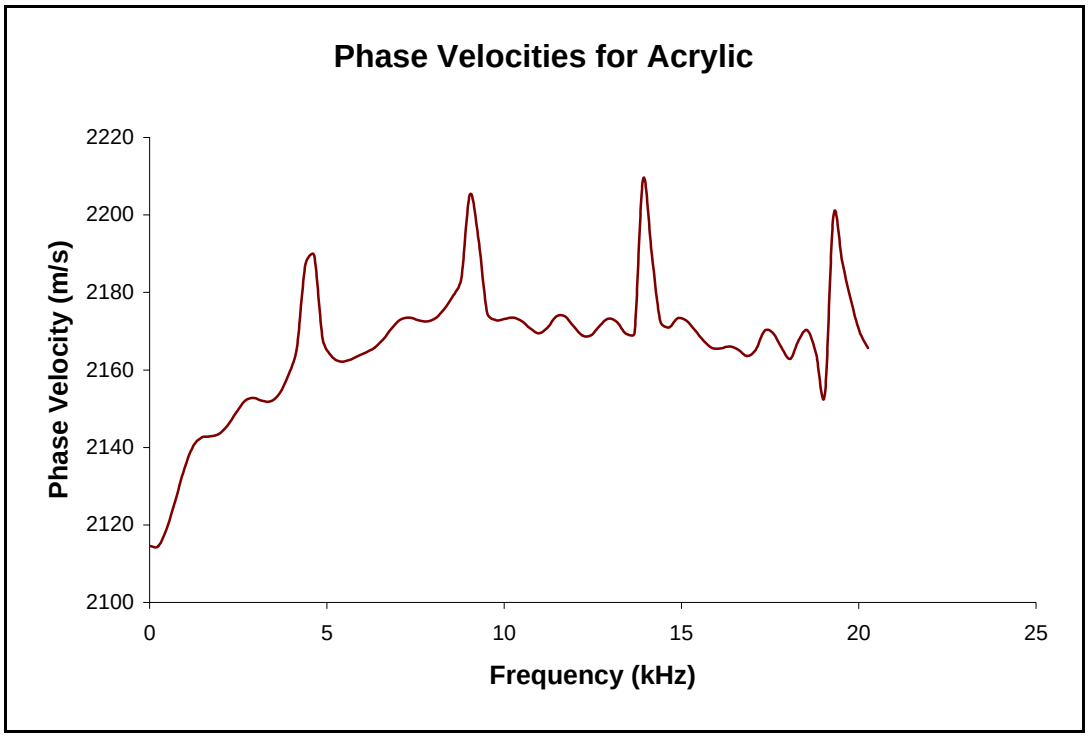


Figure 5.3c: Phase Velocities for Acrylic

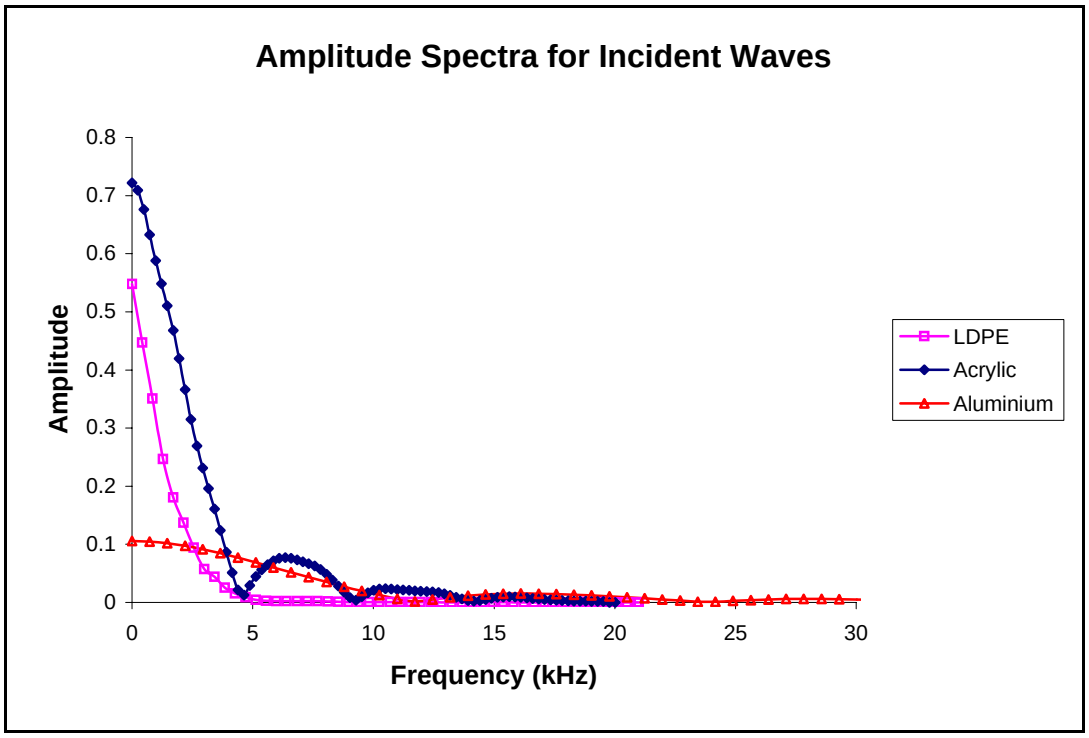


Figure 5.4a: Comparison of Amplitude Spectra

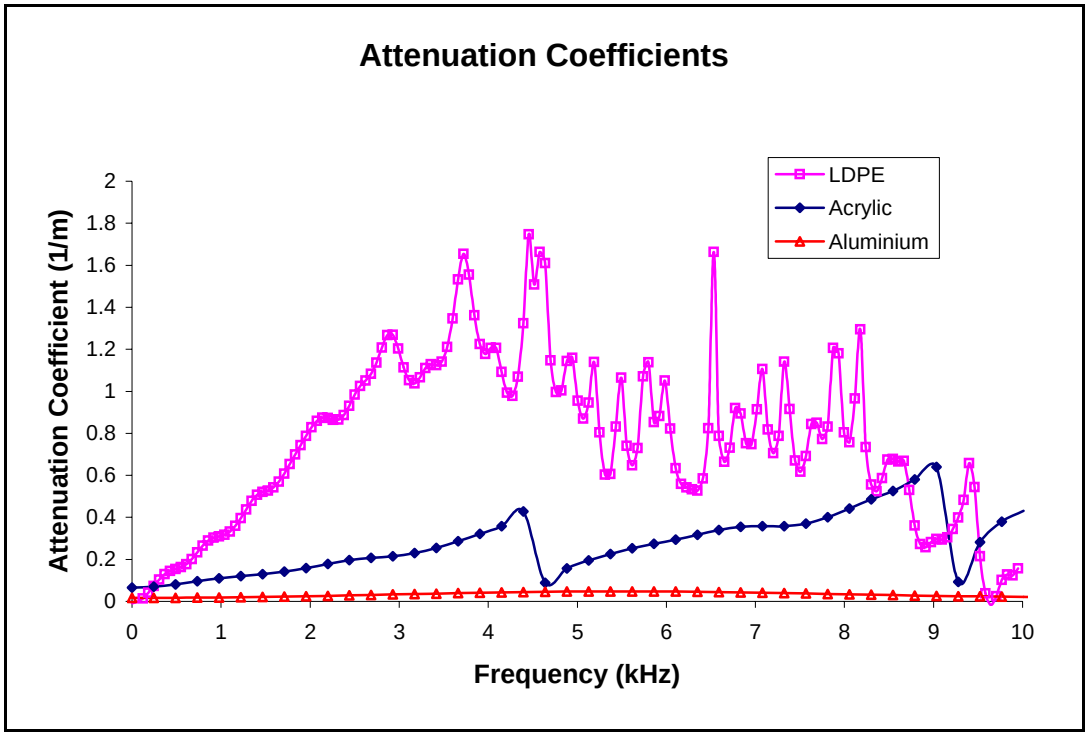


Figure 5.4b: Comparison of Attenuation Coefficients

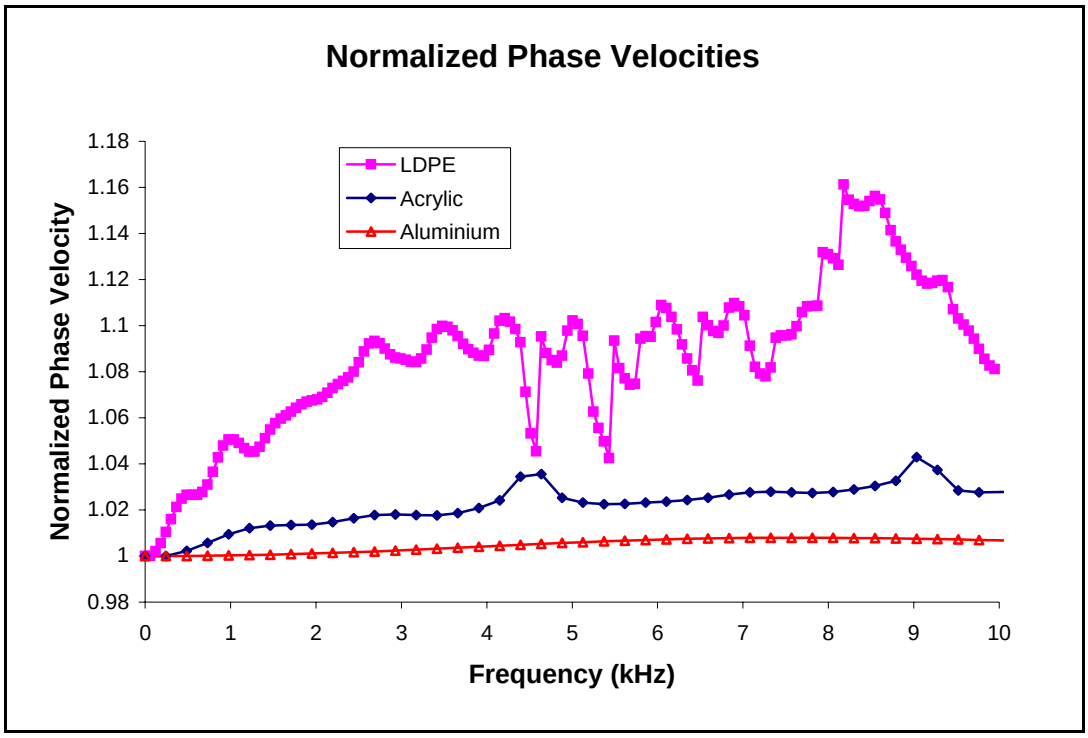


Figure 5.4c: Comparison of Normalized Phase Velocities

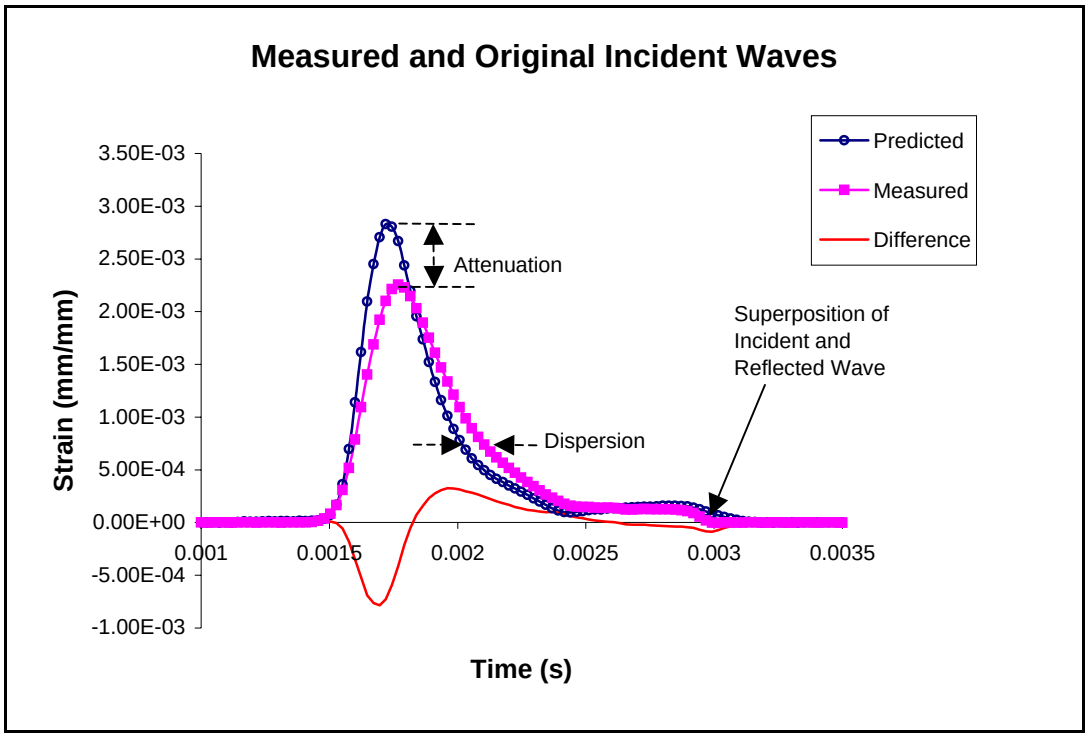


Figure 5.5a: Measured and Predicted Wave Using Elastic Wave Analysis

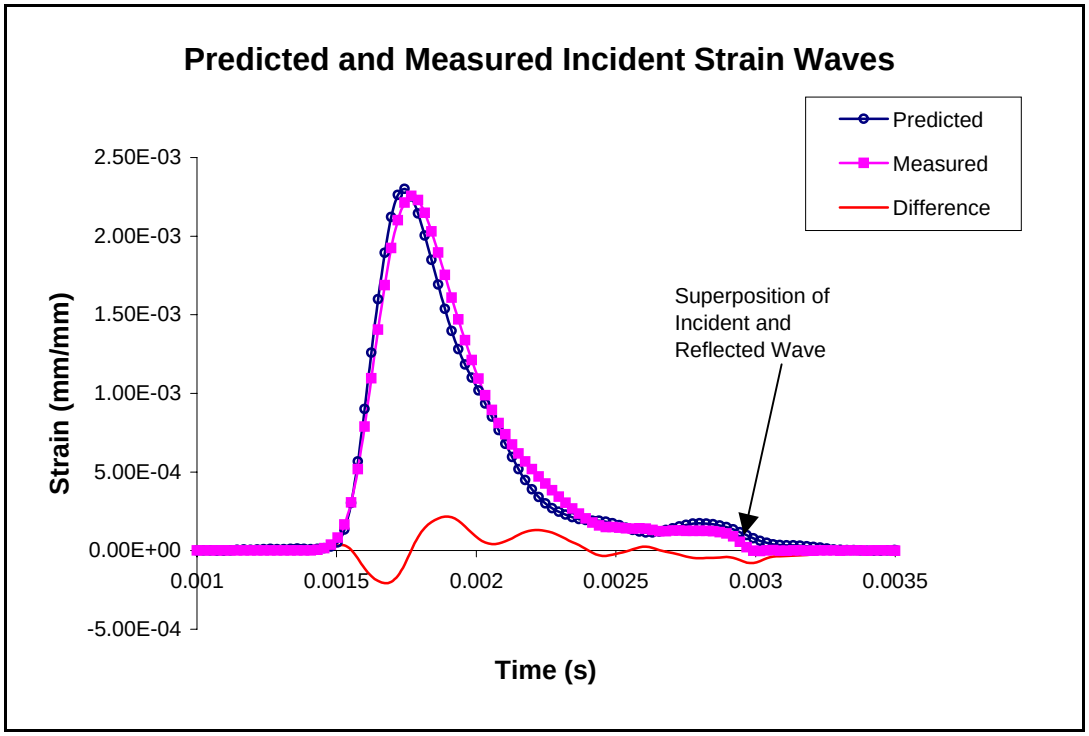


Figure 5.5b: Measured and Predicted Wave Using Viscoelastic Wave Analysis

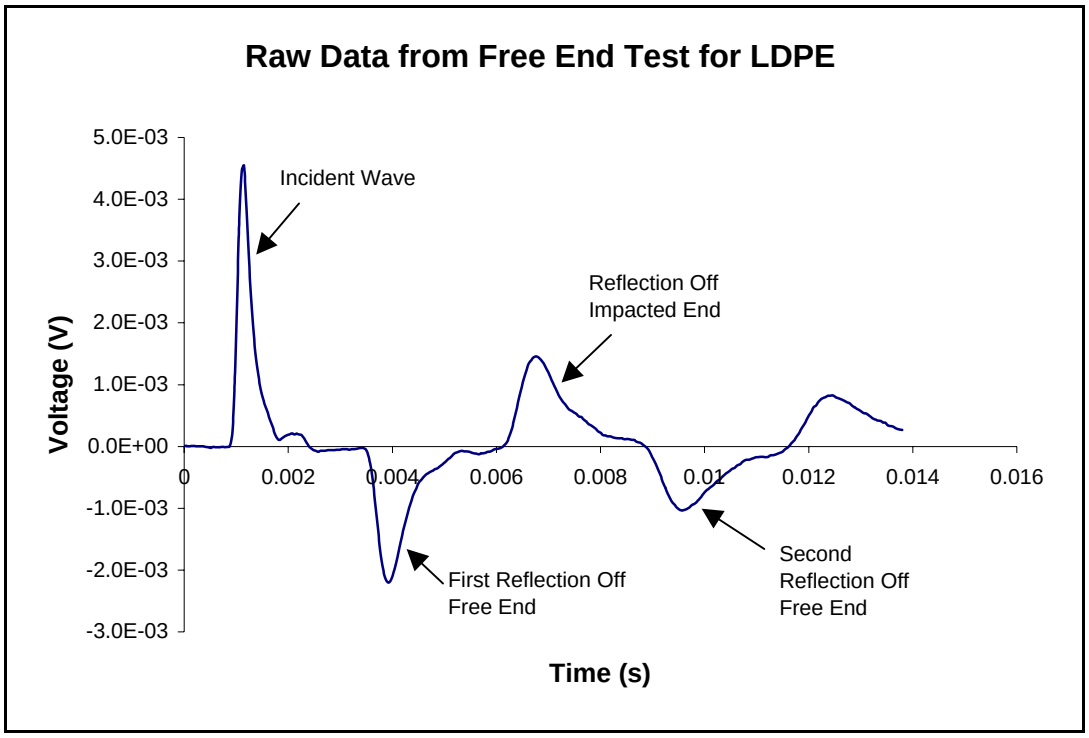


Figure 5.6a: Raw Data Acquired from Free End Test for LDPE

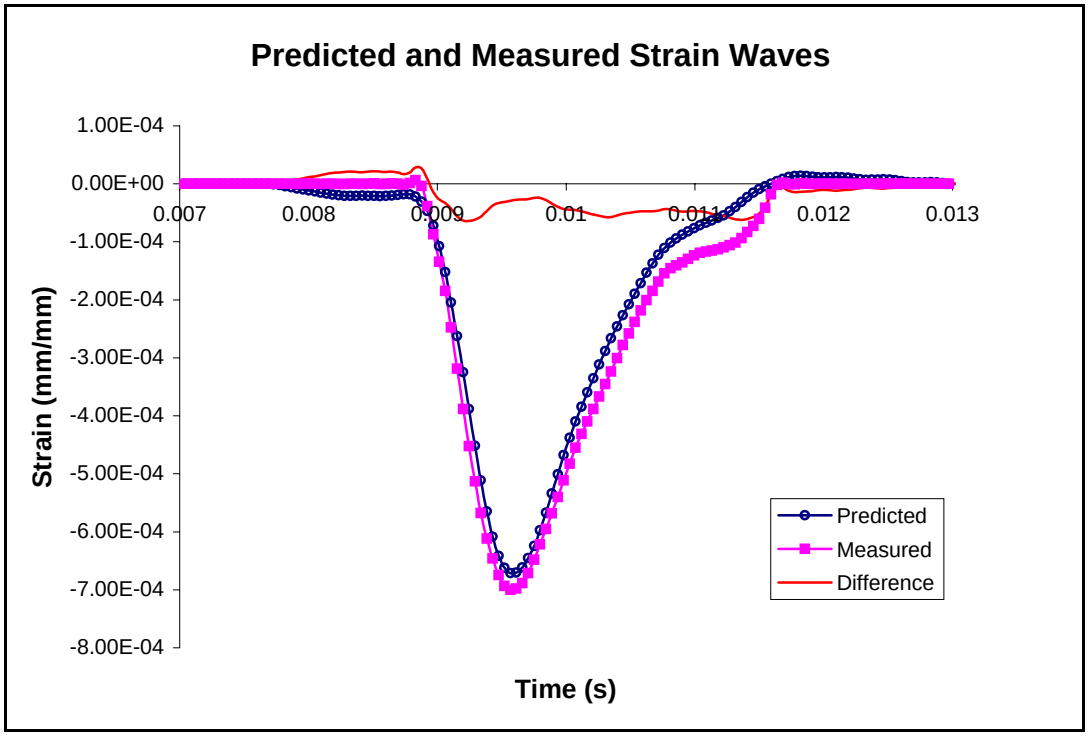


Figure 5.6b: Predicted and Measured Waves for Second Reflection

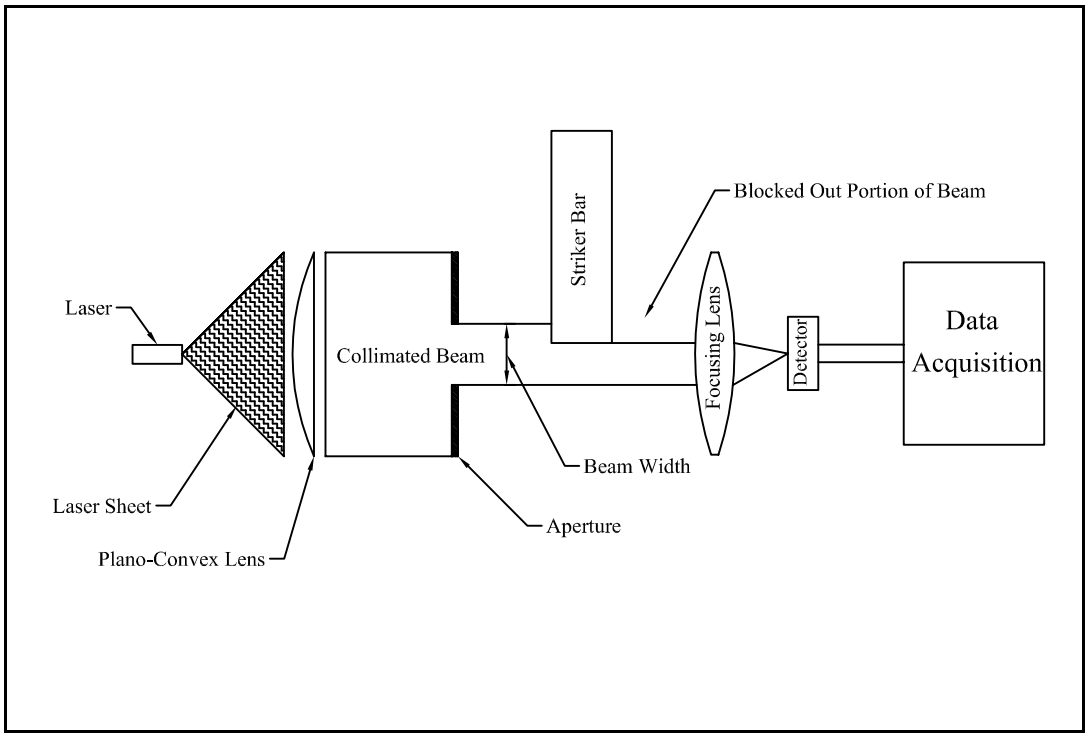


Figure 5.7: Schematic of ELVS System

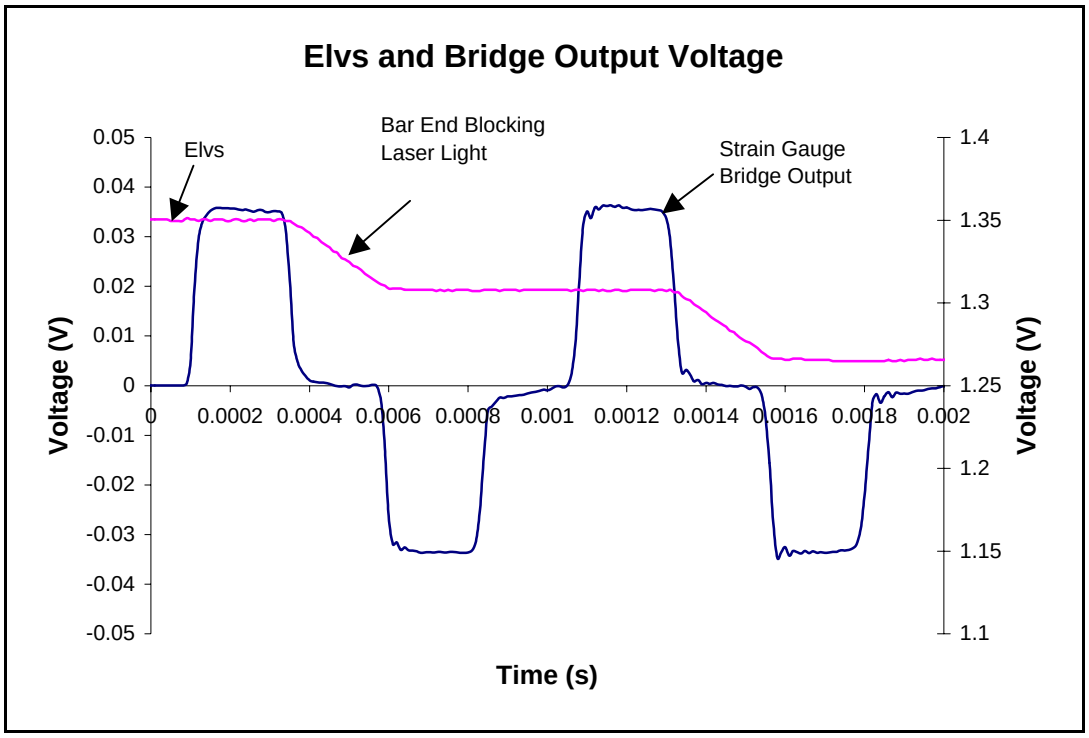


Figure 5.8a: Raw Data from ELVS Test

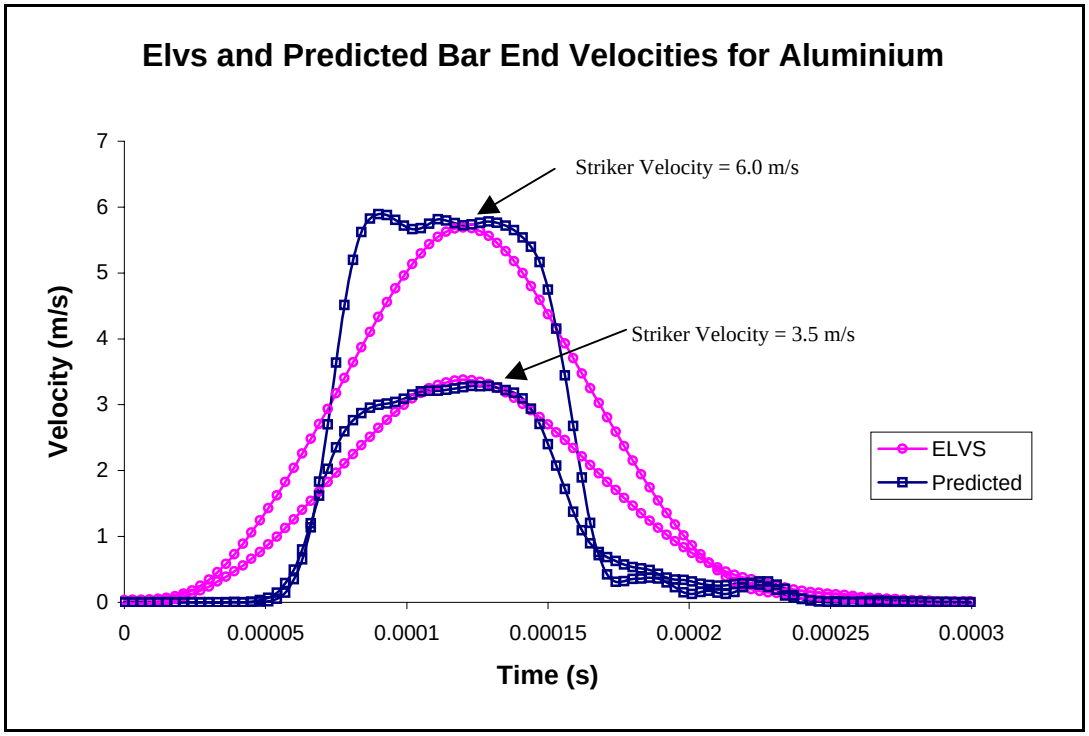


Figure 5.8b: ELVS and Predicted Results for Aluminium

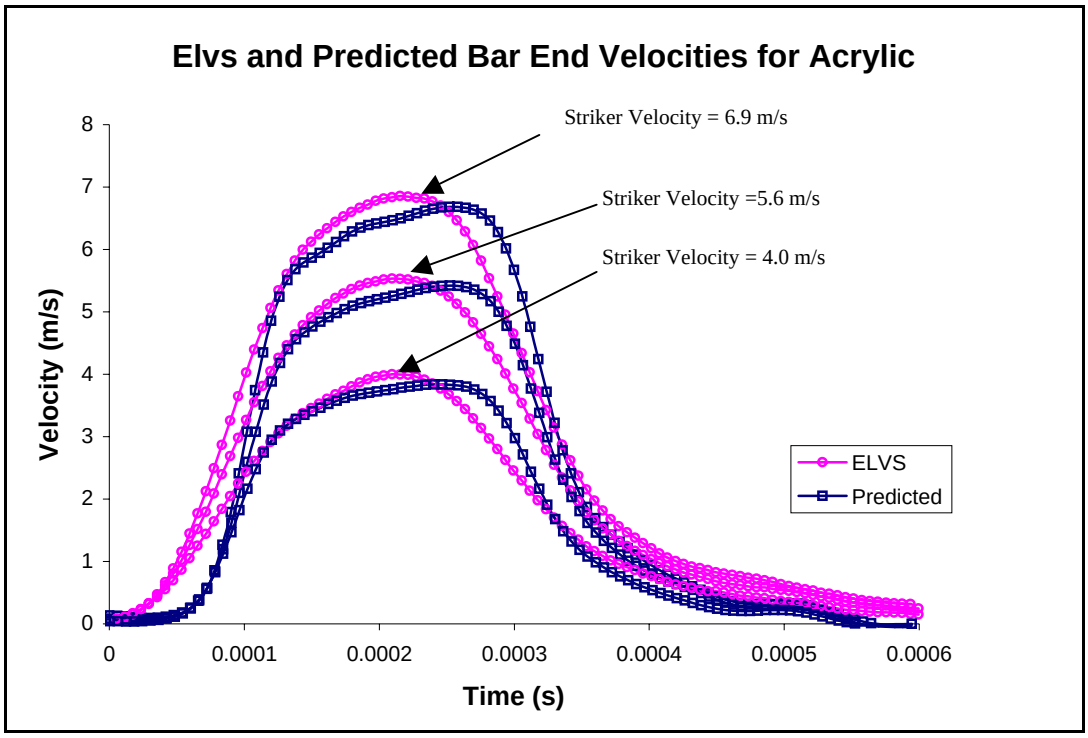


Figure 5.8c: ELVS and Predicted Results for Acrylic

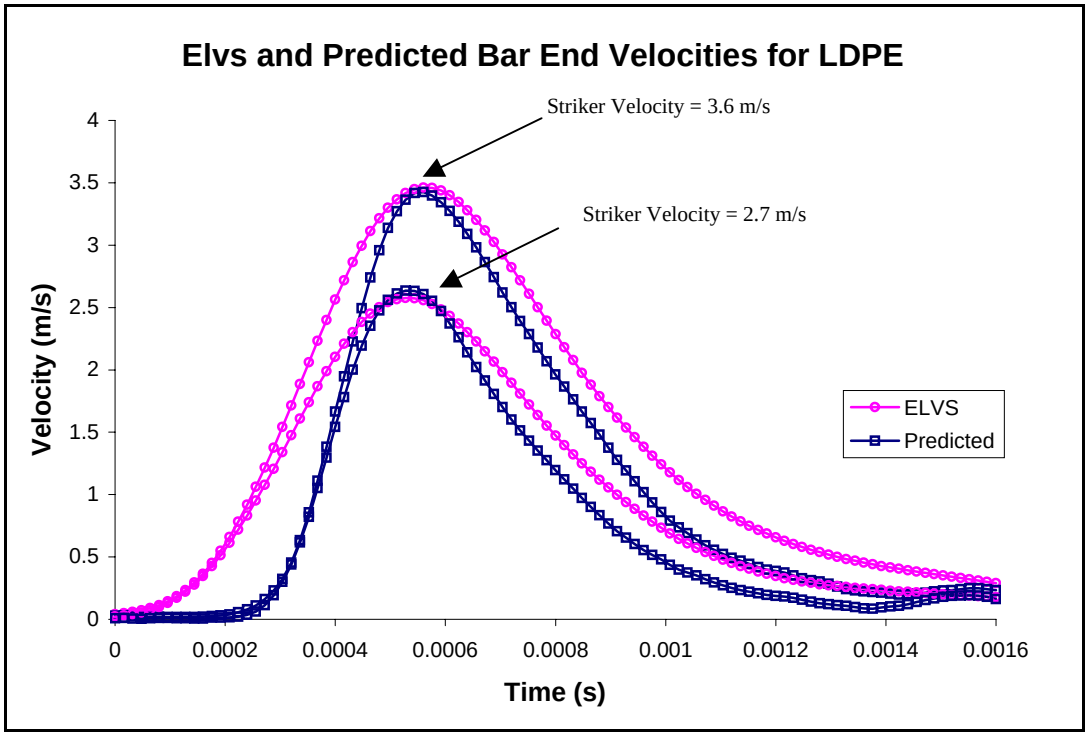


Figure 5.8d: ELVS and Predicted Results for LDPE

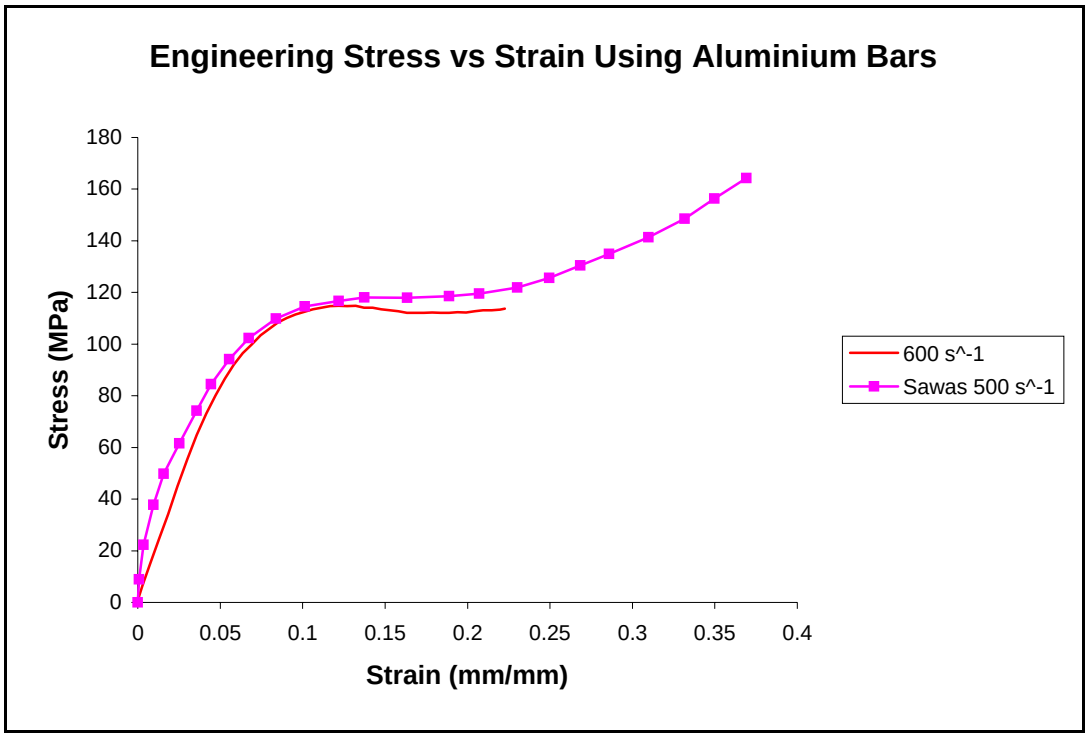


Figure 5.9a: Stress Strain Curves for Polycarbonate at 500 s^{-1} Using Aluminium Bars

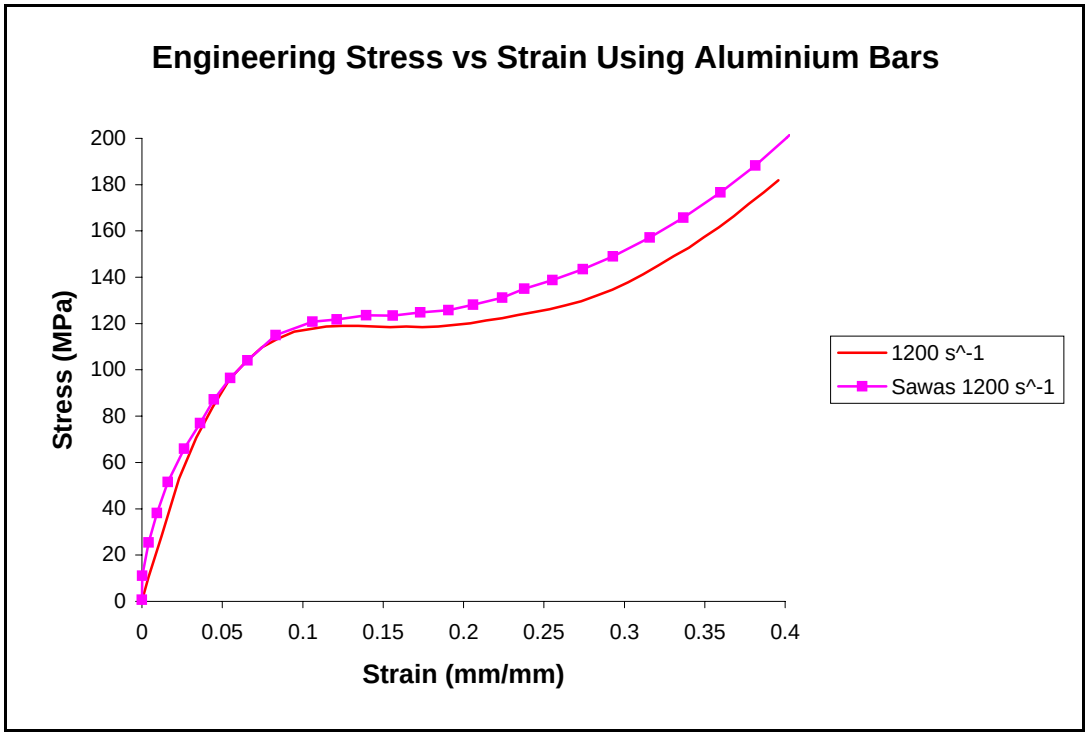


Figure 5.9b: Stress Strain Curves for Polycarbonate at 1200 s^{-1} Using Aluminium Bars

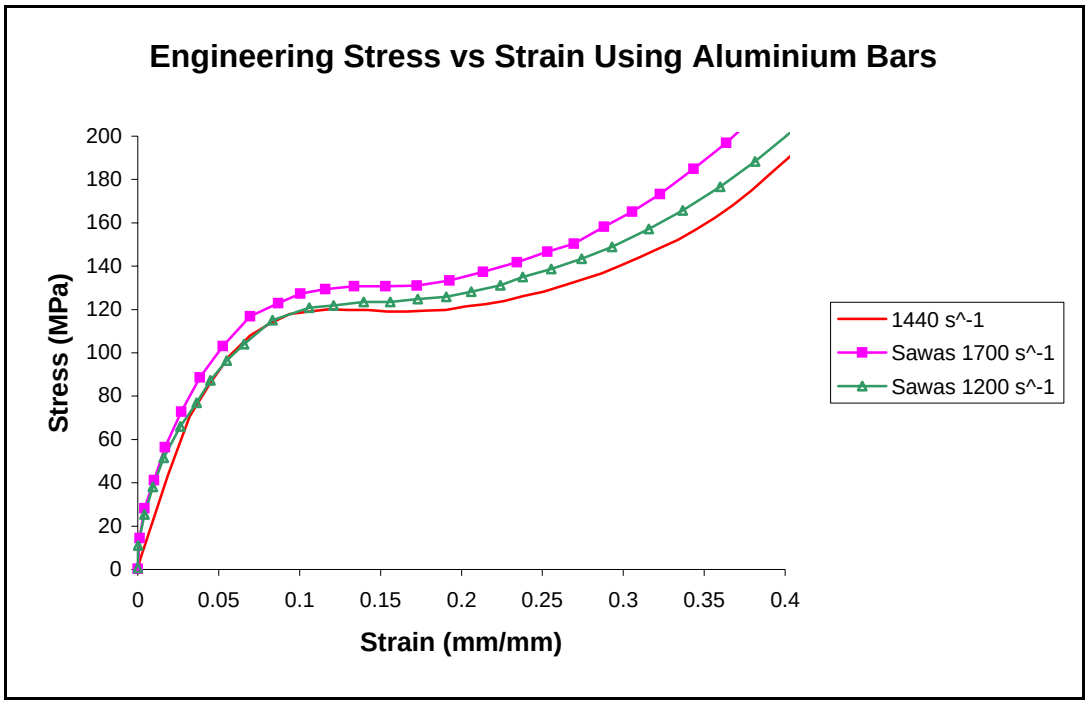


Figure 5.9c: Stress Strain Curves for Polycarbonate at 1700 s^{-1} Using Aluminium Bars

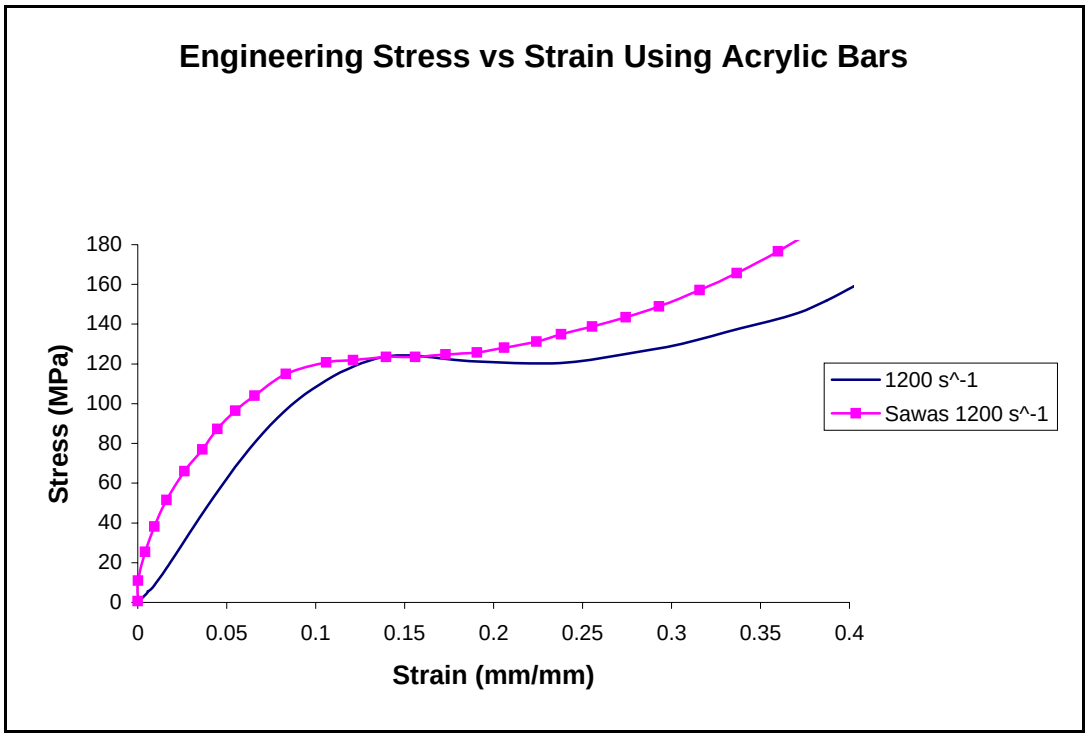


Figure 5.10a: Stress Strain Curves for Polycarbonate at 1200 s^{-1} Using Acrylic Bars

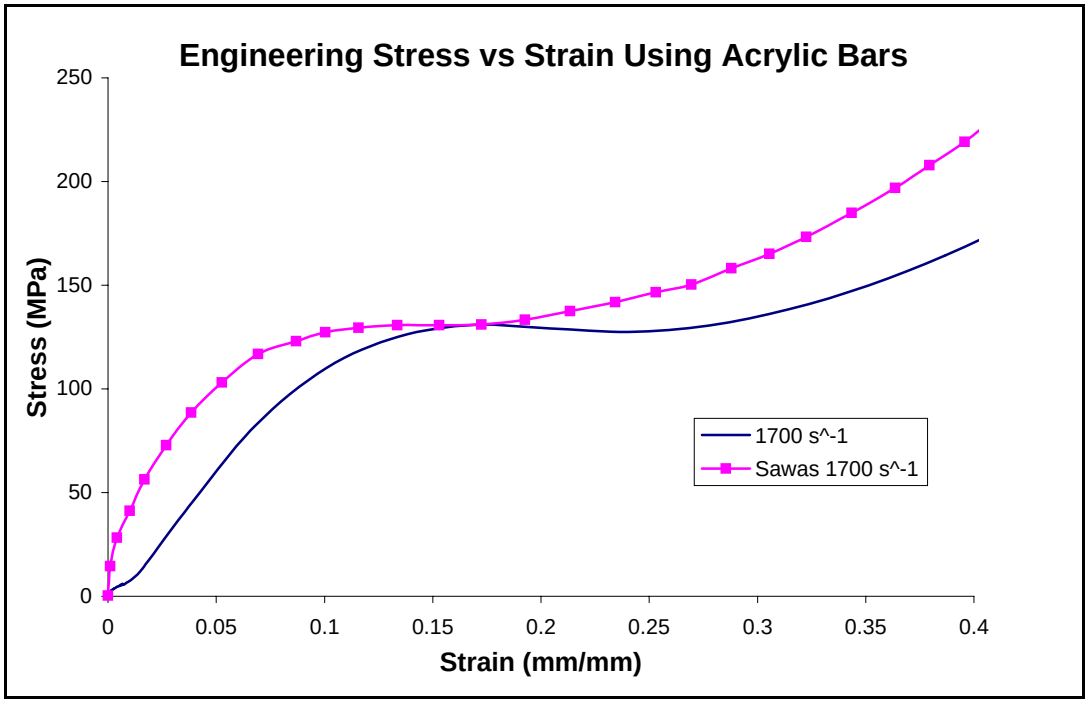


Figure 5.10b: Stress Strain Curves for Polycarbonate at 1700 s^{-1} Using Acrylic Bars

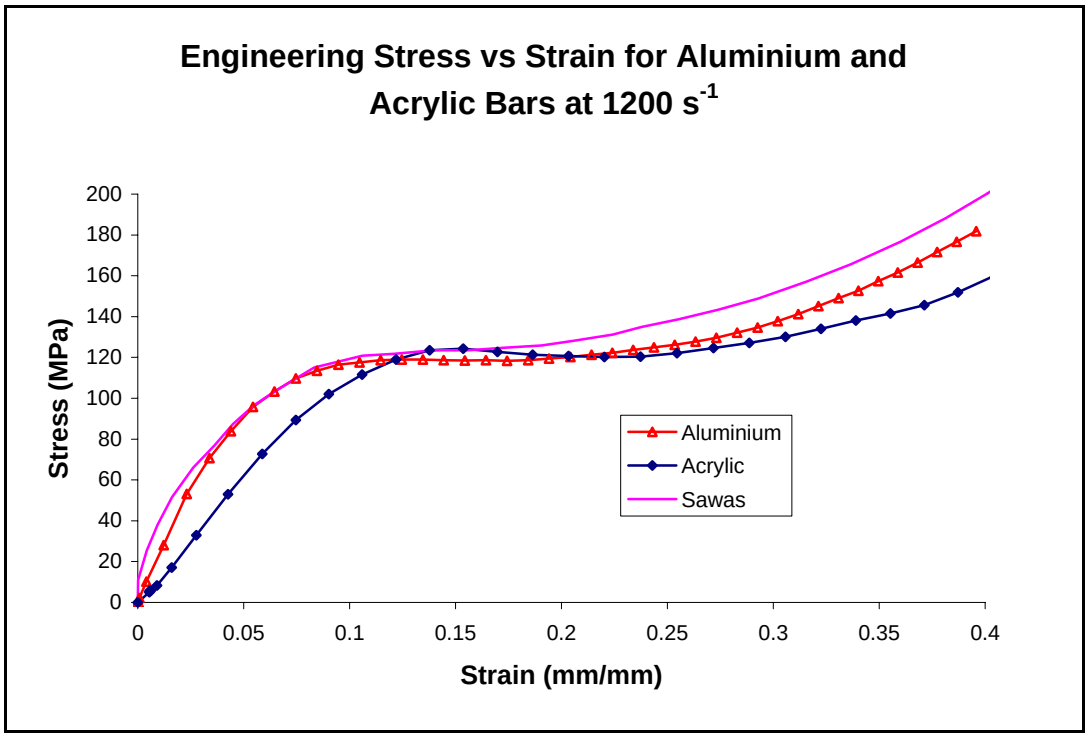


Figure 5.11: Comparison of Stress Strain Curves at 1200 s^{-1}

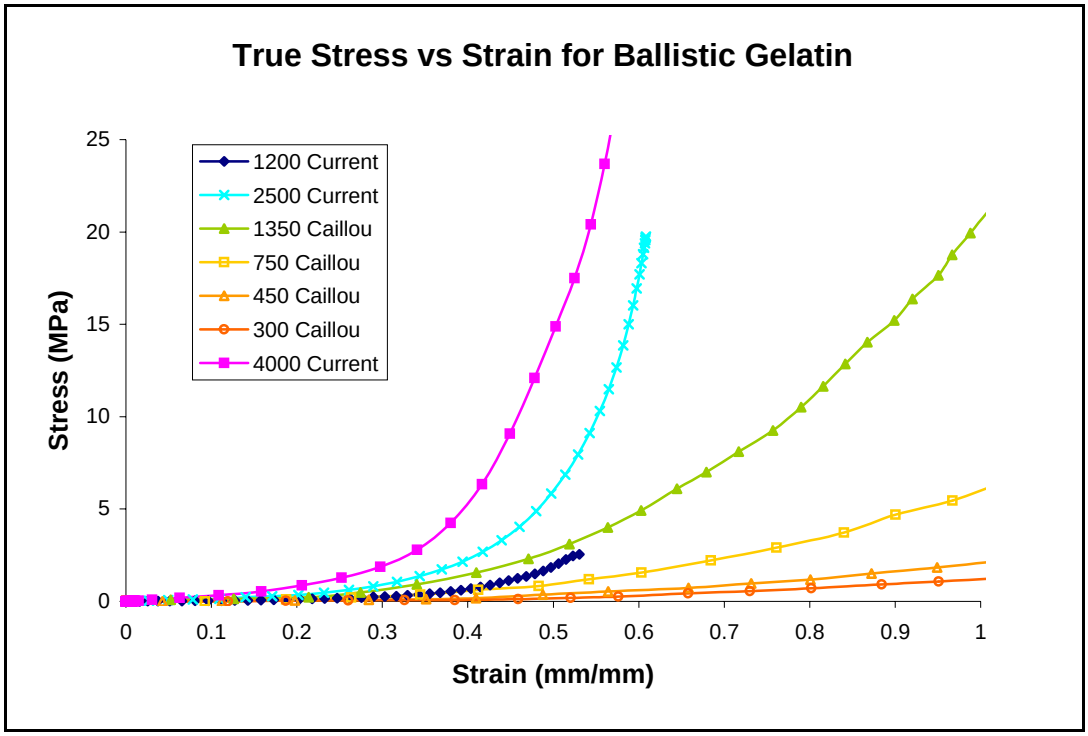


Figure 5.12: Ballistic Gelatin Results

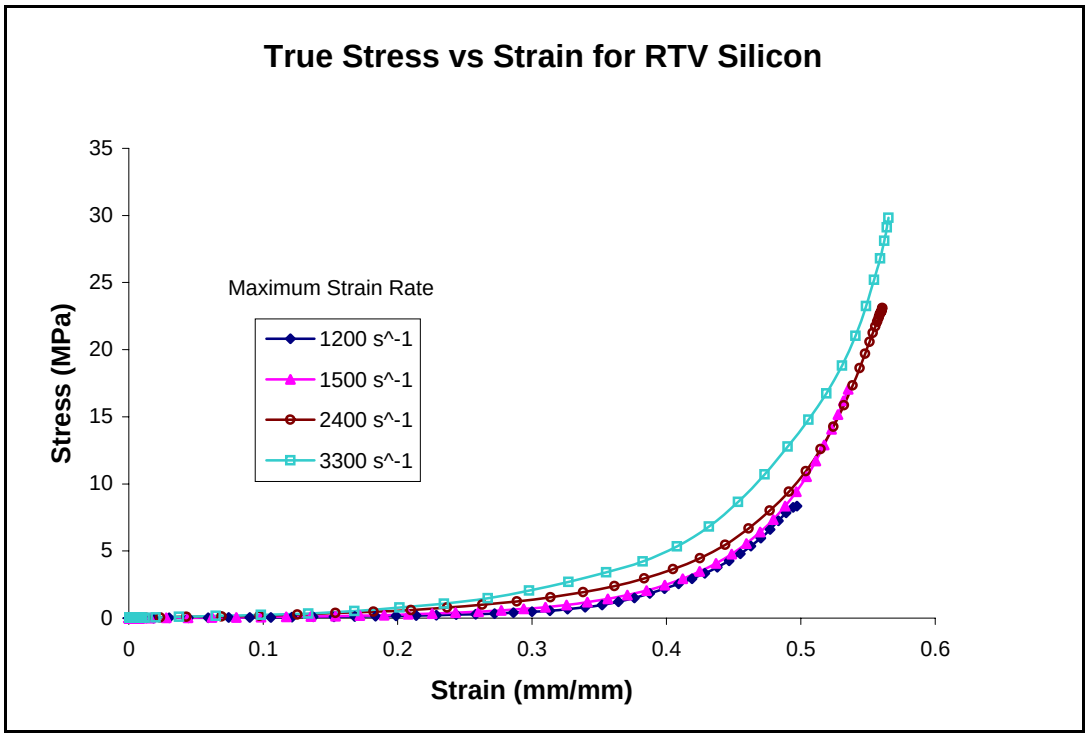


Figure 5.13: RTV Silicon Results

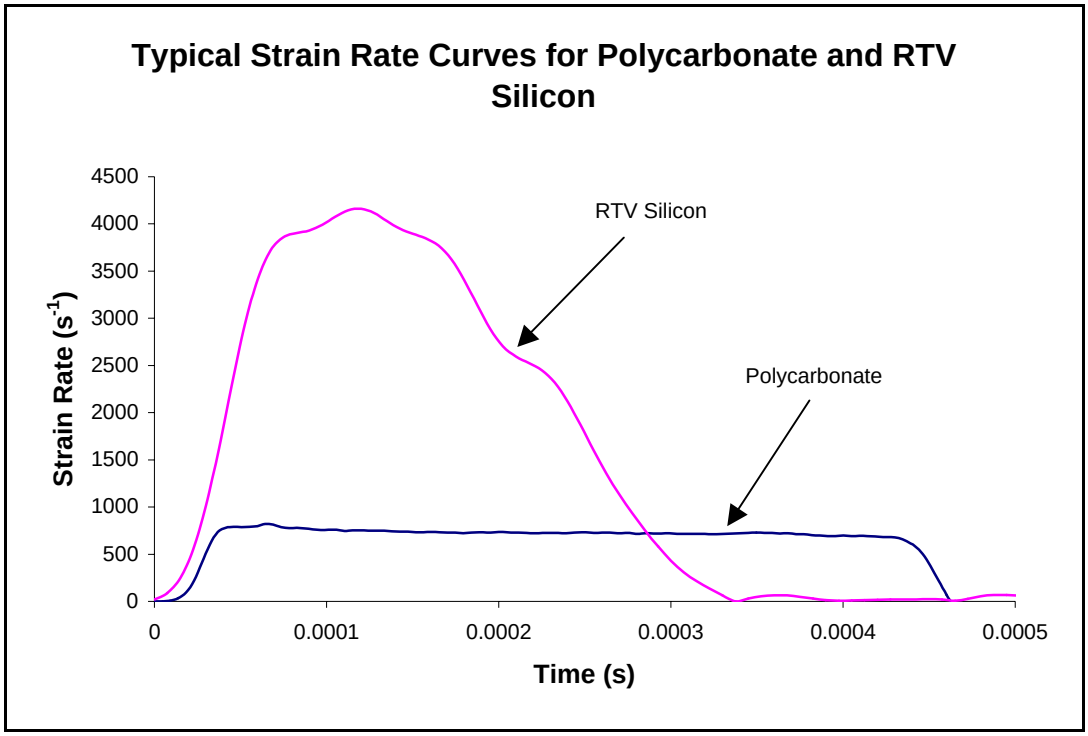


Figure 5.14: Comparison of Strain Rates

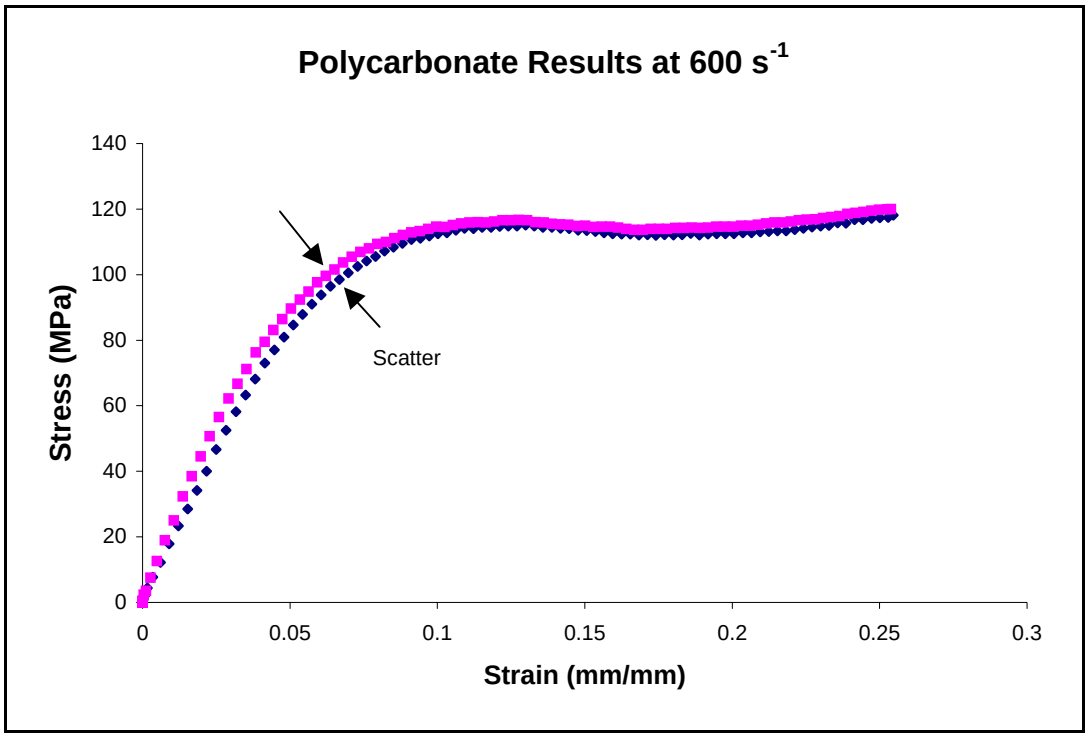
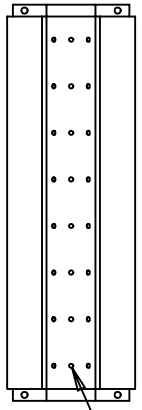
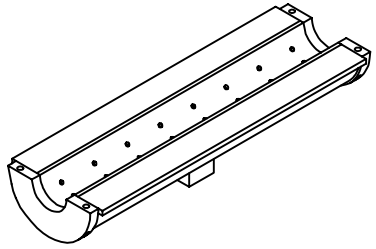


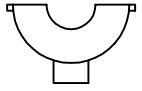
Figure 5.15: Experimental Scatter

Appendix A

DWG. NO.	SH	REV.

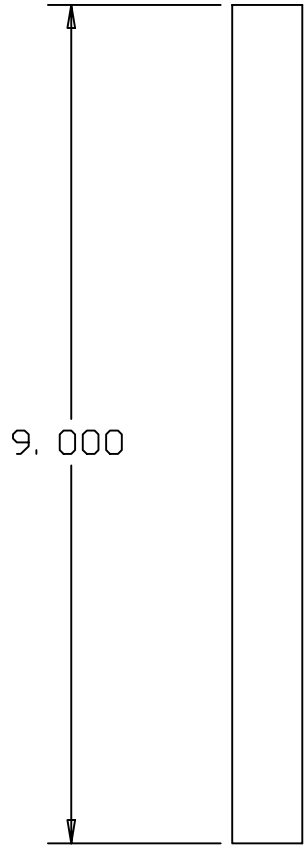


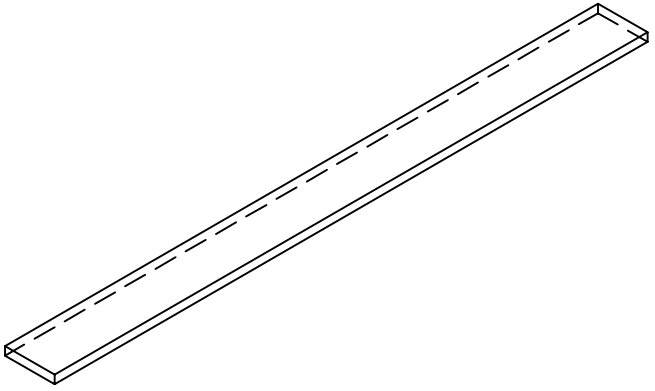


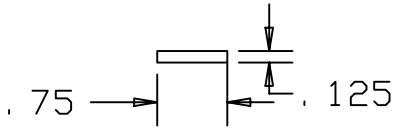


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE ALUMINIUM	PROJECT HOPKINSON BAR
	QUANTITY 1	TITLE AIR BEARING ASSEMBLY
TREATMENT NONE	NOTES	SIZE: A; REVISION NO.: 1; PART NO.:
FINISH NONE		
DATE AUGUST 31, 2000	DO NOT SCALE DRAWING	SCALE; SHEET

DWG. NO.	SH	REV.







UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN INCHES
TOLERANCES ARE:

DECIMALS
. X=±.1
. XX=±0.02
. XXX=±0.005

ANGLES
X=±0.5
X.X=±0.05

TREATMENT: NONE

FINISH: NONE

DATE: SEPT. 8, 2000

MATERIAL TYPE: 6061-T6

QUANTITY: 14

NOTES:

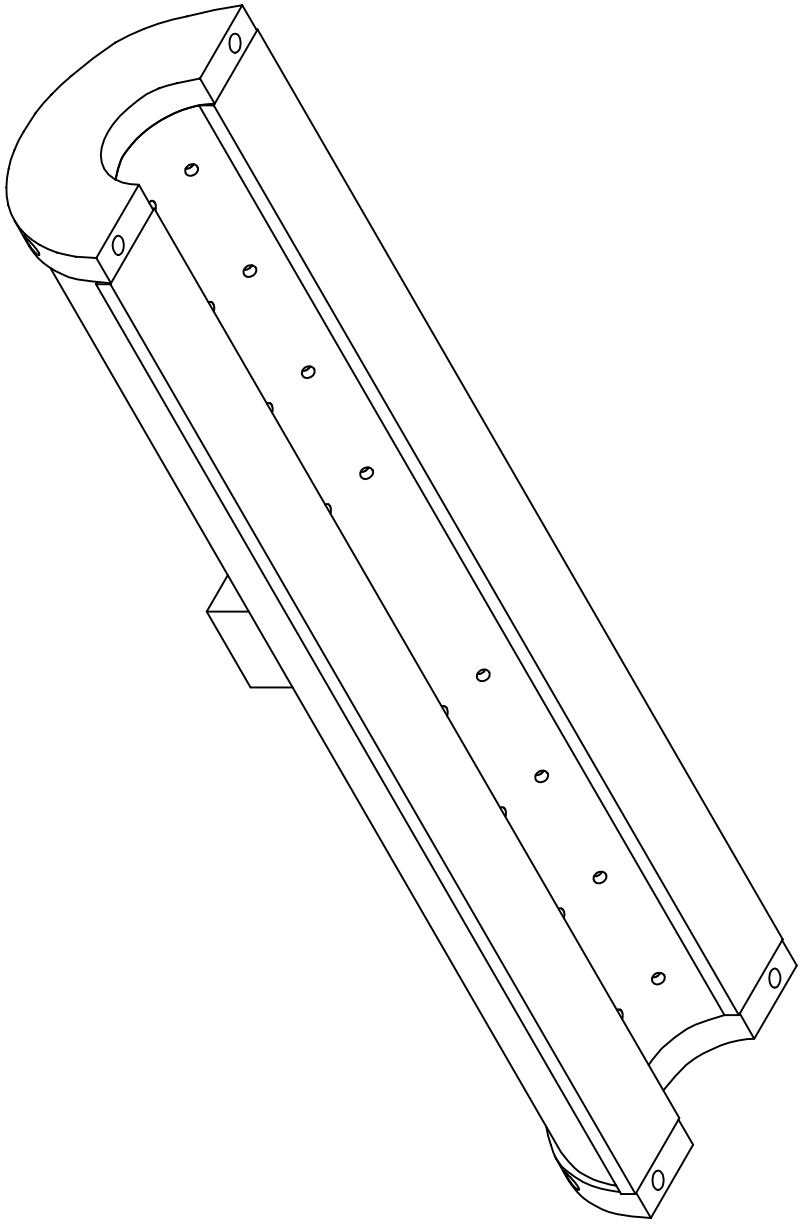
DO NOT SCALE DRAWING

PROJECT: HOPKINSON BAR

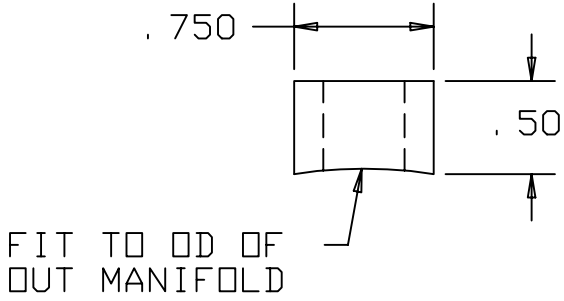
TITLE: SIDE PIECE

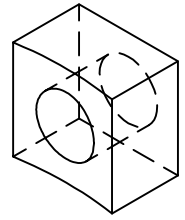
SIZE	REVISION NO.	PART NO.
A	1	

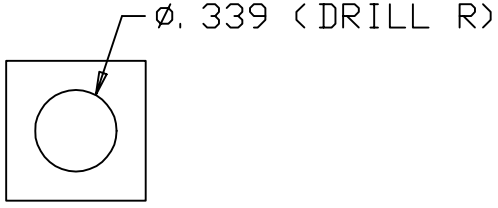
SCALE		SHEET



DWG. NO.	SH	REV.

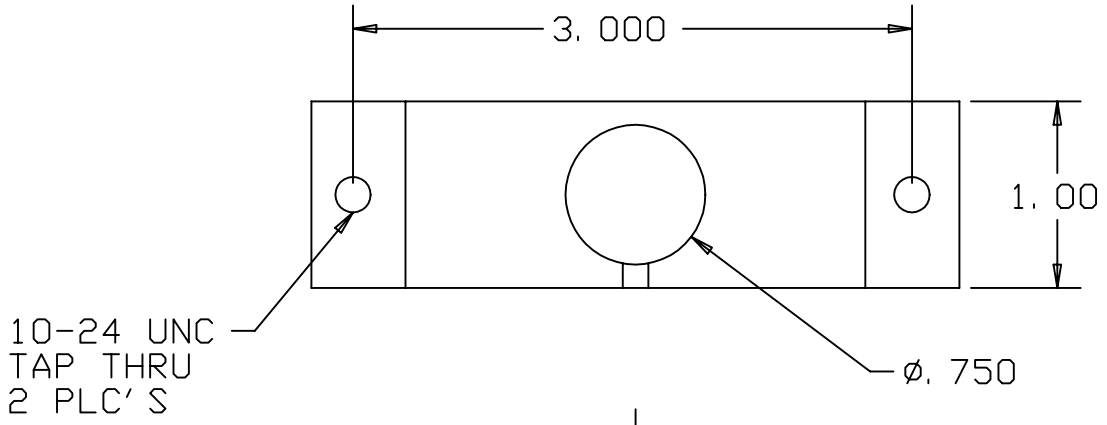


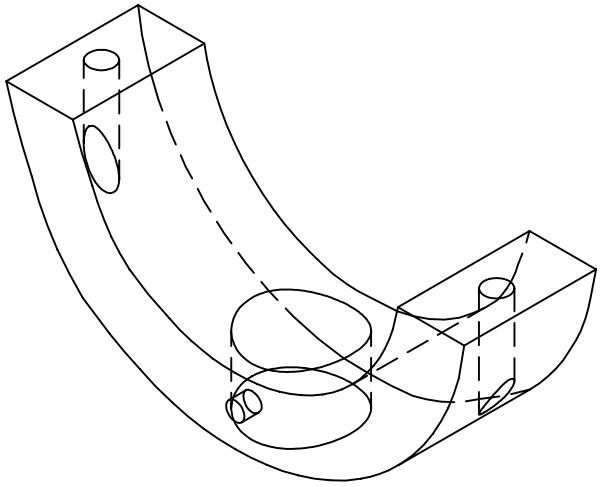


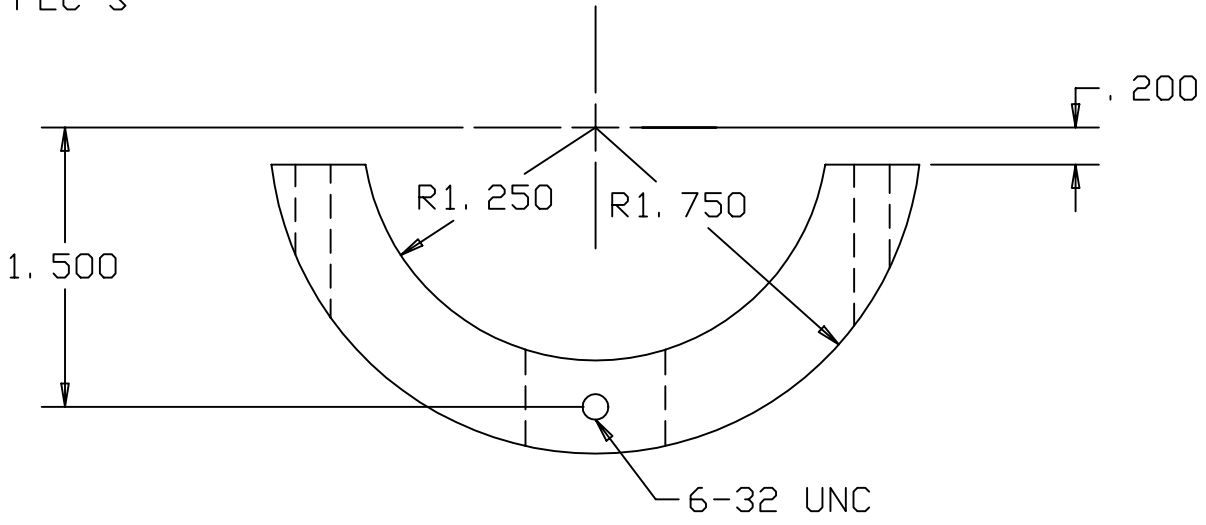


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE 6061-T6	PROJECT HOPKINSON BAR
	QUANTITY 7	TITLE AIR PLUG
TREATMENT NONE	NOTES	SIZE A | REVISION NO. 1 | PART NO.
FINISH NONE		
DATE SEPT. 8, 2000	DO NOT SCALE DRAWING	SCALE | SHEET

DWG. NO.	SH	REV.

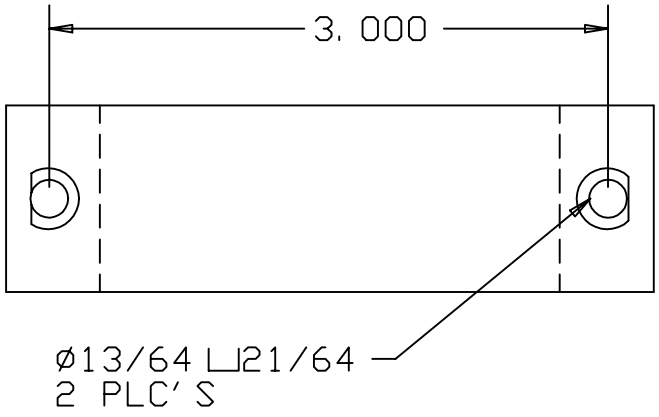


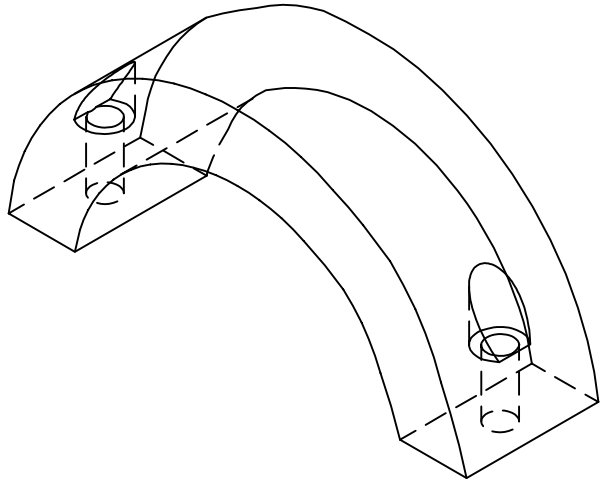


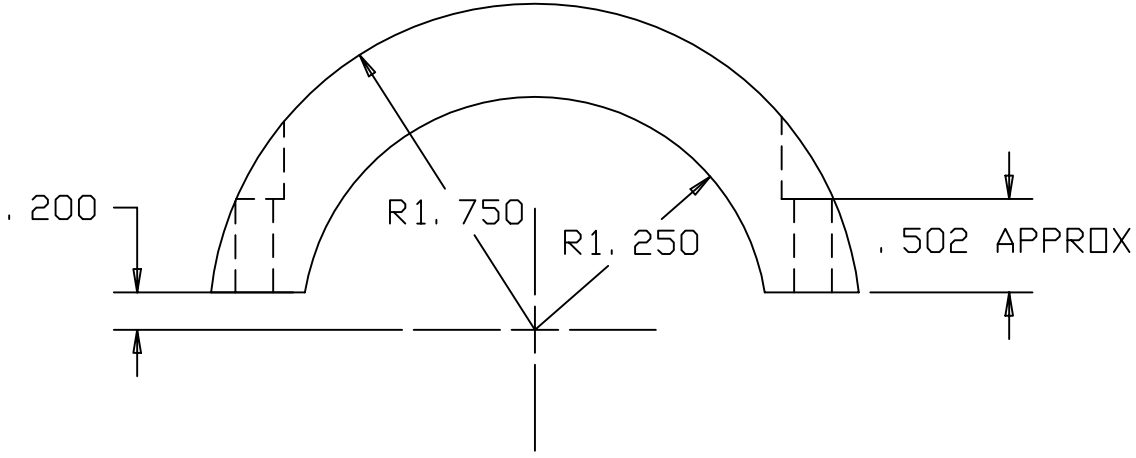


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE Aluminum 2024	PROJECT HOPKINSON BAR
TREATMENT NONE	QUANTITY 14	TITLE LOWER AIR BEARING SUPPORT
FINISH NONE	NOTES	SIZE A / REVISION NO. 1 / PART NO.
DATE SEPTEMBER 12, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

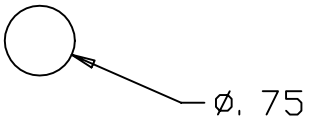


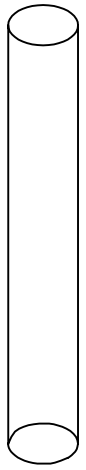


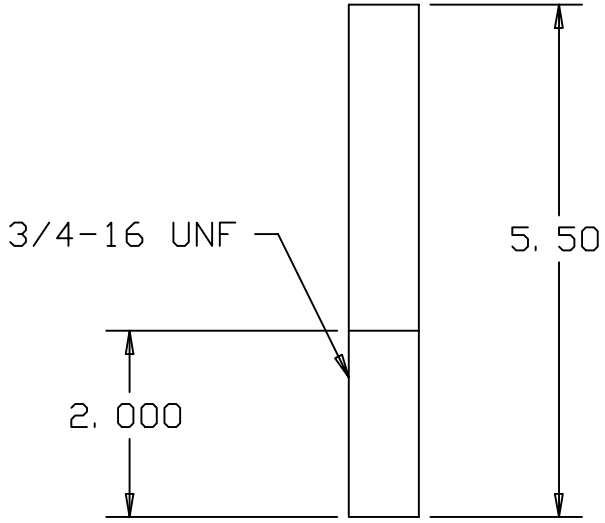


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE ALUMINUM 2024	PROJECT HOPKINSON BAR
	QUANTITY 14	TITLE UPPER AIR BEARING CLAMP
TREATMENT NONE	NOTES	SIZE A / REVISION NO. 1 / PART NO.
FINISH NONE		
DATE SEPTEMBER 12, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

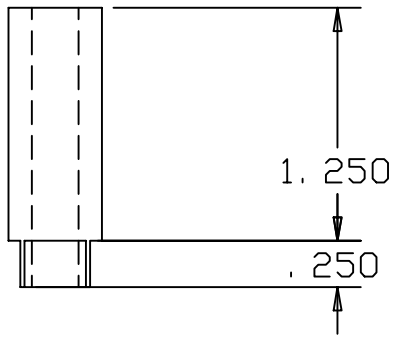


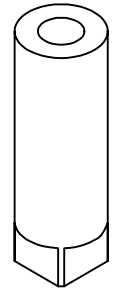


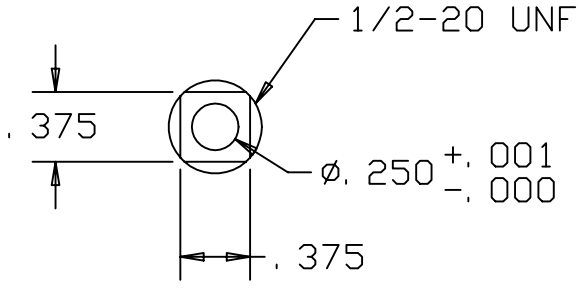


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE STEEL	PROJECT HOPKINSON BAR
	QUANTITY 14	TITLE AIR BEARING SUPPORT ROD
TREATMENT NONE	NOTES	SIZE A | REVISION NO. 1 | PART NO.
FINISH NONE		
DATE SEPTEMBER 14, 2000	DO NOT SCALE DRAWING	SCALE | | SHEET

DWG. NO.	SH	REV.

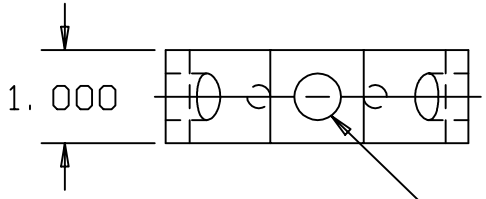




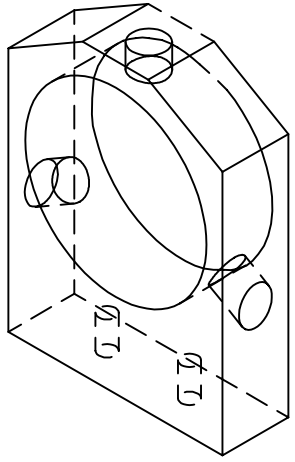


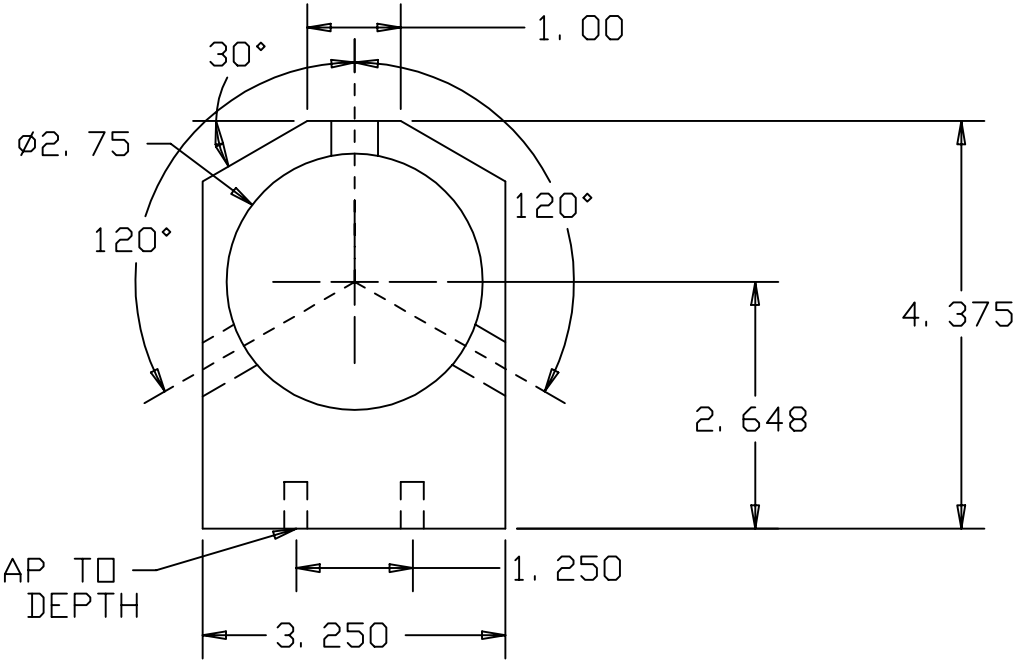
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE: BRASS	PROJECT: HOPKINSON BAR
	QUANTITY: 6	TITLE: ALIGNMENT PIN ADJUSTER
TREATMENT: NONE	NOTES	SIZE: A / REVISION NO.: 2 / PART NO.
FINISH: NONE		
DATE: SEPTEMBER 15, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.



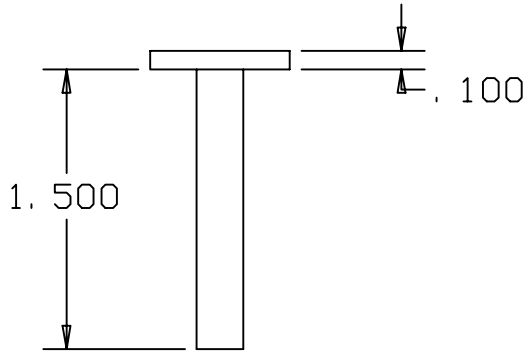
1/2-20 UNF TAP
THRU 3 PLC'S

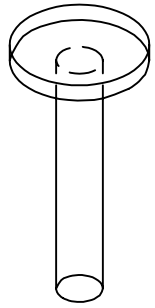


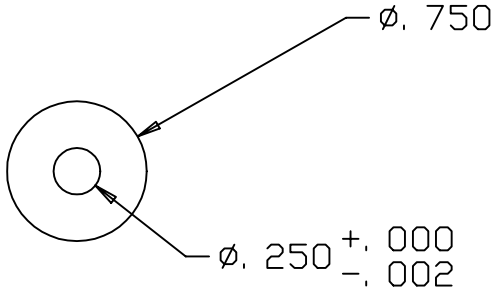


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE ALUMINUM 2024	PROJECT HOPKINSON BAR
	QUANTITY 2	TITLE ALIGNMENT HOUSING
TREATMENT NONE	NOTES	SIZE A / REVISION NO. 3 / PART NO.
FINISH NONE		
DATE SEPTEMBER 15, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

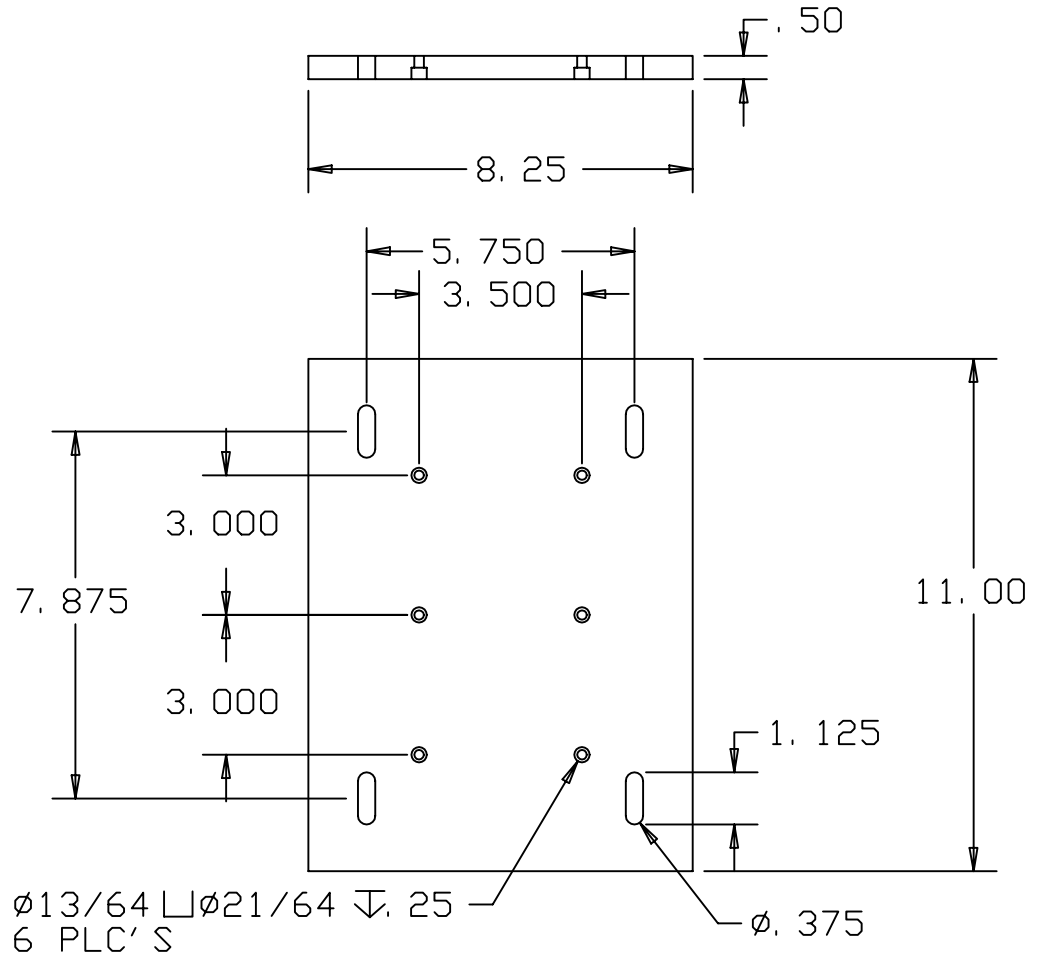


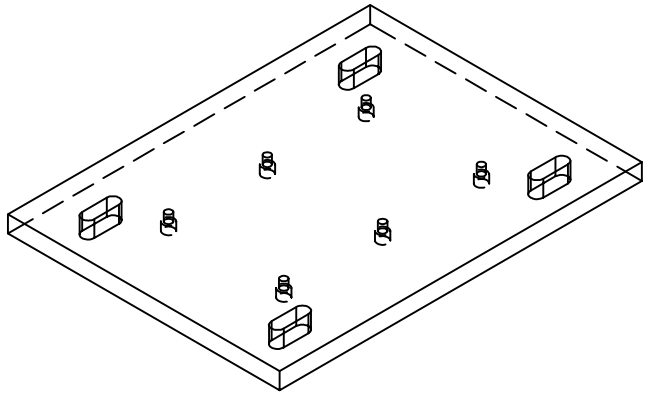




UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE Al 2024	PROJECT HOPKINSON BAR
	QUANTITY 6	TITLE ALIGNMENT PIN
TREATMENT NONE	NOTES	SIZE A | REVISION NO. 2 | PART NO.
FINISH NONE		
DATE SEPTEMBER 15, 2000	DO NOT SCALE DRAWING	SCALE | SHEET

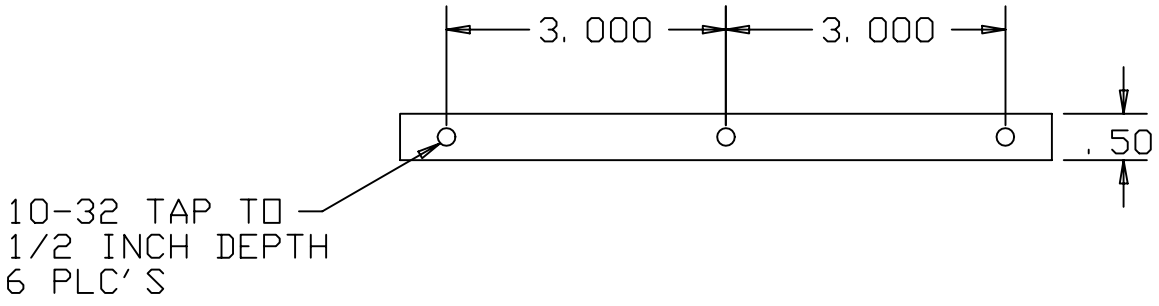
DWG. NO.	SH	REV.

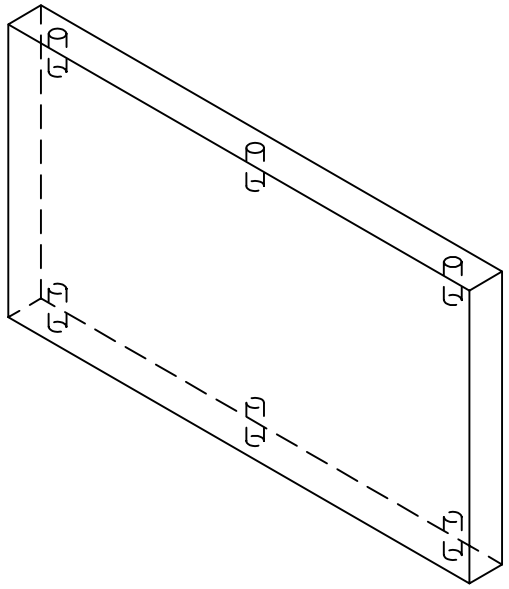


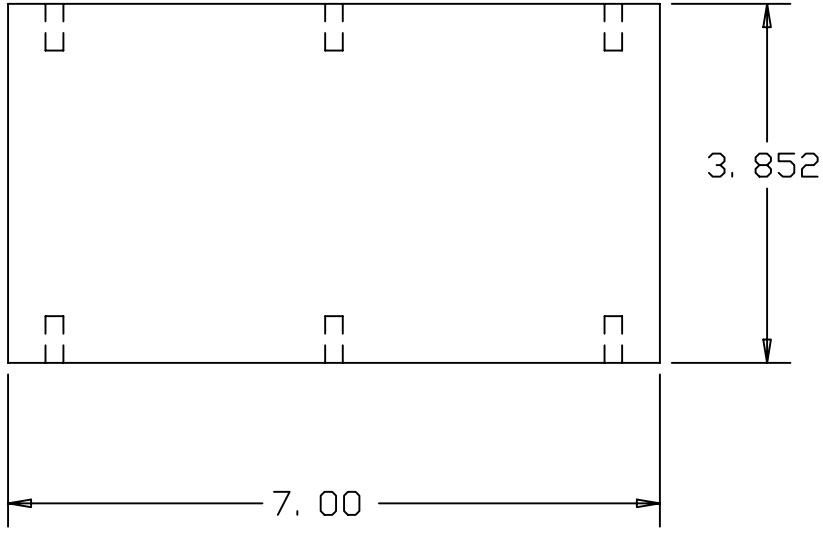


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE: MILD STEEL	PROJECT: HOPKINSON BAR
	QUANTITY: 1	TITLE: ALIGNMENT LOWER SUPPORT MOUNT
TREATMENT: NONE	NOTES:	SIZE: A / REVISION NO.: 1 / PART NO.:
FINISH: PAINTED BLACK		
DATE: SEPTEMBER 15, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

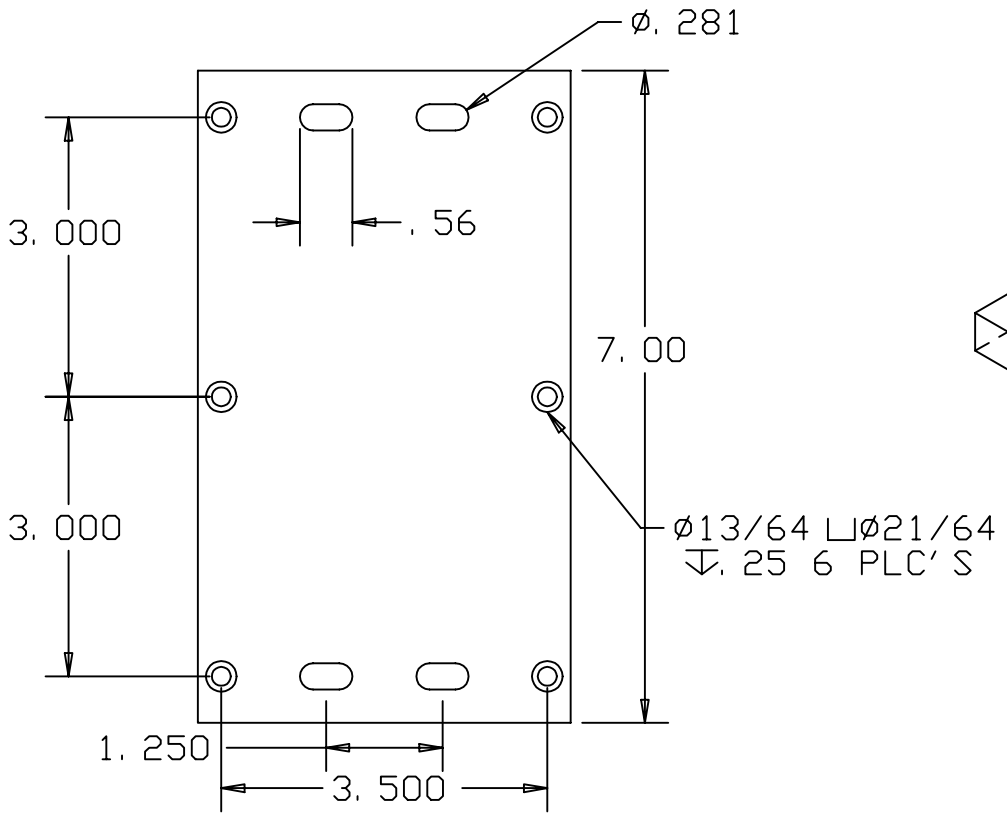


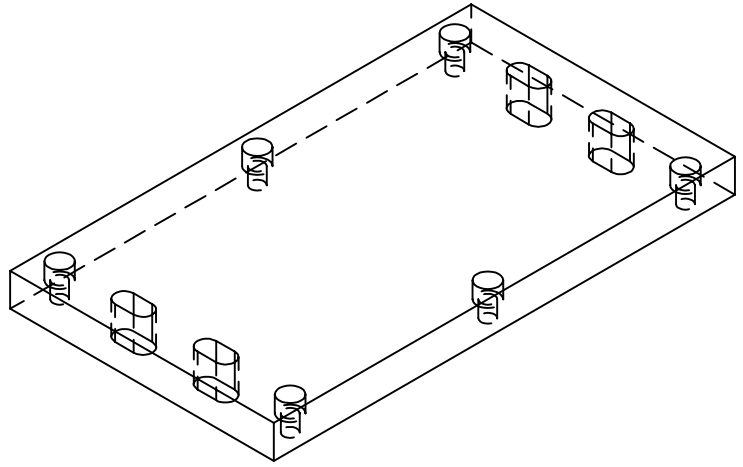




UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE: STEEL	PROJECT: HOPKINSON BAR
TREATMENT: NONE	QUANTITY: 2	TITLE: ALIGNMENT SIDE SUPPORT MOUNT
FINISH: PAINTED BLACK	NOTES: DO NOT SCALE DRAWING	SIZE: A / REVISION NO.: 1 / PART NO.:
DATE: SEPTEMBER 15, 2000		SCALE / SHEET

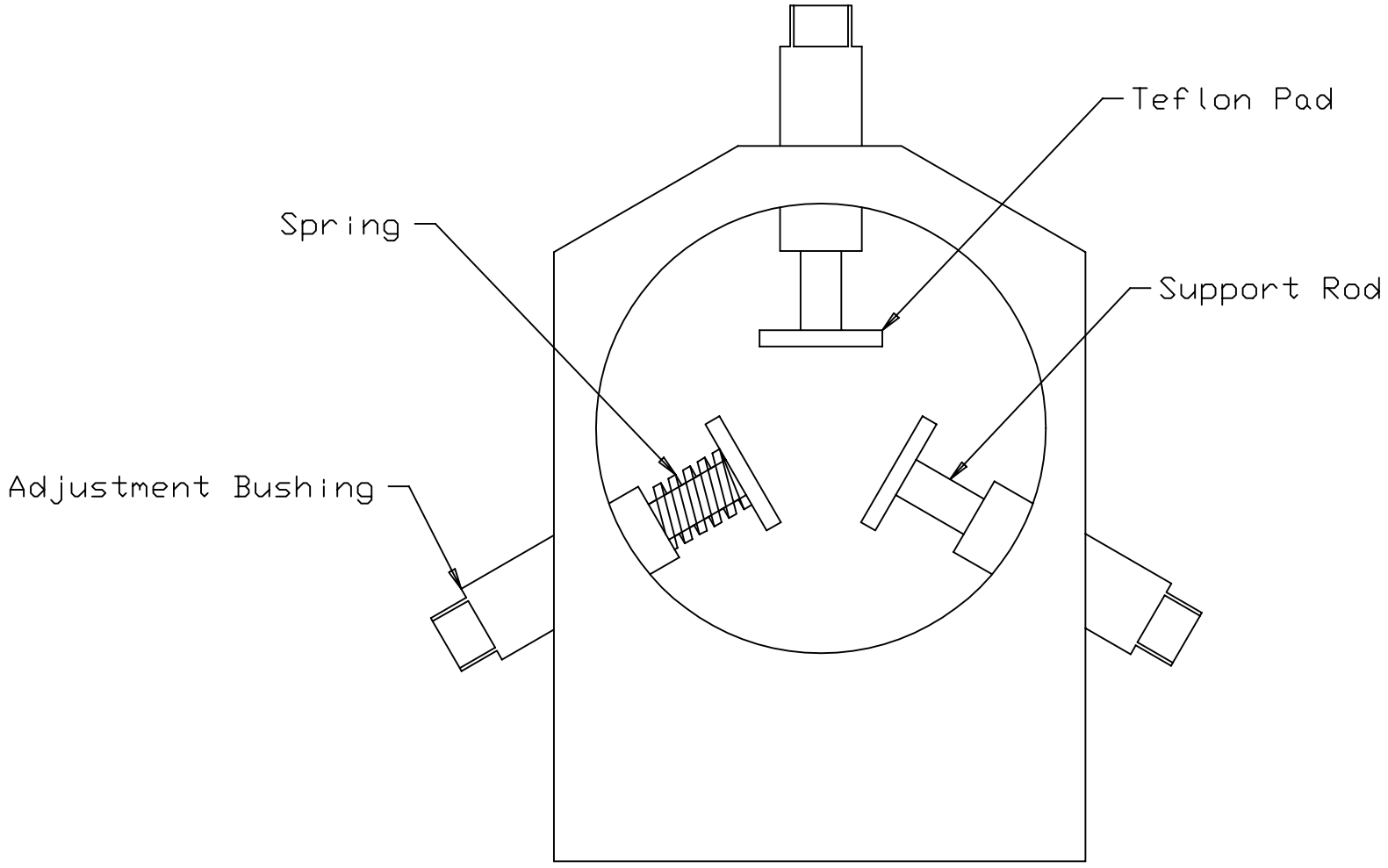
DWG. NO.	SH	REV.





UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE STEEL	PROJECT HOPKINSON BAR
	QUANTITY 1	TITLE ALIGNMENT UPPER SUPPORT MOUNT
TREATMENT NONE	NOTES	SIZE: A; REVISION NO.: 1; PART NO.:
FINISH PAINTED BLACK		
DATE SEPTEMBER 15, 2000	DO NOT SCALE DRAWING	SCALE; SHEET

DWG. NO.	SH	REV.



UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN INCHES
TOLERANCES ARE:

DECIMALS
. X=±. 1
. XX=±0. 02
. XXX=±0. 005

ANGLES
X=±0. 5
X. X=±0. 05

TREATMENT: NONE

FINISH: NONE

DATE: AUGUST 10, 2000

MATERIAL TYPE:

QUANTITY: 1

NOTES:

DO NOT SCALE DRAWING

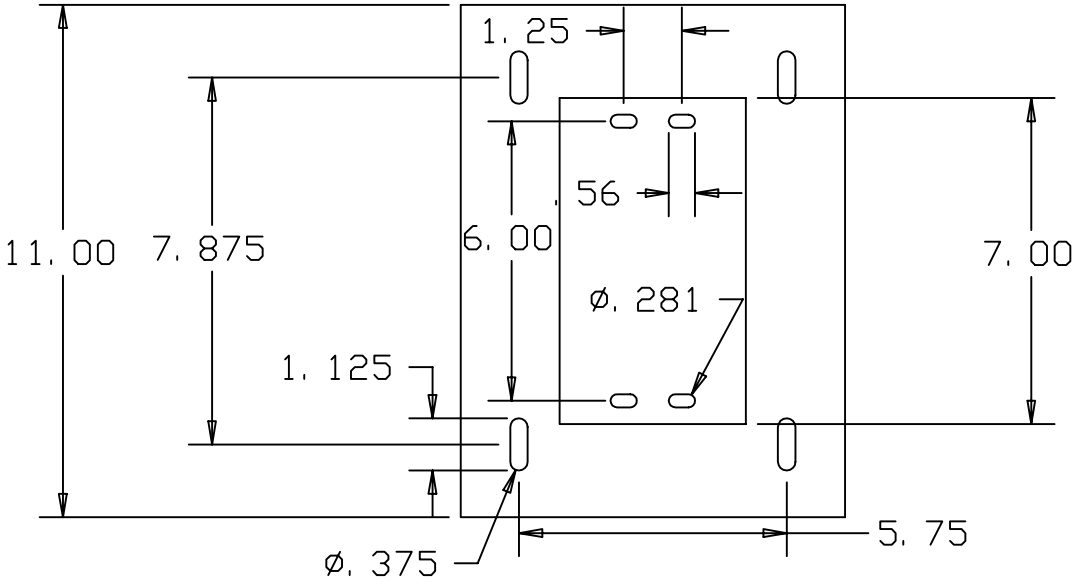
PROJECT: HOPKINSON BAR

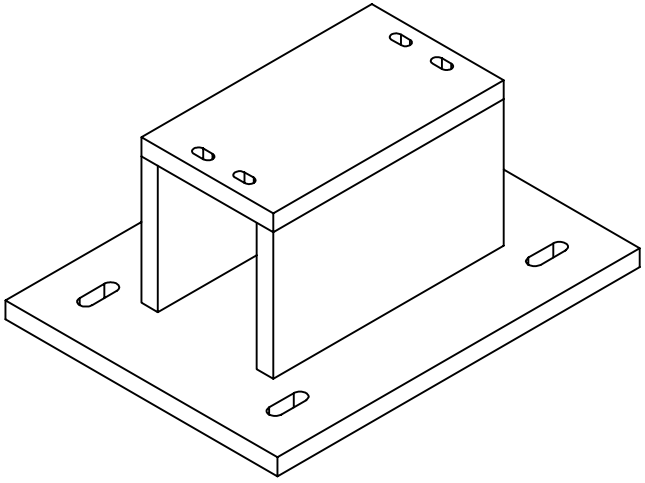
TITLE: ALIGNMENT ASSEMBLY

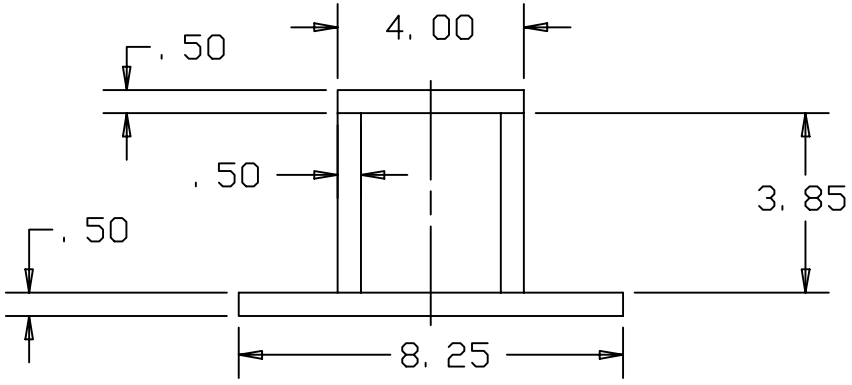
SIZE	REVISION NO.	PART NO.
A	1	

SCALE		SHEET

DWG. NO.	SH	REV.







UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES
TOLERANCES ARE:

DECIMALS
.X=±.1
.XX=±0.02
.XXX=±0.005

ANGLES
X=±0.5
X.X=±0.05

TREATMENT: NONE	MATERIAL TYPE: MILD STEEL
FINISH: PAINT BLACK	QUANTITY: 1
DATE: SEPTEMBER 12, 2000	NOTES: DO NOT SCALE DRAWING

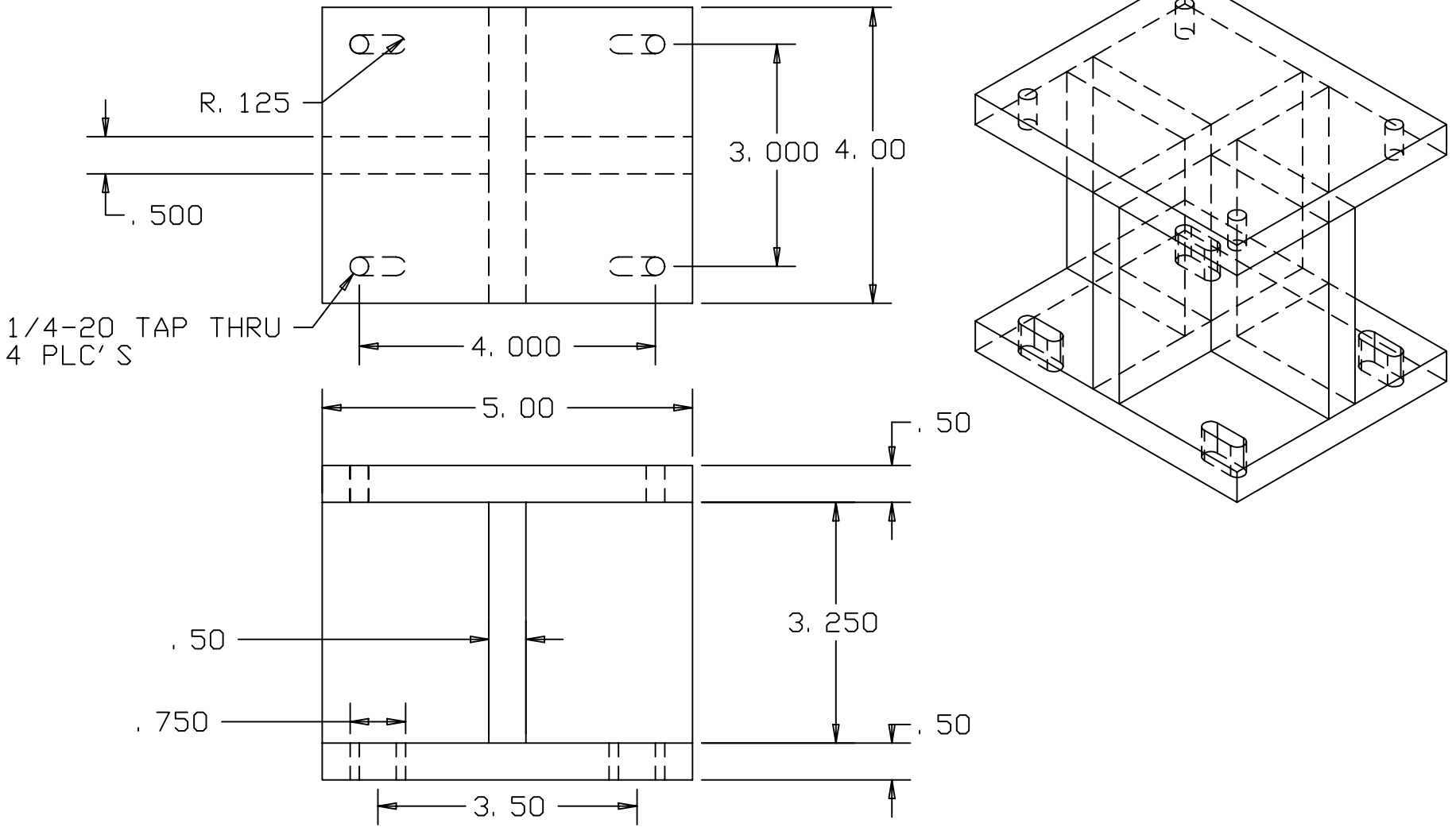
PROJECT: HOPKINSON BAR

TITLE: ALIGNMENT SUPPORT

SIZE	REVISION NO.	PART NO.
A	1	

SCALE		SHEET

DWG. NO.	SH	REV.



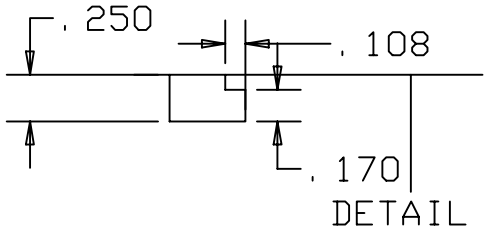
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES
TOLERANCES ARE:

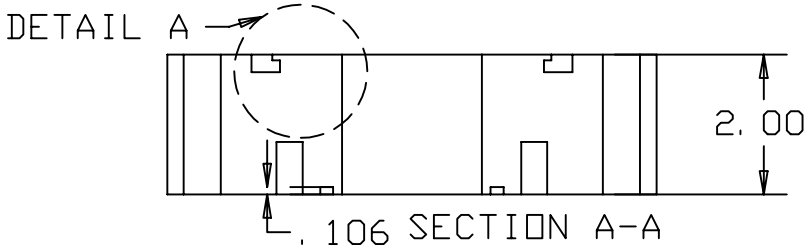
DECIMALS
. X=±. 1
. XX=±0. 02
. XXX=±0. 005

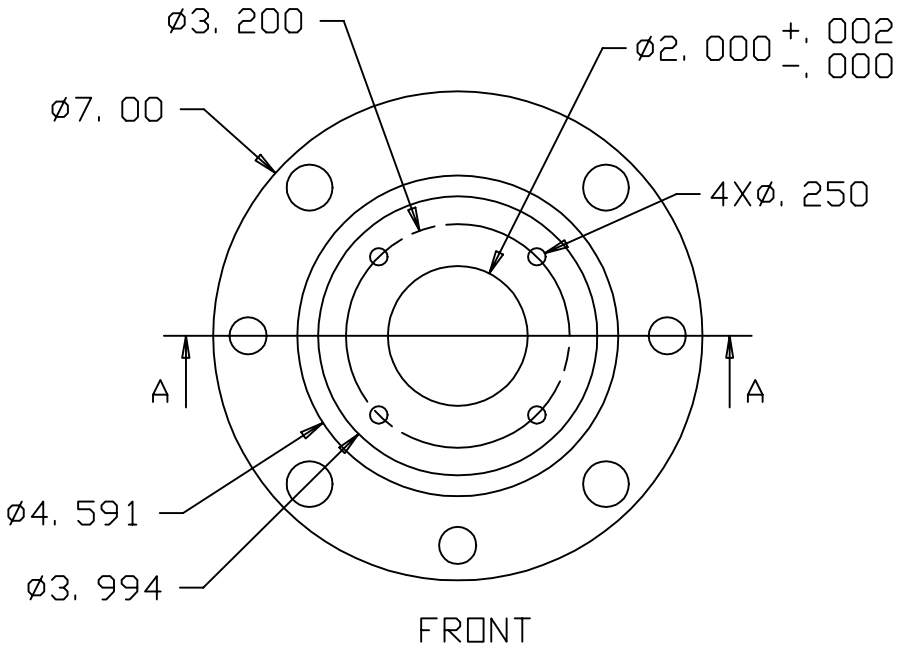
ANGLES
X=±0. 5
X. X=±0. 05

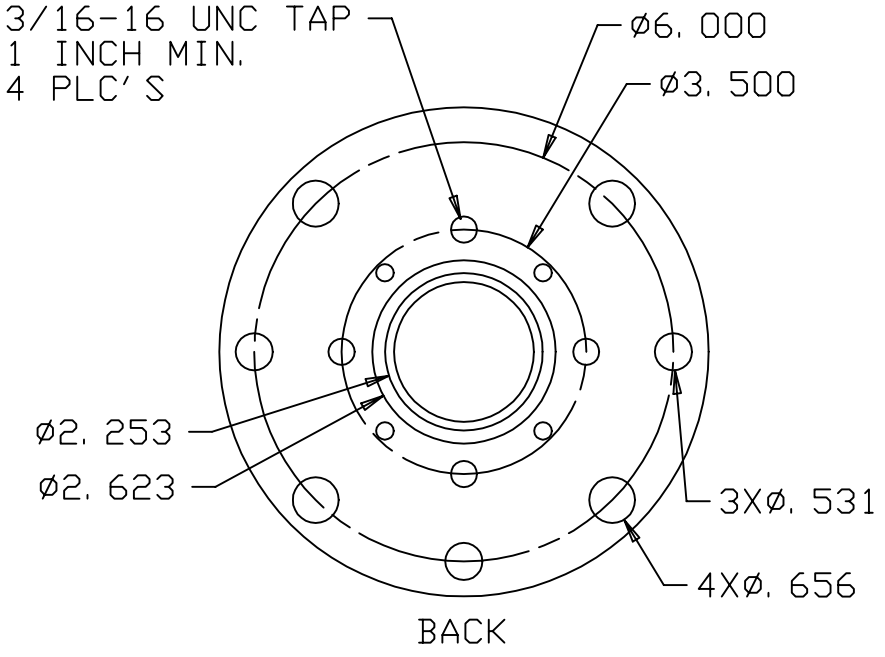
TREATMENT: NONE	MATERIAL TYPE: STEEL	PROJECT: HOPKINSON BAR
FINISH: PAINTED BLACK	QUANTITY: 3	TITLE: STRIKER HOUSING BASE
DATE: SEPTEMBER 13, 2000	NOTES: DO NOT SCALE DRAWING	SIZE: A / REVISION NO.: 1 / PART NO.:
		SCALE / SHEET

DWG. NO.	SH	REV.



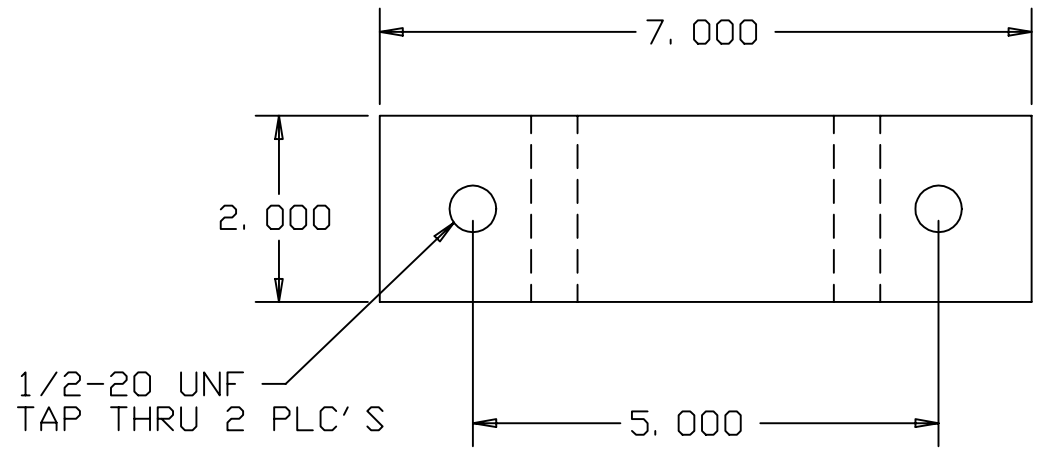


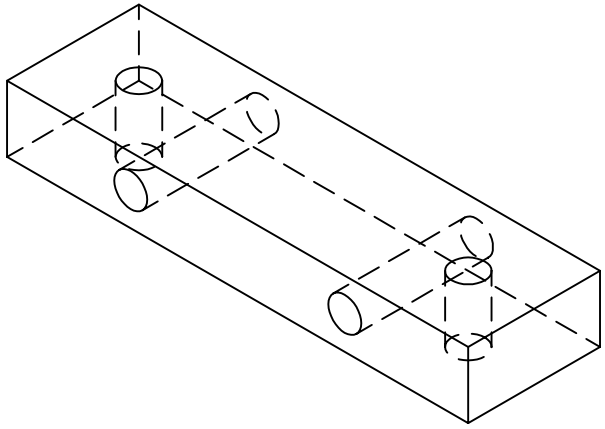


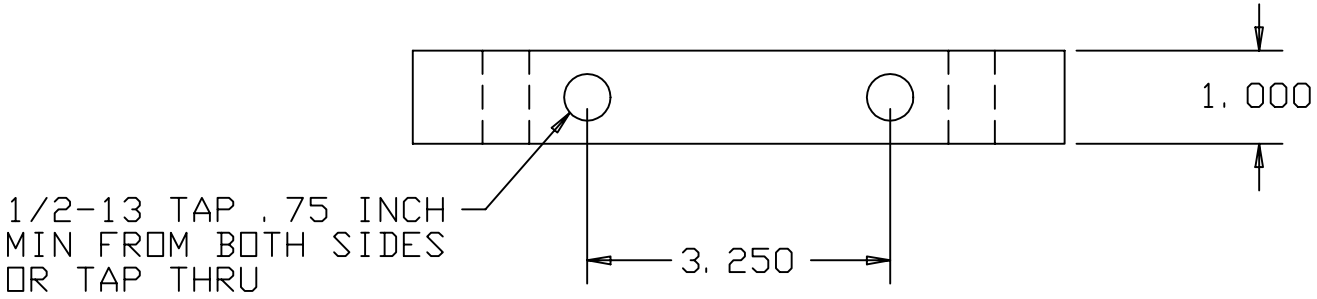


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE ALUMINUM	PROJECT RETRO GUN
	QUANTITY 1	TITLE BIG END CAP
TREATMENT NONE	NOTES	SIZE A / REVISION NO. 1 / PART NO.
FINISH NONE		
DATE FEBRUARY 14, 2001	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

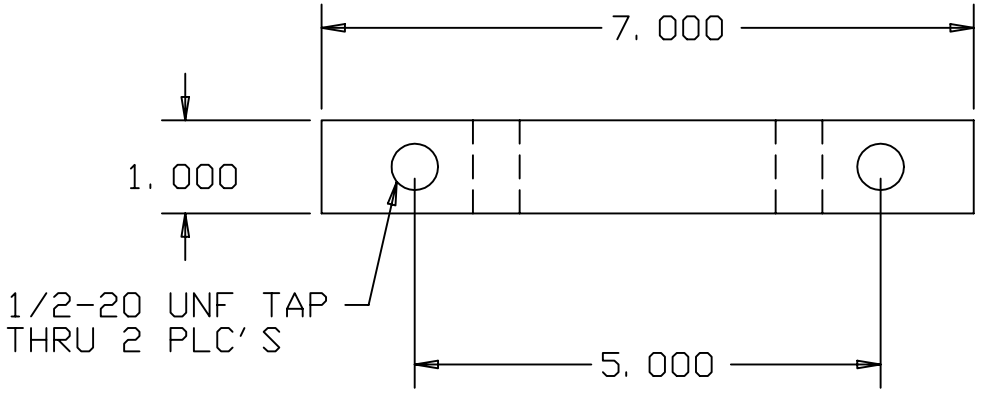


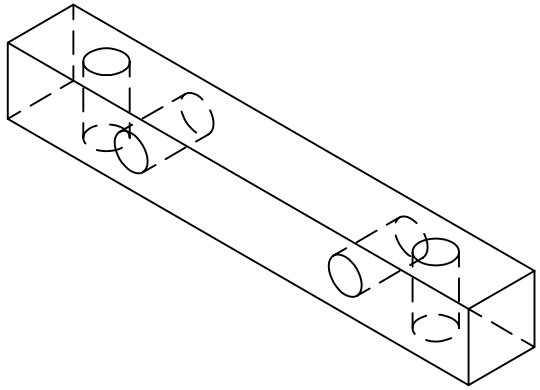


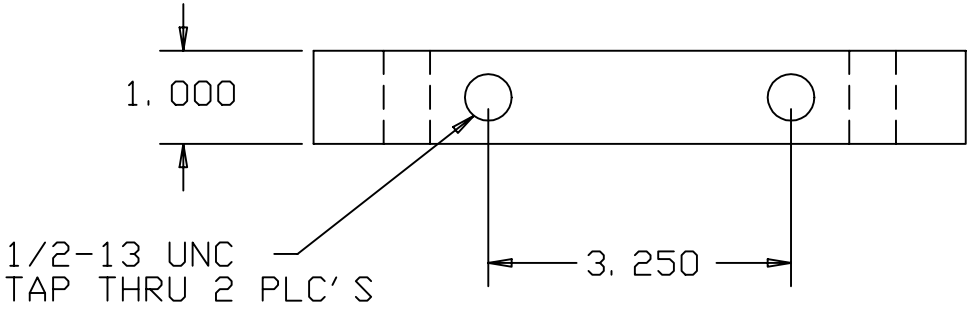


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1, . XX=±0. 02, . XXX=±0. 005 ANGLES X=±0. 5, X. X=±0. 05	MATERIAL TYPE: ALUMINUM	PROJECT: RETRO GUN
	QUANTITY: 1	TITLE: FRONT BOTTOM MOUNT
TREATMENT: NONE	NOTES	SIZE: A / REVISION NO.: 1 / PART NO.
FINISH: NONE		
DATE: FEBRUARY 14, 2001	DO NOT SCALE DRAWING	SCALE / SHEET

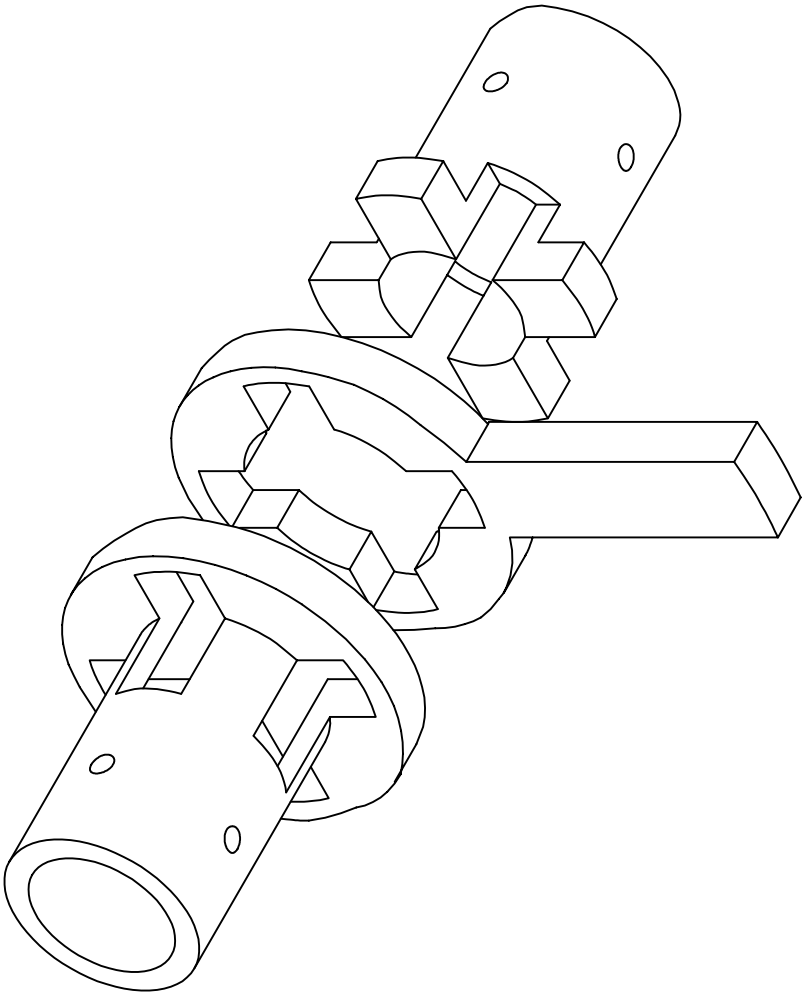
DWG. NO.	SH	REV.



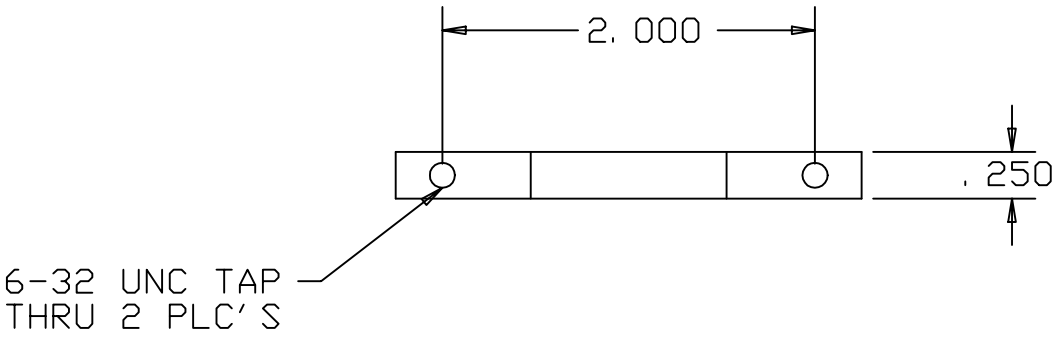


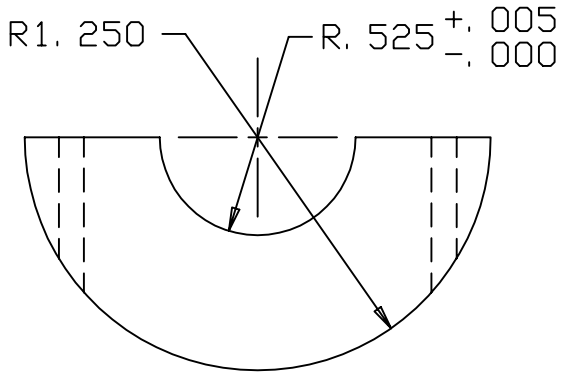


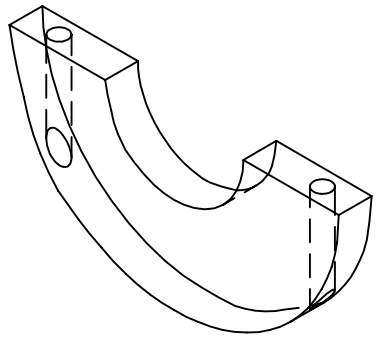
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE ALUMINUM	PROJECT RETRO GUN
	QUANTITY 1	TITLE REAR BOTTOM MOUNT
TREATMENT NONE	NOTES	SIZE A / REVISION NO. 1 / PART NO.
FINISH NONE		
DATE FEBRUARY 14, 2001	DO NOT SCALE DRAWING	SCALE / SHEET



DWG. NO.	SH	REV.

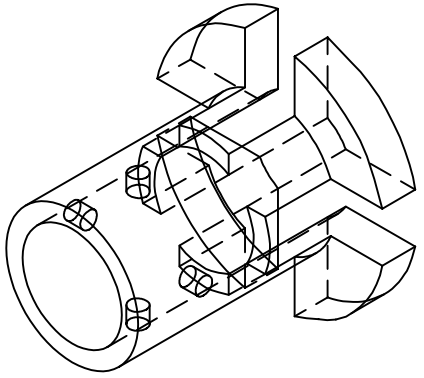


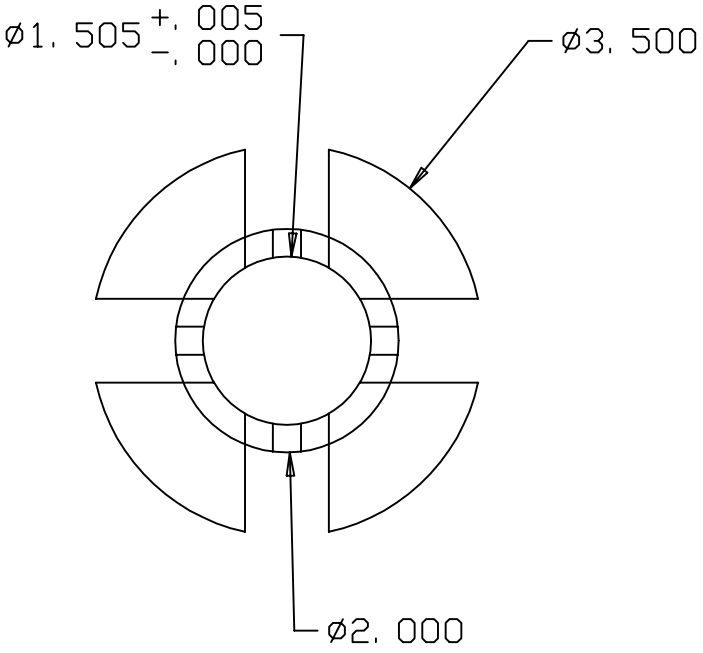


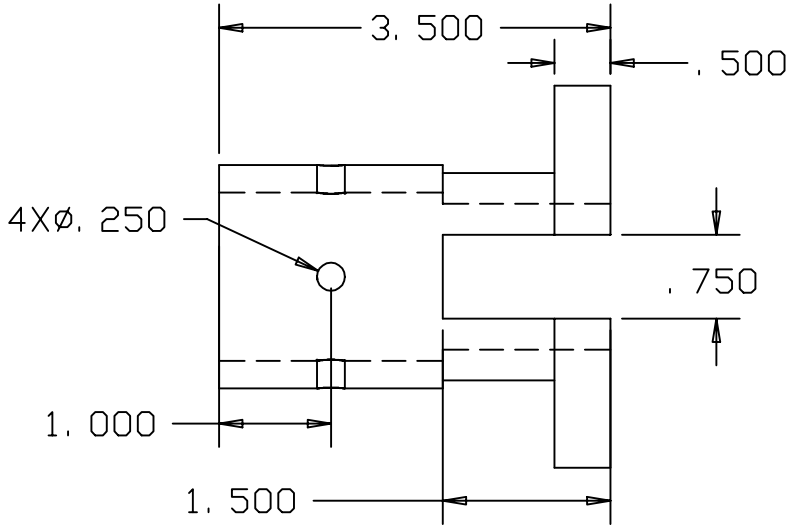


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE 6061-T6	PROJECT HOPKINSON BAR
TREATMENT NONE	QUANTITY 14	TITLE AIR BEARING END CAP
FINISH NONE	NOTES	SIZE A / REVISION NO. 1 / PART NO.
DATE SEPT. 8, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

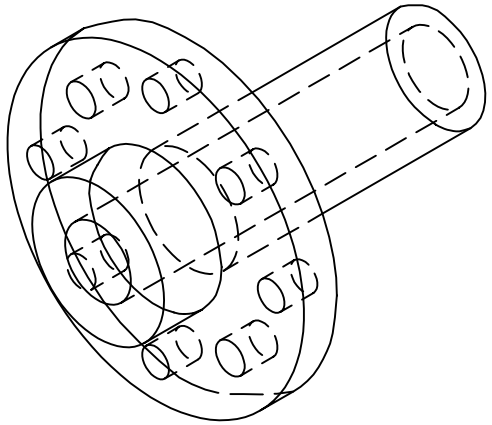


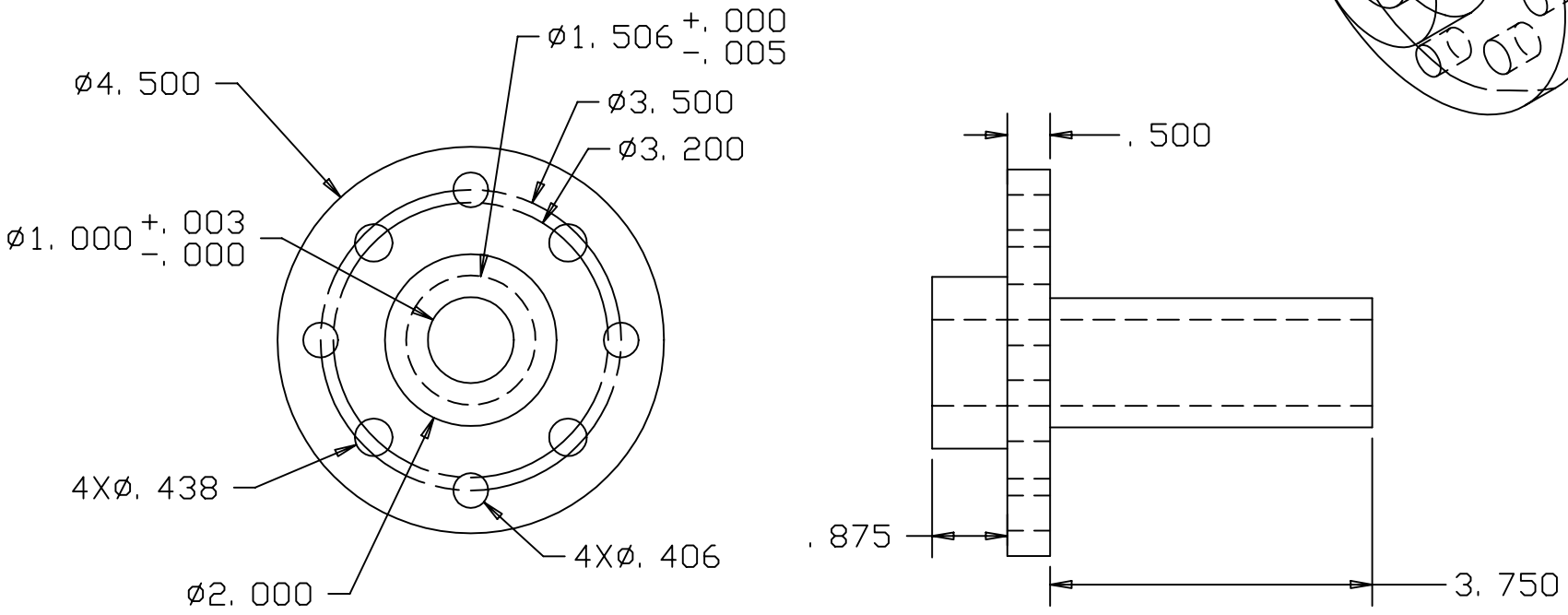




UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE STEEL	PROJECT HOPKINSON RETRO
	QUANTITY 1	TITLE FEMALE BREACH
TREATMENT NONE	NOTES	SIZE A / REVISION NO. 1 / PART NO.
FINISH NONE		
DATE APRIL 12, 2001	DO NOT SCALE DRAWING	SCALE / SHEET

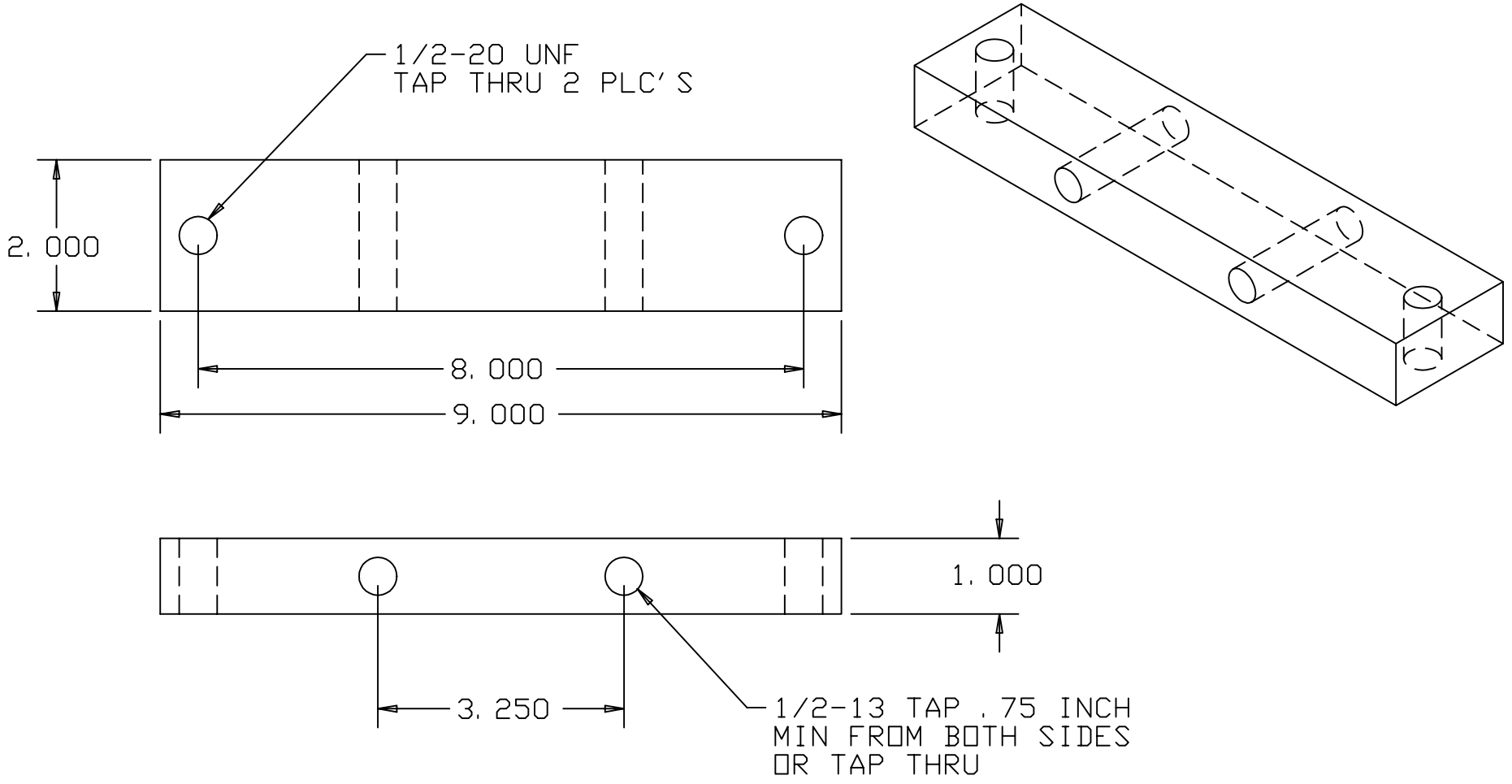
DWG. NO.	SH	REV.





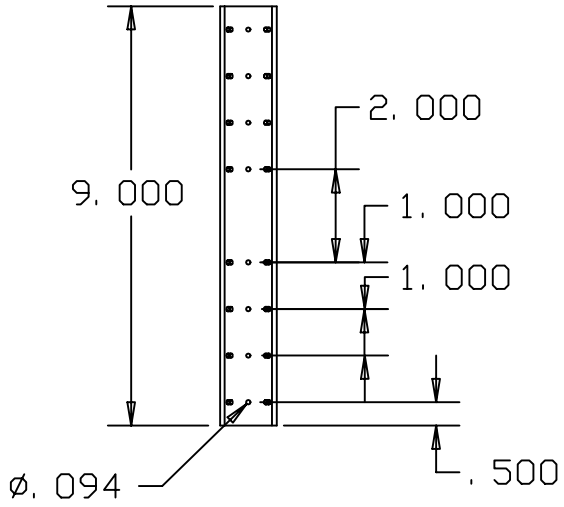
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE: BRASS	PROJECT: RETRO GUN
	QUANTITY: 1	TITLE: FLANGE
TREATMENT: NONE	NOTES	SIZE: A / REVISION NO.: 1 / PART NO.
FINISH: NONE		
DATE: FEBRUARY 14, 2001	DO NOT SCALE DRAWING	SCALE / SHEET

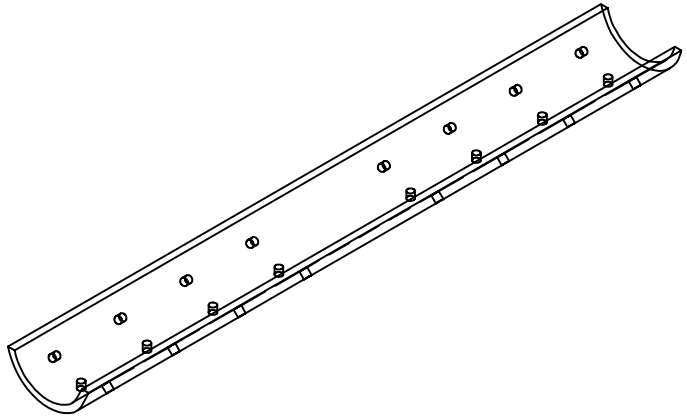
DWG. NO.	SH	REV.

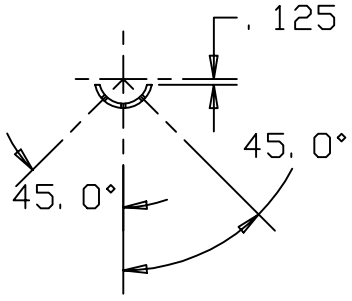


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE: ALUMINUM	PROJECT: RETRO GUN
	QUANTITY: 1	TITLE: FRONT BOTTOM MOUNT
TREATMENT: NONE	NOTES	SIZE: A / REVISION NO.: 2 / PART NO.
FINISH: NONE		
DATE: FEBRUARY 14, 2001	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.





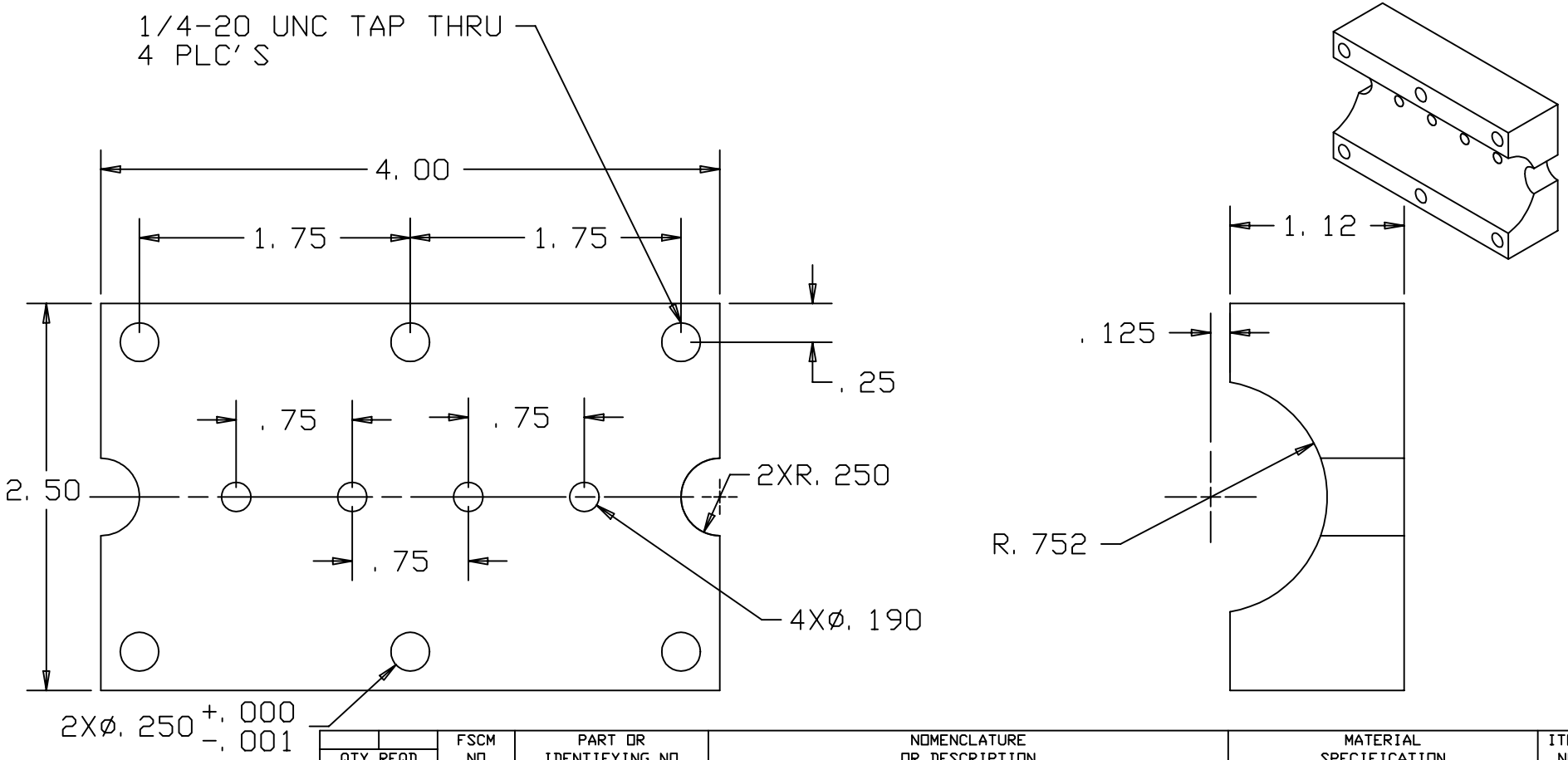


USE ALUMINUM
PIPE WITH ø1. 055 ID

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE ALUMINUM	PROJECT HOPKINSON BAR
	QUANTITY 7	TITLE IN MANIFOLD
TREATMENT NONE	NOTES	SIZE A / REVISION NO. 1 / PART NO.
FINISH NONE		
DATE AUGUST 31, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED



QTY REQD	FSCM NO	PART OR IDENTIFYING NO	NOMENCLATURE OR DESCRIPTION	MATERIAL SPECIFICATION	ITEM NO

PARTS LIST

UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN INCHES
TOLERANCES ARE:

DECIMALS
.XX= ±0.02
.XXX=±0.002

ANGLES
±.5°

DO NOT SCALE DRAWING

MATERIAL TYPE
ALUMINUM 6061

TREATMENT
ANODIZE BLACK

FINISH

SIMILAR TO | ACT. WT | CALC WT

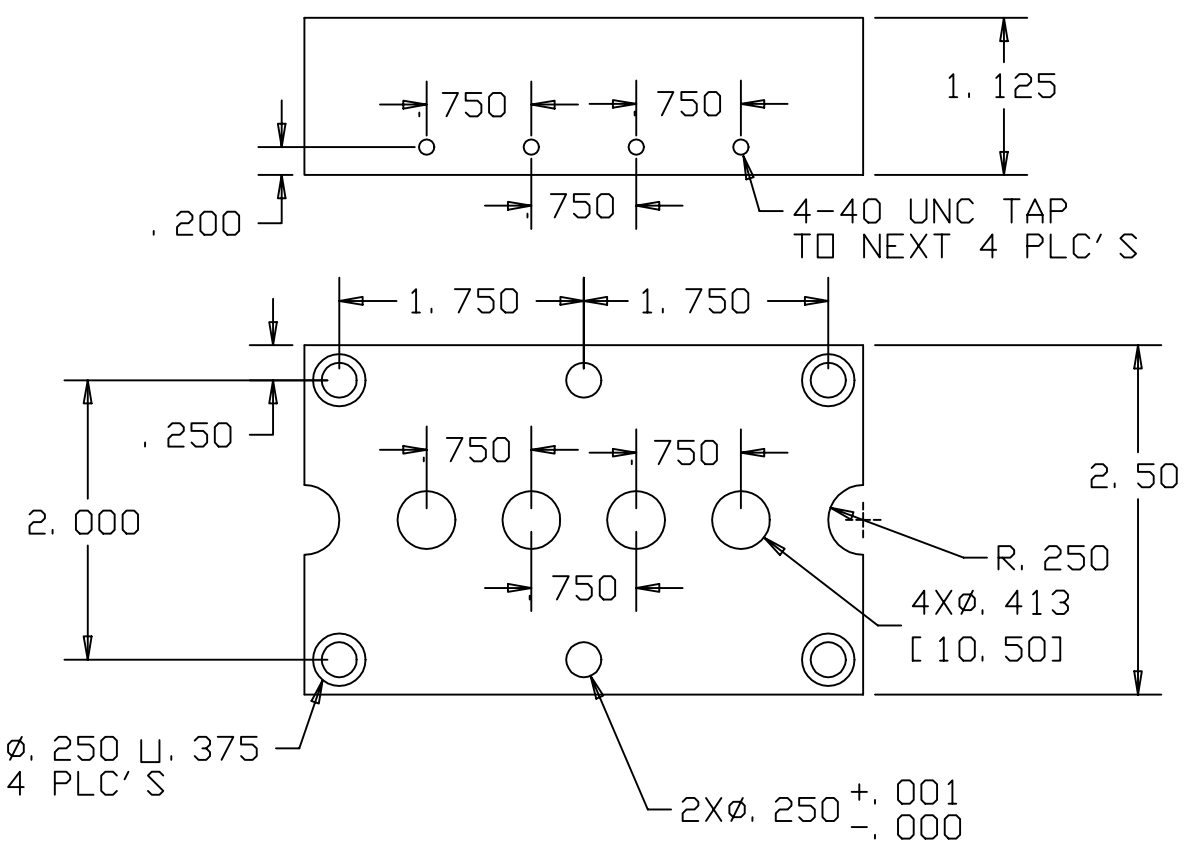
APPROVALS	DATE
DRAWN	
CHECKED	
ISSUED	

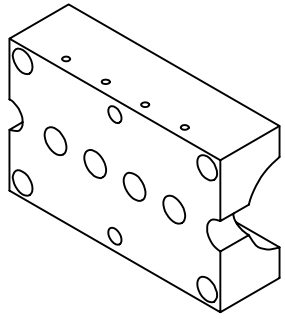
TITLE
FEMALE LASER HOUSING

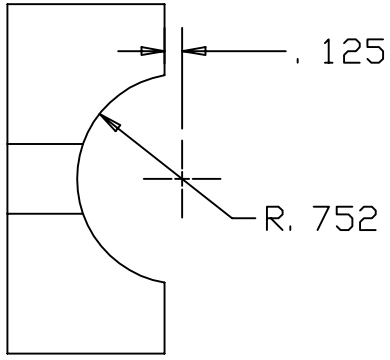
SIZE	FSCM NO.	DWG NO.
A		FEMALE HOUSING-1

SCALE		SHEET

DWG. NO.	SH	REV.

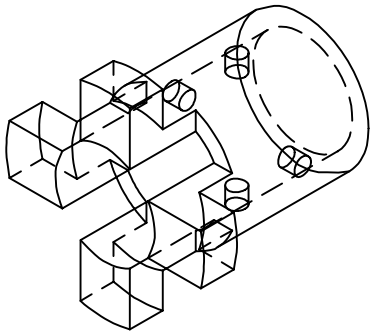


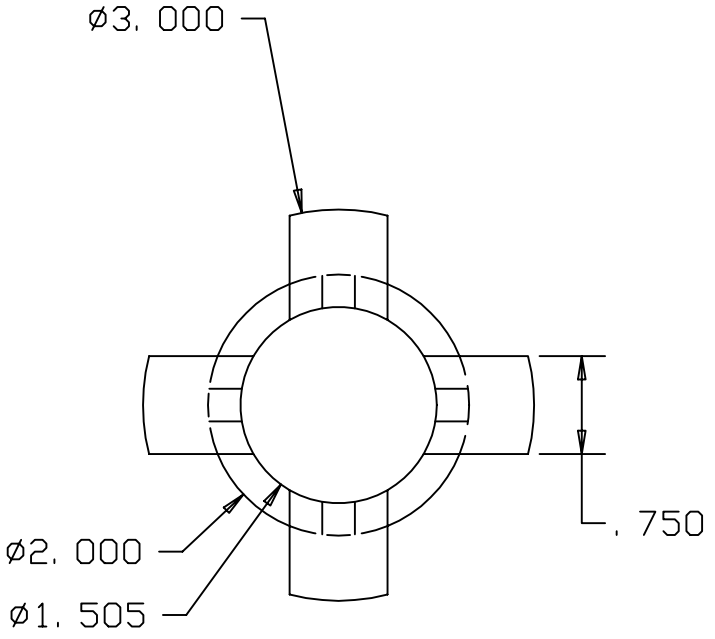


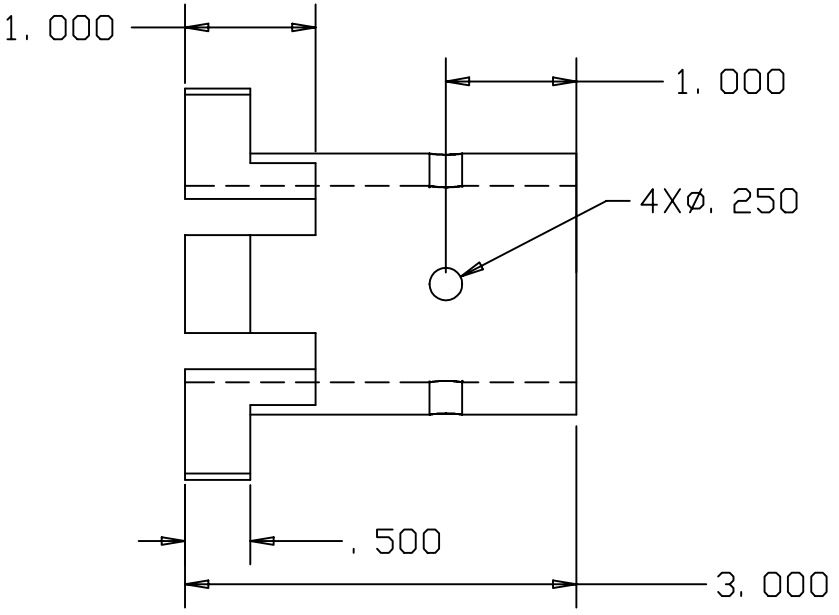


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 DO NOT SCALE DRAWING	MATERIAL TYPE ALUMINUM 6061	PROJECT HOPKINSON BAR
	QUANTITY 1	TITLE MALE LASER HOUSING
TREATMENT	NOTES	SIZE A / VERSION NO. 1 / PART NO.
FINISH BLACK ANODIZE		SCALE / SHEET
DATE AUGUST 9, 2000		

DWG. NO.	SH	REV.

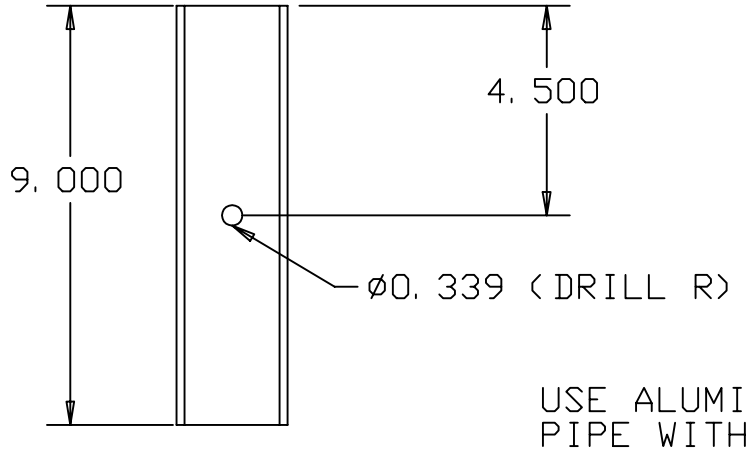


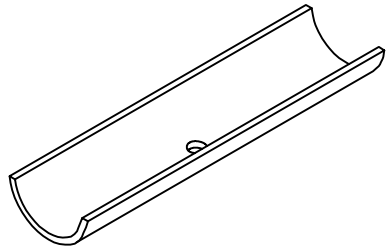




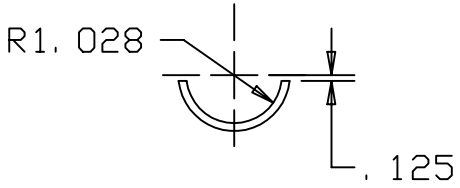
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE STEEL	PROJECT HOPKINSON RETRO
	QUANTITY 1	TITLE MALE BREACH
TREATMENT NONE	NOTES	SIZE A / REVISION NO. 1 / PART NO.
FINISH NONE		
DATE APRIL 12, 2001	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.



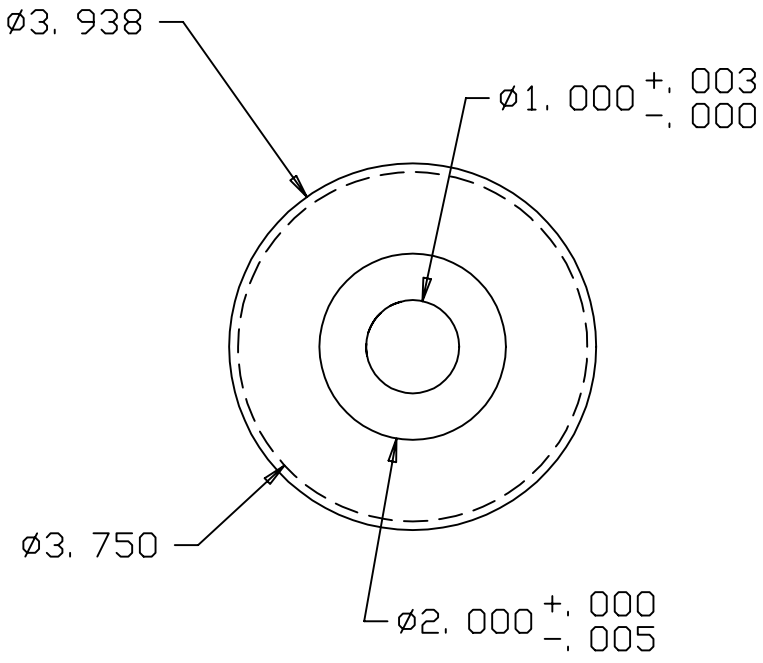


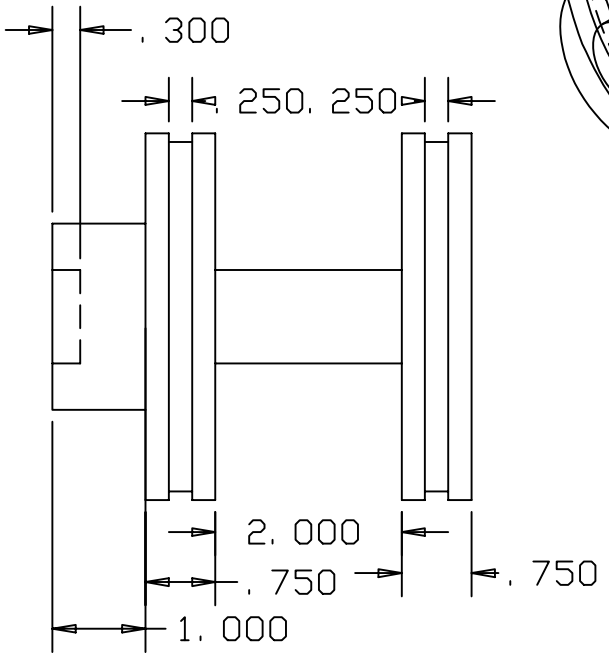
USE ALUMINUM
PIPE WITH
Ø2.400 OD

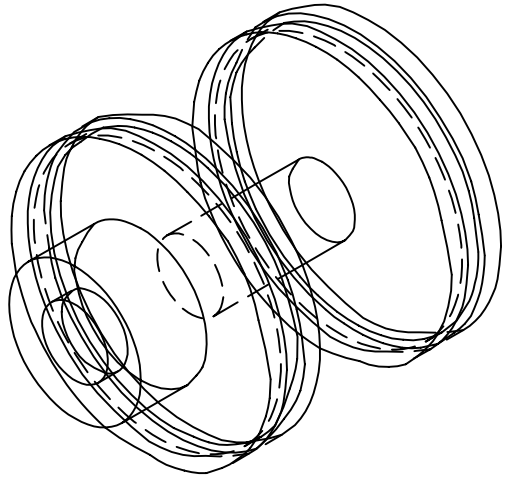


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE: ALUMINUM	PROJECT: HOPKINSON BAR
	QUANTITY: 7	TITLE: OUT MANIFOLD
TREATMENT: NONE	NOTES	SIZE: A / REVISION NO.: 1 / PART NO.
FINISH: NONE		
DATE: AUGUST 31, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.







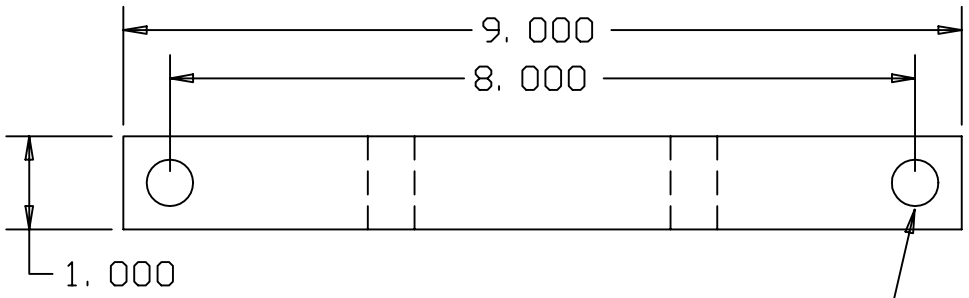
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE:

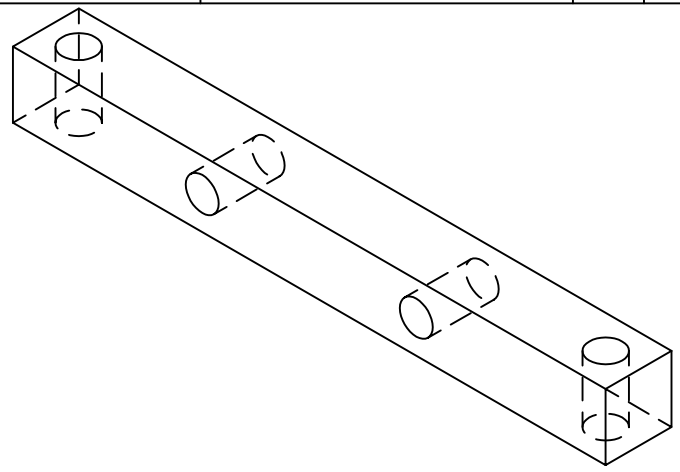
DECIMALS
. X=±. 1
. XX=±0. 02
. XXX=±0. 005

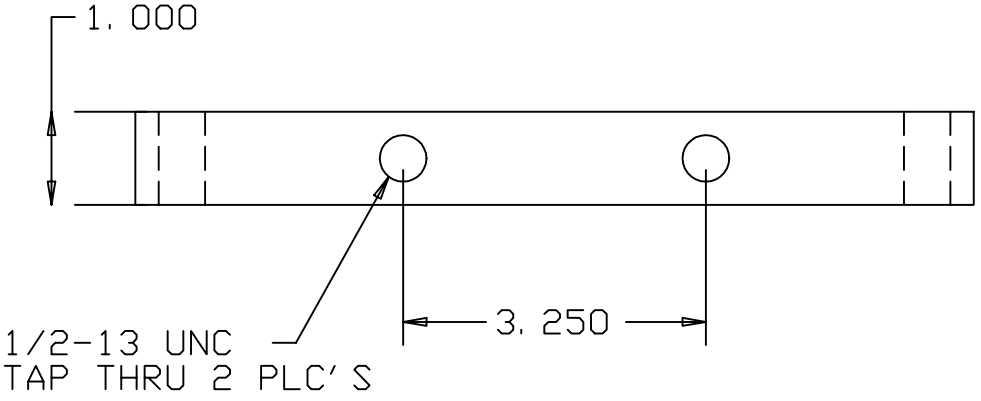
ANGLES
X=±0. 5
X. X=±0. 05

TREATMENT: NONE	MATERIAL TYPE: ALUMINUM	PROJECT: RETRO GUN
FINISH: NONE	QUANTITY: 1	TITLE: PISTON
DATE: FEBRUARY 14, 2001	NOTES: DO NOT SCALE DRAWING	SIZE: A / REVISION NO.: 1 / PART NO.
		SCALE / SHEET

DWG. NO.	SH	REV.

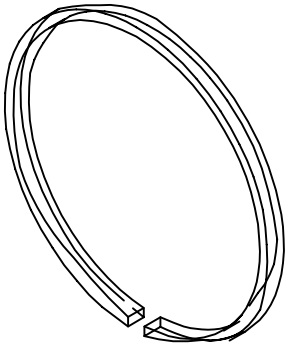


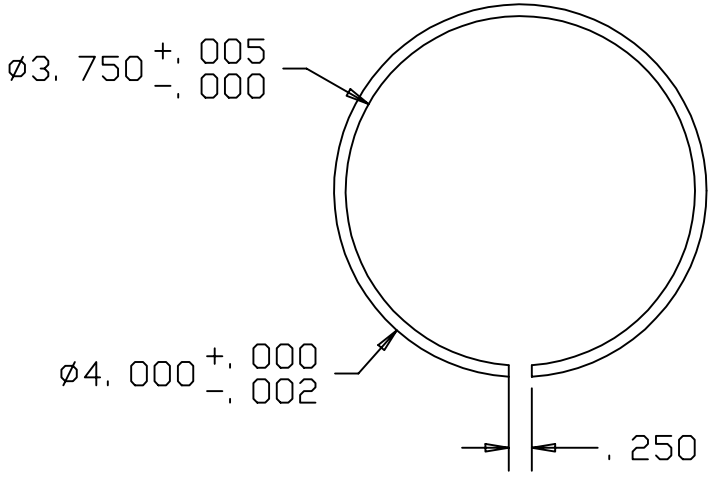


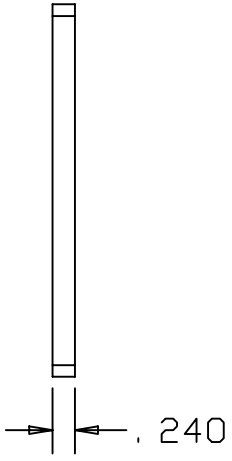


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE ALUMINUM	PROJECT RETRO GUN
	QUANTITY 1	TITLE REAR BOTTOM MOUNT
TREATMENT NONE	NOTES	SIZE A / REVISION NO. 2 / PART NO.
FINISH NONE		
DATE FEBRUARY 14, 2001	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

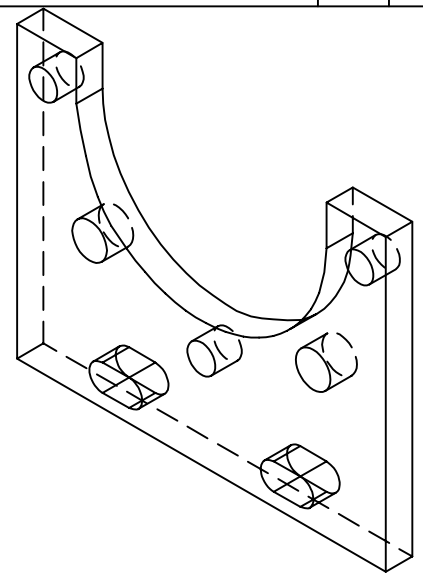


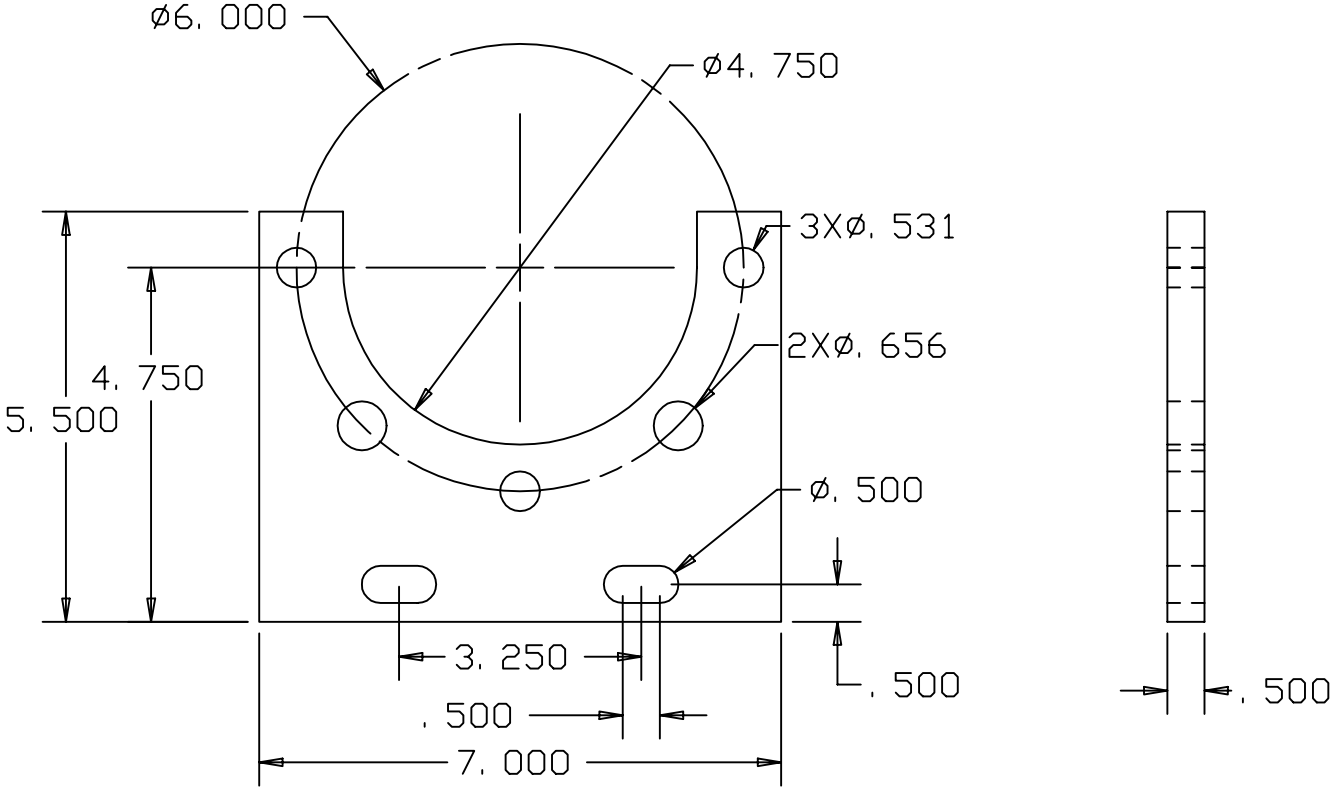




UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE: TEFLON	PROJECT: RETRO GUN
	QUANTITY: 2	TITLE: PISTON RING
TREATMENT: NONE	NOTES:	SIZE: A / REVISION NO.: 1 / PART NO.:
FINISH: NONE		
DATE: FEBRUARY 14, 2001	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.



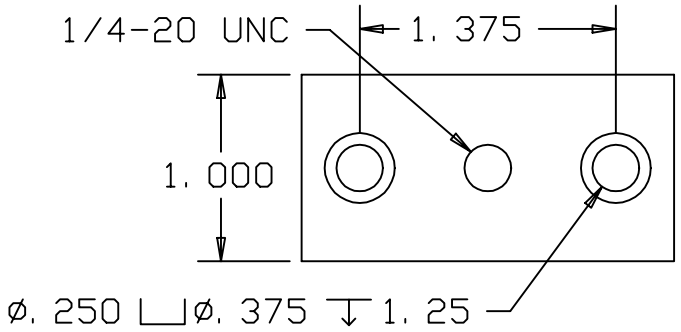


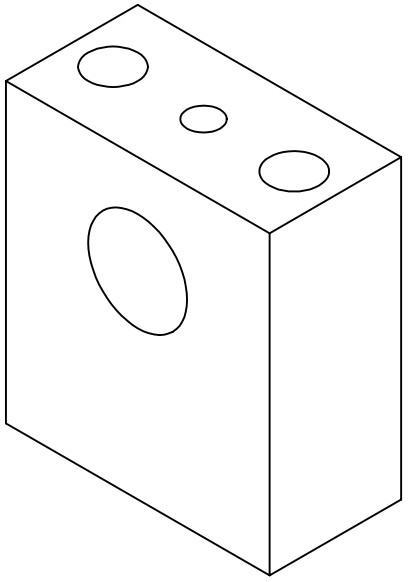
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE: ALUMINUM	PROJECT: RETRO GUN
	QUANTITY: 4	TITLE: SUPPORT SIDE
TREATMENT: NONE	NOTES	SIZE: A / REVISION NO.: 1 / PART NO.
FINISH: NONE		
DATE: FEBRUARY 14, 2001	DO NOT SCALE DRAWING	SCALE / SHEET

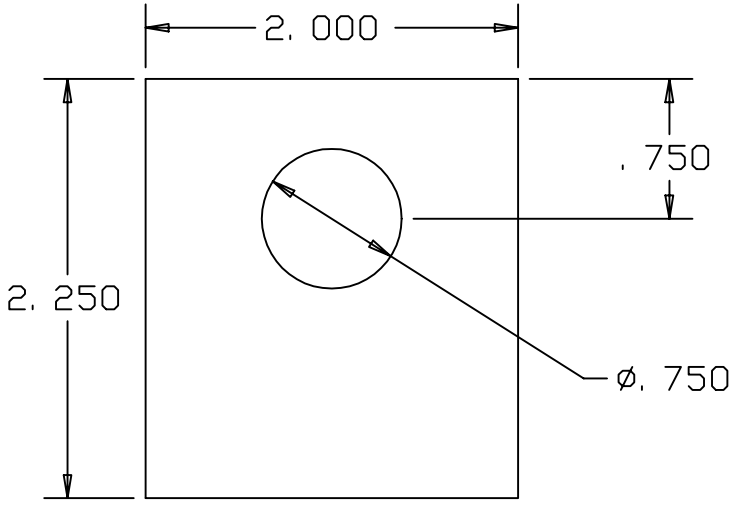
DWG. NO.	SH	REV.

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED







QTY REQD	FSCM NO	PART OR IDENTIFYING NO	NOMENCLATURE OR DESCRIPTION	MATERIAL SPECIFICATION	ITEM NO

PARTS LIST

UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN INCHES
TOLERANCES ARE:

FRACTIONS ± DECIMALS .XX= ± .XXX= ANGLES ±

DO NOT SCALE DRAWING

CONTRACT NO.

APPROVALS	DATE
DRAWN	
CHECKED	
ISSUED	

TREATMENT

FINISH

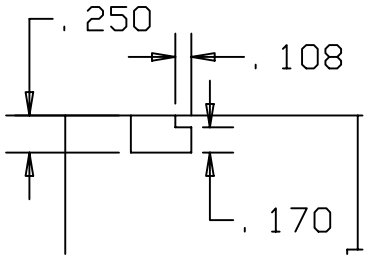
SIMILAR TO	ACT. WT	CALC WT

TITLE

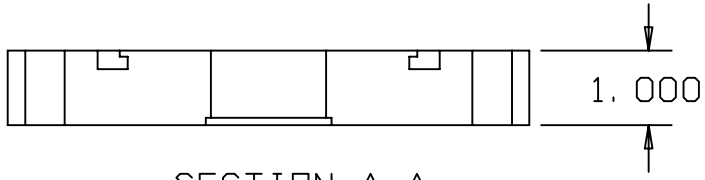
SIZE	FSCM NO.	DWG NO.
A		

SCALE		SHEET

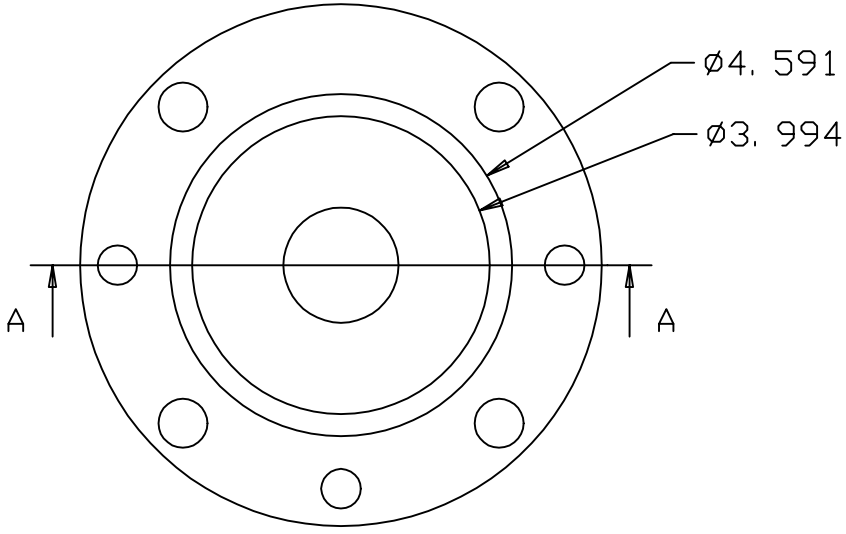
DWG. NO.	SH	REV.



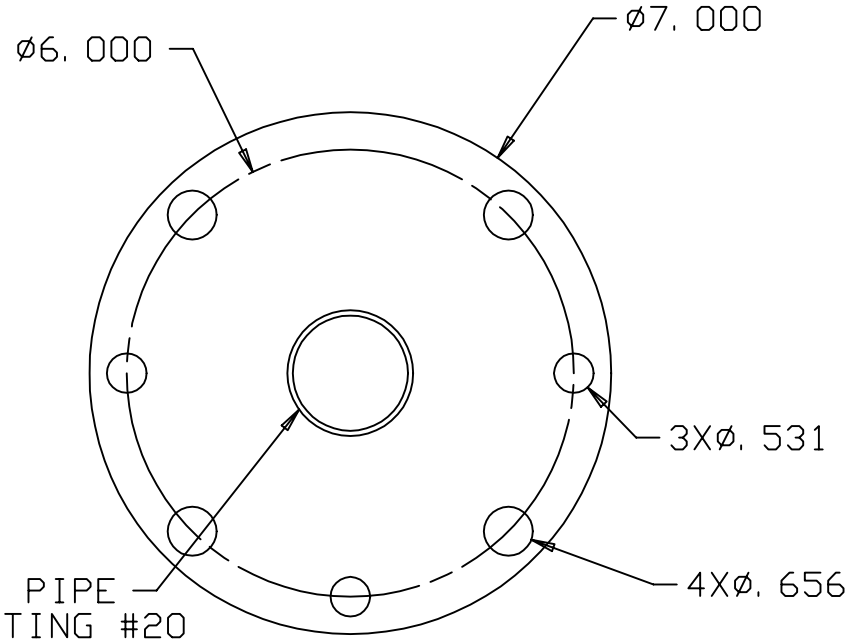
DETAIL A-A
DETAIL A



SECTION A-A



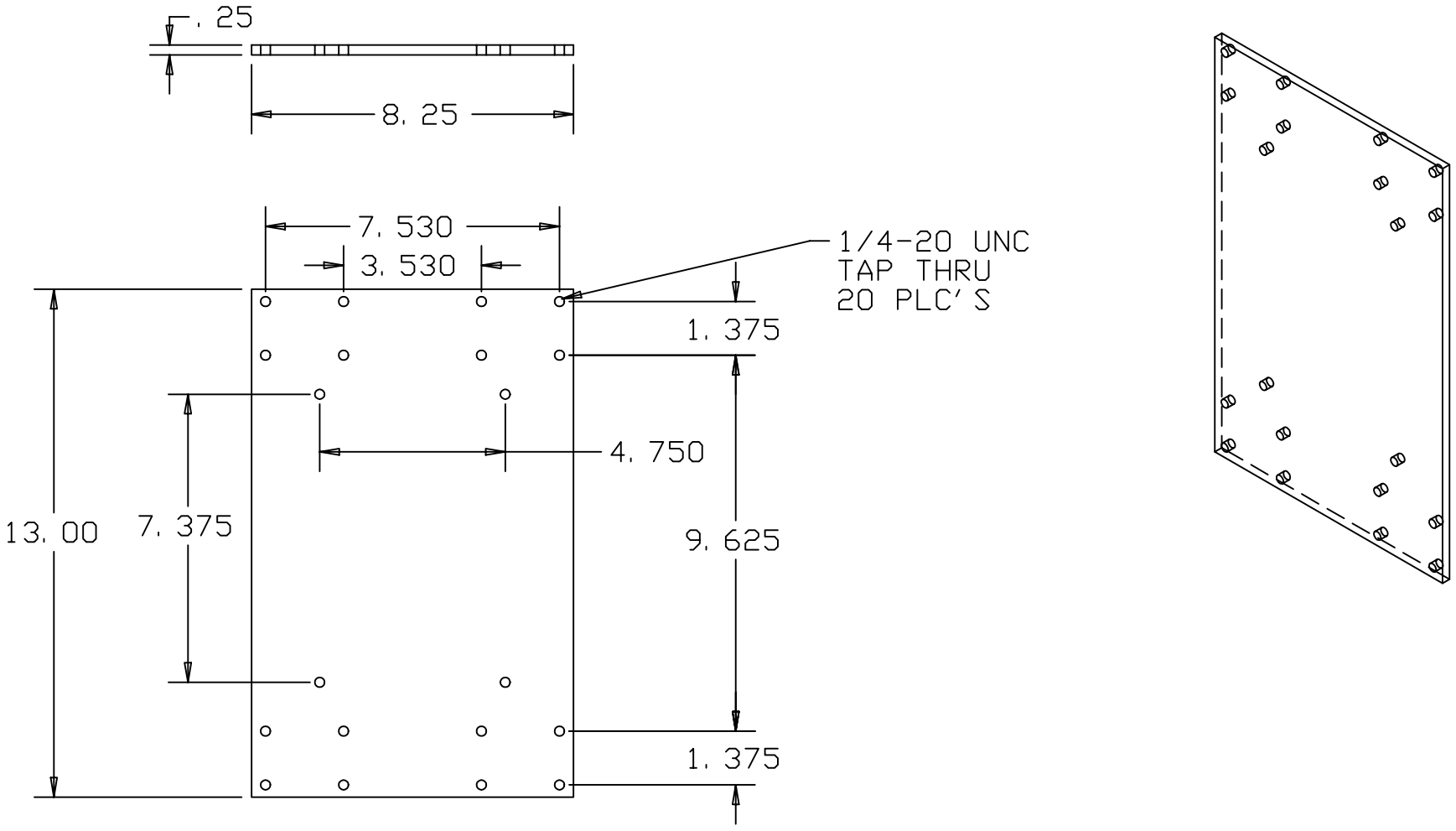
FRONT



BACK

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE ALUMINUM	PROJECT RETRO GUN
	QUANTITY 1	TITLE REAR END CAP
TREATMENT NONE	NOTES	SIZE A / REVISION NO. 1 / PART NO.
FINISH NONE		
DATE FEBRUARY 14, 2001	DO NOT SCALE DRAWING	SCALE / SHEET

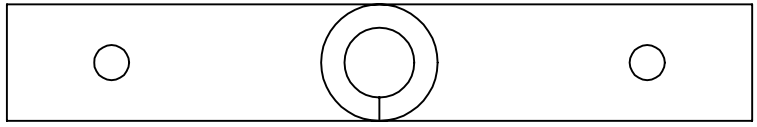
DWG. NO.	SH	REV.

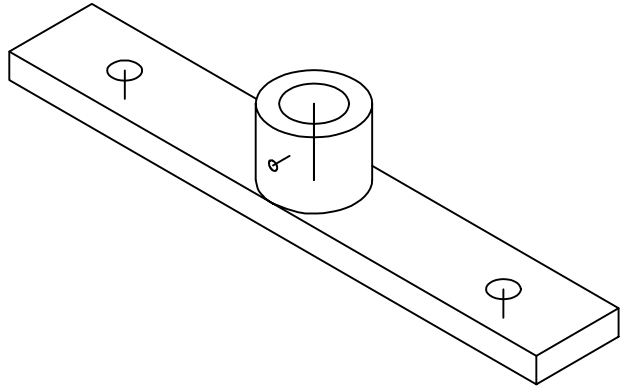


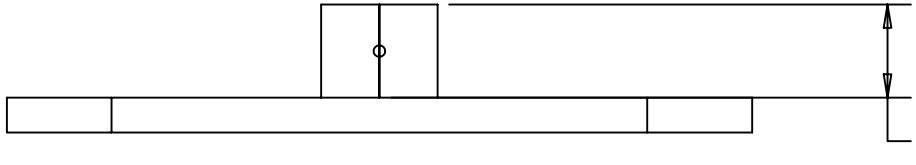
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE: ALUMINUM 2024	PROJECT: HOPKINSON BAR
	QUANTITY: 1	TITLE: GUN SUPPORT PLATE
TREATMENT: NONE	NOTES	SIZE: A / REVISION NO.: 1 / PART NO.
FINISH: NONE		
DATE: SEPTEMBER 14, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED







QTY REQD	FSCM NO	PART OR IDENTIFYING NO	NOMENCLATURE OR DESCRIPTION	MATERIAL SPECIFICATION	ITEM NO

PARTS LIST

UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN INCHES
TOLERANCES ARE:

FRACTIONS ± DECIMALS .XX= ±.XXX= ANGLES ±

DO NOT SCALE DRAWING

CONTRACT NO.

TREATMENT

FINISH

SIMILAR TO | ACT. WT | CALC WT

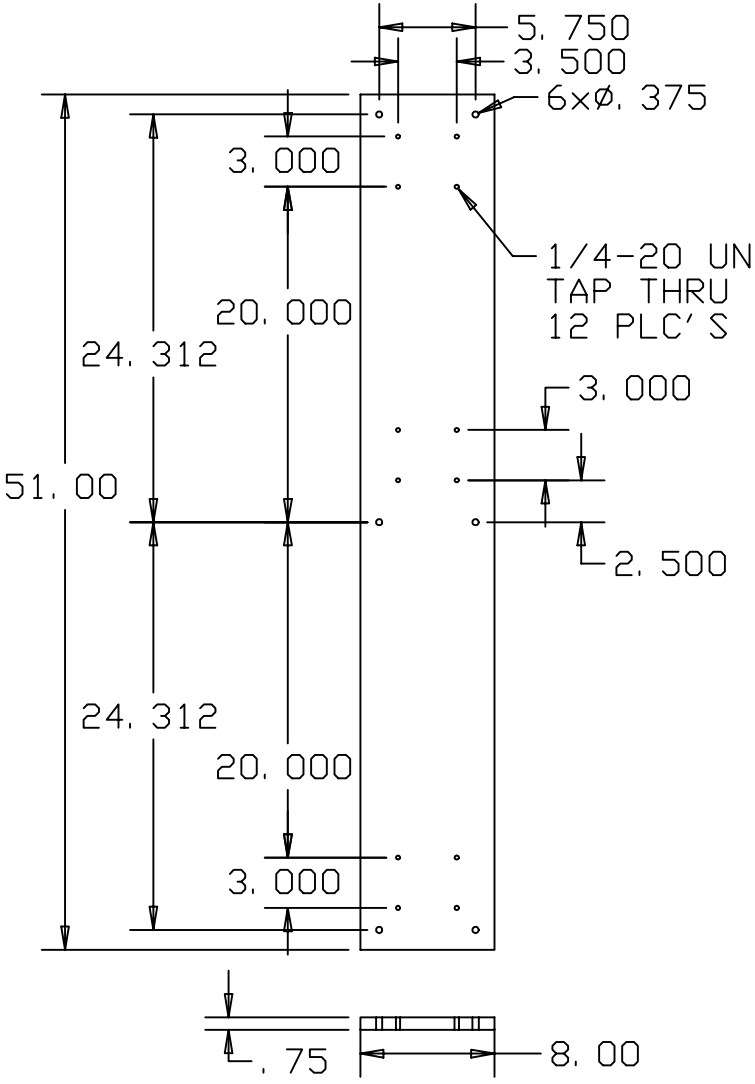
APPROVALS	DATE
DRAWN	
CHECKED	
ISSUED	

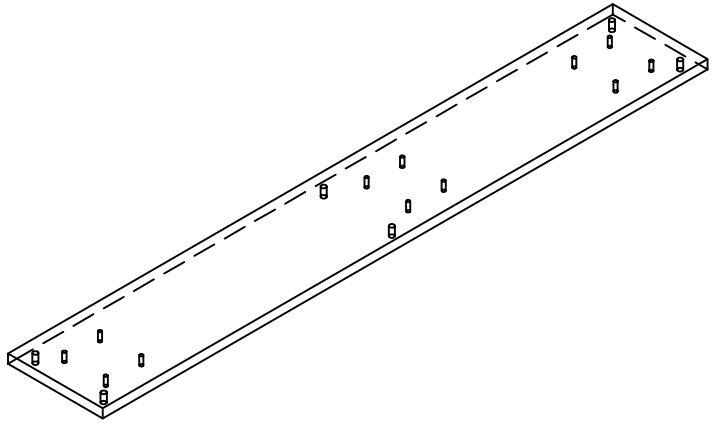
TITLE

SIZE A | FSCM NO. | DWG NO.

SCALE | SHEET

DWG. NO.	SH	REV.





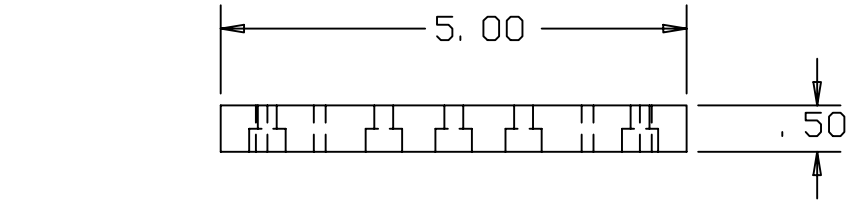
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES
TOLERANCES ARE:

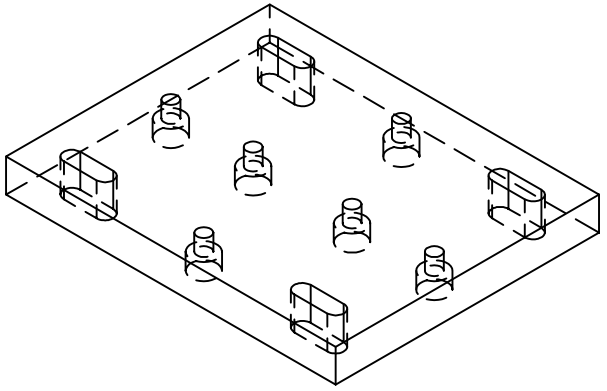
DECIMALS
. X=±. 1
. XX=±0. 02
. XXX=±0. 005

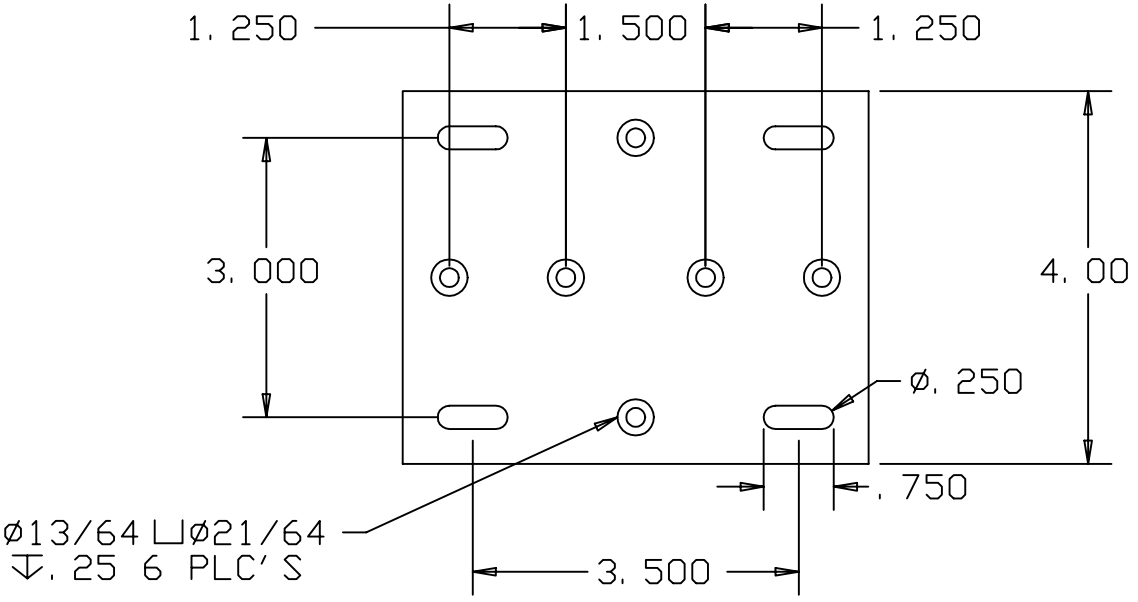
ANGLES
X=±0. 5
X. X=±0. 05

TREATMENT	NONE
FINISH	PAINTED BLACK
DATE	SEPTEMBER 14, 2000
MATERIAL TYPE	MILD STEEL
QUANTITY	1
NOTES	DO NOT SCALE DRAWING
PROJECT	HOPKINSON BAR
TITLE	STRIKER HOUSING SUPPORT PLATE
SIZE	A
REVISION NO.	1
PART NO.	
SCALE	
SHEET	

DWG. NO.	SH	REV.

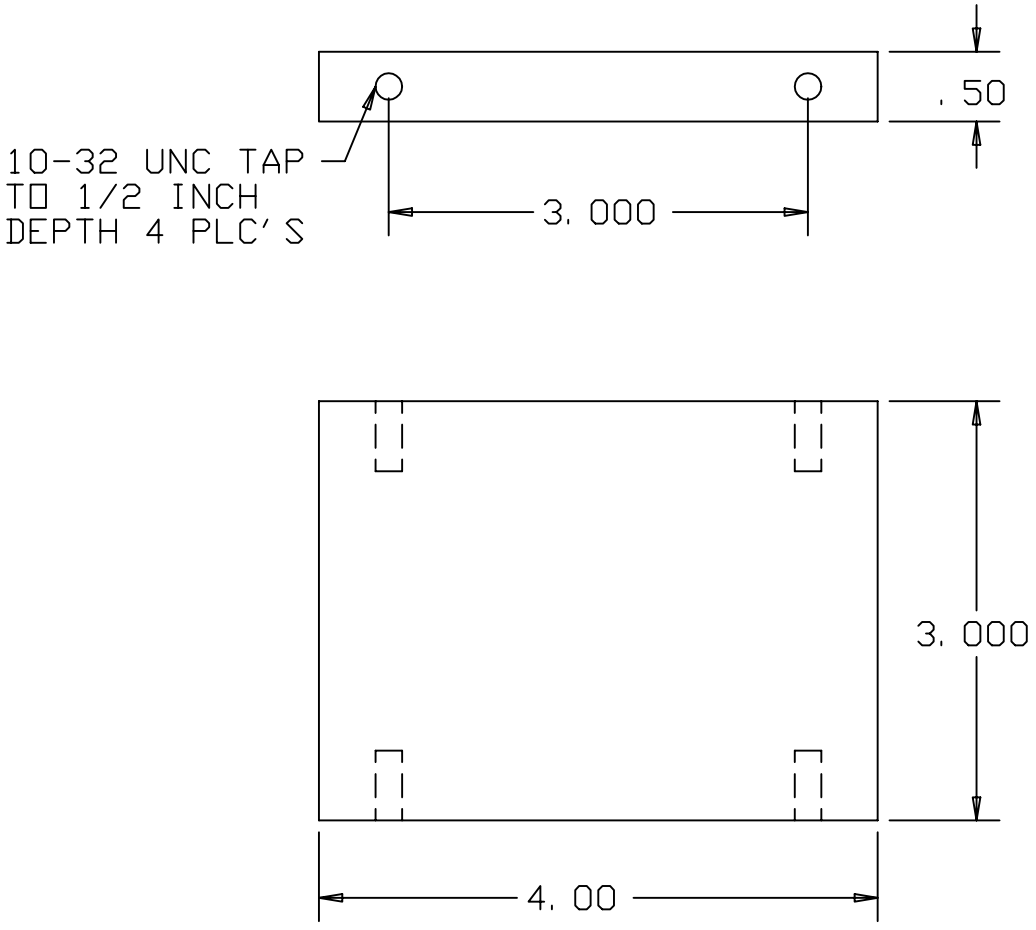


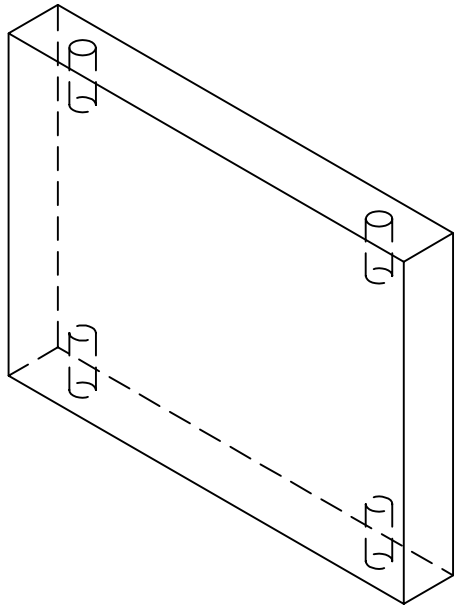




UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1, . XX=±0. 02, . XXX=±0. 005 ANGLES X=±0. 5, X. X=±0. 05	MATERIAL TYPE: STEEL	PROJECT: HOPKINSON BAR
	QUANTITY: 3	TITLE: STRIKER HOUSING MOUNT LOWER
TREATMENT: NONE	NOTES	SIZE: A / REVISION NO.: 1 / PART NO.
FINISH: PAINTED BLACK		
DATE: SEPTEMBER 15, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

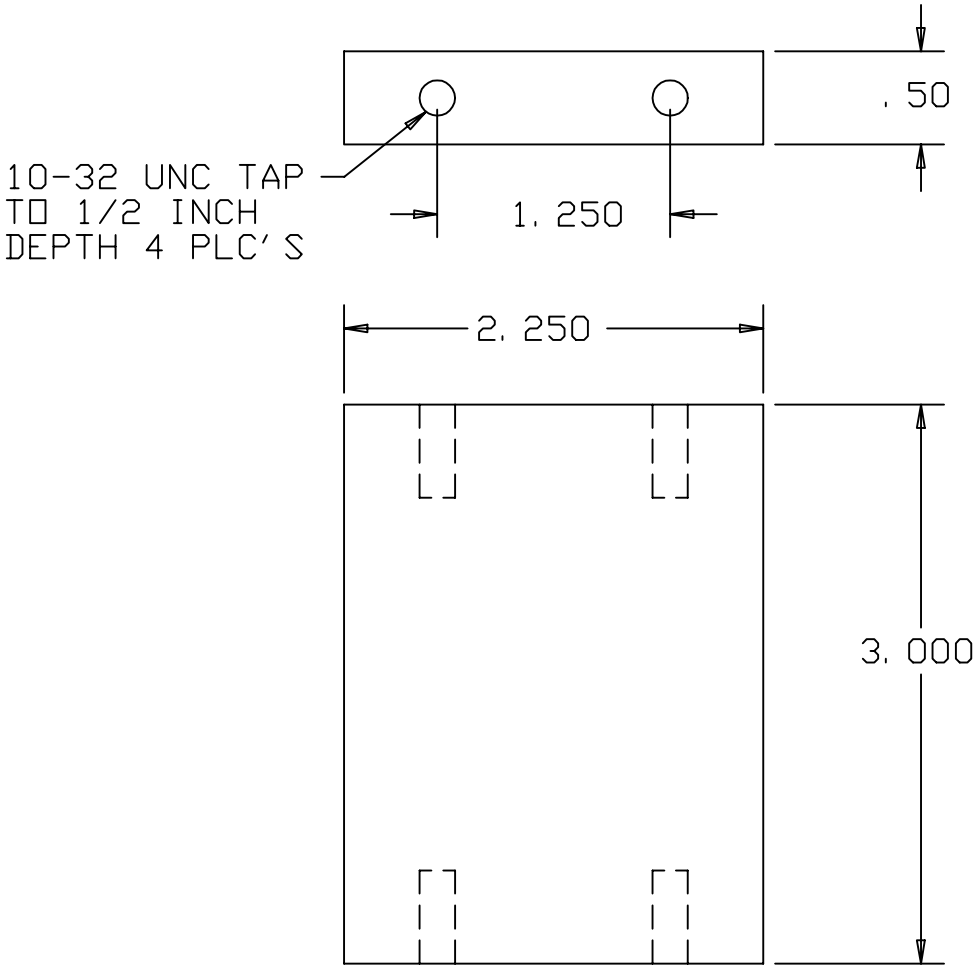
DWG. NO.	SH	REV.

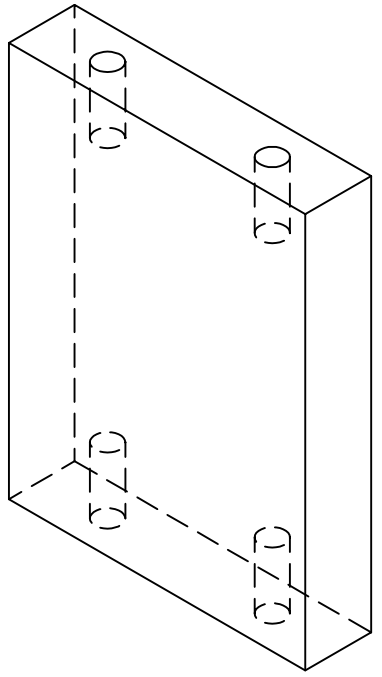




UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE: STEEL	PROJECT: HOPKINSON BAR
	QUANTITY: 3	TITLE: STRIKER HOUSING MOUNT SIDE 2
TREATMENT: NONE	NOTES	SIZE: A / REVISION NO.: 1 / PART NO.
FINISH: PAINTED BLACK		
DATE: SEPTEMBER 15, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

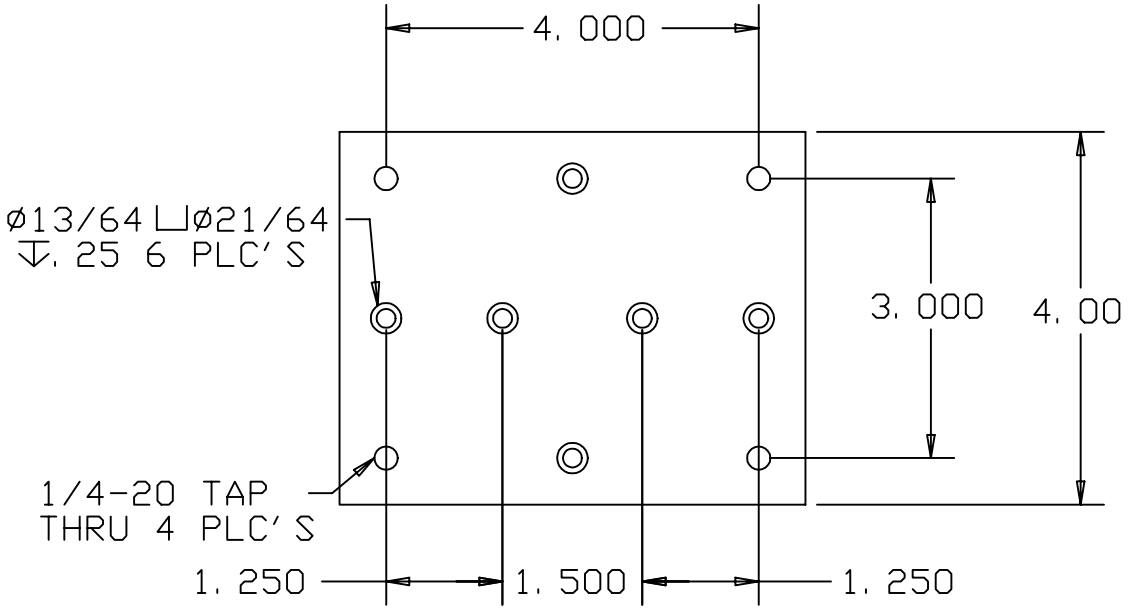
DWG. NO.	SH	REV.

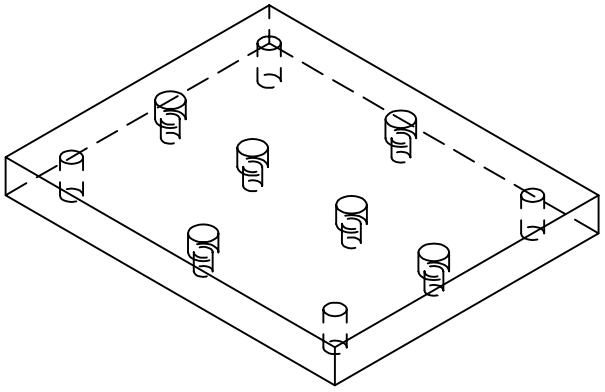


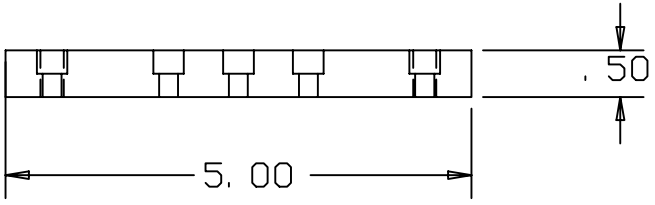


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE: STEEL	PROJECT: HOPKINSON BAR
	QUANTITY: 6	TITLE: STRIKER HOUSING MOUNT SIDE 1
TREATMENT: NONE	NOTES	SIZE: A / REVISION NO.: 1 / PART NO.
FINISH: PAINTED BLACK		
DATE: SEPTEMBER 15, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

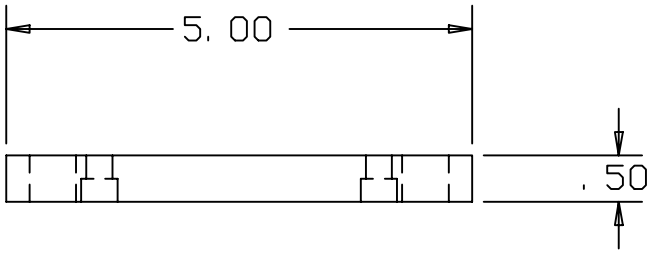


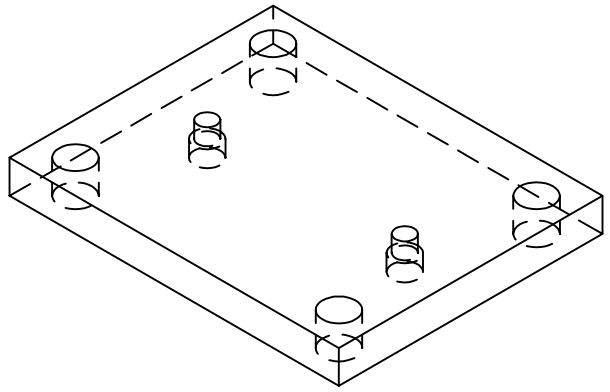


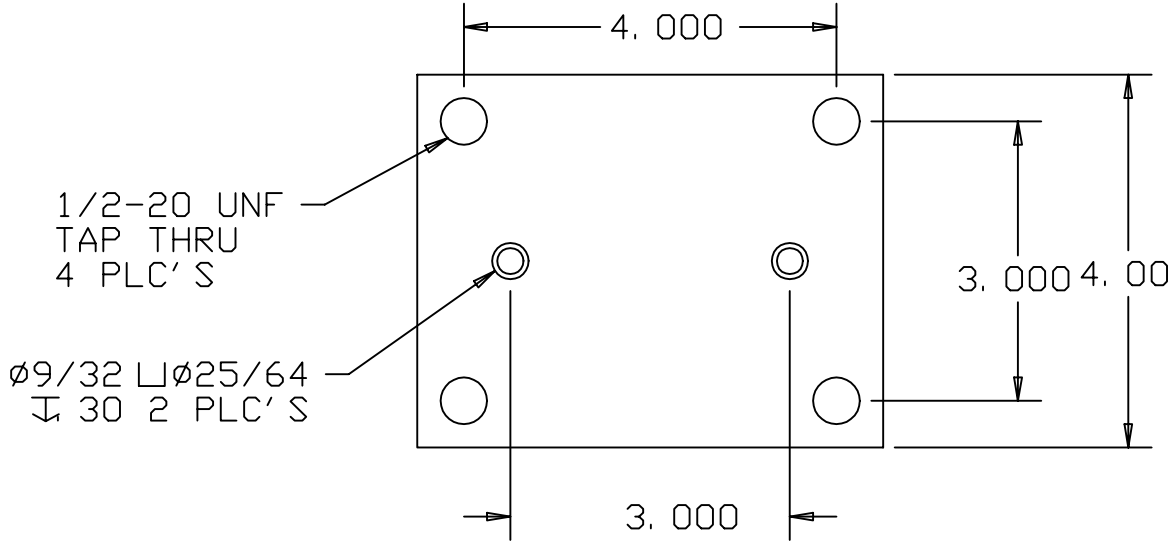


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE STEEL	PROJECT HOPKINSON BAR
	QUANTITY 3	TITLE STRIKER HOUSING MOUNT UPPER
TREATMENT NONE	NOTES	SIZE A / REVISION NO. 1 / PART NO.
FINISH PAINTED BLACK		
DATE SEPTEMBER 15, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

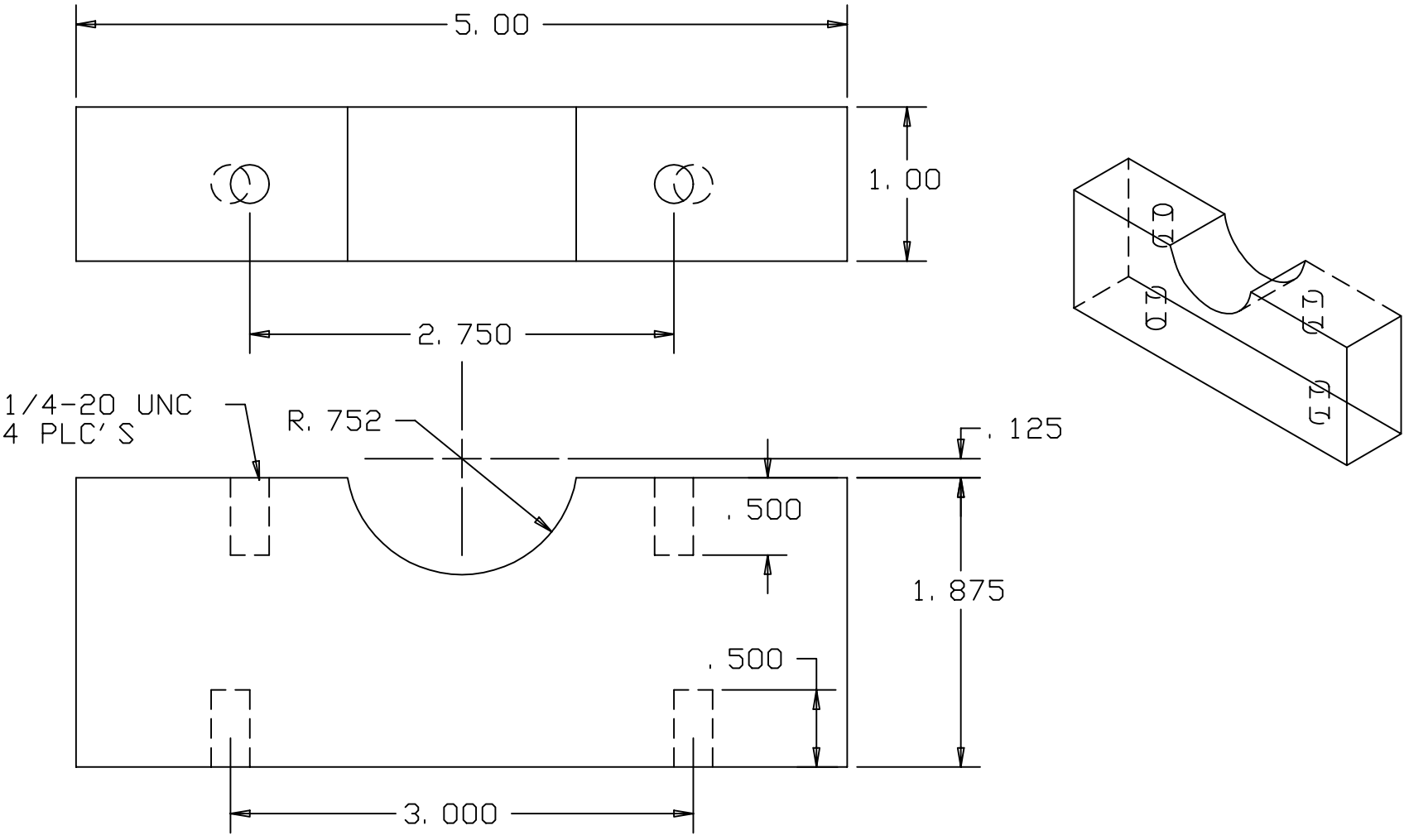






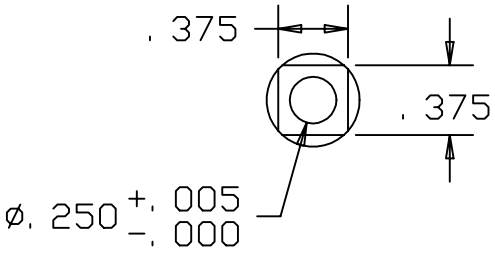
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1, . XX=±0. 02, . XXX=±0. 005 ANGLES X=±0. 5, X. X=±0. 05	MATERIAL TYPE: STEEL	PROJECT: HOPKINSON BAR
	QUANTITY: 3	TITLE: STRIKER HOUSING MOUNT UPPER PLATE
TREATMENT: NONE	NOTES	SIZE: A / REVISION NO.: 1 / PART NO.
FINISH: PAINTED BLACK		
DATE: SEPTEMBER 12, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

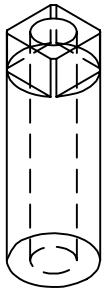
DWG. NO.	SH	REV.

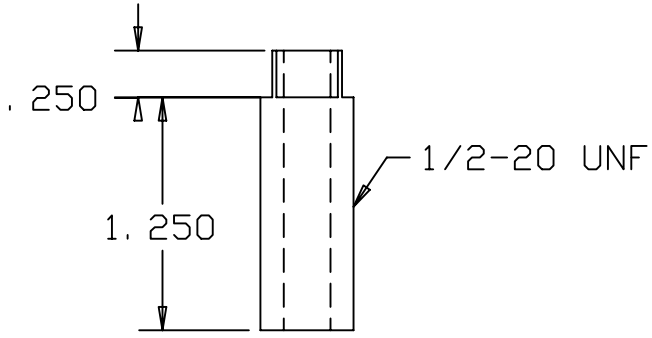


UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE ALUMINUM-2024	PROJECT HOPKINSON BAR
	QUANTITY 3	TITLE STRIKER HOUSING MOUNT SUPPORT
TREATMENT NONE	NOTES	SIZE A / REVISION NO. 1 / PART NO.
FINISH NONE		
DATE SEPTEMBER 13, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

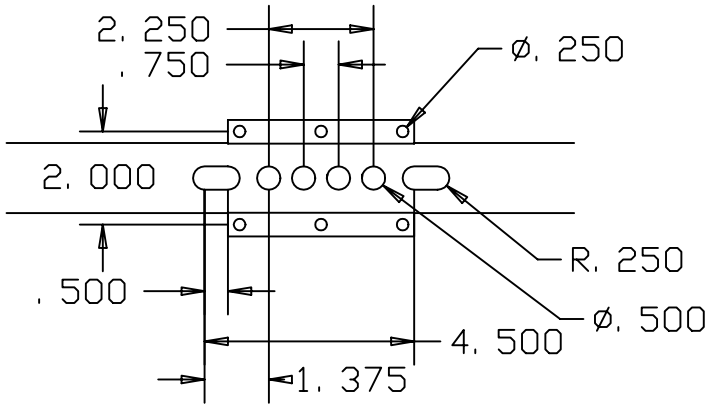


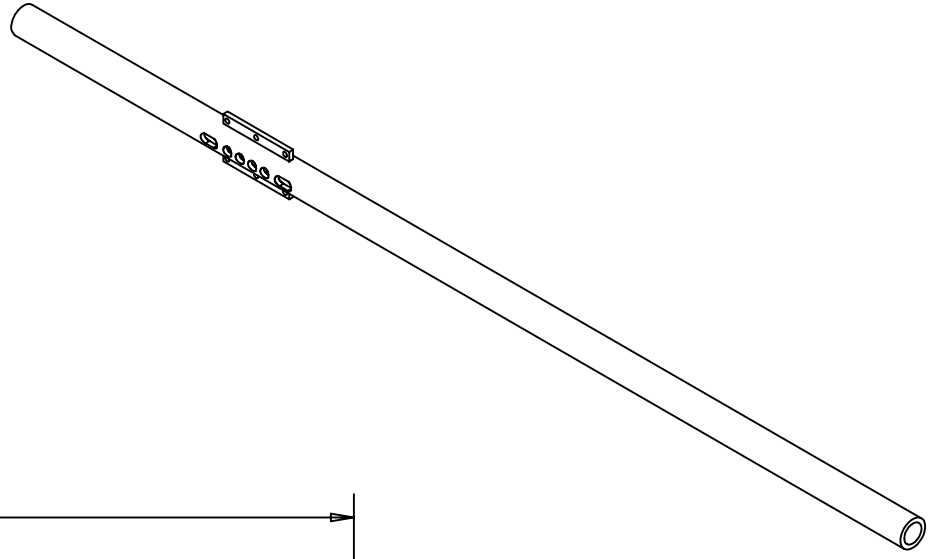


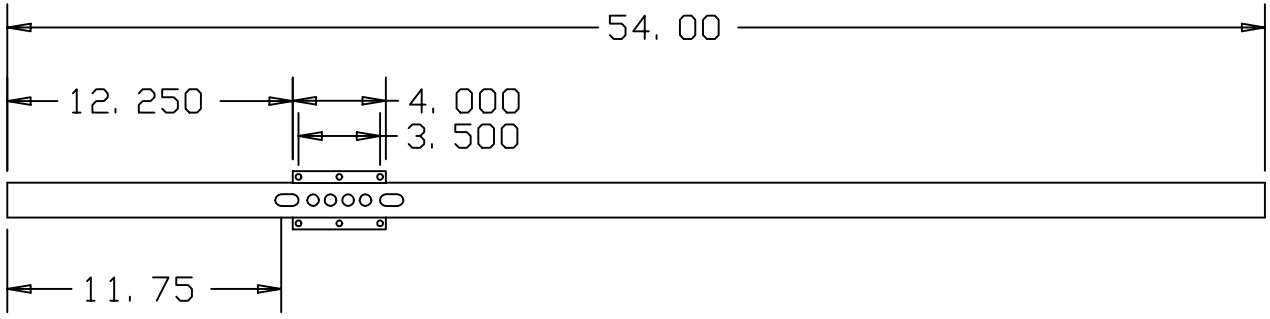


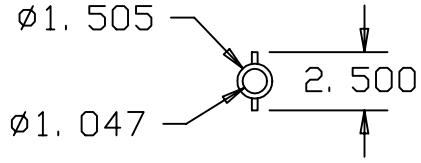
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS .X=±.1 .XX=±0.02 .XXX=±0.005 ANGLES X=±0.5 X.X=±0.05	MATERIAL TYPE: STEEL	PROJECT: HOPKINSON BAR
	QUANTITY: 12	TITLE: STRIKER HOUSING MOUNT ALIGNMENT PIN
TREATMENT: NONE	NOTES	SIZE: A / REVISION NO.: 1 / PART NO.
FINISH: NONE		
DATE: SEPTEMBER 13, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

DWG. NO.	SH	REV.

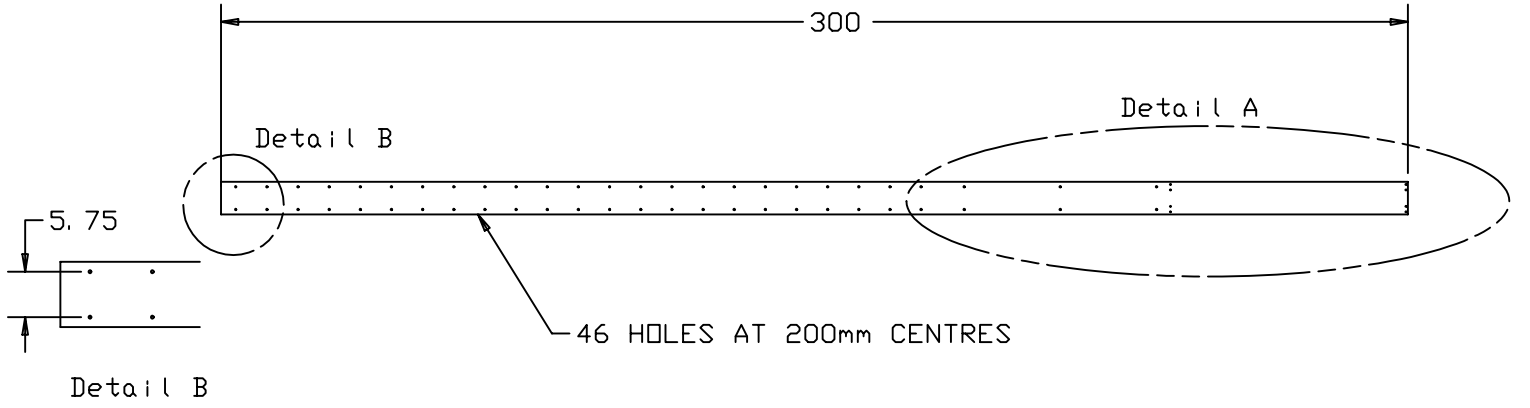




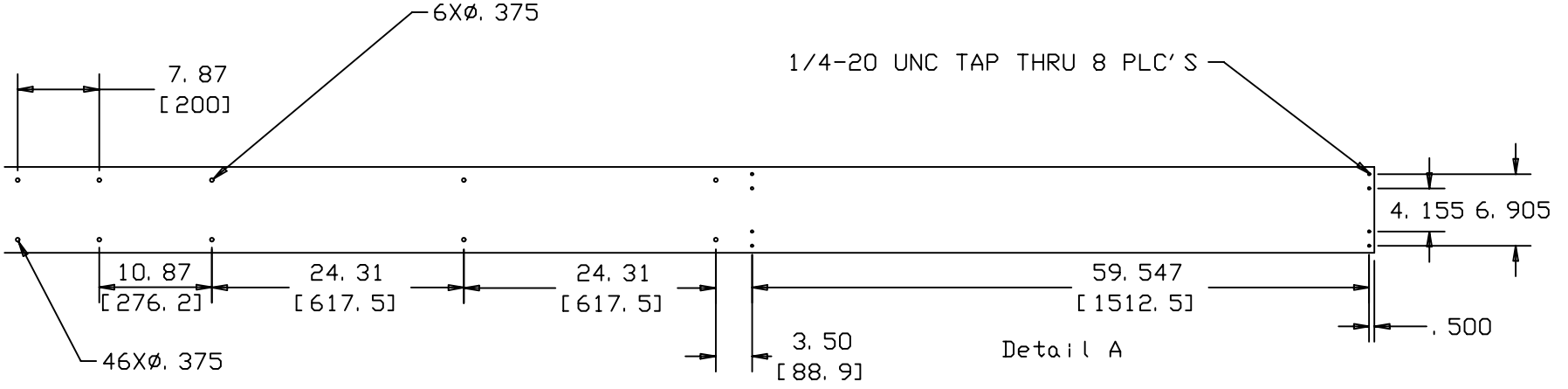




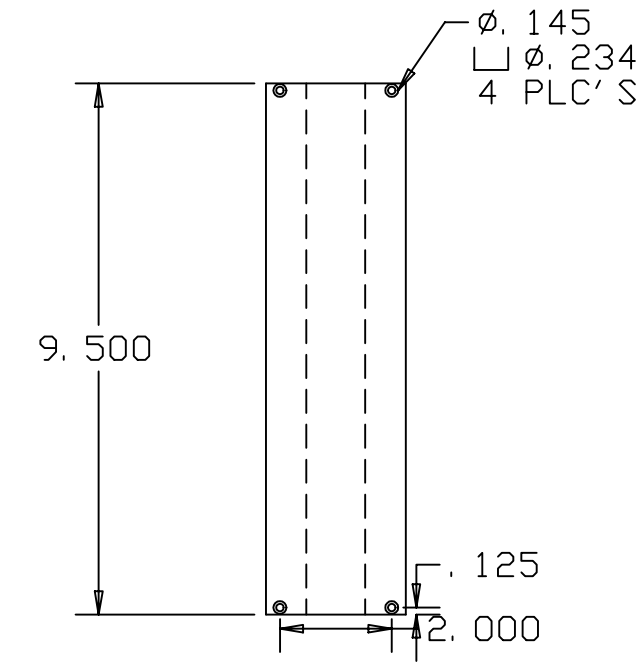
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE STEEL	PROJECT HOPKINSON BAR
	QUANTITY 1	TITLE STRIKER HOUSING
TREATMENT NONE	NOTES	SIZE A / REVISION NO. 2 / PART NO.
FINISH NONE		
DATE AUGUST 10, 2000	DO NOT SCALE DRAWING	SCALE / SHEET

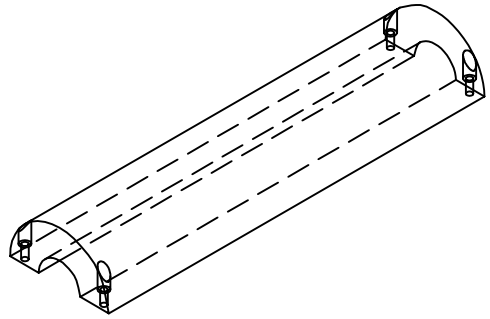


LAYOUT FOR HOLES ON TOP PART OF BEAM



DWG. NO.	SH	REV.





UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ARE: DECIMALS . X=±. 1 . XX=±0. 02 . XXX=±0. 005 ANGLES X=±0. 5 X. X=±0. 05	MATERIAL TYPE NYLON	PROJECT HOPKINSON BAR
	QUANTITY 7	TITLE UPPER AIR BEARING
TREATMENT NONE	NOTES	SIZE A | REVISION NO. 1 | PART NO.
FINISH NONE		
DATE SEPT. 8, 2000	DO NOT SCALE DRAWING	SCALE | SHEET

Appendix B

```
#if !defined(AFX_AMPLITUDEPROPDLG_H__5C0B38B1_5396_11D5_AD6C_00A0CCE1AE89__INCLUDED_)
#define AFX_AMPLITUDEPROPDLG_H__5C0B38B1_5396_11D5_AD6C_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// AmplitudePropDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CAmplitudePropDlg dialog

class CAmplitudePropDlg : public CDialog
{
// Construction
public:
    CAmplitudePropDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CAmplitudePropDlg)
    enum { IDD = IDD_AMP_PROP };
    int     m_nPoints;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAmplitudePropDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CAmplitudePropDlg)
    virtual void OnOK();
    virtual void OnCancel();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_AMPLITUDEPROPDLG_H__5C0B38B1_5396_11D5_AD6C_00A0CCE1AE89__INCLUDED_)
```

```
#if !defined(AFX_ANALYSEDLG_H__7F641B02_3A49_11D5_AD55_00A0CCE1AE89__INCLUDED_)
#define AFX_ANALYSEDLG_H__7F641B02_3A49_11D5_AD55_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// AnalyseDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CAnalyseDlg dialog

class CAnalyseDlg : public CDialog
{
// Construction
public:
    CAnalyseDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CAnalyseDlg)
    enum { IDD = IDD_ANALYSEDLG };
    double  m_dSampLo;
    double  m_dSampDf;
    double  m_dSampDo;
    double  m_dSampLf;
    CString m_szSampName;
    CString m_szCalib_I;
    CString m_szCalib_T;
    CString m_szData_I;
    CString m_szData_T;
    BOOL    m_bIncidentDispersion;
    BOOL    m_bTransmittedDispersion;
    double  m_dFreqFilter;
    int     m_nFourierAnalysis;
    int     m_nNyquist;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAnalyseDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CAnalyseDlg)
    virtual void OnCancel();
    afx_msg void OnFourier();
    afx_msg void OnNyquist();
    afx_msg void OnFilter();
    afx_msg void OnConvanalysis();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_ANALYSEDLG_H__7F641B02_3A49_11D5_AD55_00A0CCE1AE89__INCLUDED_)
```

```
#if !defined(AFX_BARDATADLG_H__901B25E3_1C7B_11D5_AD55_00A0CCE1AE89__INCLUDED_)
#define AFX_BARDATADLG_H__901B25E3_1C7B_11D5_AD55_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// BarDataDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CBarDataDlg dialog

class CBarDataDlg : public CDialog
{
// Construction
public:
    struct bar_data{
        CArray<double,double> m_dVolts;
        CArray<double,double> m_dStrain;
        CArray <double,double> m_dG;
        CArray <double,double> m_dFreq;
    }I,T;

public:
    BOOL m_bReadTransmittedBar;
    BOOL m_bReadIncidentBar;
    double m_dIDf,m_dIDT;
    double m_dTDf,m_dTDT;
    BOOL GetTransmittedBarData();
    BOOL GetIncidentBarData();
    double gammln(double xx);
    void gcf (double *gammcf,double a,double x, double *gln);
    void gser(double *gamser,double a,double x, double *gln);

    double gammq(double a,double x);
    void fit(double x[],double y[], int ndata, double sig[], int mwt, double *a, double *b, double
 *siga, double *sigb, double *chi2, double *q);

    CBarDataDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CBarDataDlg)
    enum { IDD = IDD_BARDATA };
    double  m_dILength;
    double  m_dIPRatio;
    double  m_dTDensity;
    double  m_dTDiameter;
    double  m_dTModulus;
    double  m_dTLength;
    double  m_dTPRatio;
    double  m_dIDensity;
    double  m_dIDiameter;
    double  m_dIModulus;
    CString m_szIFilename;
    CString m_szIName;
    double  m_dISlope;
    CString m_szTName;
    CString m_szTFilename;
    double  m_dTSlope;
    double  m_dIDistance;
    double  m_dTDistance;
    BOOL    m_bIPropCoeffIncluded;
    BOOL    m_bTPropCoeffIncluded;
    double  m_dIFilter;
    double  m_dTFilter;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CBarDataDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
```

```
protected:

    // Generated message map functions
    //{{AFX_MSG(CBarDataDlg)
    afx_msg void OnBrowseIncident();
    afx_msg void OnBrowseTransmitted();
    virtual void OnOK();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_BARDATADLG_H__901B25E3_1C7B_11D5_AD55_00A0CCE1AE89__INCLUDED_)
```

```
#if !defined(AFX_BARDATADLG_H__901B25E3_1C7B_11D5_AD55_00A0CCE1AE89__INCLUDED_)
#define AFX_BARDATADLG_H__901B25E3_1C7B_11D5_AD55_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// BarDataDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CBarDataDlg dialog

class CBarDataDlg : public CDialog
{
// Construction
public:
    struct bar_data{
        CArray<double,double> m_dVolts;
        CArray<double,double> m_dStrain;
        CArray <double,double> m_dG;
        CArray <double,double> m_dFreq;
    }I,T;

public:
    BOOL m_bReadTransmittedBar;
    BOOL m_bReadIncidentBar;
    double m_dIDf,m_dIDT;
    double m_dTDf,m_dTDT;
    BOOL GetTransmittedBarData();
    BOOL GetIncidentBarData();
    double gammln(double xx);
    void gcf (double *gammcf,double a,double x, double *gln);
    void gser(double *gamser,double a,double x, double *gln);

    double gammq(double a,double x);
    void fit(double x[],double y[], int ndata, double sig[], int mwt, double *a, double *b, double
 *siga, double *sigb, double *chi2, double *q);

    CBarDataDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CBarDataDlg)
    enum { IDD = IDD_BARDATA };
    double  m_dILength;
    double  m_dIPRatio;
    double  m_dTDensity;
    double  m_dTDiameter;
    double  m_dTModulus;
    double  m_dTLength;
    double  m_dTPRatio;
    double  m_dIDensity;
    double  m_dIDiameter;
    double  m_dIModulus;
    CString m_szIFilename;
    CString m_szIName;
    double  m_dISlope;
    CString m_szTName;
    CString m_szTFilename;
    double  m_dTSlope;
    double  m_dIDistance;
    double  m_dTDistance;
    BOOL    m_bIPropCoeffIncluded;
    BOOL    m_bTPropCoeffIncluded;
    double  m_dIFilter;
    double  m_dTFilter;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CBarDataDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
```

```
protected:

    // Generated message map functions
    //{{AFX_MSG(CBarDataDlg)
    afx_msg void OnBrowseIncident();
    afx_msg void OnBrowseTransmitted();
    virtual void OnOK();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_BARDATADLG_H__901B25E3_1C7B_11D5_AD55_00A0CCE1AE89__INCLUDED_)
```

```
// CSHB.h : main header file for the CSHB application
//

#if !defined(AFX_CSHB_H__437D5685_0F45_11D5_AD55_00A0CCE1AE89__INCLUDED_)
#define AFX_CSHB_H__437D5685_0F45_11D5_AD55_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CCSHBApp:
// See CSHB.cpp for the implementation of this class
//

class CCSHBApp : public CWinApp
{
public:
    CCSHBApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CCSHBApp)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation
    //{{AFX_MSG(CCSHBApp)
    afx_msg void OnAppAbout();
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_CSHB_H__437D5685_0F45_11D5_AD55_00A0CCE1AE89__INCLUDED_)
```

```
// CSHBDoc.h : interface of the CCSHBDoc class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_CSHBDOC_H__C9D5C164_88E0_4640_B1C0_BC1CBCC2C49D__INCLUDED_)
#define AFX_CSHBDOC_H__C9D5C164_88E0_4640_B1C0_BC1CBCC2C49D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


class CCSHBDoc : public CDocument
{
protected: // create from serialization only
    CCSHBDoc();
    DECLARE_DYNCREATE(CCSHBDoc)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CCSHBDoc)
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL

// Implementation
public:

    virtual ~CCSHBDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CCSHBDoc)
    afx_msg void OnFileReadIncident();
    afx_msg void OnFileReadTransmitted();
    afx_msg void OnBarProperties();
    afx_msg void OnSampleProperties();
    afx_msg void OnExport();
    afx_msg void OnAnalyseCalculateresults();
    afx_msg void OnUtilitiesFft();
    afx_msg void OnUtilitiesFileconversion();
    afx_msg void OnTempShit();
    afx_msg void OnFileNewsample();
    afx_msg void OnSeperatewave();
    afx_msg void OnUpdateCalculateResults(CCmdUI* pCmdUI);
    afx_msg void OnAnalysePropagationCalculation();
    afx_msg void OnUpdateAnalysePropcalc(CCmdUI* pCmdUI);
    afx_msg void OnUpdatePreSeperatewave(CCmdUI* pCmdUI);
    afx_msg void OnUtilitiesPropagatewave();
    afx_msg void OnUpdateUtilitiesPropagatewave(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
//Member Variables
public:


    struct raw_data{

        CArray<double,double> m_dtime;
        CArray<double,double> m_dvolts;
//      CArray<double,double> m_dpeak;
//      CArray<double,double> m_dtruepeak;
//      CArray<int,int> m_npeakindex;
//      CArray<int,int> m_ntruepeakindex;
```

```
        double m_dDT;               //timestep
        double m_dmaxy,m_dmaxx;
        double m_dminy,m_dminx;
        double m_dzero;
        int m_dmaxindex,m_dminindex;
        CString m_szComment;
        CString m_szFilename;
        COleDateTime m_dtDate;
    }data1,data2;

    struct processed_data{
//      CArray<double,double> m_dtime;
        CArray<double,double> m_dstrain;
        double m_dDf;
        //double m_dbegin;
        //double m_dend;
        double m_dmaxstrain;
        double m_dminstrain;
        int m_nbegin;
        int m_nend;

        BOOL m_bTime;
        BOOL m_bPolar;
//      double m_dtolerance_value,m_dthreshold1_value,m_dthreshold2_value;
    }transmitted,incident,reflected;

    struct sample_data{
        CArray<double,double>   m_dengstress;       //engineering stress
        CArray<double,double>   m_dengstrain;       //engineering strain
        CArray<double,double>   m_dengstrainrate;       //engineering strain rate
        double m_dLo,m_dDo,m_dLf,m_dDf;                         //orginal and final dimensions
        double m_dArea;
        CString m_szName;
        COleDateTime m_dtDate;
        double m_dmax_strainrate,m_dmin_strainrate;
        double m_dmax_strain,m_dmin_strain;
        double m_dmax_stress,m_dmin_stress;
        BOOL m_bCalculatedStrain;
        BOOL m_bCalculatedStrainRate;
        BOOL m_bCalculatedStress;
        double m_dPressure,m_dVelocity,m_dStrikerLength;


    }sample;
    struct bar_data{
        double m_dE,m_dD,m_dL,m_dDensity,m_dPratio;                 //bar properties
        double m_dDistance;
        double m_dCo,m_dArea;                           //wave velocity in bar
        CArray<double,double> m_dmV;
        double m_dSlope,m_dIntercept;                       //slope and intercept values for calib
ration
        CArray<double,double> m_dG;     // if dispersion included
        CArray<double,double> m_dVelocity;      // if dispersion included
        CArray<double,double> m_dForce;         //then each bar will have a force and a velocity
        CArray<double,double> m_dDisplacement;
        double m_dDf,m_dDT;
        double m_dFilter;
        CString m_szFilename;
        CString m_szname;
        BOOL m_bPropCoeffIncluded;
        struct data{
                BOOL m_bCalculated;
                BOOL m_bTime;
                double m_dmax;
                double m_dmin;
        }Force,Velocity,Displacement;

    }incidentbar,transmittedbar;


    struct flags{
        BOOL m_bReadIncident;
        BOOL m_bReadTransmitted;
        BOOL m_bIncidentStrain;
        BOOL m_bTransmittedStrain;
```

```
        BOOL m_bReflectedStrain;
        BOOL m_bReadIncidentBar;
        BOOL m_bReadTransmittedBar;
        BOOL m_bSeparatedIncident;
        BOOL m_bSeparatedTransmitted;
        BOOL m_bProcessedResults;
        BOOL m_bReadSampleData;
        BOOL m_bPropagateUtility;
    //  BOOL m_bData1Interpolated;
    //  BOOL m_bData2Interpolated;
        BOOL m_bRecalculate;

    }flag;

    struct scale{
        double time,volts;
        double strain,strainrate,stress;
        double force,velocity,displacement;
        double strainratetime,straintime;
    }scales;

    struct PropUtility{
        double m_dF;
        double strain,strainrate,stress;
        double force,velocity,displacement;
        double strainratetime,straintime;
    }PropUtilitydata;

public:
    BOOL CalculatePropagationCoefficient();

    CSize GetDocSize() { return m_sizeDoc; }
    static  CCSHBDoc* GetDoc();
    BOOL Export();
    BOOL CalculateResults();
    void Cmul(double r_a,double i_a,double r_b,double i_b, double *r_c, double *i_c);
    double m_dFrequencyFilter;
    void InterpolateData(int n,double dt_a,double dt_b,double data[]);
    BOOL CalculateForce();
    BOOL CalculateVelocity();
    BOOL CalculateDisplacement();
    static BOOL ChangeForm(int N,double data[],BOOL *polar);

    BOOL PropagateWave(int data1_n,double data[],double dt,int data2_n,double H[],int sign, double
 x);
    static double arctan(double imag,double real);
    BOOL GetBeginEnd(BOOL Pulse,double y[],double max,double min,double zero,int n, int *begin, in
t *end,int *begin2,int *end2,int nIntercept);
    void InitializeValues();
    void Cdiv(double r_a, double i_a,double r_b,double i_b,double *r_c,double *i_c);

    static void Realft(double data[],unsigned long n,BOOL *Time);
    static void FFT(double data[],unsigned long nn, int isign);
//  double gammln(double xx);
//  void gcf (double *gammcf,double a,double x, double *gln);
//  void gser(double *gamser,double a,double x, double *gln);

//  double gammq(double a,double x);
//  void fit(double x[],double y[], int ndata, double sig[], int mwt, double *a, double *b, double
 *siga, double *sigb, double *chi2, double *q);



    void CalculateStress();
    void CalculateStrain();
    void CalculateStrainRate();
    BOOL Volts_to_Strain(double in[],double out[],int n,double slope,double *max,double *min);



protected:
    CSize   m_sizeDoc;
    int m_nFourierAnalysis;

    CArray<double,double> m_dH;
    struct triggers{
```

```
        double m_dtolerance_value,m_dthreshold1_value,m_dthreshold2_value;
    }trigger;
};



/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_CSHBDOC_H__C9D5C164_88E0_4640_B1C0_BC1CBCC2C49D__INCLUDED_)
```

```
// CSHBView.h : interface of the CCSHBView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_CSHBVIEW_H__A4DD65E3_CFBF_4E42_8A69_AB7AF9F0E5F5__INCLUDED_)
#define AFX_CSHBVIEW_H__A4DD65E3_CFBF_4E42_8A69_AB7AF9F0E5F5__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


class CCSHBView : public CView
{
protected: // create from serialization only
    CCSHBView();
    DECLARE_DYNCREATE(CCSHBView)

// Attributes
public:
    CCSHBDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CCSHBView)
    public:
    virtual void OnDraw(CDC* pDC);  // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void OnInitialUpdate();
    protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnPrint(CDC* pDC, CPrintInfo* pInfo);
    //}}AFX_VIRTUAL

// Implementation
public:
    BOOL GraphTick(int x_origin,int y_origin,double maxx,double maxy,double scalex,double scaley);

    void AllDoneModeless();
    BOOL GraphFrequency(BOOL y_vs_t,double dx,double xo,double x[], double y[], int n, double scal
e_x, double scale_y,DWORD colour, int size);
    BOOL GraphData(BOOL y_vs_t,double dx,double xo,double x[],double y[],int n,double scale_x,doub
le scale_y,DWORD colour,int size,int x_origin,int y_origin);
    BOOL DrawAxis(CDC *pDC,int x_origin,int y_origin,int x_length,int y_length,
                CString strTitle, CString strXAxis, CString strYAxis);

    virtual ~CCSHBView();
//public:
//  struct colours{
//      DWORD Red=RGB(245,0,0);
//  }colour;
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

    int m_cyClient;
    int m_cxClient;
// Generated message map functions
protected:
    int m_nViewExtra;
    double m_dXscale;
    double m_dYscale;
    CPoint m_ptMouse;
    //{{AFX_MSG(CCSHBView)
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnViewfreq();
    afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
```

```
    afx_msg void OnCancelMode();
    afx_msg void OnPreViewindices();
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnViewVelocity();
    afx_msg void OnViewForce();
    afx_msg void OnViewStrain();
    afx_msg void OnUpdateViewVelocity(CCmdUI* pCmdUI);
    afx_msg void OnUpdateViewForce(CCmdUI* pCmdUI);
    afx_msg void OnUpdateViewStrain(CCmdUI* pCmdUI);
    afx_msg void OnViewDisplacement();
    afx_msg void OnUpdateViewDisplacement(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CRect m_rcPrintRect;
};

#ifndef _DEBUG  // debug version in CSHBView.cpp
inline CCSHBDoc* CCSHBView::GetDocument()
   { return (CCSHBDoc*)m_pDocument; }
#endif
#define RED     RGB(245,0,0)
#define BLACK   RGB(0,0,0)          //black
#define GREEN   RGB(0,255,0)        //green
#define BLUE    RGB(0,0,255)        //blue
#define YELLOW  RGB(255,255,0)      //yellow
#define MAGENTA RGB(255,0,255)      //magenta
#define CYAN    RGB(0,255,255)      //cyan
#define GRAY    RGB(127,127,127)    //gray
#define ORANGE  RGB(251,153,55)
#define BROWN   RGB(108,100,90)
/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_CSHBVIEW_H__A4DD65E3_CFBF_4E42_8A69_AB7AF9F0E5F5__INCLUDED_)
```

```
#if !defined(AFX_EXPORTDATADLG_H__AF2E8BE1_2D12_11D5_AD55_00A0CCE1AE89__INCLUDED_)
#define AFX_EXPORTDATADLG_H__AF2E8BE1_2D12_11D5_AD55_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ExportDataDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CExportDataDlg dialog

class CExportDataDlg : public CDialog
{
// Construction
public:
    BOOL m_bEngineeringAllowed;
    BOOL m_bForcesAllowed;
    BOOL m_bStrainsAllowed;
    BOOL m_bVelocitiesAllowed;
    void SetAllowed();

    CExportDataDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CExportDataDlg)
    enum { IDD = IDD_EXPORT };
    BOOL    m_bEngineeringData;
    BOOL    m_bHeader;
    BOOL    m_bStrains;
    BOOL    m_bTrueData;
    BOOL    m_bForces;
    BOOL    m_bVelocities;
    CString m_szExportFileName;
    BOOL    m_bDisplacements;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CExportDataDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CExportDataDlg)
    afx_msg void OnBrowse();
    virtual void OnOK();
    virtual void OnCancel();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_EXPORTDATADLG_H__AF2E8BE1_2D12_11D5_AD55_00A0CCE1AE89__INCLUDED_)
```

```
#if !defined(AFX_FFTUTILITYDLG_H__8D1A1631_4AF4_11D5_AD6C_00A0CCE1AE89__INCLUDED_)
#define AFX_FFTUTILITYDLG_H__8D1A1631_4AF4_11D5_AD6C_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FFTUtilityDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CFFTUtilityDlg dialog

class CFFTUtilityDlg : public CDialog
{
// Construction
public:
    BOOL OutputData();
    BOOL GetInputData();
    CFFTUtilityDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CFFTUtilityDlg)
    enum { IDD = IDD_FFTUTILITY };
    CString m_szInputFile;
    CString m_szOutputFile;
    int     m_nTransform;
    //}}AFX_DATA

    CArray <double,double> m_dData1;
    CArray <double,double> m_dData2;
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFFTUtilityDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CFFTUtilityDlg)
    afx_msg void OnBrowseIn();
    afx_msg void OnBrowseOut();
    virtual void OnOK();
    virtual void OnCancel();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FFTUTILITYDLG_H__8D1A1631_4AF4_11D5_AD6C_00A0CCE1AE89__INCLUDED_)
```

```
#if !defined(AFX_INDEXDLG_H__A6D5E483_3E76_11D5_AD60_00A0CCE1AE89__INCLUDED_)
#define AFX_INDEXDLG_H__A6D5E483_3E76_11D5_AD60_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// IndexDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CIndexDlg dialog

class CIndexDlg : public CDialog
{
// Construction
public:
    BOOL CheckTime();
    BOOL ValidateData();
    int m_nData2Size;
    int m_nData1Size;
    double m_dTDT;
    double m_dIDT;
    int m_nIndex;
    //CCSHBDoc *GetDocument();
    BOOL m_bTime;
    CIndexDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CIndexDlg)
    enum { IDD = IDD_INDICES };
    double  m_dIe;
    double  m_dIs;
    double  m_dRe;
    double  m_dRs;
    double  m_dTe;
    double  m_dTs;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CIndexDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual void PostNcDestroy();
    //}}AFX_VIRTUAL

// Implementation
protected:
    CWnd* m_pParent;
    // Generated message map functions
    //{{AFX_MSG(CIndexDlg)
    virtual void OnOK();
    afx_msg void OnViewtime();
    afx_msg void OnChooseIs();
    afx_msg void OnChooseRs();
    afx_msg void OnChooseTs();
    afx_msg void OnChooseIe();
    afx_msg void OnChooseRe();
    afx_msg void OnChooseTe();
    afx_msg void OnDestroy();
    virtual void OnCancel();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_INDEXDLG_H__A6D5E483_3E76_11D5_AD60_00A0CCE1AE89__INCLUDED_)
```

```
#if !defined(AFX_INPUTFILEDLG_H__27E76F10_538A_11D5_AD6C_00A0CCE1AE89__INCLUDED_)
#define AFX_INPUTFILEDLG_H__27E76F10_538A_11D5_AD6C_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// InputFileDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CInputFileDlg dialog

class CInputFileDlg : public CDialog
{
public:
    struct raw_data{
        CArray <double,double> m_dtime;
        CArray <double,double> m_dvolts;
        double  m_dDT;
        COleDateTime m_dtDate;
    }Data;

// Construction
public:
    int m_nZeroPoints;
    CString GetField(char *string, int offset,int size);
    BOOL GetData();
    CInputFileDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CInputFileDlg)
    enum { IDD = IDD_INPUTFILE };
    int     m_nBinary;
    CString m_szFileName;
    BOOL    m_bAmplitudeShift;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CInputFileDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CInputFileDlg)
    afx_msg void OnBrowse();
    virtual void OnOK();
    afx_msg void OnAscii();
    afx_msg void OnBinary();
    virtual void OnCancel();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_INPUTFILEDLG_H__27E76F10_538A_11D5_AD6C_00A0CCE1AE89__INCLUDED_)
```

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__437D5689_0F45_11D5_AD55_00A0CCE1AE89__INCLUDED_)
#define AFX_MAINFRM_H__437D5689_0F45_11D5_AD55_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMainFrame : public CMDIFrameWnd
{
    DECLARE_DYNAMIC(CMainFrame)
public:
    CMainFrame();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MAINFRM_H__437D5689_0F45_11D5_AD55_00A0CCE1AE89__INCLUDED_)
```

```
#ifndef _NR_UTILS_H_
#define _NR_UTILS_H_


static double dsqrarg;
#define DSQR(a) ((dsqrarg=(a))==0.0?0.0:dsqrarg*dsqrarg)

static float sqrarg;
#define SQR(a) ((sqrarg=(a))==0.0?0.0:sqrarg*sqrarg)

void nrerror(char error_text[]);
#endif  //_NR_UTILS_H_
```

```
#if !defined(AFX_PROPAGATEWAVEUTILITYDLG_H__D01D5271_7A46_11D5_AD77_00A0CCE1AE89__INCLUDED_)
#define AFX_PROPAGATEWAVEUTILITYDLG_H__D01D5271_7A46_11D5_AD77_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropagateWaveUtilityDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPropagateWaveUtilityDlg dialog

class CPropagateWaveUtilityDlg : public CDialog
{
// Construction
public:
    CPropagateWaveUtilityDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CPropagateWaveUtilityDlg)
    enum { IDD = IDD_PROPAGATEWAVE };
    double  m_dTransmittedDistance;
    double  m_dIncidentDistance;
    int     m_nNyquist;
    double  m_dFreqFilter;
    double  m_dReflectedDistance;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPropagateWaveUtilityDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CPropagateWaveUtilityDlg)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_PROPAGATEWAVEUTILITYDLG_H__D01D5271_7A46_11D5_AD77_00A0CCE1AE89__INCLUDED_)
```

```
#if !defined(AFX_PROPCOEFFDLG_H__6F355E11_4555_11D5_AD6A_00A0CCE1AE89__INCLUDED_)
#define AFX_PROPCOEFFDLG_H__6F355E11_4555_11D5_AD6A_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PropCoeffDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPropCoeffDlg dialog

class CPropCoeffDlg : public CDialog
{
// Construction
public:
    CPropCoeffDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CPropCoeffDlg)
    enum { IDD = IDD_PROPCOEFF };
    CString m_szCalibrationFile;
    CString m_szDataFile;
    BOOL    m_bUpdateFile;
    BOOL    m_bExportFile;
    BOOL    m_bComplexModulus;
    double  m_dFilter;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPropCoeffDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CPropCoeffDlg)
    afx_msg void OnDeterminecoeff();
    virtual void OnCancel();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_PROPCOEFFDLG_H__6F355E11_4555_11D5_AD6A_00A0CCE1AE89__INCLUDED_)
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by CSHB.rc
//
#define IDCONVERT                       3
#define IDD_ABOUTBOX                    100
#define IDD_NICOLETCONVERTDLG           102
#define IDR_MAINFRAME                   128
#define IDR_CSHBTYPE                    129
#define IDD_AMP_PROP                    129
#define IDD_SAMPLEDATA                  130
#define IDD_BARDATA                     131
#define IDD_EXPORT                      132
#define IDD_ANALYSEDLG                  134
#define IDD_SEPARATEWAVEVALUES          135
#define IDD_INDICES                     136
#define IDD_PROPCOEFF                   138
#define IDD_FFTUTILITY                  140
#define IDD_INPUTFILE                   141
#define IDI_HAPPY                       143
#define IDC_CROSSHAIR                   146
#define IDD_PROPAGATEWAVE               149
#define IDC_SAMPLETITLE                 1000
#define IDC_INPUTFILE                   1000
#define IDC_INPUTBROWSE                 1001
#define IDC_Do                          1002
#define IDC_OUTPUTFILE                  1002
#define IDC_Lo                          1003
#define IDC_I_BARNAME                   1003
#define IDC_OUPUTBROWSE                 1003
#define IDC_I_CALIBRATIONNAME           1004
#define IDC_INCHEADER                   1004
#define IDC_I_BARDIAMETER               1005
#define IDC_AMPLTITUDE                  1005
#define IDC_Df                          1006
#define IDC_I_BARLENGTH                 1006
#define IDC_POINTS                      1006
#define IDC_Lf                          1007
#define IDC_I_DISTANCE                  1007
#define IDC_I_BARDENSITY                1008
#define IDC_I_BAREMODULUS               1009
#define IDC_SAMPLE_DEFAULT              1009
#define IDC_I_BARPRATIO                 1010
#define IDC_T_BARDIAMETER               1011
#define IDC_T_BARLENGTH                 1012
#define IDC_T_DISTANCE                  1013
#define IDC_T_BARDENSITY                1014
#define IDC_T_BAREMODULUS               1015
#define IDC_T_BARPRATIO                 1016
#define IDC_T_BARNAME                   1017
#define IDC_T_CALIBRATIONNAME           1018
#define IDC_EDIT1                       1020
#define IDC_INPUTNAME                   1020
#define IDC_FILENAME                    1020
#define IDC_DISPFILTER                  1020
#define IDC_EXPORTFILE                  1020
#define IDC_INCDATE                     1020
#define IDC_INCIDENTDIST                1020
#define IDC_I_FILTER                    1020
#define IDC_RS                          1021
#define IDC_OUTPUTNAME                  1021
#define IDC_TRANSMITTEDDIST             1021
#define IDC_T_FILTER                    1021
#define IDC_EXP_ENGSTRESS               1022
#define IDC_IE                          1022
#define IDC_EXP_ENGDATA                 1022
#define IDC_DISPFILTER2                 1022
#define IDC_EXP_INCIDENT_T              1023
#define IDC_RE                          1023
#define IDC_EXP_STRAINS                 1023
#define IDC_REFLECTEDDIST               1023
#define IDC_EXP_TRANSMITTED_T           1024
#define IDC_BROWSE_INCIDENT             1024
#define IDC_TE                          1024
#define IDC_EXP_REFLECTED_T             1025
#define IDC_TS                          1025
```

```
#define IDC_EXP_INCIDENT_F              1026
#define IDC_EXP_HEADER                  1026
#define IDC_EXP_TRUESTRESS              1027
#define IDC_I_SLOPE                     1027
#define IDC_EXP_TRUEDATA                1027
#define IDC_EXP_ENGSTRAIN               1028
#define IDC_I_INTERCEPT                 1028
#define IDC_EXP_TRUESTRAIN              1029
#define IDC_BROWSE_TRANSMITTED          1029
#define IDC_EXP_TRANSMITTED_F           1030
#define IDC_T_SLOPE                     1030
#define IDC_EXP_STRAINRATE              1031
#define IDC_T_INTERCEPT                 1031
#define IDC_EXP_REFLECTED_F             1032
#define IDC_EXP_COMPINCI                1033
#define IDC_DATE                        1034
#define IDC_EXP_COMPTRAN                1034
#define IDC_SAMP_NAME                   1044
#define IDC_SAMP_DO                     1045
#define IDC_SAMP_LO                     1046
#define IDC_SAMP_DF                     1047
#define IDC_SAMP_LF                     1048
#define IDC_DATA_I                      1049
#define IDC_DATA_I2                     1050
#define IDC_DATA_T                      1050
#define IDC_DATA_I3                     1051
#define IDC_CALIB_I                     1051
#define IDC_DATA_I4                     1052
#define IDC_CALIB_T                     1052
#define IDANALYSE                       1053
#define IDC_Default                     1055
#define IDC_THRESHOLD1                  1056
#define IDC_THRESHOLD2                  1057
#define IDC_IS                          1057
#define IDC_TOLERANCE                   1058
#define IDC_SEPARATETRANSMITTED         1064
#define IDC_SEPARATEINCIDENT            1065
#define IDC_DETERMINECOEFF              1066
#define IDC_VIEWFREQ                    1067
#define IDC_UPDATEFILE                  1068
#define IDC_BROWSE_IN                   1069
#define IDC_BROWSE_OUT                  1070
#define IDC_FFT                         1072
#define IDC_IFFT                        1073
#define IDC_INCDISP                     1076
#define IDC_FOURIER                     1076
#define IDC_CONVANALYSIS                1077
#define IDC_INCIDENT_V                  1078
#define IDC_VELOCITIES                  1078
#define IDC_TRANSMITTED_V               1079
#define IDC_INCIDENT_F                  1080
#define IDC_FORCES                      1080
#define IDC_TRANSMITTED_F               1081
#define IDC_IDISPAV                     1081
#define IDC_DISPLACEMENTS               1081
#define IDC_TDISPAV                     1082
#define IDC_EXPFILE                     1083
#define IDC_BINARY                      1084
#define IDC_COMPLEXMOD                  1084
#define IDC_ASCII                       1085
#define IDC_BROWSE                      1086
#define IDC_AMPSHIFT                    1087
#define IDC_VIEWINDEX                   1088
#define IDC_VIEWTIME                    1088
#define IDC_START                       1089
#define IDC_END                         1090
#define IDC_TRANSDATE                   1097
#define IDC_MONTHCALENDAR3              1101
#define IDC_CHOOSE_IS                   1102
#define IDC_CHOOSE_RS                   1103
#define IDC_CHOOSE_TS                   1104
#define IDC_CHOOSE_IE                   1105
#define IDC_CHOOSE_RE                   1106
#define IDC_CHOOSE_TE                   1107
#define IDC_BUTTON3                     1108
#define IDC_INCDISPERSION               1111
```

```
#define IDC_INTERCEPT                   1113
#define IDC_PEAK                        1114
#define IDC_PRESSURE                    1115
#define IDC_VELOCITY                    1116
#define IDC_NYQUIST                     1116
#define IDC_FILTER                      1117
#define IDC_STRIKERL                    1117
#define ID_FILE_READ_INCIDENT           32771
#define ID_FILE_READ_TRANSMITTED        32772
#define ID_CALCULATE_SEPERATEWAVE       32773
#define ID_CALCULATE_DRAWWAVE           32774
#define ID_DRAWDATA                     32775
#define ID_BAR_PROPERTIES               32776
#define ID_SAMPLE_PROPERTIES            32777
#define ID_DRAWPROCESSEDDATA            32780
#define ID_CALCULATERESULTS             32781
#define ID_EXPORT                       32783
#define ID_FFT                          32784
#define ID_PROP_PREPROCESS              32785
#define ID_PROP_CALC                    32786
#define ID_ANALYSE_CALCULATERESULTS     32788
#define ID_PRE_VIEWINDICES              32789
#define ID_UTILITIES_FILECONVERSION     32790
#define ID_UTILITIES_FFT                32791
#define ID_UTILITIES_PROPCALC           32792
#define ID_TEMP_SHIT                    32793
#define ID_FILE_NEWSAMPLE               32794
#define ID_PRE_SEPERATEWAVE             32796
#define ID_POST_VIEWVEL                 32797
#define ID_POST_VWFOR                   32798
#define ID_POST_VWSTRN                  32799
#define ID_POST_VWDISP                  32800
#define ID_ANALYSE_PROPCALC             32801
#define ID_UTILITIES_PROPAGATEWAVE      32809

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        150
#define _APS_NEXT_COMMAND_VALUE         32810
#define _APS_NEXT_CONTROL_VALUE         1118
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

```
#if !defined(AFX_SAMPLEDATADLG_H__8DFCFC20_1C89_11D5_AD55_00A0CCE1AE89__INCLUDED_)
#define AFX_SAMPLEDATADLG_H__8DFCFC20_1C89_11D5_AD55_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// SampleDataDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CSampleDataDlg dialog

class CSampleDataDlg : public CDialog
{
// Construction
public:
    COleDateTime m_dtDate;
    CSampleDataDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CSampleDataDlg)
    enum { IDD = IDD_SAMPLEDATA };
    CDateTimeCtrl   m_dctrlDate;
    double  m_dDf;
    double  m_dDo;
    double  m_dLf;
    double  m_dLo;
    CString m_szName;
    COleDateTime    m_dtIDate;
    COleDateTime    m_dtTDate;
    double  m_dPressure;
    double  m_dVelocity;
    double  m_dStrikerLength;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSampleDataDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CSampleDataDlg)
    afx_msg void OnSampleDefault();
    virtual void OnOK();
    afx_msg void OnDateTimeChange(NMHDR* pNMHDR, LRESULT* pResult);
    virtual BOOL OnInitDialog();
    virtual void OnCancel();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SAMPLEDATADLG_H__8DFCFC20_1C89_11D5_AD55_00A0CCE1AE89__INCLUDED_)
```

```
#if !defined(AFX_SEPARATEVALUESDLG_H__A6D5E482_3E76_11D5_AD60_00A0CCE1AE89__INCLUDED_)
#define AFX_SEPARATEVALUESDLG_H__A6D5E482_3E76_11D5_AD60_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// SeparateValuesDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CSeparateValuesDlg dialog

class CSeparateValuesDlg : public CDialog
{
// Construction
public:
    CSeparateValuesDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CSeparateValuesDlg)
    enum { IDD = IDD_SEPARATEWAVEVALUES };
    double  m_dthreshold1;
    double  m_dthreshold2;
    double  m_dtolerance;
    BOOL    m_bSeparateTransmitted;
    BOOL    m_bSeparateIncident;
    int     m_nIntercept;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSeparateValuesDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CSeparateValuesDlg)
    virtual void OnOK();
    afx_msg void OnDefault();
    virtual void OnCancel();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SEPARATEVALUESDLG_H__A6D5E482_3E76_11D5_AD60_00A0CCE1AE89__INCLUDED_)
```

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__437D5687_0F45_11D5_AD55_00A0CCE1AE89__INCLUDED_)
#define AFX_STDAFX_H__437D5687_0F45_11D5_AD55_00A0CCE1AE89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC Automation classes
#include <afxdtctl.h>       // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>         // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__437D5687_0F45_11D5_AD55_00A0CCE1AE89__INCLUDED_)
#include <afxtempl.h>    // MFC templates I added to account for CArray usage
```

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////
//////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////
//////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////
//////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

#ifdef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////
//////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
    "#define _AFX_NO_OLE_RESOURCES\r\n"
```

```
    "#define _AFX_NO_TRACKER_RESOURCES\r\n"
    "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
    "\r\n"
    "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\r\n"
    "#ifdef _WIN32\r\n"
    "LANGUAGE 9, 1\r\n"
    "#pragma code_page(1252)\r\n"
    "#endif //_WIN32\r\n"
    "#include ""res\\CSHB.rc2""  // non-Microsoft Visual C++ edited re
sources\r\n"
    "#include ""afxres.rc""         // Standard components\r\n"
    "#include ""afxprint.rc""       // printing/print preview resource
s\r\n"
    "#endif\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED


/////////////////////////////////////////////////////////////////////////
///////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON    DISCARDABLE     "res\\CSHB.ico"
IDR_CSHBTYPE            ICON    DISCARDABLE     "res\\CSHBDoc.ico"

/////////////////////////////////////////////////////////////////////////
///////
//
// Bitmap
//

IDR_MAINFRAME           BITMAP  MOVEABLE PURE   "res\\Toolbar.bmp"

/////////////////////////////////////////////////////////////////////////
///////
//
// Toolbar
//

IDR_MAINFRAME TOOLBAR DISCARDABLE  16, 15
BEGIN
    BUTTON      ID_FILE_NEW
```

```
    BUTTON      ID_FILE_OPEN
    BUTTON      ID_FILE_SAVE
    SEPARATOR
    BUTTON      ID_EDIT_CUT
    BUTTON      ID_EDIT_COPY
    BUTTON      ID_EDIT_PASTE
    SEPARATOR
    BUTTON      ID_FILE_PRINT
    SEPARATOR
    BUTTON      ID_APP_ABOUT
END


////////////////////////////////////////////////////////////////////////
///////
//
// Menu
//

IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",                ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",            ID_FILE_OPEN
        MENUITEM SEPARATOR
        MENUITEM "P&rint Setup...",             ID_FILE_PRINT_SETUP
        MENUITEM SEPARATOR
        MENUITEM "Recent File",                 ID_FILE_MRU_FILE1, GRA
YED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About CSHB...",              ID_APP_ABOUT
    END
END

IDR_CSHBTYPE MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
```

```
        MENUITEM "Read Incident Data",          ID_FILE_READ_INCIDENT
        MENUITEM "Read Transmitted Data",       ID_FILE_READ_TRANSMITT
ED
        MENUITEM "&New\tCtrl+N",                ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",            ID_FILE_OPEN, INACTIVE
        MENUITEM "&Close",                      ID_FILE_CLOSE
        MENUITEM "&Save\tCtrl+S",               ID_FILE_SAVE
        MENUITEM "Save &As...",                 ID_FILE_SAVE_AS
        MENUITEM SEPARATOR
        MENUITEM "&Print...\tCtrl+P",           ID_FILE_PRINT, INACTIV
E
        MENUITEM "Print Pre&view",              ID_FILE_PRINT_PREVIEW
        , INACTIVE
        MENUITEM "P&rint Setup...",             ID_FILE_PRINT_SETUP
        , INACTIVE
        MENUITEM SEPARATOR
        MENUITEM "Recent File",                 ID_FILE_MRU_FILE1, GRA
YED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
//    POPUP "Manipulate"
  //  BEGIN
    //    MENUITEM "Seperate Waves",              ID_MANIPULATE_SEPERA
TEWAVE
      //  MENUITEM "View Indices",                ID_MANIPULATE_VIEWIN
DICES
    //END
    POPUP "Properties"
    BEGIN
        MENUITEM "&Bar Properties ...",         ID_BAR_PROPERTIES
        MENUITEM "&Sample Properties",          ID_SAMPLE_PROPERTIES
    END
    POPUP "Analyse"
    BEGIN
        MENUITEM "Calculate Results",           ID_ANALYSE_CALCULATERE
SULTS
    END
    POPUP "Post-Process"
    BEGIN
        MENUITEM "Draw Processed Data",         ID_DRAWPROCESSEDDATA
        MENUITEM "Export",                      ID_EXPORT
    END
    POPUP "Utilities"
    BEGIN
        MENUITEM "FFT",                         ID_FFT
        MENUITEM "Draw FFT",                    ID_DRAWFFT
        MENUITEM "Calculate Results",           ID_CALCULATERESULTS
```

```
    END
    POPUP "Propagation"
    BEGIN
        MENUITEM "Preprocess",                  ID_PROP_PREPROCESS
        MENUITEM "FFT",                         ID_FFT
        MENUITEM "Calculate",                   ID_PROP_CALC
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About CSHB...",              ID_APP_ABOUT
    END
    POPUP "&Window"
    BEGIN
        MENUITEM "&New Window",                 ID_WINDOW_NEW
        MENUITEM "&Cascade",                    ID_WINDOW_CASCADE
        MENUITEM "&Tile",                       ID_WINDOW_TILE_HORZ
        MENUITEM "&Arrange Icons",              ID_WINDOW_ARRANGE
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo\tCtrl+Z",               ID_EDIT_UNDO
        MENUITEM SEPARATOR
        MENUITEM "Cu&t\tCtrl+X",                ID_EDIT_CUT
        MENUITEM "&Copy\tCtrl+C",               ID_EDIT_COPY
        MENUITEM "&Paste\tCtrl+V",              ID_EDIT_PASTE
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
    END
END


//////////////////////////////////////////////////////////////////////
///////
//
// Accelerator
//

IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
    "N",            ID_FILE_NEW,            VIRTKEY, CONTROL
    "O",            ID_FILE_OPEN,           VIRTKEY, CONTROL
    "S",            ID_FILE_SAVE,           VIRTKEY, CONTROL
    "P",            ID_FILE_PRINT,          VIRTKEY, CONTROL
    "Z",            ID_EDIT_UNDO,           VIRTKEY, CONTROL
    "X",            ID_EDIT_CUT,            VIRTKEY, CONTROL
```

```
    "C",            ID_EDIT_COPY,           VIRTKEY, CONTROL
    "V",            ID_EDIT_PASTE,          VIRTKEY, CONTROL
    VK_BACK,        ID_EDIT_UNDO,           VIRTKEY, ALT
    VK_DELETE,      ID_EDIT_CUT,            VIRTKEY, SHIFT
    VK_INSERT,      ID_EDIT_COPY,           VIRTKEY, CONTROL
    VK_INSERT,      ID_EDIT_PASTE,          VIRTKEY, SHIFT
    VK_F6,          ID_NEXT_PANE,           VIRTKEY
    VK_F6,          ID_PREV_PANE,           VIRTKEY, SHIFT
END


//////////////////////////////////////////////////////////////////////
///////
//
// Dialog
//

IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 235, 79
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About CSHB"
FONT 8, "MS Sans Serif"
BEGIN
    ICON            IDR_MAINFRAME,IDC_STATIC,107,24,21,20
    LTEXT           "CSHB Version 1.0",IDC_STATIC,46,13,58,8,SS_NOPREF
IX
    LTEXT           "Copyright (C) 2001",IDC_STATIC,130,13,59,8
    DEFPUSHBUTTON   "OK",IDOK,92,58,50,14,WS_GROUP
END

IDD_SAMPLEDATA DIALOG DISCARDABLE  0, 0, 289, 290
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Sample Data"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDC_SAMPLETITLE,24,30,200,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_Do,53,205,62,14,ES_AUTOHSCROLL
    EDITTEXT        IDC_Lo,53,227,62,14,ES_AUTOHSCROLL
    EDITTEXT        IDC_Df,173,204,62,14,ES_AUTOHSCROLL
    EDITTEXT        IDC_Lf,173,228,62,14,ES_AUTOHSCROLL
    CONTROL         "DateTimePicker3",IDC_DATE,"SysDateTimePick32",
                    DTS_RIGHTALIGN | WS_TABSTOP,37,73,145,16
    DEFPUSHBUTTON   "Apply",IDOK,79,265,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,159,265,50,14
    GROUPBOX        "Sample Title",IDC_STATIC,17,15,218,33
    GROUPBOX        "Date of Test",IDC_STATIC,17,53,218,121
    GROUPBOX        "Sample Dimensions",IDC_STATIC,18,180,253,72
    LTEXT           "Do:",IDC_STATIC,36,208,12,8
    LTEXT           "Lo:",IDC_STATIC,36,230,12,8
```

```
    LTEXT           "Df:",IDC_STATIC,161,208,12,8
    LTEXT           "Lf:",IDC_STATIC,161,230,12,8
    LTEXT           "mm",IDC_STATIC,240,208,12,8
    LTEXT           "mm",IDC_STATIC,240,230,12,8
    LTEXT           "mm",IDC_STATIC,119,208,12,8
    LTEXT           "mm",IDC_STATIC,119,230,12,8
    GROUPBOX        "Original",IDC_STATIC,31,192,106,53
    GROUPBOX        "Final",IDC_STATIC,153,192,106,53
END

IDD_BARDATA DIALOG DISCARDABLE  0, 0, 418, 313
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Bar Properties"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "Apply",IDOK,150,292,50,14
    GROUPBOX        "Calibration File",IDC_STATIC,14,16,178,50
    EDITTEXT        IDC_I_BARNAME,21,84,136,12,ES_AUTOHSCROLL
    GROUPBOX        "Bar Name",IDC_STATIC,14,71,178,31
    EDITTEXT        IDC_I_CALIBRATIONNAME,21,28,136,12,ES_AUTOHSCROLL
    LTEXT           "Diameter:",IDC_STATIC,42,121,31,8
    GROUPBOX        "Geometry",IDC_STATIC,14,110,178,63
    EDITTEXT        IDC_I_BARDIAMETER,77,120,57,13,ES_AUTOHSCROLL
    LTEXT           "Length:",IDC_STATIC,48,139,25,8
    EDITTEXT        IDC_I_BARLENGTH,77,138,57,13,ES_AUTOHSCROLL
    GROUPBOX        "Material Properties",IDC_STATIC,14,182,178,63
    LTEXT           "Density:",IDC_STATIC,47,196,26,8
    LTEXT           "Elastic Modulus:",IDC_STATIC,21,213,52,8
    LTEXT           "Poisson's Ratio:",IDC_STATIC,22,230,51,8
    EDITTEXT        IDC_I_BARDENSITY,77,193,57,13,ES_AUTOHSCROLL
    EDITTEXT        IDC_I_BAREMODULUS,77,210,57,13,ES_AUTOHSCROLL
    EDITTEXT        IDC_I_BARPRATIO,77,227,57,13,ES_AUTOHSCROLL
    LTEXT           "kg/m^3",IDC_STATIC,138,196,26,8
    LTEXT           "N/m^2",IDC_STATIC,138,212,23,8
    LTEXT           "m",IDC_STATIC,138,121,8,8
    LTEXT           "m",IDC_STATIC,138,139,8,8
    LTEXT           "Diameter:",IDC_STATIC,253,121,31,8
    GROUPBOX        "Geometry",IDC_STATIC,222,111,178,63
    EDITTEXT        IDC_T_BARDIAMETER,285,120,57,13,ES_AUTOHSCROLL
    LTEXT           "Length:",IDC_STATIC,259,139,25,8
    EDITTEXT        IDC_T_BARLENGTH,285,138,57,13,ES_AUTOHSCROLL
    GROUPBOX        "Material Properties",IDC_STATIC,222,182,177,63
    LTEXT           "Density:",IDC_STATIC,258,196,26,8
    LTEXT           "Elastic Modulus:",IDC_STATIC,232,213,52,8
    LTEXT           "Poisson's Ratio:",IDC_STATIC,233,230,51,8
    EDITTEXT        IDC_T_BARDENSITY,285,193,57,13,ES_AUTOHSCROLL
    EDITTEXT        IDC_T_BAREMODULUS,285,210,57,13,ES_AUTOHSCROLL
    EDITTEXT        IDC_T_BARPRATIO,285,227,57,13,ES_AUTOHSCROLL
```

```
    LTEXT           "kg/m^3",IDC_STATIC,345,196,26,8
    LTEXT           "N/m^2",IDC_STATIC,345,213,23,8
    LTEXT           "m",IDC_STATIC,345,121,8,8
    LTEXT           "m",IDC_STATIC,345,139,8,8
    GROUPBOX        "Incident Bar",IDC_STATIC,7,7,196,278
    GROUPBOX        "Transmitted Bar",IDC_STATIC,215,7,196,278
    GROUPBOX        "Calibration File",IDC_STATIC,222,16,177,50
    EDITTEXT        IDC_T_BARNAME,232,84,136,12,ES_AUTOHSCROLL
    GROUPBOX        "Bar Name",IDC_STATIC,222,71,177,31
    EDITTEXT        IDC_T_CALIBRATIONNAME,232,28,136,12,ES_AUTOHSCROLL
    PUSHBUTTON      "Browse",IDC_BROWSE_INCIDENT,71,47,50,14
    GROUPBOX        "Calibration Parameters",IDC_STATIC,14,248,178,26
    LTEXT           "Strain=",IDC_STATIC,41,260,23,8
    LTEXT           "mV+",IDC_STATIC,107,260,18,8
    EDITTEXT        IDC_I_SLOPE,65,258,38,12,ES_AUTOHSCROLL | WS_DISAB
LED
    EDITTEXT        IDC_I_INTERCEPT,127,258,30,12,ES_AUTOHSCROLL |
                    WS_DISABLED
    PUSHBUTTON      "Browse",IDC_BROWSE_TRANSMITTED,285,47,50,14
    GROUPBOX        "Calibration Parameters",IDC_STATIC,222,248,177,26
    LTEXT           "Strain=",IDC_STATIC,244,260,23,8
    LTEXT           "mV+",IDC_STATIC,311,260,18,8
    EDITTEXT        IDC_T_SLOPE,268,258,38,12,ES_AUTOHSCROLL | WS_DISA
BLED
    EDITTEXT        IDC_T_INTERCEPT,330,258,30,12,ES_AUTOHSCROLL |
                    WS_DISABLED
    PUSHBUTTON      "Cancel",IDCANCEL,218,292,50,14
    LTEXT           " Gauge to Interface Distance:",IDC_STATIC,26,158,
94,8
    EDITTEXT        IDC_I_DISTANCE,122,156,53,13,ES_AUTOHSCROLL
    LTEXT           "m",IDC_STATIC,177,158,8,8
    LTEXT           " Gauge to Interface Distance:",IDC_STATIC,230,158
,94,8
    EDITTEXT        IDC_T_DISTANCE,326,156,53,13,ES_AUTOHSCROLL
    LTEXT           "m",IDC_STATIC,381,158,8,8
END

IDD_EXPORT DIALOG DISCARDABLE  0, 0, 278, 242
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Export"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "Export",IDOK,69,221,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,151,221,50,14
    GROUPBOX        "Items to Export",IDC_STATIC,24,14,225,196
    CONTROL         "Engineering Stress",IDC_EXP_ENGSTRESS,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,152,40,75,10
    CONTROL         "True Stress",IDC_EXP_TRUESTRESS,"Button",
```

```
                    BS_AUTOCHECKBOX | WS_TABSTOP,152,112,52,10
    CONTROL         "Engineering Strain",IDC_EXP_ENGSTRAIN,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,152,64,73,10
    CONTROL         "True Strain",IDC_EXP_TRUESTRAIN,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,152,136,51,10
    CONTROL         "Strain Rate",IDC_EXP_STRAINRATE,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,152,88,51,10
    CONTROL         "Incident Waveform",IDC_EXP_INCIDENT_T,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,36,40,76,10
    CONTROL         "Tranmitted Waveform",IDC_EXP_TRANSMITTED_T,"Butto
n",
                    BS_AUTOCHECKBOX | WS_TABSTOP,36,54,84,10
    CONTROL         "Reflected Waveform",IDC_EXP_REFLECTED_T,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,36,67,81,10
    CONTROL         "Incident Waveform ",IDC_EXP_INCIDENT_F,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,36,111,78,10
    CONTROL         "Transmitted Waveform",IDC_EXP_TRANSMITTED_F,"Butt
on",
                    BS_AUTOCHECKBOX | WS_TABSTOP,36,124,87,10
    CONTROL         "Reflected Waveform",IDC_EXP_REFLECTED_F,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,36,137,81,10
    GROUPBOX        "Frequency Domain",IDC_STATIC,30,96,104,63
    GROUPBOX        "Time Domain",IDC_STATIC,30,24,104,63
    GROUPBOX        "Post Processed Results",IDC_STATIC,146,24,97,135
    GROUPBOX        "Values at Interface",IDC_STATIC,30,163,213,42
    CONTROL         "Incident Velocity",IDC_INCIDENT_V,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,36,174,78,10
    CONTROL         "Transmitted Velocity",IDC_TRANSMITTED_V,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,36,187,78,10
    CONTROL         "Incident Force",IDC_INCIDENT_F,"Button",BS_AUTOCH
ECKBOX |
                    WS_TABSTOP,152,174,78,10
    CONTROL         "Transmitted Force",IDC_TRANSMITTED_F,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,152,187,78,10
END

IDD_ANALYSEDLG DIALOG DISCARDABLE  0, 0, 334, 170
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Analyse Data"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "Analyse",IDOK,95,149,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,176,149,50,14
    GROUPBOX        "Sample Data",IDC_STATIC,7,14,109,118
    LTEXT           "Name:",IDC_STATIC,15,31,22,8
    EDITTEXT        IDC_SAMP_NAME,38,30,73,12,ES_AUTOHSCROLL | WS_DISA
BLED
    LTEXT           "Do:",IDC_STATIC,15,51,22,8
```

```
    EDITTEXT        IDC_SAMP_DO,38,50,34,12,ES_AUTOHSCROLL | WS_DISABL
ED
    LTEXT           "Lo:",IDC_STATIC,15,71,22,8
    EDITTEXT        IDC_SAMP_LO,38,70,34,12,ES_AUTOHSCROLL | WS_DISABL
ED
    LTEXT           "Df:",IDC_STATIC,15,91,22,8
    EDITTEXT        IDC_SAMP_DF,38,90,34,12,ES_AUTOHSCROLL | WS_DISABL
ED
    LTEXT           "Lf:",IDC_STATIC,15,111,22,8
    EDITTEXT        IDC_SAMP_LF,38,110,34,12,ES_AUTOHSCROLL | WS_DISAB
LED
    GROUPBOX        "Raw Data Files",IDC_STATIC,124,15,203,58
    LTEXT           "Incident Bar:",IDC_STATIC,129,31,41,8
    EDITTEXT        IDC_DATA_I,181,29,99,13,ES_AUTOHSCROLL | WS_DISABL
ED
    LTEXT           "Transmitted Bar:",IDC_STATIC,129,52,52,8
    EDITTEXT        IDC_DATA_T,181,50,99,13,ES_AUTOHSCROLL | WS_DISABL
ED
    LTEXT           "Incident Bar:",IDC_STATIC,129,99,41,8
    EDITTEXT        IDC_CALIB_I,181,96,99,13,ES_AUTOHSCROLL | WS_DISAB
LED
    LTEXT           "Transmitted Bar:",IDC_STATIC,129,117,52,8
    EDITTEXT        IDC_CALIB_T,181,115,99,13,ES_AUTOHSCROLL | WS_DISA
BLED
    GROUPBOX        "Calibration Files",IDC_STATIC,124,74,203,58
    LTEXT           "mm",IDC_STATIC,80,51,12,8
    LTEXT           "mm",IDC_STATIC,80,71,12,8
    LTEXT           "mm",IDC_STATIC,80,91,12,8
    LTEXT           "mm",IDC_STATIC,80,111,12,8
    CONTROL         "Include Dispersion Analysis",IDC_INCDISP,"Button"
,
                    BS_AUTORADIOBUTTON | WS_GROUP,59,138,101,10
    CONTROL         "Conventional Anaylsis",IDC_CONVANALYSIS,"Button",
                    BS_AUTORADIOBUTTON,189,138,85,10
    CONTROL         "",IDC_IDISPAV,"Button",BS_AUTOCHECKBOX | WS_DISAB
LED |
                    WS_TABSTOP,293,98,16,8
    CONTROL         "",IDC_TDISPAV,"Button",BS_AUTOCHECKBOX | WS_DISAB
LED |
                    WS_TABSTOP,293,116,11,9
    LTEXT           "Dispersion",IDC_STATIC,283,80,34,8
    LTEXT           "Available",IDC_STATIC,285,89,30,8
END

IDD_SEPARATEWAVEVALUES DIALOG DISCARDABLE  0, 0, 283, 103
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Separate Wave Values"
FONT 8, "MS Sans Serif"
```

```
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,55,82,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,183,82,50,14
    PUSHBUTTON      "Default",IDC_Default,119,82,50,14
    LTEXT           "Primary Edge Trigger Threshold:",IDC_STATIC,19,32
,102,8
    EDITTEXT        IDC_THRESHOLD1,131,30,31,13,ES_AUTOHSCROLL
    LTEXT           "of maximum value",IDC_STATIC,165,32,57,8
    LTEXT           "Secondary Edge Trigger Threshold:",IDC_STATIC,8,4
8,113,
                    8
    EDITTEXT        IDC_THRESHOLD2,131,47,31,13,ES_AUTOHSCROLL
    LTEXT           "of first edge",IDC_STATIC,165,48,57,8
    LTEXT           "Filter:",IDC_STATIC,103,64,18,8
    EDITTEXT        IDC_TOLERANCE,131,63,31,13,ES_AUTOHSCROLL
    LTEXT           "of maximum value",IDC_STATIC,167,64,57,8
    CONTROL         "Separate Transmitted",IDC_SEPARATETRANSMITTED,"Bu
tton",
                    BS_AUTOCHECKBOX | WS_TABSTOP,157,14,83,10
    CONTROL         "Separate Incident",IDC_SEPARATEINCIDENT,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,43,14,72,10
END

IDD_INDICES DIALOG DISCARDABLE  0, 0, 228, 103
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Starting and End Points"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDC_IS,65,28,32,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_IE,134,28,32,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_RS,65,42,32,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_RE,134,42,32,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_TS,65,56,32,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_TE,134,56,32,12,ES_AUTOHSCROLL
    DEFPUSHBUTTON   "OK",IDOK,58,82,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,118,82,50,14
    GROUPBOX        "Starting Index",IDC_STATIC,53,16,61,58
    GROUPBOX        "End Index",IDC_STATIC,121,16,61,58
    LTEXT           "Incident",IDC_STATIC,14,29,26,8
    LTEXT           "Reflected",IDC_STATIC,14,43,32,8
    LTEXT           "Transmitted",IDC_STATIC,14,57,38,8
END

IDD_PROPCOEFF DIALOG DISCARDABLE  0, 0, 205, 98
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Propagation Coefficient"
FONT 8, "MS Sans Serif"
BEGIN
```

```
    DEFPUSHBUTTON   "OK",IDOK,48,77,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,108,77,50,14
    GROUPBOX        "Files",IDC_STATIC,7,7,183,48
    LTEXT           "Data File:",IDC_STATIC,32,20,31,8
    EDITTEXT        IDC_DATA_I,66,17,99,13,ES_AUTOHSCROLL | WS_DISABLE
D
    LTEXT           "Calibration File:",IDC_STATIC,15,34,48,8
    EDITTEXT        IDC_CALIB_I,66,33,99,13,ES_AUTOHSCROLL | WS_DISABL
ED
    CONTROL         "Update Calibration File",IDC_UPDATEFILE,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,7,62,87,10
    CONTROL         "Export to File",IDC_EXPFILE,"Button",BS_AUTOCHECK
BOX |
                    WS_TABSTOP,111,62,78,9
END

IDD_FFTUTILITY DIALOG DISCARDABLE  0, 0, 219, 118
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "FFT Utility"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "Browse",IDC_BROWSE_IN,143,21,50,14
    PUSHBUTTON      "Browse",IDC_BROWSE_OUT,143,45,50,14
    CONTROL         "Forward Transform",IDC_FFT,"Button",BS_AUTORADIOB
UTTON |
                    WS_GROUP,25,63,75,10
    CONTROL         "Inverse Transform",IDC_IFFT,"Button",BS_AUTORADIO
BUTTON,
                    25,76,73,10
    DEFPUSHBUTTON   "OK",IDOK,52,90,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,116,90,50,14
    EDITTEXT        IDC_INPUTNAME,51,21,85,13,ES_AUTOHSCROLL
    LTEXT           "Input File:",IDC_STATIC,18,23,32,8
    EDITTEXT        IDC_OUTPUTNAME,51,45,85,13,ES_AUTOHSCROLL
    LTEXT           "Output File:",IDC_STATIC,13,47,37,8
END

IDD_INPUTFILE DIALOG DISCARDABLE  0, 0, 230, 68
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Input File"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,60,47,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,120,47,50,14
    LTEXT           "File:",IDC_STATIC,14,16,14,8
    EDITTEXT        IDC_FILENAME,30,14,123,13,ES_AUTOHSCROLL
    CONTROL         "Binary",IDC_BINARY,"Button",BS_AUTORADIOBUTTON |
                    WS_GROUP,28,32,35,10
```

```
    CONTROL         "Ascii",IDC_ASCII,"Button",BS_AUTORADIOBUTTON,85,3
2,31,
                    10
    PUSHBUTTON      "Browse",IDC_BROWSE,162,14,50,14
    CONTROL         "Amplitude Shift",IDC_AMPSHIFT,"Button",BS_AUTOCHE
CKBOX |
                    WS_TABSTOP,138,32,63,10
END

IDD_DIALOG1 DIALOG DISCARDABLE  0, 0, 186, 79
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,32,58,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,103,58,50,14
    LTEXT           "Number of Points to Use:",IDC_STATIC,20,17,80,8
    EDITTEXT        IDC_,61,33,73,12,ES_AUTOHSCROLL
END

IDD_AMP_PROP DIALOG DISCARDABLE  0, 0, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Amplitude Shift"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,33,74,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,102,74,50,14
    LTEXT           "Number of point to use as zero:",IDC_STATIC,21,17
,112,8
    EDITTEXT        IDC_POINTS,57,37,70,13,ES_AUTOHSCROLL
END

IDD_NICOLETCONVERT_DIALOG DIALOGEX 0, 0, 372, 106
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_APPWINDOW
CAPTION "Nicolet Conversion Utility"
FONT 8, "MS Sans Serif", 0, 0, 0x1
BEGIN
    PUSHBUTTON      "Browse",IDC_INPUTBROWSE,308,15,50,14
    PUSHBUTTON      "Browse",IDC_OUPUTBROWSE,308,43,50,14
    CONTROL         "Include Header",IDC_INCHEADER,"Button",BS_AUTOCHE
CKBOX |
                    WS_TABSTOP,103,65,88,9
    CONTROL         "Amplitude Shift",IDC_AMPLTITUDE,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,205,65,63,10
    DEFPUSHBUTTON   "Convert",IDCONVERT,120,85,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,202,85,50,14
    LTEXT           "Input File:",IDC_STATIC,22,19,32,8
```

```
    EDITTEXT        IDC_INPUTFILE,57,16,242,13,ES_AUTOHSCROLL
    LTEXT           "Output File:",IDC_STATIC,17,46,37,8
    EDITTEXT        IDC_OUTPUTFILE,57,44,242,13,ES_AUTOHSCROLL
END


#ifndef _MAC
/////////////////////////////////////////////////////////////////////////////
///////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904B0"
        BEGIN
            VALUE "CompanyName", "\0"
            VALUE "FileDescription", "CSHB MFC Application\0"
            VALUE "FileVersion", "1, 0, 0, 1\0"
            VALUE "InternalName", "CSHB\0"
            VALUE "LegalCopyright", "Copyright (C) 2001\0"
            VALUE "LegalTrademarks", "\0"
            VALUE "OriginalFilename", "CSHB.EXE\0"
            VALUE "ProductName", "CSHB Application\0"
            VALUE "ProductVersion", "1, 0, 0, 1\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END

#endif    // !_MAC
```

```
//////////////////////////////////////////////////////////////////////
///////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 228
        TOPMARGIN, 7
        BOTTOMMARGIN, 72
    END

    IDD_SAMPLEDATA, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 282
        VERTGUIDE, 24
        VERTGUIDE, 115
        VERTGUIDE, 121
        VERTGUIDE, 235
        VERTGUIDE, 240
        TOPMARGIN, 7
        BOTTOMMARGIN, 283
        HORZGUIDE, 233
        HORZGUIDE, 252
        HORZGUIDE, 256
        HORZGUIDE, 259
    END

    IDD_BARDATA, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 411
        VERTGUIDE, 14
        VERTGUIDE, 21
        VERTGUIDE, 41
        VERTGUIDE, 73
        VERTGUIDE, 96
        VERTGUIDE, 136
        VERTGUIDE, 192
        VERTGUIDE, 222
```

```
    VERTGUIDE, 399
    TOPMARGIN, 7
    BOTTOMMARGIN, 306
    HORZGUIDE, 239
END

IDD_EXPORT, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 271
    VERTGUIDE, 243
    TOPMARGIN, 7
    BOTTOMMARGIN, 235
END

IDD_ANALYSEDLG, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 327
    VERTGUIDE, 124
    TOPMARGIN, 7
    BOTTOMMARGIN, 163
    HORZGUIDE, 132
END

IDD_SEPARATEWAVEVALUES, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 276
    TOPMARGIN, 7
    BOTTOMMARGIN, 96
END

IDD_INDICES, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 221
    TOPMARGIN, 7
    BOTTOMMARGIN, 96
END

IDD_PROPCOEFF, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 198
    TOPMARGIN, 7
    BOTTOMMARGIN, 91
END
```

```
    IDD_FFTUTILITY, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 212
        TOPMARGIN, 7
        BOTTOMMARGIN, 111
    END

    IDD_INPUTFILE, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 223
        TOPMARGIN, 7
        BOTTOMMARGIN, 61
    END

    IDD_DIALOG1, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 179
        TOPMARGIN, 7
        BOTTOMMARGIN, 72
    END

    IDD_AMP_PROP, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 179
        TOPMARGIN, 7
        BOTTOMMARGIN, 88
    END
END
#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////
///////
//
// String Table
//

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    IDR_MAINFRAME           "CSHB"
    IDR_CSHBTYPE            "\nCSHB\nCSHB\n\n\nCSHB.Document\nCSHB Doc
ument"
END
```

```
STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    AFX_IDS_APP_TITLE       "CSHB"
    AFX_IDS_IDLEMESSAGE     "Ready"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_INDICATOR_EXT        "EXT"
    ID_INDICATOR_CAPS       "CAP"
    ID_INDICATOR_NUM        "NUM"
    ID_INDICATOR_SCRL       "SCRL"
    ID_INDICATOR_OVR        "OVR"
    ID_INDICATOR_REC        "REC"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_NEW             "Create a new document\nNew"
    ID_FILE_OPEN            "Open an existing document\nOpen"
    ID_FILE_CLOSE           "Close the active document\nClose"
    ID_FILE_SAVE            "Save the active document\nSave"
    ID_FILE_SAVE_AS         "Save the active document with a new name\
nSave As"
    ID_FILE_PAGE_SETUP      "Change the printing options\nPage Setup"
    ID_FILE_PRINT_SETUP     "Change the printer and printing options\n
Print Setup"
    ID_FILE_PRINT           "Print the active document\nPrint"
    ID_FILE_PRINT_PREVIEW   "Display full pages\nPrint Preview"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_APP_ABOUT            "Display program information, version numb
er and
copyright\nAbout"
    ID_APP_EXIT             "Quit the application; prompts to save doc
uments\nExit"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_MRU_FILE1       "Open this document"
    ID_FILE_MRU_FILE2       "Open this document"
    ID_FILE_MRU_FILE3       "Open this document"
    ID_FILE_MRU_FILE4       "Open this document"
    ID_FILE_MRU_FILE5       "Open this document"
```

```
    ID_FILE_MRU_FILE6       "Open this document"
    ID_FILE_MRU_FILE7       "Open this document"
    ID_FILE_MRU_FILE8       "Open this document"
    ID_FILE_MRU_FILE9       "Open this document"
    ID_FILE_MRU_FILE10      "Open this document"
    ID_FILE_MRU_FILE11      "Open this document"
    ID_FILE_MRU_FILE12      "Open this document"
    ID_FILE_MRU_FILE13      "Open this document"
    ID_FILE_MRU_FILE14      "Open this document"
    ID_FILE_MRU_FILE15      "Open this document"
    ID_FILE_MRU_FILE16      "Open this document"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_NEXT_PANE            "Switch to the next window pane\nNext Pane
"
    ID_PREV_PANE            "Switch back to the previous window pane\n
Previous Pane"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_WINDOW_NEW           "Open another window for the active docume
nt\nNew Window"
    ID_WINDOW_ARRANGE       "Arrange icons at the bottom of the window
\nArrange Icons"
    ID_WINDOW_CASCADE       "Arrange windows so they overlap\nCascade
Windows"
    ID_WINDOW_TILE_HORZ     "Arrange windows as non-overlapping tiles\
nTile Windows"
    ID_WINDOW_TILE_VERT     "Arrange windows as non-overlapping tiles\
nTile Windows"
    ID_WINDOW_SPLIT         "Split the active window into panes\nSplit
"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_EDIT_CLEAR           "Erase the selection\nErase"
    ID_EDIT_CLEAR_ALL       "Erase everything\nErase All"
    ID_EDIT_COPY            "Copy the selection and put it on the Clip
board\nCopy"
    ID_EDIT_CUT             "Cut the selection and put it on the Clipb
oard\nCut"
    ID_EDIT_FIND            "Find the specified text\nFind"
    ID_EDIT_PASTE           "Insert Clipboard contents\nPaste"
    ID_EDIT_REPEAT          "Repeat the last action\nRepeat"
```

```
    ID_EDIT_REPLACE         "Replace specific text with different text
\nReplace"
    ID_EDIT_SELECT_ALL      "Select the entire document\nSelect All"
    ID_EDIT_UNDO            "Undo the last action\nUndo"
    ID_EDIT_REDO            "Redo the previously undone action\nRedo"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_VIEW_TOOLBAR         "Show or hide the toolbar\nToggle ToolBar"
    ID_VIEW_STATUS_BAR      "Show or hide the status bar\nToggle Statu
sBar"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCSIZE          "Change the window size"
    AFX_IDS_SCMOVE          "Change the window position"
    AFX_IDS_SCMINIMIZE      "Reduce the window to an icon"
    AFX_IDS_SCMAXIMIZE      "Enlarge the window to full size"
    AFX_IDS_SCNEXTWINDOW    "Switch to the next document window"
    AFX_IDS_SCPREVWINDOW    "Switch to the previous document window"
    AFX_IDS_SCCLOSE         "Close the active window and prompts to sa
ve the documents"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCRESTORE       "Restore the window to normal size"
    AFX_IDS_SCTASKLIST      "Activate Task List"
    AFX_IDS_MDICHILD        "Activate this window"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_PREVIEW_CLOSE   "Close print preview mode\nCancel Preview"
END

#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////
///////



#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////
///////
//
```

```
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif //_WIN32
#include "res\CSHB.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#include "afxprint.rc"       // printing/print preview resources
#endif

////////////////////////////////////////////////////////////////////////
///////
#endif    // not APSTUDIO_INVOKED
```

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////
//////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////
//////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////
//////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

#ifdef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////
//////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
    "#define _AFX_NO_OLE_RESOURCES\r\n"
```

```
    "#define _AFX_NO_TRACKER_RESOURCES\r\n"
    "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
    "\r\n"
    "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\r\n"
    "#ifdef _WIN32\r\n"
    "LANGUAGE 9, 1\r\n"
    "#pragma code_page(1252)\r\n"
    "#endif //_WIN32\r\n"
    "#include ""res\\CSHB.rc2""  // non-Microsoft Visual C++ edited re
sources\r\n"
    "#include ""afxres.rc""         // Standard components\r\n"
    "#include ""afxprint.rc""       // printing/print preview resource
s\r\n"
    "#endif\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED


/////////////////////////////////////////////////////////////////////////
///////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON    DISCARDABLE     "res\\CSHB.ico"
IDR_CSHBTYPE            ICON    DISCARDABLE     "res\\CSHBDoc.ico"

/////////////////////////////////////////////////////////////////////////
///////
//
// Bitmap
//

IDR_MAINFRAME           BITMAP  MOVEABLE PURE   "res\\Toolbar.bmp"

/////////////////////////////////////////////////////////////////////////
///////
//
// Toolbar
//

IDR_MAINFRAME TOOLBAR DISCARDABLE  15, 15
BEGIN
    BUTTON      ID_FILE_NEWSAMPLE
```

```
    BUTTON      ID_FILE_READ_INCIDENT
    BUTTON      ID_FILE_READ_TRANSMITTED
    BUTTON      ID_PRE_SEPERATEWAVE
    BUTTON      ID_SAMPLE_PROPERTIES
    SEPARATOR
    BUTTON      ID_ANALYSE_CALCULATERESULTS
    SEPARATOR
    BUTTON      ID_FILE_NEW
    SEPARATOR
    BUTTON      ID_APP_ABOUT
END


//////////////////////////////////////////////////////////////////////
///////
//
// Menu
//

IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",                ID_FILE_NEW
        MENUITEM SEPARATOR
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About CSHB...",              ID_APP_ABOUT
    END
END

IDR_CSHBTYPE MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "New Sample",                  ID_FILE_NEWSAMPLE
        MENUITEM "Read Incident Data",          ID_FILE_READ_INCIDENT
        MENUITEM "Read Transmitted Data",       ID_FILE_READ_TRANSMITT
ED
        MENUITEM SEPARATOR
```

```
        MENUITEM "&New\tCtrl+N",                ID_FILE_NEW
        MENUITEM "&Close",                      ID_FILE_CLOSE
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
    POPUP "&PreProcess"
    BEGIN
        MENUITEM "&Seperate Waves",             ID_PRE_SEPERATEWAVE
        MENUITEM "&View Indices",               ID_PRE_VIEWINDICES, IN
ACTIVE
        POPUP "&Properties"
        BEGIN
            MENUITEM "&Sample Properties",          ID_SAMPLE_PROPERTI
ES
            MENUITEM "&Bar Properties",             ID_BAR_PROPERTIES
        END
    END
    POPUP "&Analyse"
    BEGIN
        MENUITEM "Calculate Results",           ID_ANALYSE_CALCULATERE
SULTS
        MENUITEM SEPARATOR
        MENUITEM SEPARATOR
        MENUITEM "Propagation Coefficient Calculation", ID_ANALYSE_PRO
PCALC
    END
    POPUP "P&ost-Process"
    BEGIN
        MENUITEM "Export",                      ID_EXPORT
        MENUITEM SEPARATOR
        MENUITEM SEPARATOR
        MENUITEM "View Displacement",           ID_POST_VWDISP
        MENUITEM "View Velocity",               ID_POST_VIEWVEL
        MENUITEM "View Force",                  ID_POST_VWFOR
        MENUITEM "View Strain",                 ID_POST_VWSTRN, CHECKE
D
    END
    POPUP "&Utilities"
    BEGIN
        MENUITEM "FFT Conversion",              ID_UTILITIES_FFT
        MENUITEM "Propagate Waves",             ID_UTILITIES_PROPAGATE
WAVE
        MENUITEM "Nicolet File Conversion",     ID_UTILITIES_FILECONVE
RSION
    END
    POPUP "&Window"
    BEGIN
        MENUITEM "&New Window",                 ID_WINDOW_NEW
```

```
        MENUITEM "&Cascade",                    ID_WINDOW_CASCADE
        MENUITEM "&Tile",                       ID_WINDOW_TILE_HORZ
        MENUITEM "&Arrange Icons",              ID_WINDOW_ARRANGE
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About CSHB...",              ID_APP_ABOUT
    END
END


//////////////////////////////////////////////////////////////////////
///////
//
// Accelerator
//

IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
    "N",            ID_FILE_NEW,            VIRTKEY, CONTROL
    "O",            ID_FILE_OPEN,           VIRTKEY, CONTROL
    "S",            ID_FILE_SAVE,           VIRTKEY, CONTROL
    "P",            ID_FILE_PRINT,          VIRTKEY, CONTROL
    "Z",            ID_EDIT_UNDO,           VIRTKEY, CONTROL
    "X",            ID_EDIT_CUT,            VIRTKEY, CONTROL
    "C",            ID_EDIT_COPY,           VIRTKEY, CONTROL
    "V",            ID_EDIT_PASTE,          VIRTKEY, CONTROL
    VK_BACK,        ID_EDIT_UNDO,           VIRTKEY, ALT
    VK_DELETE,      ID_EDIT_CUT,            VIRTKEY, SHIFT
    VK_INSERT,      ID_EDIT_COPY,           VIRTKEY, CONTROL
    VK_INSERT,      ID_EDIT_PASTE,          VIRTKEY, SHIFT
    VK_F6,          ID_NEXT_PANE,           VIRTKEY
    VK_F6,          ID_PREV_PANE,           VIRTKEY, SHIFT
END


//////////////////////////////////////////////////////////////////////
///////
//
// Dialog
//

IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 235, 79
```

```
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About CSHB"
FONT 8, "MS Sans Serif"
BEGIN
    ICON            IDR_MAINFRAME,IDC_STATIC,107,24,21,20
    LTEXT           "CSHB Version 1.0",IDC_STATIC,46,13,58,8,SS_NOPREF
IX
    LTEXT           "Copyright (C) 2001",IDC_STATIC,130,13,59,8
    DEFPUSHBUTTON   "OK",IDOK,92,58,50,14,WS_GROUP
END

IDD_SAMPLEDATA DIALOG DISCARDABLE  0, 0, 242, 287
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Sample Data"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDC_SAMPLETITLE,24,30,200,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_Do,43,159,38,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_Lo,43,181,38,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_Df,156,159,38,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_Lf,156,181,38,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_VELOCITY,64,223,38,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_PRESSURE,177,222,38,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_STRIKERL,64,239,38,12,ES_AUTOHSCROLL
    CONTROL         "DateTimePicker3",IDC_DATE,"SysDateTimePick32",
                    DTS_RIGHTALIGN | DTS_LONGDATEFORMAT | WS_TABSTOP,8
1,99,
                    145,16
    DEFPUSHBUTTON   "Apply",IDOK,55,263,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,135,263,50,14
    GROUPBOX        "Sample Title",IDC_STATIC,7,15,228,33
    GROUPBOX        "Date of Test",IDC_STATIC,7,54,228,72
    GROUPBOX        "Sample Dimensions",IDC_STATIC,7,134,228,72
    LTEXT           "Do:",IDC_STATIC,29,162,12,8
    LTEXT           "Lo:",IDC_STATIC,29,183,12,8
    LTEXT           "Df:",IDC_STATIC,144,162,12,8
    LTEXT           "Lf:",IDC_STATIC,144,183,12,8
    LTEXT           "mm",IDC_STATIC,198,162,12,8
    LTEXT           "mm",IDC_STATIC,198,184,12,8
    LTEXT           "mm",IDC_STATIC,85,162,12,8
    LTEXT           "mm",IDC_STATIC,85,184,12,8
    GROUPBOX        "Original",IDC_STATIC,21,146,82,53
    GROUPBOX        "Final",IDC_STATIC,136,146,82,53
    LTEXT           "Sample Date:",IDC_STATIC,24,103,44,8
    EDITTEXT        IDC_INCDATE,115,65,113,12,ES_AUTOHSCROLL | ES_READ
ONLY
    LTEXT           "Incident Data Date:",IDC_STATIC,24,68,63,8
    LTEXT           "Transmitted Data Date:",IDC_STATIC,24,82,74,8
```

```
    EDITTEXT        IDC_TRANSDATE,115,81,113,12,ES_AUTOHSCROLL | ES_RE
ADONLY
    GROUPBOX        "Input Conditions",IDC_STATIC,7,210,228,46
    LTEXT           "Striker Velocity:",IDC_STATIC,13,224,50,8
    LTEXT           "Gun Pressure:",IDC_STATIC,131,224,46,8
    LTEXT           "m/s",IDC_STATIC,107,224,13,8
    LTEXT           "psi",IDC_STATIC,219,224,10,8
    LTEXT           "Striker Length:",IDC_STATIC,13,240,47,8
    LTEXT           "m",IDC_STATIC,107,240,8,8
END

IDD_BARDATA DIALOG DISCARDABLE  0, 0, 418, 364
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Bar Properties"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "Apply",IDOK,183,343,50,14
    GROUPBOX        "Calibration File",IDC_STATIC,14,16,178,50
    EDITTEXT        IDC_I_BARNAME,21,84,164,12,ES_AUTOHSCROLL | ES_REA
DONLY
    GROUPBOX        "Bar Name",IDC_STATIC,14,71,178,31
    EDITTEXT        IDC_I_CALIBRATIONNAME,21,28,164,12,ES_AUTOHSCROLL
|
                    ES_READONLY
    LTEXT           "Diameter:",IDC_STATIC,42,121,31,8
    GROUPBOX        "Geometry",IDC_STATIC,14,110,178,63
    EDITTEXT        IDC_I_BARDIAMETER,77,120,57,13,ES_AUTOHSCROLL |
                    ES_READONLY
    LTEXT           "Length:",IDC_STATIC,48,139,25,8
    EDITTEXT        IDC_I_BARLENGTH,77,138,57,13,ES_AUTOHSCROLL |
                    ES_READONLY
    GROUPBOX        "Material Properties",IDC_STATIC,14,182,178,63
    LTEXT           "Density:",IDC_STATIC,47,196,26,8
    LTEXT           "Elastic Modulus:",IDC_STATIC,21,213,52,8
    LTEXT           "Poisson's Ratio:",IDC_STATIC,22,230,51,8
    EDITTEXT        IDC_I_BARDENSITY,77,193,57,13,ES_AUTOHSCROLL |
                    ES_READONLY
    EDITTEXT        IDC_I_BAREMODULUS,77,210,57,13,ES_AUTOHSCROLL |
                    ES_READONLY
    EDITTEXT        IDC_I_BARPRATIO,77,227,57,13,ES_AUTOHSCROLL |
                    ES_READONLY
    LTEXT           "kg/m^3",IDC_STATIC,138,196,26,8
    LTEXT           "N/m^2",IDC_STATIC,138,212,23,8
    LTEXT           "m",IDC_STATIC,138,121,8,8
    LTEXT           "m",IDC_STATIC,138,139,8,8
    LTEXT           "Diameter:",IDC_STATIC,253,121,31,8
    GROUPBOX        "Geometry",IDC_STATIC,222,111,178,63
    EDITTEXT        IDC_T_BARDIAMETER,285,120,57,13,ES_AUTOHSCROLL |
```

```
                    ES_READONLY
    LTEXT           "Length:",IDC_STATIC,259,139,25,8
    EDITTEXT        IDC_T_BARLENGTH,285,138,57,13,ES_AUTOHSCROLL |
                    ES_READONLY
    GROUPBOX        "Material Properties",IDC_STATIC,222,182,177,63
    LTEXT           "Density:",IDC_STATIC,258,196,26,8
    LTEXT           "Elastic Modulus:",IDC_STATIC,232,213,52,8
    LTEXT           "Poisson's Ratio:",IDC_STATIC,233,230,51,8
    EDITTEXT        IDC_T_BARDENSITY,285,193,57,13,ES_AUTOHSCROLL |
                    ES_READONLY
    EDITTEXT        IDC_T_BAREMODULUS,285,210,57,13,ES_AUTOHSCROLL |
                    ES_READONLY
    EDITTEXT        IDC_T_BARPRATIO,285,227,57,13,ES_AUTOHSCROLL |
                    ES_READONLY
    LTEXT           "kg/m^3",IDC_STATIC,345,196,26,8
    LTEXT           "N/m^2",IDC_STATIC,345,213,23,8
    LTEXT           "m",IDC_STATIC,345,121,8,8
    LTEXT           "m",IDC_STATIC,345,139,8,8
    GROUPBOX        "Incident Bar",IDC_STATIC,7,7,196,324
    GROUPBOX        "Transmitted Bar",IDC_STATIC,215,7,196,325
    GROUPBOX        "Calibration File",IDC_STATIC,222,16,177,50
    EDITTEXT        IDC_T_BARNAME,229,85,164,12,ES_AUTOHSCROLL | ES_RE
ADONLY
    GROUPBOX        "Bar Name",IDC_STATIC,222,71,177,31
    EDITTEXT        IDC_T_CALIBRATIONNAME,229,28,164,12,ES_AUTOHSCROLL
 |
                    ES_READONLY
    PUSHBUTTON      "Browse",IDC_BROWSE_INCIDENT,71,47,50,14
    GROUPBOX        "Strain Calibration Parameter",IDC_STATIC,14,248,1
78,26
    LTEXT           "Strain=",IDC_STATIC,73,260,23,8
    LTEXT           "V",IDC_STATIC,140,260,8,8
    EDITTEXT        IDC_I_SLOPE,97,258,38,12,ES_AUTOHSCROLL | ES_READO
NLY
    PUSHBUTTON      "Browse",IDC_BROWSE_TRANSMITTED,285,47,50,14
    GROUPBOX        "Strain Calibration Parameter",IDC_STATIC,222,248,
177,26
    LTEXT           "Strain=",IDC_STATIC,281,260,23,8
    LTEXT           "V",IDC_STATIC,347,260,8,8
    EDITTEXT        IDC_T_SLOPE,305,257,38,12,ES_AUTOHSCROLL | ES_READ
ONLY
    LTEXT           " Gauge to Interface Distance:",IDC_STATIC,26,158,
94,8
    EDITTEXT        IDC_I_DISTANCE,122,156,53,13,ES_AUTOHSCROLL |
                    ES_READONLY
    LTEXT           "m",IDC_STATIC,177,158,8,8
    LTEXT           " Gauge to Interface Distance:",IDC_STATIC,229,158
,94,8
```

```
    EDITTEXT        IDC_T_DISTANCE,326,156,53,13,ES_AUTOHSCROLL |
                    ES_READONLY
    LTEXT           "m",IDC_STATIC,381,158,8,8
    CONTROL         "",IDC_IDISPAV,"Button",BS_AUTOCHECKBOX | WS_DISAB
LED |
                    WS_TABSTOP,159,292,16,8
    LTEXT           "Propagation Coefficient Included",IDC_STATIC,50,2
92,104,
                    8
    CONTROL         "",IDC_TDISPAV,"Button",BS_AUTOCHECKBOX | WS_DISAB
LED |
                    WS_TABSTOP,367,292,16,8
    LTEXT           "Propagation Coefficient Included",IDC_STATIC,257,
292,
                    104,8
    LTEXT           "Frequency Filter :",IDC_STATIC,50,308,55,8
    EDITTEXT        IDC_I_FILTER,110,307,38,12,ES_AUTOHSCROLL | ES_REA
DONLY
    LTEXT           "Frequency Filter :",IDC_STATIC,257,308,55,8
    EDITTEXT        IDC_T_FILTER,317,307,38,12,ES_AUTOHSCROLL | ES_REA
DONLY
    GROUPBOX        "Appended Data",IDC_STATIC,222,279,177,46
    GROUPBOX        "Appended Data",IDC_STATIC,14,279,177,46
    LTEXT           "Hz",IDC_STATIC,155,308,24,8
    LTEXT           "Hz",IDC_STATIC,361,308,16,10
END

IDD_EXPORT DIALOG DISCARDABLE  0, 0, 202, 174
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Export"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDC_EXPORTFILE,22,18,117,12,ES_AUTOHSCROLL
    PUSHBUTTON      "Browse",IDC_BROWSE,145,17,50,14
    CONTROL         "Include Header",IDC_EXP_HEADER,"Button",BS_AUTOCH
ECKBOX |
                    WS_TABSTOP,22,51,65,10
    CONTROL         "True Stress, Strain and Strain Rate Data",
                    IDC_EXP_TRUEDATA,"Button",BS_AUTOCHECKBOX | WS_TAB
STOP,
                    22,65,143,10
    CONTROL         "Engineering Stress, Strain and Strain Rate Data",
                    IDC_EXP_ENGDATA,"Button",BS_AUTOCHECKBOX | WS_TABS
TOP,22,
                    79,165,10
    CONTROL         "Strain Waveforms",IDC_EXP_STRAINS,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,22,93,72,10
    CONTROL         "Forces",IDC_FORCES,"Button",BS_AUTOCHECKBOX |
```

```
                    WS_TABSTOP,22,107,78,10
    CONTROL         "Velocities",IDC_VELOCITIES,"Button",BS_AUTOCHECKB
OX |
                    WS_TABSTOP,22,121,78,10
    CONTROL         "Displacements",IDC_DISPLACEMENTS,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,22,135,78,10
    DEFPUSHBUTTON   "OK",IDOK,44,153,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,107,153,50,14
    GROUPBOX        "Items to Export",IDC_STATIC,7,38,188,113
    LTEXT           "File:",IDC_STATIC,7,19,14,8
END

IDD_ANALYSEDLG DIALOG DISCARDABLE  0, 0, 362, 186
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Analyse Data"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "Analyse",IDOK,92,165,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,219,165,50,14
    GROUPBOX        "Sample Data",IDC_STATIC,7,14,109,118
    LTEXT           "Name:",IDC_STATIC,15,31,22,8
    EDITTEXT        IDC_SAMP_NAME,38,30,73,12,ES_AUTOHSCROLL
    LTEXT           "Do:",IDC_STATIC,15,51,22,8
    EDITTEXT        IDC_SAMP_DO,38,50,34,12,ES_AUTOHSCROLL | ES_READON
LY
    LTEXT           "Lo:",IDC_STATIC,15,71,22,8
    EDITTEXT        IDC_SAMP_LO,38,70,34,12,ES_AUTOHSCROLL | ES_READON
LY
    LTEXT           "Df:",IDC_STATIC,15,91,22,8
    EDITTEXT        IDC_SAMP_DF,38,90,34,12,ES_AUTOHSCROLL | ES_READON
LY
    LTEXT           "Lf:",IDC_STATIC,15,111,22,8
    EDITTEXT        IDC_SAMP_LF,38,110,34,12,ES_AUTOHSCROLL | ES_READO
NLY
    GROUPBOX        "Raw Data Files",IDC_STATIC,124,15,231,58
    LTEXT           "Incident Bar:",IDC_STATIC,129,31,41,8
    EDITTEXT        IDC_DATA_I,181,29,134,13,ES_AUTOHSCROLL | ES_READO
NLY
    LTEXT           "Transmitted Bar:",IDC_STATIC,129,52,52,8
    EDITTEXT        IDC_DATA_T,181,50,134,13,ES_AUTOHSCROLL | ES_READO
NLY
    LTEXT           "Incident Bar:",IDC_STATIC,129,99,41,8
    EDITTEXT        IDC_CALIB_I,181,96,134,13,ES_AUTOHSCROLL | ES_READ
ONLY
    LTEXT           "Transmitted Bar:",IDC_STATIC,129,117,52,8
    EDITTEXT        IDC_CALIB_T,181,115,134,13,ES_AUTOHSCROLL | ES_REA
DONLY
    GROUPBOX        "Calibration Files",IDC_STATIC,124,74,231,58
```

```
    LTEXT           "mm",IDC_STATIC,80,51,12,8
    LTEXT           "mm",IDC_STATIC,80,71,12,8
    LTEXT           "mm",IDC_STATIC,80,91,12,8
    LTEXT           "mm",IDC_STATIC,80,111,12,8
    CONTROL         "Fourier Analysis",IDC_FOURIER,"Button",
                    BS_AUTORADIOBUTTON | WS_GROUP,203,150,65,10
    CONTROL         "Conventional Anaylsis",IDC_CONVANALYSIS,"Button",
                    BS_AUTORADIOBUTTON,203,138,85,10
    CONTROL         "",IDC_IDISPAV,"Button",BS_AUTOCHECKBOX | WS_DISAB
LED |
                    WS_TABSTOP,329,98,16,8
    CONTROL         "",IDC_TDISPAV,"Button",BS_AUTOCHECKBOX | WS_DISAB
LED |
                    WS_TABSTOP,329,116,11,9
    LTEXT           "Propagation Coefficient",IDC_STATIC,275,80,74,8
    LTEXT           "Available",IDC_STATIC,317,88,30,8
    EDITTEXT        IDC_DISPFILTER,111,139,41,12,ES_AUTOHSCROLL |
                    WS_DISABLED
    LTEXT           "Hz",IDC_STATIC,155,140,10,8
    CONTROL         "Use Nyquist",IDC_NYQUIST,"Button",BS_AUTORADIOBUT
TON |
                    WS_DISABLED | WS_GROUP,39,152,54,10
    CONTROL         "Filter Frequency",IDC_FILTER,"Button",
                    BS_AUTORADIOBUTTON | WS_DISABLED,39,139,66,10
END

IDD_SEPARATEWAVEVALUES DIALOG DISCARDABLE  0, 0, 283, 114
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Separate Wave Values"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,55,93,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,183,93,50,14
    PUSHBUTTON      "Default",IDC_Default,119,93,50,14
    LTEXT           "Primary Edge Trigger Threshold:",IDC_STATIC,19,32
,102,8
    EDITTEXT        IDC_THRESHOLD1,131,30,31,13,ES_AUTOHSCROLL
    LTEXT           "of maximum value",IDC_STATIC,165,32,57,8
    LTEXT           "Secondary Edge Trigger Threshold:",IDC_STATIC,8,4
8,113,
                    8
    EDITTEXT        IDC_THRESHOLD2,131,47,31,13,ES_AUTOHSCROLL
    LTEXT           "of first edge",IDC_STATIC,165,48,57,8
    LTEXT           "Filter:",IDC_STATIC,103,64,18,8
    EDITTEXT        IDC_TOLERANCE,131,63,31,13,ES_AUTOHSCROLL
    LTEXT           "of maximum value",IDC_STATIC,167,64,57,8
    CONTROL         "Separate Transmitted",IDC_SEPARATETRANSMITTED,"Bu
tton",
```

```
                    BS_AUTOCHECKBOX | WS_TABSTOP,157,14,83,10
    CONTROL         "Separate Incident",IDC_SEPARATEINCIDENT,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,43,14,72,10
    CONTROL         "Use Intercept Method",IDC_INTERCEPT,"Button",
                    BS_AUTORADIOBUTTON | WS_GROUP,35,80,89,10
    CONTROL         "Use Peak Detection Method",IDC_PEAK,"Button",
                    BS_AUTORADIOBUTTON,151,79,106,10
END

IDD_INDICES DIALOG DISCARDABLE  0, 0, 248, 92
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CAPTION "Starting and End Points"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDC_IS,59,19,45,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_IE,143,19,45,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_RS,59,33,45,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_RE,143,33,45,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_TS,59,46,45,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_TE,143,46,45,12,ES_AUTOHSCROLL
    DEFPUSHBUTTON   "OK",IDOK,27,71,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,98,71,50,14
    GROUPBOX        "Start Index",IDC_START,54,7,76,58
    GROUPBOX        "End Index",IDC_END,136,7,76,58
    LTEXT           "Incident",IDC_STATIC,22,21,26,8
    LTEXT           "Reflected",IDC_STATIC,16,35,32,8
    LTEXT           "Transmitted",IDC_STATIC,10,48,38,8
    PUSHBUTTON      "View Time",IDC_VIEWTIME,169,71,50,14
    PUSHBUTTON      "",IDC_CHOOSE_IS,110,21,10,8
    PUSHBUTTON      "",IDC_CHOOSE_RS,110,35,10,8
    PUSHBUTTON      "",IDC_CHOOSE_TS,110,48,10,8
    PUSHBUTTON      "",IDC_CHOOSE_IE,194,21,10,8
    PUSHBUTTON      "",IDC_CHOOSE_RE,194,35,10,8
    PUSHBUTTON      "",IDC_CHOOSE_TE,194,48,10,8
END

IDD_PROPCOEFF DIALOG DISCARDABLE  0, 0, 264, 116
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Propagation Coefficient"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,50,95,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,135,95,50,14
    GROUPBOX        "Files",IDC_STATIC,7,7,250,65
    LTEXT           "Data File:",IDC_STATIC,32,20,31,8
    EDITTEXT        IDC_DATA_I,66,17,183,13,ES_AUTOHSCROLL | ES_READON
LY
    LTEXT           "Calibration File:",IDC_STATIC,15,34,48,8
```

```
    EDITTEXT        IDC_CALIB_I,66,33,183,13,ES_AUTOHSCROLL | ES_READO
NLY
    CONTROL         "Update Calibration File",IDC_UPDATEFILE,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,15,54,87,10
    CONTROL         "Export to File",IDC_EXPFILE,"Button",BS_AUTOCHECK
BOX |
                    WS_TABSTOP,148,55,78,9
    CONTROL         "Export Complex Modulus",IDC_COMPLEXMOD,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,15,76,93,10
    LTEXT           "Filter:",IDC_STATIC,133,77,18,8
    EDITTEXT        IDC_FILTER,153,75,44,13,ES_AUTOHSCROLL
    LTEXT           "Hz",IDC_STATIC,202,78,10,8
END

IDD_FFTUTILITY DIALOG DISCARDABLE  0, 0, 219, 118
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "FFT Utility"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "Browse",IDC_BROWSE_IN,143,21,50,14
    PUSHBUTTON      "Browse",IDC_BROWSE_OUT,143,45,50,14
    CONTROL         "Forward Transform",IDC_FFT,"Button",BS_AUTORADIOB
UTTON |
                    WS_GROUP,25,63,75,10
    CONTROL         "Inverse Transform",IDC_IFFT,"Button",BS_AUTORADIO
BUTTON,
                    25,76,73,10
    DEFPUSHBUTTON   "OK",IDOK,52,90,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,116,90,50,14
    EDITTEXT        IDC_INPUTNAME,51,21,85,13,ES_AUTOHSCROLL
    LTEXT           "Input File:",IDC_STATIC,18,23,32,8
    EDITTEXT        IDC_OUTPUTNAME,51,45,85,13,ES_AUTOHSCROLL
    LTEXT           "Output File:",IDC_STATIC,13,47,37,8
END

IDD_INPUTFILE DIALOG DISCARDABLE  0, 0, 346, 69
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Input File"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,102,48,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,193,48,50,14
    LTEXT           "File:",IDC_STATIC,14,16,14,8
    EDITTEXT        IDC_FILENAME,30,14,250,13,ES_AUTOHSCROLL
    CONTROL         "Binary",IDC_BINARY,"Button",BS_AUTORADIOBUTTON |
                    WS_GROUP,47,32,35,10
    CONTROL         "Ascii",IDC_ASCII,"Button",BS_AUTORADIOBUTTON,145,
32,31,
```

```
                    10
    PUSHBUTTON      "Browse",IDC_BROWSE,289,14,50,14
    CONTROL         "Amplitude Shift",IDC_AMPSHIFT,"Button",BS_AUTOCHE
CKBOX |
                    WS_TABSTOP,239,31,63,10
END

IDD_AMP_PROP DIALOG DISCARDABLE  0, 0, 159, 53
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Amplitude Shift"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,20,32,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,89,32,50,14
    LTEXT           "Number of point to use as zero:",IDC_STATIC,7,14,
99,8
    EDITTEXT        IDC_POINTS,109,12,30,13,ES_AUTOHSCROLL
END

IDD_NICOLETCONVERTDLG DIALOGEX 0, 0, 372, 106
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_APPWINDOW
CAPTION "Nicolet Conversion Utility"
FONT 8, "MS Sans Serif", 0, 0, 0x1
BEGIN
    PUSHBUTTON      "Browse",IDC_INPUTBROWSE,308,15,50,14
    PUSHBUTTON      "Browse",IDC_OUPUTBROWSE,308,43,50,14
    CONTROL         "Include Header",IDC_INCHEADER,"Button",BS_AUTOCHE
CKBOX |
                    WS_TABSTOP,103,65,88,9
    CONTROL         "Amplitude Shift",IDC_AMPLTITUDE,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,205,65,63,10
    DEFPUSHBUTTON   "Convert",IDCONVERT,120,85,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,202,85,50,14
    LTEXT           "Input File:",IDC_STATIC,22,19,32,8
    EDITTEXT        IDC_INPUTFILE,57,16,242,13,ES_AUTOHSCROLL
    LTEXT           "Output File:",IDC_STATIC,17,46,37,8
    EDITTEXT        IDC_OUTPUTFILE,57,44,242,13,ES_AUTOHSCROLL
END

IDD_PROPAGATEWAVE DIALOG DISCARDABLE  0, 0, 203, 122
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Propagate Wave"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDC_INCIDENTDIST,92,15,60,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_REFLECTEDDIST,92,32,60,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_TRANSMITTEDDIST,92,49,60,12,ES_AUTOHSCROLL
```

```
    CONTROL         "Filter Frequency",IDC_FILTER,"Button",
                    BS_AUTORADIOBUTTON,28,73,66,10
    EDITTEXT        IDC_DISPFILTER2,100,73,41,12,ES_AUTOHSCROLL
    CONTROL         "Use Nyquist",IDC_NYQUIST,"Button",BS_AUTORADIOBUT
TON |
                    WS_GROUP,28,86,54,10
    DEFPUSHBUTTON   "OK",IDOK,45,101,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,108,101,50,14
    LTEXT           "Incident Wave:",IDC_STATIC,36,16,50,8
    LTEXT           "Transmitted Wave:",IDC_STATIC,25,52,61,8
    LTEXT           "m",IDC_STATIC,155,18,8,8
    LTEXT           "m",IDC_STATIC,155,52,8,8
    LTEXT           "Hz",IDC_STATIC,146,75,10,8
    LTEXT           "Reflected Wave:",IDC_STATIC,31,34,55,8
    LTEXT           "m",IDC_STATIC,155,35,8,8
END


#ifndef _MAC
/////////////////////////////////////////////////////////////////////////
///////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904B0"
        BEGIN
            VALUE "CompanyName", "\0"
            VALUE "FileDescription", "CSHB MFC Application\0"
            VALUE "FileVersion", "1, 0, 0, 1\0"
            VALUE "InternalName", "CSHB\0"
            VALUE "LegalCopyright", "Copyright (C) 2001\0"
            VALUE "LegalTrademarks", "\0"
```

```
            VALUE "OriginalFilename", "CSHB.EXE\0"
            VALUE "ProductName", "CSHB Application\0"
            VALUE "ProductVersion", "1, 0, 0, 1\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END

#endif    // !_MAC


/////////////////////////////////////////////////////////////////////////////
///////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 228
        TOPMARGIN, 7
        BOTTOMMARGIN, 72
    END

    IDD_BARDATA, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 411
        VERTGUIDE, 14
        VERTGUIDE, 21
        VERTGUIDE, 50
        VERTGUIDE, 73
        VERTGUIDE, 96
        VERTGUIDE, 136
        VERTGUIDE, 148
        VERTGUIDE, 192
        VERTGUIDE, 222
        VERTGUIDE, 229
        VERTGUIDE, 258
        VERTGUIDE, 355
        VERTGUIDE, 399
```

```
    TOPMARGIN, 7
    BOTTOMMARGIN, 357
    HORZGUIDE, 239
    HORZGUIDE, 250
    HORZGUIDE, 300
    HORZGUIDE, 319
    HORZGUIDE, 332
END

IDD_EXPORT, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 195
    VERTGUIDE, 22
    VERTGUIDE, 139
    TOPMARGIN, 7
    BOTTOMMARGIN, 167
END

IDD_ANALYSEDLG, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 355
    VERTGUIDE, 124
    VERTGUIDE, 329
    TOPMARGIN, 7
    BOTTOMMARGIN, 179
    HORZGUIDE, 132
END

IDD_SEPARATEWAVEVALUES, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 276
    TOPMARGIN, 7
    BOTTOMMARGIN, 107
END

IDD_INDICES, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 241
    VERTGUIDE, 54
    VERTGUIDE, 60
    VERTGUIDE, 110
    VERTGUIDE, 138
    VERTGUIDE, 144
    VERTGUIDE, 194
```

```
    VERTGUIDE, 212
    TOPMARGIN, 7
    BOTTOMMARGIN, 85
    HORZGUIDE, 25
    HORZGUIDE, 39
    HORZGUIDE, 52
END

IDD_PROPCOEFF, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 257
    TOPMARGIN, 7
    BOTTOMMARGIN, 109
END

IDD_FFTUTILITY, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 212
    TOPMARGIN, 7
    BOTTOMMARGIN, 111
END

IDD_INPUTFILE, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 339
    TOPMARGIN, 7
    BOTTOMMARGIN, 62
END

IDD_AMP_PROP, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 152
    TOPMARGIN, 7
    BOTTOMMARGIN, 46
    HORZGUIDE, 25
END

IDD_PROPAGATEWAVE, DIALOG
BEGIN
    LEFTMARGIN, 7
    RIGHTMARGIN, 196
    VERTGUIDE, 92
    TOPMARGIN, 7
    BOTTOMMARGIN, 115
```

```
    END
END
#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////
///////
//
// Cursor
//

IDC_CROSSHAIR           CURSOR  DISCARDABLE     "res\\cursor1.cur"

////////////////////////////////////////////////////////////////////////
///////
//
// String Table
//

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    IDR_MAINFRAME           "CSHB"
    IDR_CSHBTYPE            "\nCSHB\nCSHB\n\n\nCSHB.Document\nCSHB Doc
ument"
END

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    AFX_IDS_APP_TITLE       "CSHB"
    AFX_IDS_IDLEMESSAGE     "Ready"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_INDICATOR_EXT        "EXT"
    ID_INDICATOR_CAPS       "CAP"
    ID_INDICATOR_NUM        "NUM"
    ID_INDICATOR_SCRL       "SCRL"
    ID_INDICATOR_OVR        "OVR"
    ID_INDICATOR_REC        "REC"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_NEW             "Create a new document\nNew"
    ID_FILE_OPEN            "Open an existing document\nOpen"
    ID_FILE_CLOSE           "Close the active document\nClose"
    ID_FILE_SAVE            "Save the active document\nSave"
```

```
    ID_FILE_SAVE_AS         "Save the active document with a new name\
nSave As"
    ID_FILE_PAGE_SETUP      "Change the printing options\nPage Setup"
    ID_FILE_PRINT_SETUP     "Change the printer and printing options\n
Print Setup"
    ID_FILE_PRINT           "Print the active document\nPrint"
    ID_FILE_PRINT_PREVIEW   "Display full pages\nPrint Preview"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_APP_ABOUT            "Display program information, version numb
er and
copyright\nAbout"
    ID_APP_EXIT             "Quit the application; prompts to save doc
uments\nExit"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_MRU_FILE1       "Open this document"
    ID_FILE_MRU_FILE2       "Open this document"
    ID_FILE_MRU_FILE3       "Open this document"
    ID_FILE_MRU_FILE4       "Open this document"
    ID_FILE_MRU_FILE5       "Open this document"
    ID_FILE_MRU_FILE6       "Open this document"
    ID_FILE_MRU_FILE7       "Open this document"
    ID_FILE_MRU_FILE8       "Open this document"
    ID_FILE_MRU_FILE9       "Open this document"
    ID_FILE_MRU_FILE10      "Open this document"
    ID_FILE_MRU_FILE11      "Open this document"
    ID_FILE_MRU_FILE12      "Open this document"
    ID_FILE_MRU_FILE13      "Open this document"
    ID_FILE_MRU_FILE14      "Open this document"
    ID_FILE_MRU_FILE15      "Open this document"
    ID_FILE_MRU_FILE16      "Open this document"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_NEXT_PANE            "Switch to the next window pane\nNext Pane
"
    ID_PREV_PANE            "Switch back to the previous window pane\n
Previous Pane"
END

STRINGTABLE DISCARDABLE
BEGIN
```

```
    ID_WINDOW_NEW           "Open another window for the active docume
nt\nNew Window"
    ID_WINDOW_ARRANGE       "Arrange icons at the bottom of the window
\nArrange Icons"
    ID_WINDOW_CASCADE       "Arrange windows so they overlap\nCascade
Windows"
    ID_WINDOW_TILE_HORZ     "Arrange windows as non-overlapping tiles\
nTile Windows"
    ID_WINDOW_TILE_VERT     "Arrange windows as non-overlapping tiles\
nTile Windows"
    ID_WINDOW_SPLIT         "Split the active window into panes\nSplit
"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_EDIT_CLEAR           "Erase the selection\nErase"
    ID_EDIT_CLEAR_ALL       "Erase everything\nErase All"
    ID_EDIT_COPY            "Copy the selection and put it on the Clip
board\nCopy"
    ID_EDIT_CUT             "Cut the selection and put it on the Clipb
oard\nCut"
    ID_EDIT_FIND            "Find the specified text\nFind"
    ID_EDIT_PASTE           "Insert Clipboard contents\nPaste"
    ID_EDIT_REPEAT          "Repeat the last action\nRepeat"
    ID_EDIT_REPLACE         "Replace specific text with different text
\nReplace"
    ID_EDIT_SELECT_ALL      "Select the entire document\nSelect All"
    ID_EDIT_UNDO            "Undo the last action\nUndo"
    ID_EDIT_REDO            "Redo the previously undone action\nRedo"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_VIEW_TOOLBAR         "Show or hide the toolbar\nToggle ToolBar"
    ID_VIEW_STATUS_BAR      "Show or hide the status bar\nToggle Statu
sBar"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCSIZE          "Change the window size"
    AFX_IDS_SCMOVE          "Change the window position"
    AFX_IDS_SCMINIMIZE      "Reduce the window to an icon"
    AFX_IDS_SCMAXIMIZE      "Enlarge the window to full size"
    AFX_IDS_SCNEXTWINDOW    "Switch to the next document window"
    AFX_IDS_SCPREVWINDOW    "Switch to the previous document window"
    AFX_IDS_SCCLOSE         "Close the active window and prompts to sa
```

```
ve the documents"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCRESTORE       "Restore the window to normal size"
    AFX_IDS_SCTASKLIST      "Activate Task List"
    AFX_IDS_MDICHILD        "Activate this window"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_PREVIEW_CLOSE   "Close print preview mode\nCancel Preview"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_ANALYSE_CALCULATERESULTS "\nCalculate Results"
    ID_PRE_VIEWINDICES      "View Indicies"
    ID_FILE_NEWSAMPLE       "\nNew Sample"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_READ_INCIDENT   "Read New Incident Data Set\nRead Incident
 Data"
    ID_FILE_READ_TRANSMITTED "\nRead Transmitted Data"
    ID_SAMPLE_PROPERTIES    "\nSample Properties"
END

#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////
///////



#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////
///////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
```

```
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif //_WIN32
#include "res\CSHB.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#include "afxprint.rc"       // printing/print preview resources
#endif

/////////////////////////////////////////////////////////////////////////////
///////
#endif    // not APSTUDIO_INVOKED
```

```
// AnalyseDlg.cpp : implementation file
//

#include "stdafx.h"
#include "CSHB.h"
#include "AnalyseDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAnalyseDlg dialog


CAnalyseDlg::CAnalyseDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CAnalyseDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CAnalyseDlg)
    m_dSampLo = 0.0;
    m_dSampDf = 0.0;
    m_dSampDo = 0.0;
    m_dSampLf = 0.0;
    m_szSampName = _T("");
    m_szCalib_I = _T("");
    m_szCalib_T = _T("");
    m_szData_I = _T("");
    m_szData_T = _T("");
    m_bIncidentDispersion = FALSE;
    m_bTransmittedDispersion = FALSE;
    m_dFreqFilter = 0.0;
    m_nFourierAnalysis = -1;
    m_nNyquist = -1;
    //}}AFX_DATA_INIT



}


void CAnalyseDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAnalyseDlg)
    DDX_Text(pDX, IDC_SAMP_LO, m_dSampLo);
    DDX_Text(pDX, IDC_SAMP_DF, m_dSampDf);
    DDX_Text(pDX, IDC_SAMP_DO, m_dSampDo);
    DDX_Text(pDX, IDC_SAMP_LF, m_dSampLf);
    DDX_Text(pDX, IDC_SAMP_NAME, m_szSampName);
    DDX_Text(pDX, IDC_CALIB_I, m_szCalib_I);
    DDX_Text(pDX, IDC_CALIB_T, m_szCalib_T);
    DDX_Text(pDX, IDC_DATA_I, m_szData_I);
    DDX_Text(pDX, IDC_DATA_T, m_szData_T);
    DDX_Check(pDX, IDC_IDISPAV, m_bIncidentDispersion);
    DDX_Check(pDX, IDC_TDISPAV, m_bTransmittedDispersion);
    DDX_Text(pDX, IDC_DISPFILTER, m_dFreqFilter);
    DDX_Radio(pDX, IDC_FOURIER, m_nFourierAnalysis);
    DDX_Radio(pDX, IDC_NYQUIST, m_nNyquist);
    //}}AFX_DATA_MAP
    if (m_nFourierAnalysis==0)
    {
            GetDlgItem(IDC_FILTER)->EnableWindow(TRUE);
            GetDlgItem(IDC_NYQUIST)->EnableWindow(TRUE);
            if (m_nNyquist!=0)
            {GetDlgItem(IDC_DISPFILTER)->EnableWindow(TRUE);
            }else{
                GetDlgItem(IDC_DISPFILTER)->EnableWindow(FALSE);
            }
    }
}


BEGIN_MESSAGE_MAP(CAnalyseDlg, CDialog)
    //{{AFX_MSG_MAP(CAnalyseDlg)
```

```
    ON_BN_CLICKED(IDC_FOURIER, OnFourier)
    ON_BN_CLICKED(IDC_NYQUIST, OnNyquist)
    ON_BN_CLICKED(IDC_FILTER, OnFilter)
    ON_BN_CLICKED(IDC_CONVANALYSIS, OnConvanalysis)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAnalyseDlg message handlers

void CAnalyseDlg::OnCancel()
{
    // TODO: Add extra cleanup here

    CDialog::OnCancel();
}

void CAnalyseDlg::OnFourier()
{
    GetDlgItem(IDC_FILTER)->EnableWindow(TRUE);
    GetDlgItem(IDC_NYQUIST)->EnableWindow(TRUE);
    if(m_nNyquist!=0)
    {GetDlgItem(IDC_DISPFILTER)->EnableWindow(TRUE);}
    else
    {GetDlgItem(IDC_DISPFILTER)->EnableWindow(FALSE);}
}

void CAnalyseDlg::OnNyquist()
{
    GetDlgItem(IDC_DISPFILTER)->EnableWindow(FALSE);

}

void CAnalyseDlg::OnFilter()
{
    GetDlgItem(IDC_DISPFILTER)->EnableWindow(TRUE);

}

void CAnalyseDlg::OnConvanalysis()
{
    GetDlgItem(IDC_FILTER)->EnableWindow(FALSE);
    GetDlgItem(IDC_DISPFILTER)->EnableWindow(FALSE);
    GetDlgItem(IDC_NYQUIST)->EnableWindow(FALSE);

}
```

```
// BarDataDlg.cpp : implementation file
//

#include "stdafx.h"
#include "CSHB.h"
#include "BarDataDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
//I added
#define PI 3.14159265
#define ITMAX 100
#define EPS 3.0e-7
#define FPMIN 1.0e-30
#include "math.h"
#include "nrutil.h"
#include "fstream.h"
/////////////////////////////////////////////////////////////////////////////
// CBarDataDlg dialog


CBarDataDlg::CBarDataDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CBarDataDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CBarDataDlg)
    m_dILength = 0.0;
    m_dIPRatio = 0.0;
    m_dTDensity = 0.0;
    m_dTDiameter = 0.0;
    m_dTModulus = 0.0;
    m_dTLength = 0.0;
    m_dTPRatio = 0.0;
    m_dIDensity = 0.0;
    m_dIDiameter = 0.0;
    m_dIModulus = 0.0;
    m_szIFilename = _T("");
    m_szIName = _T("");
    m_dISlope = 0.0;
    m_szTName = _T("");
    m_szTFilename = _T("");
    m_dTSlope = 0.0;
    m_dIDistance = 0.0;
    m_dTDistance = 0.0;
    m_bIPropCoeffIncluded = FALSE;
    m_bTPropCoeffIncluded = FALSE;
    m_dIFilter = 0.0;
    m_dTFilter = 0.0;
    //}}AFX_DATA_INIT
    m_bReadIncidentBar=FALSE;
    m_bReadTransmittedBar=FALSE;
}


void CBarDataDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CBarDataDlg)
    DDX_Text(pDX, IDC_I_BARLENGTH, m_dILength);
    DDX_Text(pDX, IDC_I_BARPRATIO, m_dIPRatio);
    DDV_MinMaxDouble(pDX, m_dIPRatio, 0., 0.5);
    DDX_Text(pDX, IDC_T_BARDENSITY, m_dTDensity);
    DDX_Text(pDX, IDC_T_BARDIAMETER, m_dTDiameter);
    DDX_Text(pDX, IDC_T_BAREMODULUS, m_dTModulus);
    DDX_Text(pDX, IDC_T_BARLENGTH, m_dTLength);
    DDX_Text(pDX, IDC_T_BARPRATIO, m_dTPRatio);
    DDV_MinMaxDouble(pDX, m_dTPRatio, 0., 0.5);
    DDX_Text(pDX, IDC_I_BARDENSITY, m_dIDensity);
    DDX_Text(pDX, IDC_I_BARDIAMETER, m_dIDiameter);
    DDX_Text(pDX, IDC_I_BAREMODULUS, m_dIModulus);
    DDX_Text(pDX, IDC_I_CALIBRATIONNAME, m_szIFilename);
    DDX_Text(pDX, IDC_I_BARNAME, m_szIName);
    DDX_Text(pDX, IDC_I_SLOPE, m_dISlope);
    DDX_Text(pDX, IDC_T_BARNAME, m_szTName);
```

```
    DDX_Text(pDX, IDC_T_CALIBRATIONNAME, m_szTFilename);
    DDX_Text(pDX, IDC_T_SLOPE, m_dTSlope);
    DDX_Text(pDX, IDC_I_DISTANCE, m_dIDistance);
    DDX_Text(pDX, IDC_T_DISTANCE, m_dTDistance);
    DDX_Check(pDX, IDC_IDISPAV, m_bIPropCoeffIncluded);
    DDX_Check(pDX, IDC_TDISPAV, m_bTPropCoeffIncluded);
    DDX_Text(pDX, IDC_I_FILTER, m_dIFilter);
    DDX_Text(pDX, IDC_T_FILTER, m_dTFilter);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CBarDataDlg, CDialog)
    //{{AFX_MSG_MAP(CBarDataDlg)
    ON_BN_CLICKED(IDC_BROWSE_INCIDENT, OnBrowseIncident)
    ON_BN_CLICKED(IDC_BROWSE_TRANSMITTED, OnBrowseTransmitted)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CBarDataDlg message handlers

void CBarDataDlg::OnBrowseIncident()
{
    CDialog::UpdateData(TRUE);
    m_dIModulus=0;
    m_dILength=0;
    m_dIDistance=0;
    m_dIPRatio=0;
    m_dIDensity=0;
    m_dIDiameter=0;
    m_dISlope=0;

    m_szIFilename="";
    m_szIName="";
    m_dIDf=0;
    m_dIDT=0;
    m_bIPropCoeffIncluded=FALSE;

    GetIncidentBarData();
    CDialog::UpdateData(FALSE);
}

void CBarDataDlg::fit(double x[], double y[], int ndata, double sig[], int mwt, double *a, double
*b, double *siga, double *sigb, double *chi2, double *q)
{
    //linear regression model of data  "Numerical Recipes in C 2nd Edition" pg 665-666
    int i;
    double wt,t,sxoss,sx=0.0,sy=0.0,st2=0.0,ss;
    double sxy=0,sx2=0;

    *b=0.0;
    if (mwt){
        ss=0.0;
        for (i=0;i<ndata;i++){          //with weights
            wt=1.0/DSQR(sig[i]);
            ss+=wt;
            sx+=x[i]*wt;
            sy+=y[i]*wt;
        }
    }else{
        for (i=0;i<ndata;i++){          //without weights
            sx+=x[i];                   //sum of x's
            sy+=y[i];                   //sum of y's
            sxy+=x[i]*y[i];
            sx2+=(x[i]*x[i]);
        }
        ss=ndata;
    }



    sxoss=sx/ss;
    if (mwt){
        for (i=0;i<ndata;i++){
            t=(x[i]-sxoss)/sig[i];
```

```
            st2+=t*t;
            *b+=t*y[i]/sig[i];
        }
    }else{
        for (i=0;i<ndata;i++){
            t=x[i]-sxoss;
            st2+=t*t;
            *b+=t*y[i];
        }
    }

    *b/=st2;
    *a=(sy-sx*(*b))/ss;

//  *siga=sqrt((1.0+sx*sx/(ss*st2))/ss);
    //*sigb=sqrt(1.0/st2);
/*
    *chi2=0.0;
    *q=1.0;

    if(mwt==0){
        for (i=0;i<ndata;i++)
                *chi2+=DSQR(y[i]-(*a)-(*b)*x[i]);
            sigdat=sqrt((*chi2)/(ndata-2));
            *siga*=sigdat;
            *sigb*=sigdat;
    }else{
        for (i=0;i<ndata;i++)
                *chi2+=DSQR(y[i]-(*a)-(*b)*x[i]/sig[i]);
        if (ndata>2) *q=gammq(0.5*(ndata-2),0.5*(*chi2)); //Equation 15.2.12
    }
    */
}

double CBarDataDlg::gammln(double xx)
{
    double x,y,tmp,ser;
    static double cof[6]={76.18009172947146,-86.50532032941677,24.01409824083091,
        -1.231739572450155,0.1208650973866179e-2,-0.5395239384953e-5};
    int j;
    y=x=xx;
    tmp=x+5.5;
    tmp-=(x+0.5)*log(tmp);
    ser=1.000000000190015;
    for (j=0;j<=5;j++) ser+=cof[j]/++y;
    return -tmp+log(2.5066282746310005*ser/x);
}
void CBarDataDlg::gser(double *gamser, double a, double x, double *gln)
{

//  void nrerror( char error_text[]);
    int n;
    double sum,del,ap;

    *gln=gammln(a);
    if (x<=0.0){
//      if (x<0.0) nrerror ("x less than 0 in routine gser");
        *gamser=0.0;
        return;
    }else{
        ap=a;
        del=sum=1.0/a;
        for (n=1;n<=ITMAX;n++){
            ++ap;
            del *=x/ap;
            sum +=del;
            if (fabs(del)<fabs(sum)*EPS){
//              *gamser=sum*exp(-x+a*log(x)-(*gln));
                return;
            }
        }
//      nrerror("a too large,ITMAX too small in gser");
        return;
    }
}
```

```
void CBarDataDlg::gcf(double *gammcf, double a, double x, double *gln)
{

//  void nrerror(char error_text[]);
    int i;
    double an,b,c,d,del,h;


    *gln=gammln(a);
    b=x+1.0-a;
    c=1.0/FPMIN;
    d=1.0/b;
    h=d;

    for (i=1;i<=ITMAX;i++){
        an=-i*(i-a);
        b+=2.0;
        d=an*d+b;
        if (fabs(d)<FPMIN) d=FPMIN;
        c=b+an/c;
        if (fabs(c)<FPMIN) c=FPMIN;
        d=1.0/d;
        del=d*c;
        h *=del;
        if (fabs(del-1.0)<EPS) break;
    }
//  if (i>ITMAX) nrerror("a too large, ITMAX too small in gcf");
//  *gammcf=exp(-x+a*log(x)-(*gln))*h;

}

double CBarDataDlg::gammq(double a, double x)
{
    double gamser,gammcf,gln;

//  if (x<0.0||a<=0.0)nrerror("Invalid arugements in gammq");
    if(x<(a+1.0)){
        gser(&gamser,a,x,&gln);
        return 1.0-gamser;
    }else{
        gcf(&gammcf,a,x,&gln);
        return gammcf;
    }
}

BOOL CBarDataDlg::GetIncidentBarData()
{
    char string[1000];
    char seps[]   = " ,\t";



    BOOL stopflag = false;


    OPENFILENAME OpenFileName;
    TCHAR szFile[256000] = "\0";    // File list
    static char szFilter[] = "Calibration Files (*.cal)\0*.cal\0"
                             "All Files (*.*)\0*.*\0\0";

    I.m_dStrain.RemoveAll();
    I.m_dVolts.RemoveAll();

    OpenFileName.lStructSize = sizeof(OPENFILENAME);
    OpenFileName.hwndOwner = NULL;
    OpenFileName.hInstance = NULL;
    OpenFileName.lpstrFilter = szFilter;
    OpenFileName.lpstrCustomFilter = NULL;
    OpenFileName.nMaxCustFilter = 0;
    OpenFileName.nFilterIndex = 0;
    OpenFileName.lpstrFile = szFile;
    OpenFileName.nMaxFile = sizeof(szFile);
    OpenFileName.lpstrFileTitle = NULL;
    OpenFileName.nMaxFileTitle = 0;
    OpenFileName.lpstrInitialDir = m_szIFilename;
    OpenFileName.lpstrTitle = "Select Incident Bar Calibration File";
```

```
OpenFileName.nFileOffset = 0;
OpenFileName.nFileExtension = 0;
OpenFileName.lpstrDefExt = "";
OpenFileName.lpfnHook = NULL;
OpenFileName.lpTemplateName = NULL;
OpenFileName.Flags = OFN_EXPLORER | OFN_FILEMUSTEXIST;

// Open dialog box
GetOpenFileName(&OpenFileName);
// Filenames are stored in szFile;
// Assign the value of szFile to m_FileList
m_szIFilename = szFile;
// Update the information in the textbox to show the current files selected

ifstream ifsin(m_szIFilename,ios::in|ios::nocreate);

if (!ifsin)
{   AfxMessageBox("Can't Open File",MB_ICONHAND);
    return FALSE;
}
ifsin.getline(string,1000,'\n');
do{

    if (string[0]=='*')
    {
        if ((strncmp(string,"*NAME",5))==0)         //check for name of bar
        {
            ifsin.getline(string,1000,'\n');    //gets next line after identifier
            m_szIName=string;
        }

        if ((strncmp(string,"*GEOMETRY",5))==0)
        {
            ifsin.getline(string,1000,'\n');                //gets bar geomerty
            m_dIDiameter=(atof( strtok( string, seps))) ;
            m_dILength=(atof( strtok( NULL, seps))) ;
            m_dIDistance=(atof( strtok( NULL, seps)));
        }

        if ((strncmp(string,"*PROPERTIES",7))==0)
        {
            ifsin.getline(string,1000,'\n');                //gets bar properties
            m_dIDensity=(atof( strtok( string, seps))) ;
            m_dIModulus=(atof( strtok( NULL, seps))) ;
            m_dIPRatio=(atof( strtok( NULL, seps))) ;
        }

        if ((strncmp(string,"*FILTER",7))==0)
        {
            ifsin.getline(string,1000,'\n');                //gets bar properties
            m_dIFilter=(atof( strtok( string, seps))) ;

        }
        if ((strncmp(string,"*CALIBRATION",5))==0)
        {   ifsin.getline(string,1000,'\n');

            while(((string[0])!='*')&&(ifsin!=0))
            {
                if ((string[0] != '"')&&(string[0]!='\0'))
                {   I.m_dVolts.Add((atof( strtok( string, seps ))));
                    I.m_dStrain.Add((atof( strtok( NULL, seps ))));
                }
                ifsin.getline(string,1000,'\n');
            }
        }
        if ((strncmp(string,"*PROPAGATION",7))==0)
        {   ifsin.getline(string,1000,'\n');
            I.m_dFreq.RemoveAll();
            I.m_dG.RemoveAll();
            while(((string[0])!='*')&&(ifsin!=0))
            {
                if ((string[0] != '"')&&(string[0]!='\0'))
                {   I.m_dFreq.Add((atof( strtok( string, seps ))));
                    I.m_dG.Add((atof( strtok( NULL, seps ))));
                    I.m_dG.Add((atof( strtok( NULL, seps ))));
                }
```

```
                    ifsin.getline(string,1000,'\n');
                }
                m_dIDf=I.m_dFreq[1]-I.m_dFreq[0];
                m_dIDT=1/(I.m_dG.GetSize()*m_dIDf);
                m_bIPropCoeffIncluded=TRUE;
            }

        }
        if(string[0]!='*')
        {ifsin.getline(string,1000,'\n');}
    }   while(ifsin!=0);

    ifsin.close();
    double temp;
    I.m_dFreq.RemoveAll();
    fit(&I.m_dVolts[0],&I.m_dStrain[0],I.m_dStrain.GetSize(),0,0,&temp,&m_dISlope,0,0,0,0);

    // Set intercept =0 since at zero force there must be zero strain
    m_bReadIncidentBar=TRUE;

    return TRUE;


}

void CBarDataDlg::OnBrowseTransmitted()
{

    // Get the current information entered
    CDialog::UpdateData(TRUE);
    m_dTModulus=0;
    m_dTLength=0;
    m_dTDistance=0;
    m_dTPRatio=0;
    m_dTDensity=0;
    m_dTDiameter=0;
    m_dTSlope=0;

    m_szTFilename="";
    m_szTName="";
    m_dTDf=0;
    m_dTDT=0;
    m_bTPropCoeffIncluded=FALSE;

    GetTransmittedBarData();
    CDialog::UpdateData(FALSE);

}

BOOL CBarDataDlg::GetTransmittedBarData()
{
    char string[1000];
    char seps[]   = " ,\t";

    bool stopflag = false;

    T.m_dStrain.RemoveAll();
    T.m_dVolts.RemoveAll();

    OPENFILENAME OpenFileName;
    TCHAR szFile[256000] = "\0";    // File list
    static char szFilter[] = "Calibration Files (*.cal)\0*.cal\0"
                             "All Files (*.*)\0*.*\0\0";


    OpenFileName.lStructSize = sizeof(OPENFILENAME);
    OpenFileName.hwndOwner = NULL;
    OpenFileName.hInstance = NULL;
    OpenFileName.lpstrFilter = szFilter;
    OpenFileName.lpstrCustomFilter = NULL;
    OpenFileName.nMaxCustFilter = 0;
    OpenFileName.nFilterIndex = 0;
    OpenFileName.lpstrFile = szFile;
    OpenFileName.nMaxFile = sizeof(szFile);
    OpenFileName.lpstrFileTitle = NULL;
    OpenFileName.nMaxFileTitle = 0;
```

```
    OpenFileName.lpstrInitialDir = m_szTFilename;
    OpenFileName.lpstrTitle = "Select Transmitted Bar Calibration File";
    OpenFileName.nFileOffset = 0;
    OpenFileName.nFileExtension = 0;
    OpenFileName.lpstrDefExt = "";
    OpenFileName.lpfnHook = NULL;
    OpenFileName.lpTemplateName = NULL;
    OpenFileName.Flags = OFN_EXPLORER | OFN_FILEMUSTEXIST;

    // Open dialog box
    GetOpenFileName(&OpenFileName);
    // Filenames are stored in szFile;
    // Assign the value of szFile to m_FileList
    m_szTFilename = szFile;
    // Update the information in the textbox to show the current files selected

    ifstream ifsin(m_szTFilename,ios::in|ios::nocreate);

    if (!ifsin)
    {   AfxMessageBox("Can't Open File",MB_ICONHAND);
        return FALSE;
    }
    ifsin.getline(string,1000,'\n');
    do{

        if (string[0]=='*')
        {
            if ((strncmp(string,"*NAME",5))==0)         //check for name of bar
            {
                ifsin.getline(string,1000,'\n');    //gets next line after identifier
                m_szTName=string;
            }

            if ((strncmp(string,"*GEOMETRY",5))==0)
            {
                ifsin.getline(string,1000,'\n');                //gets bar geomerty
                m_dTDiameter=(atof( strtok( string, seps))) ;
                m_dTLength=(atof( strtok( NULL, seps))) ;
                m_dTDistance=(atof( strtok( NULL, seps)));
            }

            if ((strncmp(string,"*PROPERTIES",7))==0)
            {
                ifsin.getline(string,1000,'\n');                //gets bar properties
                m_dTDensity=(atof( strtok( string, seps))) ;
                m_dTModulus=(atof( strtok( NULL, seps))) ;
                m_dTPRatio=(atof( strtok( NULL, seps))) ;
            }

            if ((strncmp(string,"*FILTER",5))==0)
            {
                ifsin.getline(string,1000,'\n');                //gets bar properties
                m_dTFilter=(atof( strtok( string, seps))) ;
            }
            if ((strncmp(string,"*CALIBRATION",5))==0)
            {   ifsin.getline(string,1000,'\n');
                T.m_dStrain.RemoveAll();
                T.m_dVolts.RemoveAll();
                while(((string[0])!='*')&&(ifsin!=0))
                {   if ((string[0] != '"')&&(string[0]!='\0'))
                    {   T.m_dVolts.Add((atof( strtok( string, seps ))));
                        T.m_dStrain.Add((atof( strtok( NULL, seps )))); //times bye 10^-3 to get i
nto volts
                    }
                    ifsin.getline(string,1000,'\n');
                }
            }
            if ((strncmp(string,"*PROPAGATION",7))==0)
            {   ifsin.getline(string,1000,'\n');
                T.m_dFreq.RemoveAll();
                T.m_dG.RemoveAll();
                while(((string[0])!='*')&&(ifsin!=0))
                {
                    if ((string[0] != '"')&&(string[0]!='\0'))
                    {   T.m_dFreq.Add((atof( strtok( string, seps ))));
                        T.m_dG.Add((atof( strtok( NULL, seps ))));
```

```
                        T.m_dG.Add((atof( strtok( NULL, seps ))));
                    }
                    ifsin.getline(string,1000,'\n');
                }
                m_dTDf=T.m_dFreq[1]-T.m_dFreq[0];
                m_dTDT=1/(T.m_dG.GetSize()*m_dTDf);
                m_bTPropCoeffIncluded=TRUE;
            }
        }
        if(string[0]!='*')
        {ifsin.getline(string,1000,'\n');}
    }   while(ifsin!=0);
    T.m_dFreq.RemoveAll();
    ifsin.close();

    double temp;

    fit(&T.m_dVolts[0],&T.m_dStrain[0],T.m_dStrain.GetSize(),0,0,&temp,&m_dTSlope,0,0,0,0);

    m_bReadTransmittedBar=TRUE;
    return TRUE;

}




void CBarDataDlg::OnOK()
{
    // TODO: Add extra validation here

    CDialog::OnOK();
}
```

```
// ChildFrm.cpp : implementation of the CChildFrame class
//

#include "stdafx.h"
#include "CSHB.h"

#include "ChildFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CChildFrame

IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
    //{{AFX_MSG_MAP(CChildFrame)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CChildFrame construction/destruction

CChildFrame::CChildFrame()
{
    // TODO: add member initialization code here

}

CChildFrame::~CChildFrame()
{
}

BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    if( !CMDIChildWnd::PreCreateWindow(cs) )
        return FALSE;

    return TRUE;
}



/////////////////////////////////////////////////////////////////////////////
// CChildFrame diagnostics

#ifdef _DEBUG
void CChildFrame::AssertValid() const
{
    CMDIChildWnd::AssertValid();
}

void CChildFrame::Dump(CDumpContext& dc) const
{
    CMDIChildWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CChildFrame message handlers
```

```
// ConvertUtilityDlg.cpp : implementation file
//

#include "stdafx.h"
#include "cshb.h"
#include "ConvertUtilityDlg.h"
#include "AmplitudePropDlg.h"
#include "math.h"
#include "fstream.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CConvertUtilityDlg dialog


CConvertUtilityDlg::CConvertUtilityDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CConvertUtilityDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CConvertUtilityDlg)
    m_bAmplitudeShift = FALSE;
    m_bIncludeHeader = FALSE;
    m_szInputFile = _T("");
    m_szOutFile = _T("");
    //}}AFX_DATA_INIT
    m_bAmplitudeShift=TRUE;
    m_bIncludeHeader=TRUE;
}


void CConvertUtilityDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CConvertUtilityDlg)
    DDX_Check(pDX, IDC_AMPLTITUDE, m_bAmplitudeShift);
    DDX_Check(pDX, IDC_INCHEADER, m_bIncludeHeader);
    DDX_Text(pDX, IDC_INPUTFILE, m_szInputFile);
    DDX_Text(pDX, IDC_OUTPUTFILE, m_szOutFile);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CConvertUtilityDlg, CDialog)
    //{{AFX_MSG_MAP(CConvertUtilityDlg)
    ON_COMMAND(ID_UTILITIES_FILECONVERSION, OnUtilitiesFileconversion)
    ON_BN_CLICKED(IDC_INPUTBROWSE, OnInputbrowse)
    ON_BN_CLICKED(IDC_OUPUTBROWSE, OnOuputbrowse)
    ON_BN_CLICKED(IDCONVERT, OnConvert)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CConvertUtilityDlg message handlers

void CConvertUtilityDlg::OnUtilitiesFileconversion()
{
    // TODO: Add your command handler code here

}

void CConvertUtilityDlg::OnInputbrowse()
{
            // Calls the dialog box to select files
    bool stopflag = false;
    OPENFILENAME OpenFileName;
    TCHAR szFile[256000] = "\0";    // File list
    static char szFilter[] ="WFT Files (*.wft)\0*.wft\0"
                            "All Files (*.*)\0*.*\0\0";

    OpenFileName.lStructSize = sizeof(OPENFILENAME);
    OpenFileName.hwndOwner = NULL;
    OpenFileName.hInstance = NULL;
    OpenFileName.lpstrFilter = szFilter;
```

```
    OpenFileName.lpstrCustomFilter = NULL;
    OpenFileName.nMaxCustFilter = 0;
    OpenFileName.nFilterIndex = 0;
    OpenFileName.lpstrFile = szFile;
    OpenFileName.nMaxFile = sizeof(szFile);
    OpenFileName.lpstrFileTitle = NULL;
    OpenFileName.nMaxFileTitle = 0;
    OpenFileName.lpstrInitialDir = NULL;//data1.m_szFilename;
    OpenFileName.lpstrTitle = "Import Data";
    OpenFileName.nFileOffset = 0;
    OpenFileName.nFileExtension = 0;
    OpenFileName.lpstrDefExt = "";
    OpenFileName.lpfnHook = NULL;
    OpenFileName.lpTemplateName = NULL;
    OpenFileName.Flags = OFN_EXPLORER | OFN_FILEMUSTEXIST;

    // Open dialog box
    GetOpenFileName(&OpenFileName);
    // Assign the value of szFile to m_FileList
    m_szInputFile = szFile;
    // Update the information in the textbox to show the current files selected
    CDialog::UpdateData(FALSE);
    m_bReadInput=TRUE;

}

void CConvertUtilityDlg::OnOuputbrowse()
{
        // Calls the dialog box to select files

    bool stopflag = false;
    OPENFILENAME SaveFileName;
    TCHAR szFile[256000] = "\0";    // File list
    static char szFilter[] = "AD Files (*.ad)\0*.ad\0"
                             "Text Files (*.txt)\0*.txt\0"
                             "All Files (*.*)\0*.*\0\0";

    char seps[]   = ".";

    LPTSTR string = new TCHAR[m_szInputFile.GetLength()+1];
    _tcscpy(string, m_szInputFile);

    string=strtok( string, seps );
    strcat(string,".ad");



    LPTSTR szName = string;


        SaveFileName.lStructSize = sizeof(OPENFILENAME);
        SaveFileName.hwndOwner = NULL;
        SaveFileName.hInstance = NULL;
        SaveFileName.lpstrFilter = szFilter;
        SaveFileName.lpstrCustomFilter = NULL;
        SaveFileName.nMaxCustFilter = 0;
        SaveFileName.nFilterIndex = 0;
        SaveFileName.lpstrFile = szName;
        SaveFileName.nMaxFile = sizeof(szFile);
        SaveFileName.lpstrFileTitle = NULL;
        SaveFileName.nMaxFileTitle = 0;
        SaveFileName.lpstrInitialDir =NULL;
        SaveFileName.lpstrTitle = "Export Data to File";
        SaveFileName.nFileOffset = 0;
        SaveFileName.nFileExtension = 0;
        SaveFileName.lpstrDefExt = "";
        SaveFileName.lpfnHook = NULL;
        SaveFileName.lpTemplateName = NULL;
        SaveFileName.Flags = OFN_EXPLORER |OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT;


    // Open dialog box
        GetSaveFileName(&SaveFileName);
        m_szOutFile=szName;
        CDialog::UpdateData(FALSE);
```

```
        m_bReadOutput=TRUE;

}

void CConvertUtilityDlg::OnConvert()
{
    CString strMessage;
    CAmplitudePropDlg dlg;

    UpdateData();
    //check to see if name was typed in
    if(m_bReadInput==FALSE)
    {   if(m_szInputFile!="")
        {m_bReadInput=TRUE;}}
    if(m_bReadOutput==FALSE)
    {   if(m_szOutFile!="")
        {m_bReadOutput=TRUE;}}

    if((m_bReadInput)&&(m_bReadOutput))
    {

        if(m_bAmplitudeShift)
        {
            if (dlg.DoModal()==IDOK)
            {
                m_nZeroPoints=dlg.m_nPoints;
                BeginWaitCursor();
                Convert();
                EndWaitCursor();
            }
        }else
        {
                BeginWaitCursor();
                Convert();
                EndWaitCursor();
        }


    }else{
        strMessage="Both input and output files required";
        MessageBox(strMessage,"Error",MB_ICONERROR);
    }


}

BOOL CConvertUtilityDlg::Convert()
{
    struct file_info{
        int Header_size,File_size,year,month,day;
        int time,data_count,Vertical_zero;
        double Vertical_norm,User_vertical_norm,User_vertical_zero;
        double User_horizontal_zero,User_horizontal_norm;
        CString Title,User_vertical_label,User_horizontal_label;
        CString User_notes,Audit,Nicolet_Digitizer_Type;
        int Bytes_per_data_point,Resolution,Process_flag,Data_compression;
        int Number_of_Segments,Length_of_Segments,Number_timebases,Length_Zone1;
        double Horz_norm_Zone1,Horz_zero_Zone1;
        int Time_Domain;
    }file;


    char string[1750];
    int i=0;
    char highbyte[1],lowbyte[1];
    CArray <double,double> xpoint;
    CArray <double,double> ypoint;
    CString strMessage;
    int number;
    int hour=0,minute=0,second=0;
    double dzero=0;



    ifstream binary(m_szInputFile,ios::nocreate | ios::binary);
```

```
if (!binary)
{
    strMessage="Could Not Open Input File";
    MessageBox(strMessage,"Error",MB_ICONERROR);
    return 0;
}

binary.getline(string,1750,'\n');

file.Time_Domain=atoi(GetField(string,4,2));
file.Header_size=atoi(GetField(string,8,12));
file.File_size=atoi(GetField(string,20,12));

file.year=atoi(GetField(string,125,3));
file.month=atoi(GetField(string,128,3));
file.day=atoi(GetField(string,131,3));
file.time=atoi(GetField(string,134,12));//divide by 1e3 to get into seconds

file.data_count=atoi(GetField(string,146,12));

file.Vertical_zero=atoi(GetField(string,158,12));
file.Vertical_norm=atof(GetField(string,170,24));
file.User_vertical_zero=atof(GetField(string,194,24));
file.User_vertical_norm=atof(GetField(string,218,24));
file.User_vertical_label=GetField(string,242,11);

file.Horz_zero_Zone1=atof(GetField(string,1060,24));
file.Horz_norm_Zone1=atof(GetField(string,1036,24));
file.User_horizontal_zero=atof(GetField(string,253,24));
file.User_horizontal_norm=atof(GetField(string,277,24));
file.User_horizontal_label=GetField(string,301,11);

file.Title=GetField(string,44,81);

file.User_notes=GetField(string,312,129);
file.Audit=GetField(string,441,196);
file.Nicolet_Digitizer_Type=GetField(string,637,21);

file.Bytes_per_data_point=atoi(GetField(string,658,3));
file.Resolution=atoi(GetField(string,661,3));

file.Process_flag=atoi(GetField(string,826,3));
file.Data_compression=atoi(GetField(string,829,3));
file.Number_of_Segments=atoi(GetField(string,832,12));
file.Length_of_Segments=atoi(GetField(string,844,12));
file.Number_timebases=atoi(GetField(string,856,12));
file.Length_Zone1=atoi(GetField(string,1024,12));



if(file.Time_Domain!=1)
{
    strMessage="Can Only Convert Time Domain Data";
    MessageBox(strMessage,"Error",MB_ICONERROR);

    return 0;
}

if(file.Data_compression!=0)
{
    strMessage="Data Has Been Compressed";
    MessageBox(strMessage,"Warning",MB_ICONEXCLAMATION );

    return 0;
}
if(file.Number_of_Segments!=1)
{
    strMessage="Will Only Convert Segment 1";
    MessageBox(strMessage,"Warning",MB_ICONEXCLAMATION );

}
if(file.Process_flag!=0)
{
    strMessage="Warning:Data Has Been Processed on Oscilliscope";
    MessageBox(strMessage,"Warning",MB_ICONEXCLAMATION );
```

```
}

binary.seekg(file.Header_size);



binary.read(lowbyte, 1);
binary.read(highbyte, 1);
i=0;
do{
    number=0x00;

    number=(BYTE(highbyte[0])<<8)+BYTE(lowbyte[0]);
    if (number>=int(pow(2,15)))
    {
        number-=int(pow(2,16));
    }
    xpoint.Add(((i*file.Horz_norm_Zone1)+file.Horz_zero_Zone1)*file.User_horizontal_norm+
    file.User_horizontal_zero);
    ypoint.Add(((number-file.Vertical_zero)*file.Vertical_norm)*file.User_vertical_norm+
    file.User_vertical_zero);

    binary.read(lowbyte, 1);
    binary.read(highbyte, 1);
    i++;

}while (i<file.Length_Zone1);


binary.close();

ofstream outfile(m_szOutFile,ios::out);

if (!binary)
{
    strMessage="Could Not Open Output File";
    MessageBox(strMessage,"Error",MB_ICONERROR);
    return 0;
}

if (m_bAmplitudeShift)
{//pAmplitudeShift->GetCheck()){
    if (m_nZeroPoints!=0)
    {   for (i=0;i<m_nZeroPoints;i++)
        {
            dzero+=ypoint[i];
        }
        dzero/=m_nZeroPoints;
    }else
    {   dzero=0;}
}else
{   dzero=0;
}

if (m_bIncludeHeader==TRUE){
    hour=int(file.time/3600e3);
    minute=int((file.time-hour*3600e3)/60e3);
    second=int((file.time-hour*3600e3-minute*60e3)/1e3);
    outfile<<"\"File:\t"<<m_szInputFile<<endl;
    outfile<<"\"Date:\t";
    outfile.fill('0');
    outfile.width(2);

    outfile<<file.year<<"/";
            outfile.fill('0');
    outfile.width(2);

    outfile<<file.month<<"/";
            outfile.fill('0');
    outfile.width(2);

    outfile <<file.day<<"\t";
            outfile.fill('0');
    outfile.width(2);
```

```
        outfile<<hour<<":";
                outfile.fill('0');
        outfile.width(2);

        outfile<<minute<<":";
                outfile.fill('0');
        outfile.width(2);

        outfile<<second<<endl;
        outfile<<"\"Number of Data Points:\t"<<file.data_count<<endl;
        outfile<<"\"Vertical Label:\t"<<file.User_vertical_label<<endl;
        outfile<<"\"Horizontal Label:\t"<<file.User_horizontal_label<<endl;
        outfile<<"\"Channel Title:\t"<<file.User_notes<<endl;
        outfile<<"\"Data Shifted by:\t"<<dzero<<endl;
        outfile<<"\"Time(s)\t"<<file.User_vertical_label<<endl;
    }

    for (i=0;i<file.Length_Zone1;i++)
    {
        outfile<<xpoint[i]<<"\t"<<ypoint[i]-dzero<<endl;
    }
    outfile.close();
    MessageBox("Conversion Completed","Done");
    return TRUE;
}

CString CConvertUtilityDlg::GetField(char *string, int offset, int size)
{
    CString combine;
    int i=0;

    for (i=offset;i<(offset+size);i++)
    {
        combine+=string[i];
    }

    return combine;
}

void CConvertUtilityDlg::OnCancel()
{
    // TODO: Add extra cleanup here

    CDialog::OnCancel();
}
```

```
// CSHB.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "CSHB.h"

#include "MainFrm.h"
#include "ChildFrm.h"
#include "CSHBDoc.h"
#include "CSHBView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCSHBApp

BEGIN_MESSAGE_MAP(CCSHBApp, CWinApp)
    //{{AFX_MSG_MAP(CCSHBApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCSHBApp construction

CCSHBApp::CCSHBApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CCSHBApp object

CCSHBApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CCSHBApp initialization

BOOL CCSHBApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();         // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();   // Call this when linking to MFC statically
#endif

    // Change the registry key under which our settings are stored.
    // TODO: You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));

    LoadStdProfileSettings();  // Load standard INI file options (including MRU)

    // Register the application's document templates.  Document templates
    //  serve as the connection between documents, frame windows and views.

    CMultiDocTemplate* pDocTemplate;
```

```
    pDocTemplate = new CMultiDocTemplate(
        IDR_CSHBTYPE,
        RUNTIME_CLASS(CCSHBDoc),
        RUNTIME_CLASS(CChildFrame), // custom MDI child frame
        RUNTIME_CLASS(CCSHBView));
    AddDocTemplate(pDocTemplate);

    // create main MDI Frame window
    CMainFrame* pMainFrame = new CMainFrame;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    m_pMainWnd = pMainFrame;

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The main window has been initialized, so show and update it.
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();

    return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
```

```
void CCSHBApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CCSHBApp message handlers
```

```
// CSHBDoc.cpp : implementation of the CCSHBDoc class
//

#include "stdafx.h"
#include "CSHB.h"
#include "Data.h"       //I added


//include "recipes.cpp"
#include "CSHBDoc.h"
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include "BarDataDlg.h"
#include "SampleDataDlg.h"
#include "ExportDataDlg.h"
#include "AnalyseDlg.h"
#include "SeparateValuesDlg.h"
#include "IndexDlg.h"
#include "PropCoeffDlg.h"
#include "FFTUtilityDlg.h"
#include "InputFileDlg.h"
#include "ConvertUtilityDlg.h"
#include "PropagateWaveUtilityDlg.h"
#include <afxdisp.h>    //for COleDateTime

#define     PI      3.14159265359
#define     e       2.718281828459045

#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr;

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCSHBDoc

IMPLEMENT_DYNCREATE(CCSHBDoc, CDocument)

BEGIN_MESSAGE_MAP(CCSHBDoc, CDocument)
    //{{AFX_MSG_MAP(CCSHBDoc)
    ON_COMMAND(ID_FILE_READ_INCIDENT, OnFileReadIncident)
    ON_COMMAND(ID_FILE_READ_TRANSMITTED, OnFileReadTransmitted)
    ON_COMMAND(ID_BAR_PROPERTIES, OnBarProperties)
    ON_COMMAND(ID_SAMPLE_PROPERTIES, OnSampleProperties)
    ON_COMMAND(ID_EXPORT, OnExport)
    ON_COMMAND(ID_ANALYSE_CALCULATERESULTS, OnAnalyseCalculateresults)
    ON_COMMAND(ID_UTILITIES_FFT, OnUtilitiesFft)
    ON_COMMAND(ID_UTILITIES_FILECONVERSION, OnUtilitiesFileconversion)
    ON_COMMAND(ID_TEMP_SHIT, OnTempShit)
    ON_COMMAND(ID_FILE_NEWSAMPLE, OnFileNewsample)
    ON_COMMAND(ID_PRE_SEPERATEWAVE, OnSeperatewave)
    ON_UPDATE_COMMAND_UI(ID_ANALYSE_CALCULATERESULTS, OnUpdateCalculateResults)
    ON_COMMAND(ID_ANALYSE_PROPCALC, OnAnalysePropagationCalculation)
    ON_UPDATE_COMMAND_UI(ID_ANALYSE_PROPCALC, OnUpdateAnalysePropcalc)
    ON_UPDATE_COMMAND_UI(ID_PRE_SEPERATEWAVE, OnUpdatePreSeperatewave)
    ON_COMMAND(ID_UTILITIES_PROPAGATEWAVE, OnUtilitiesPropagatewave)
    ON_UPDATE_COMMAND_UI(ID_UTILITIES_PROPAGATEWAVE, OnUpdateUtilitiesPropagatewave)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCSHBDoc construction/destruction

CCSHBDoc::CCSHBDoc()
{
    // TODO: add one-time construction code here

}

CCSHBDoc::~CCSHBDoc()
```

```
{
}

BOOL CCSHBDoc::OnNewDocument()
{
    InitializeValues();

    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    return TRUE;
}



/////////////////////////////////////////////////////////////////////////////
// CCSHBDoc serialization

void CCSHBDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}

/////////////////////////////////////////////////////////////////////////////
// CCSHBDoc diagnostics

#ifdef _DEBUG
void CCSHBDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CCSHBDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CCSHBDoc commands




void CCSHBDoc::OnFileReadIncident()
{
        CInputFileDlg dlg;
    int i=0;


    if (dlg.DoModal()==IDOK)
    {
        if((flag.m_bProcessedResults)||(incidentbar.Velocity.m_bCalculated)||(transmittedbar.Veloc
ity.m_bCalculated))
        {flag.m_bRecalculate=TRUE;}
        flag.m_bSeparatedIncident=FALSE;
        data1.m_szFilename=dlg.m_szFileName;
        data1.m_szComment="";
        data1.m_dmaxy=-100000000;
        data1.m_dminy=1000000000;
        flag.m_bSeparatedIncident=FALSE;
        data1.m_dtime.RemoveAll();
        data1.m_dvolts.RemoveAll();

        data1.m_dDT=dlg.Data.m_dDT;
```

```
        for(i=0;i<dlg.Data.m_dvolts.GetSize();i++)
        {
            data1.m_dvolts.Add(dlg.Data.m_dvolts[i]);
            if ((data1.m_dvolts[i])>(data1.m_dmaxy))
            {data1.m_dmaxy=data1.m_dvolts[i];}
            if ((data1.m_dvolts[i]<data1.m_dminy))
            {data1.m_dminy=data1.m_dvolts[i];}

        }
        dlg.Data.m_dvolts.RemoveAll();
        if((dlg.m_bAmplitudeShift)&&(dlg.m_nZeroPoints>0))
        {
            for (i=0;i<dlg.m_nZeroPoints;i++)
            {data1.m_dzero+=data1.m_dvolts[i];}
            data1.m_dzero/=dlg.m_nZeroPoints;
        }else
        {
            data1.m_dzero=0;
        }

        flag.m_bReadIncident=TRUE;
        UpdateAllViews(NULL);
        data1.m_dtDate=dlg.Data.m_dtDate;
    }



//  ReadIncidentData();

}

void CCSHBDoc::OnFileReadTransmitted()
{
    CInputFileDlg dlg;
    int i=0;


    if (dlg.DoModal()==IDOK)
    {
        if((flag.m_bProcessedResults)||(incidentbar.Velocity.m_bCalculated)||(transmittedbar.Veloc
ity.m_bCalculated))
        {flag.m_bRecalculate=TRUE;}
        data2.m_szFilename=dlg.m_szFileName;
        data2.m_szComment="";
        data2.m_dmaxy=-100000000;
        data2.m_dminy=1000000000;
        flag.m_bSeparatedTransmitted=FALSE;
        data2.m_dtime.RemoveAll();
        data2.m_dvolts.RemoveAll();

        data2.m_dDT=dlg.Data.m_dDT;

        for(i=0;i<dlg.Data.m_dvolts.GetSize();i++)
        {
            data2.m_dvolts.Add(dlg.Data.m_dvolts[i]);
            if ((data2.m_dvolts[i])>(data2.m_dmaxy))
            {data2.m_dmaxy=data2.m_dvolts[i];}
            if ((data2.m_dvolts[i]<data2.m_dminy))
            {data2.m_dminy=data2.m_dvolts[i];}

        }
        dlg.Data.m_dvolts.RemoveAll();
        if((dlg.m_bAmplitudeShift)&&(dlg.m_nZeroPoints>0))
        {
            for (i=0;i<dlg.m_nZeroPoints;i++)
            {data2.m_dzero+=data2.m_dvolts[i];}
            data2.m_dtDate=dlg.Data.m_dtDate;
            data2.m_dzero/=dlg.m_nZeroPoints;
        }else
        {
            data2.m_dzero=0;
        }
        flag.m_bReadTransmitted=TRUE;
        flag.m_bSeparatedTransmitted=FALSE;
        UpdateAllViews(NULL);
```

```
    }

}


BOOL CCSHBDoc::Volts_to_Strain(double in[],double out[], int n, double slope, double *max, double
*min)
{
    int i=0;

    *max=-1e20;
    *min=1e20;
    for (i=0;i<n;i++)
    {   out[i]=(in[i]*slope);
        if (out[i]>*max)
        {*max=out[i];}
        if (out[i]<*min)
        {*min=out[i];}
    }

    return TRUE;
}

void CCSHBDoc::CalculateStrainRate()
{
    double dv_real=0,dv_imag=0,strain_rate_real=0,strain_rate_imag=0;
    int i=0,N=0;
    sample.m_dengstrainrate.RemoveAll();

    if((m_nFourierAnalysis==0)&&(transmittedbar.Velocity.m_bCalculated)&&
        (incidentbar.Velocity.m_bCalculated))
    {   //Dispersion Included
        for (i=0;i<incidentbar.m_dVelocity.GetSize();i+=2)
        {
            sample.m_dengstrainrate.Add(incidentbar.m_dVelocity[i]-
                transmittedbar.m_dVelocity[i]);
            sample.m_dengstrainrate.Add(incidentbar.m_dVelocity[i+1]-
                transmittedbar.m_dVelocity[i+1]);
        }
        //change to time domain
        Realft(&sample.m_dengstrainrate[0]-1,sample.m_dengstrainrate.GetSize(),
            &sample.m_bCalculatedStrainRate);
        for (i=0;i<sample.m_dengstrainrate.GetSize();i++)
        {
            sample.m_dengstrainrate[i]=sample.m_dengstrainrate[i]/sample.m_dLo;
        }
    }else{
        for (i=0;i<reflected.m_dstrain.GetSize();i++)
        {
            strain_rate_real=(-2*incidentbar.m_dCo)/(sample.m_dLo)*reflected.m_dstrain[i];
            sample.m_dengstrainrate.Add(strain_rate_real);
        }
    }

    //find maximum and minimum strain rate
    sample.m_dmax_strainrate=-1e20;
    sample.m_dmin_strainrate=1e20;
    for (i=0;i<sample.m_dengstrainrate.GetSize();i++)
    {   if (sample.m_dengstrainrate[i]>sample.m_dmax_strainrate)
            {sample.m_dmax_strainrate=sample.m_dengstrainrate[i];}
        if (sample.m_dengstrainrate[i]<sample.m_dmin_strainrate)
            {sample.m_dmin_strainrate=sample.m_dengstrainrate[i];}
    }
    sample.m_bCalculatedStrainRate=TRUE;
}

void CCSHBDoc::CalculateStrain()
{
    double strain=0,integral=0;
    int i=0;
    sample.m_dengstrain.RemoveAll();
    sample.m_dmax_strain=-100000000000000;
    sample.m_dmin_strain=100000000000;


    if (m_nFourierAnalysis==0)      //Dispersion Included
```

```
    {   //integrate strain rate to find strain
        sample.m_dengstrain.Add(0);
        for (i=1;i<sample.m_dengstrainrate.GetSize();i++)
        {
            integral+=(data1.m_dDT)*            //trapeziodal integration
                (sample.m_dengstrainrate[i]+sample.m_dengstrainrate[i-1])/2;
            strain=integral;
            sample.m_dengstrain.Add(strain);
            if (strain>sample.m_dmax_strain)
                {sample.m_dmax_strain=strain;}
            if (strain<sample.m_dmin_strain)
                {sample.m_dmin_strain=strain;}
        }


    }else{
        sample.m_dengstrain.Add(0);
        //integrate reflected strain to find strain
        for (i=1;i<reflected.m_dstrain.GetSize();i++)
        {
            integral+=(data1.m_dDT)*            //trapeziodal integration
                (reflected.m_dstrain[i]+reflected.m_dstrain[i-1])/2;

            strain=-(2*incidentbar.m_dCo/sample.m_dLo)*integral;
            sample.m_dengstrain.Add(strain);
            if (strain>sample.m_dmax_strain)
                {sample.m_dmax_strain=strain;}
            if (strain<sample.m_dmin_strain)
                {sample.m_dmin_strain=strain;}

        }
    }

    sample.m_bCalculatedStrain=TRUE;

}

void CCSHBDoc::CalculateStress()
{
    double stress_real=0,stress_imag=0,dForce_real=0,dForce_imag=0;
    int i=0,N=0;

    sample.m_dengstress.RemoveAll();

    if((m_nFourierAnalysis==0)&&(incidentbar.Force.m_bCalculated)
        &&(transmittedbar.Force.m_bCalculated))
    {   //Dispersion Included
        for (i=0;i<incidentbar.m_dForce.GetSize();i+=2)
        {

            sample.m_dengstress.Add(transmittedbar.m_dForce[i]);
            sample.m_dengstress.Add(transmittedbar.m_dForce[i+1]);
        }

        //change to time domain.
        Realft(&sample.m_dengstress[0]-1,sample.m_dengstress.GetSize()
            ,&sample.m_bCalculatedStress);

        for (i=0;i<incidentbar.m_dForce.GetSize();i++)
        { sample.m_dengstress[i]=sample.m_dengstress[i]/(sample.m_dArea);
        }

    }else{
        for (i=0;i<(transmitted.m_dstrain.GetSize());i++)//transmitted.m_nend-transmitted.m_nbegin
);i++)
        {
            stress_real=incidentbar.m_dE*(incidentbar.m_dArea/sample.m_dArea)*transmitted.m_dstrai
n[i];
            sample.m_dengstress.Add(stress_real);
        }
    }

    //find max and min stress
    sample.m_dmax_stress=-1e20;
    sample.m_dmin_stress=1e20;
```

```
    for (i=0;i<sample.m_dengstress.GetSize();i++)
    {
        if (sample.m_dengstress[i]>sample.m_dmax_stress)
            {sample.m_dmax_stress=sample.m_dengstress[i];}
        if (sample.m_dengstress[i]<sample.m_dmin_stress)
            {sample.m_dmin_stress=sample.m_dengstress[i];}
    }
    sample.m_bCalculatedStress=TRUE;

}

void CCSHBDoc::OnBarProperties()
{
    CBarDataDlg dlg;
    int i=0;
    //gets current values
    dlg.m_dIModulus=incidentbar.m_dE;
    dlg.m_dIDiameter=incidentbar.m_dD;
    dlg.m_dILength=incidentbar.m_dL;
    dlg.m_dIDistance=incidentbar.m_dDistance;
    dlg.m_dIPRatio=incidentbar.m_dPratio;
    dlg.m_dIDensity=incidentbar.m_dDensity;
    dlg.m_dISlope=incidentbar.m_dSlope;

    dlg.m_szIFilename=incidentbar.m_szFilename;
    dlg.m_szIName=incidentbar.m_szname;
    dlg.m_bIPropCoeffIncluded=incidentbar.m_bPropCoeffIncluded;
    dlg.m_dIFilter=incidentbar.m_dFilter;

    dlg.m_dTModulus=transmittedbar.m_dE;
    dlg.m_dTDiameter=transmittedbar.m_dD;
    dlg.m_dTLength=transmittedbar.m_dL;
    dlg.m_dTDistance=transmittedbar.m_dDistance;
    dlg.m_dTPRatio=transmittedbar.m_dPratio;
    dlg.m_dTDensity=transmittedbar.m_dDensity;
    dlg.m_dTSlope=transmittedbar.m_dSlope;


    dlg.m_szTFilename=transmittedbar.m_szFilename;
    dlg.m_szTName=transmittedbar.m_szname;
    dlg.m_bTPropCoeffIncluded=transmittedbar.m_bPropCoeffIncluded;
    dlg.m_dTFilter=transmittedbar.m_dFilter;

    if (dlg.DoModal()==IDOK)
    {

        // Retrieve Dialog Data
        if(dlg.m_bReadIncidentBar){
            flag.m_bReadIncidentBar=TRUE;
            incidentbar.m_dE=dlg.m_dIModulus;
            incidentbar.m_dD=dlg.m_dIDiameter;
            incidentbar.m_dL=dlg.m_dILength;
            incidentbar.m_dDistance=dlg.m_dIDistance;
            incidentbar.m_dPratio=dlg.m_dIPRatio;
            incidentbar.m_dDensity=dlg.m_dIDensity;
            incidentbar.m_dCo=sqrt(incidentbar.m_dE/incidentbar.m_dDensity);
            incidentbar.m_dArea=pow(incidentbar.m_dD,2)*PI/4;
            incidentbar.m_dSlope=dlg.m_dISlope;

            incidentbar.m_szFilename=dlg.m_szIFilename;
            incidentbar.m_szname=dlg.m_szIName;
            incidentbar.m_bPropCoeffIncluded=dlg.m_bIPropCoeffIncluded;
            incidentbar.m_dDf=dlg.m_dIDf;
            incidentbar.m_dDT=dlg.m_dIDT;
            incidentbar.m_dFilter=dlg.m_dIFilter;

            incidentbar.m_dG.RemoveAll();
            if (incidentbar.m_bPropCoeffIncluded)
            {   for (i=0;i<dlg.I.m_dG.GetSize();i++)
            {incidentbar.m_dG.Add(dlg.I.m_dG[i]);   }
            }
            dlg.I.m_dG.RemoveAll();
        }

        if(dlg.m_bReadTransmittedBar){
            flag.m_bReadTransmittedBar=TRUE;
```

```
            transmittedbar.m_dE=dlg.m_dTModulus;
            transmittedbar.m_dD=dlg.m_dTDiameter;
            transmittedbar.m_dL=dlg.m_dTLength;
            transmittedbar.m_dDistance=dlg.m_dTDistance;
            transmittedbar.m_dPratio=dlg.m_dTPRatio;
            transmittedbar.m_dDensity=dlg.m_dTDensity;
            transmittedbar.m_dCo=sqrt(transmittedbar.m_dE/transmittedbar.m_dDensity);
            transmittedbar.m_dArea=pow(transmittedbar.m_dD,2)*PI/4;
            transmittedbar.m_dSlope=dlg.m_dTSlope;

            transmittedbar.m_szFilename=dlg.m_szTFilename;
            transmittedbar.m_szname=dlg.m_szTName;
            transmittedbar.m_bPropCoeffIncluded=dlg.m_bTPropCoeffIncluded;
            transmittedbar.m_dDf=dlg.m_dTDf;
            transmittedbar.m_dDT=dlg.m_dTDT;
            transmittedbar.m_dFilter=dlg.m_dTFilter;

            transmittedbar.m_dG.RemoveAll();
            if (transmittedbar.m_bPropCoeffIncluded)
            {   for (i=0;i<dlg.T.m_dG.GetSize();i++)
            {transmittedbar.m_dG.Add(dlg.T.m_dG[i]);}
            }
            dlg.T.m_dG.RemoveAll();
        }
//set filter frequency to lowest applicable filter


        if((flag.m_bProcessedResults)||(incidentbar.Velocity.m_bCalculated)||(transmittedbar.Veloc
ity.m_bCalculated))
        {flag.m_bRecalculate=TRUE;}
    }

}

 void CCSHBDoc::OnSampleProperties()
{
    CSampleDataDlg dlg;
    //get current values
    //dlg.m_date=sample.m_myDate;
    dlg.m_dDf=sample.m_dDf*1000;
    dlg.m_dDo=sample.m_dDo*1000;
    dlg.m_dLf=sample.m_dLf*1000;
    dlg.m_dLo=sample.m_dLo*1000;

    dlg.m_dPressure=sample.m_dPressure;
    dlg.m_dVelocity=sample.m_dVelocity;
    dlg.m_dStrikerLength=sample.m_dStrikerLength;
    dlg.m_dtIDate=data1.m_dtDate;
    dlg.m_dtTDate=data2.m_dtDate;
    dlg.m_szName=sample.m_szName;
    if ((flag.m_bReadIncident)&&(!flag.m_bReadSampleData))
    {dlg.m_dtDate=data1.m_dtDate;}
    else if((flag.m_bReadTransmitted)&&(!flag.m_bReadSampleData))
    {dlg.m_dtDate=data2.m_dtDate;}
    else if(flag.m_bReadSampleData)
    {dlg.m_dtDate=sample.m_dtDate;}
    else                                // no valid date so choose current
    {       dlg.m_dtDate= COleDateTime::GetCurrentTime();
    }

    if (dlg.DoModal()==IDOK)
    {

        // Retrieve Dialog Data
            sample.m_dtDate=dlg.m_dtDate;
            sample.m_dDf=dlg.m_dDf/1000;
            sample.m_dLf=dlg.m_dLf/1000;
            sample.m_dDo=dlg.m_dDo/1000;
            sample.m_dLo=dlg.m_dLo/1000;
            sample.m_szName=dlg.m_szName;
            sample.m_dtDate=dlg.m_dtDate;
            sample.m_dArea=pow(sample.m_dDo,2)*PI/4;
            sample.m_dStrikerLength=dlg.m_dStrikerLength;
            sample.m_dVelocity=dlg.m_dVelocity;
            sample.m_dPressure=dlg.m_dPressure;
```

```
        if ((fabs(sample.m_dDo)>5e-7)&&(fabs(sample.m_dLo)>5e-7))
        {
            flag.m_bReadSampleData=TRUE;
        }else
        {   AfxMessageBox("Do and Lo Must be Greater Than 5e-7.\n Sample Data Not Valid.",MB_ICONE
RROR);
            flag.m_bReadSampleData=FALSE;}

        if((flag.m_bProcessedResults)||(incidentbar.Velocity.m_bCalculated)||(transmittedbar.Veloc
ity.m_bCalculated))
        {flag.m_bRecalculate=TRUE;}

    }

    UpdateAllViews(NULL);

}




void CCSHBDoc::FFT(double data[], unsigned long nn, int isign)
{
    //This alogrithm is taken from Numerical Recipes in C 2nd Edition pg507
    //isign=1 replaces data by discrete Fourier Transform or if isign=-1 inverse of Fourier transf
orm
    //data is a real array of  length 2*nn
    //nn must be an integer power of 2 Check for before entering
    unsigned long n, a,mmax,m,j,istep,i;
    double wtemp,wr,wpr,wpi,wi,theta;
    double tempr,tempi;


    n=nn<<1;
    j=1;
    for (i=1;i<n;i+=2){                         //does bit reversal
        if (j>i){
            SWAP(data[j],data[i]);
            SWAP(data[j+1],data[i+1]);
        }
        m=n>>1;
        while (m>=2&&j>m){
            j-=m;
            m>>=1;
        }
        j+=m;
    }

    mmax=2;
    while (n>mmax){                             //outer Loop executed log(2)nn times
        istep=mmax<<1;
        theta=isign*(2*PI/mmax);    //Initialize trigonometric recurrence
        wtemp=sin(0.5*theta);
        wpr=-2.0*wtemp*wtemp;
        wpi=sin(theta);
        wr=1.0;
        wi=0.0;

        for (m=1;m<mmax;m+=2){
            for(i=m;i<=n;i+=istep){
                j=i+mmax;
                tempr=wr*data[j]-wi*data[j+1];
                tempi=wr*data[j+1]+wi*data[j];
                data[j]=data[i]-tempr;
                data[j+1]=data[i+1]-tempi;
                data[i]+=tempr;
                data[i+1]+=tempi;
            }
            wr=(wtemp=wr)*wpr-wi*wpi+wr;
            wi=wi*wpr+wtemp*wpi+wi;
        }
        mmax=istep;
    }
    if (isign==-1){
        for (a=1;a<=n;a++)
```

```
        {data[a]=data[a]/nn;}
    }


}




void CCSHBDoc::OnExport()
{

    Export();
}


void CCSHBDoc::Realft(double data[], unsigned long n,BOOL *Time)
{
//FFT for real data. Taken from Numerical Recipes in C 2nd edition pg 513
// a more efficient way of transforming data when it is real
//multiply by 2/n for inverse tranform ?


    unsigned long a,i,i1,i2,i3,i4,np3;
    double c1=0.5,c2,h1r,h1i,h2r,h2i;
    double wr,wi,wpr,wpi,wtemp,theta;
    int isign;

    if (!*Time){
        isign=-1;
        for (a=2;a<=n;a+=2)
        {data[a]=-data[a];}
        *Time=TRUE;
    }else
    {
        isign=1;
        *Time=FALSE;
    }

    theta=PI/(double) (n>>1);
    if (isign==1){

        c2=-0.5;
        FFT(data,n>>1,1);
    }else{
        c2=0.5;
        theta=-theta;
    }

    wtemp=sin(0.5*theta);
    wpr=-2.0*wtemp*wtemp;
    wpi=sin(theta);
    wr=1.0+wpr;
    wi=wpi;
    np3=n+3;
    for (i=2;i<=(n>>2);i++){
        i4=1+(i3=np3-(i2=1+(i1=i+i-1)));
        h1r=c1*(data[i1]+data[i3]);
        h1i=c1*(data[i2]-data[i4]);
        h2r=-c2*(data[i2]+data[i4]);
        h2i=c2*(data[i1]-data[i3]);
        data[i1]=h1r+wr*h2r-wi*h2i;
        data[i2]=h1i+wr*h2i+wi*h2r;
        data[i3]=h1r-wr*h2r+wi*h2i;
        data[i4]=-h1i+wr*h2i+wi*h2r;
        wr=(wtemp=wr)*wpr-wi*wpi+wr;
        wi=wi*wpr+wtemp*wpi+wi;
    }
    if(isign==1){
        data[1]=(h1r=data[1])+data[2];
        data[2]=h1r-data[2];
    }else {
        data[1]=c1*((h1r=data[1])+data[2]);
        data[2]=c1*(h1r-data[2]);
        FFT(data,n>>1,-1);
```

```
    }
    if (isign==1){
        for (a=2;a<=n;a+=2)
        {data[a]=(-data[a]);}
    }
}


void CCSHBDoc::Cdiv(double r_a, double i_a, double r_b, double i_b, double *r_c, double *i_c)
{
//Complex Division
//c=a/b
    double r,den;

    if ((fabs(r_b)>=fabs(i_b))&&(r_b!=0||i_b!=0)){
        if (r_b!=0)
        {   r=i_b/r_b;
            den=r_b+r*i_b;
            *r_c=(r_a+r*i_a)/den;
            *i_c=(i_a-r*r_a)/den;
        }else
        {   *r_c=i_a/i_b;
            *i_c=-r_a/i_b;
        }

    }else if ((fabs(r_b)<fabs(i_b))&&(r_b!=0||i_b!=0))
    {
        if(i_b!=0)
        {   r=r_b/i_b;
            den=i_b+r*r_b;
            *r_c=(r_a*r+i_a)/den;
            *i_c=(i_a*r-r_a)/den;
        }else
        {   *r_c=r_a/r_b;
            *i_c=i_a/r_b;
        }
    }else
    {   //if b=0 then set c=0
        *r_c=0;
        *i_c=0;
    }


}

void CCSHBDoc::Cmul(double r_a, double i_a, double r_b, double i_b, double *r_c, double *i_c)
{
//Complex multiplication
//c=a*b
    *r_c=r_a*r_b-i_a*i_b;
    *i_c=i_a*r_b+r_a*i_b;

}


void CCSHBDoc::InitializeValues()
{
        OnFileNewsample();
        /*sample.m_dDo=0;
        sample.m_dLo=0;
        sample.m_szName="";
        sample.m_dArea=0;
        sample.m_dVelocity=0;
        sample.m_dPressure=0;
*/
        sample.m_dStrikerLength=0;
        incidentbar.m_dE=0;
        incidentbar.m_dD=0;
        incidentbar.m_dL=0;
        incidentbar.m_dDistance=0;
        incidentbar.m_dPratio=0;
        incidentbar.m_dDensity=0;
        incidentbar.m_dCo=0;
        incidentbar.m_dArea=0;
        incidentbar.m_dSlope=0;
        incidentbar.m_dIntercept=0;
```

```
        incidentbar.m_szFilename="";
        incidentbar.m_bPropCoeffIncluded=FALSE;
        incidentbar.m_szname="";
        incidentbar.m_dFilter=0;


        transmittedbar.m_dE=0;
        transmittedbar.m_dD=0;
        transmittedbar.m_dL=0;
        transmittedbar.m_dDistance=0;
        transmittedbar.m_dPratio=0;
        transmittedbar.m_dDensity=0;
        transmittedbar.m_dCo=0;
        transmittedbar.m_dArea=0;
        transmittedbar.m_dSlope=0;
        transmittedbar.m_dIntercept=0;
        transmittedbar.m_szFilename="";
        transmittedbar.m_bPropCoeffIncluded=FALSE;
        transmittedbar.m_szname="";
        transmittedbar.m_dFilter=0;

        m_dFrequencyFilter=5e3;
        flag.m_bReadIncidentBar=FALSE;
        flag.m_bReadTransmittedBar=FALSE;

}




BOOL CCSHBDoc::GetBeginEnd(BOOL Pulse,double y[],double max, double min, double zero,int n, int *b
egin, int *end,int *begin2, int *end2, int nIntercept)
{
    int last_index=0,zero_index=0,lower_bound=0;
    double threshold=0,threshold2=0,Tolerance=0;
    int i=0,j=0,a=0,b=0;

    double difference=0;
    double slope[2];
    int offset=0;
    CArray <double,double> dpeak;
    CArray <int,int> npeakindex;
    CArray <double,double> dtruepeak;
    CArray <int,int> ntruepeakindex;



    dpeak.RemoveAll();
    npeakindex.RemoveAll();
    dtruepeak.RemoveAll();
    ntruepeakindex.RemoveAll();
////////////////////////////Peak Finding Section//////////////////////
////
//Find the begining index.
    threshold=max*trigger.m_dthreshold1_value;      //set threshold to cross to 10% of max

                                            //the first pulse must cross this threshold
    i=0;
    Tolerance=max*trigger.m_dtolerance_value;

    do{
        i++;
    }while ((i<n)&&(y[i]<threshold));


//  zero=0;
    do{
        i--;
    }while ((i>0)&&(y[i]>zero));   //reads values back from threshold until zero

    *begin=i;       // the begining index is a peak (a valley)


    if(nIntercept==0) //find where curves intercept axis
    { //intercept method assumes both curves go through zero at beginning and end points
```

```
        do
        {
            i++;
        }while ((i<n)&&(y[i]>zero));
        *end=i;

        if(!Pulse)
        {
            do
            {
                i++;
            }while ((i<n)&&(y[i]>min*trigger.m_dthreshold1_value));
            lower_bound=i;
            do
            {
                i--;
            }while ((i>0)&&(i>*end)&&(y[i]<zero));
            *begin2=i;
            i=lower_bound;
            do
            {
                i++;
            }while ((i<n)&&(y[i]<zero));
            *end2=i;
        }
// some error checking
        if(*begin>n)
        {*begin=n;}
        if(*begin2>n)
        {*begin2=n;}
        if(*end>n)
        {
            *end=n;
        }else if(*end<*begin)
        {
            *end=*begin;
        }

        if(*end2>n)
        {
            *end2=n;
        }else if(*end2<*begin2)
        {
            *end2=*begin2;
        }



        return TRUE;


    }


    dpeak.Add(y[*begin]);
    npeakindex.Add(*begin);


    a=1;
    for (i=*begin;i<n;i++)
    {   slope[0]=y[i]-y[i-1];
        slope[1]=y[i-1]-y[i-2];

        if(((slope[0]>=0)&&(slope[1]<0))||((slope[0]<=0)&&(slope[1]>0)))
        {       dpeak.Add(y[i-1]);
                npeakindex.Add(i-1);
                a++;
        }

    }

//last point of data is a peak
    dpeak.Add(y[i-1]);
    npeakindex.Add(i-1);
    offset=0;
    for (i=2;i<(dpeak.GetSize()-offset-1);i++)
```

```
    {

        dpeak[i]=dpeak[i+offset];
        npeakindex[i]=npeakindex[i+offset];

        if (fabs(dpeak[i]-dpeak[i-1])<2*Tolerance)
        {
            a=0;
            while ((fabs(dpeak[i+offset]-dpeak[i-1])<2*Tolerance)
                &&((i+offset)<(dpeak.GetSize()-1)))
            {
                    offset++;
                    if ((dpeak[i-1]>dpeak[i-2])&&       //maximum
                        (dpeak[i+offset]>dpeak[i-1]))
                    {

                        dpeak[i-1]=dpeak[i+offset];
                        npeakindex[i-1]=npeakindex[i+offset];
                    }else if((dpeak[i-1]<dpeak[i-2])&&  //minimum
                        (dpeak[i+offset]<dpeak[i-1]))
                    {

                        dpeak[i-1]=dpeak[i+offset];
                        npeakindex[i-1]=npeakindex[i+offset];
                    }

            }//end while
            dpeak[i]=dpeak[i+offset];
            npeakindex[i]=npeakindex[i+offset];
        }//end if

    }


    dtruepeak.Add(dpeak[0]);        //assume first peak is an actual peak
    ntruepeakindex.Add(npeakindex[0]);



    for (i=2;i<(dpeak.GetSize()-offset);i++){               //makes sure that they are all peaks a
nd not inflection points
        if( ((dpeak[i]>dpeak[i-1])&&(dpeak[i-2]>dpeak[i-1]))
            ||  ((dpeak[i]<dpeak[i-1])&&(dpeak[i-2]<dpeak[i-1])) )
        {   dtruepeak.Add(dpeak[i-1]);
            ntruepeakindex.Add(npeakindex[i-1]);
        }
    }
    dtruepeak.Add(dpeak[dpeak.GetSize()-1]);        //assume last peak is an actual peak
    ntruepeakindex.Add(npeakindex[dpeak.GetSize()-1]);
////////////////////////////////Begin and End Finding Section////////////


    if (Pulse)          // finding transmitted wave
    {
            i=1;
            do
            {
                i++;
            }while ((dtruepeak[i]>=zero)&&(i<(dtruepeak.GetSize()-1)));

            last_index=i;
        //all values after  y goes through zero are ignored
        //compares difference of peaks and next valley to original
            i=0;
            difference=dtruepeak[1]-dtruepeak[0];
            do
            {
                i+=2; //look at valleys only

            }while ((i<=last_index)&&((dtruepeak[i-1]-dtruepeak[i])<
                (difference)*trigger.m_dthreshold2_value));

            *end=ntruepeakindex[i];


            if(y[*end]<0)
```

```
            {
                i=ntruepeakindex[i-1];
                do
                {i++;
                }while (y[i]>zero);
                *end=i;

            }
            *end2=0;
            *begin2=0;
            return TRUE;
    }else  //find incident and reflected wave
    {
//checks to see if wave goes through zero

            i=1;
            do
            {
                i++;

            }while ((dtruepeak[i]>zero)&&(i<dtruepeak.GetSize()));


            zero_index=i;

//all values after  y goes through zero are ignored
//compares difference of peaks and next valley to original
            difference=dtruepeak[1]-dtruepeak[0];
            i=0;
            do
            {
                i+=2; //look at valleys only
            }while ((i<=zero_index)&&((dtruepeak[i-1]-dtruepeak[i])<
                (difference)*trigger.m_dthreshold2_value));
            last_index=i;//last peak that is possibly part of the incident pulse
            *end=ntruepeakindex[i];

//if valley is below zero then find the point where the curve crosses zero
            if(y[*end]<0)
            {
                i=ntruepeakindex[i-1];
                do
                {i++;
                }while (y[i]>zero);
                *end=i;
            }
            //*end is end of incident pulse

            threshold=min*trigger.m_dthreshold1_value;      //set threshold to cross to 10% of min



            i=last_index;       //start checking from end of incident pulse
            while ((i<(dtruepeak.GetSize()-1))&&(dtruepeak[i]>threshold))
            {

                i++;
            }
//i is the index of the peak that exceed the threshold
            lower_bound=ntruepeakindex[i-1];
            j=ntruepeakindex[i];
//check to see if curve passes through zero before hitting valley
            while ((j>=lower_bound)&&(y[j]<0)&&(j>0)){
                j--;
            }
            *begin2=j;
            dtruepeak[i-1]=y[j];
            ntruepeakindex[i-1]=j;
            difference=dtruepeak[i]-dtruepeak[i-1];


            //start look at valleys at begining of reflected pulse
            i-=1;// start at zero point on reflected pulse
            do
            {
                i+=2; //look at valleys only
```

```
            }while ((i<(dtruepeak.GetSize()))&& (fabs(dtruepeak[i-1]-dtruepeak[i])< fabs(differenc
e*trigger.m_dthreshold2_value)));

            *end2=ntruepeakindex[i];
            if(y[*end2]>0)
            {
                i=ntruepeakindex[i-1];
                do
                {i++;
                }while (y[i]<zero);
                *end2=i;

            }
        // some error checking
        if(*begin>n)
        {*begin=n;}
        if(*begin2>n)
        {*begin2=n;}
        if(*end>n)
        {
            *end=n;
        }else if(*end<*begin)
        {
            *end=*begin;
        }

        if(*end2>n)
        {
            *end2=n;
        }else if(*end2<*begin2)
        {
            *end2=*begin2;
        }



    return TRUE;
    }//end else


}



double CCSHBDoc::arctan(double imag, double real)
{

    double angle=0,ans=0;
    if ((real!=0)&&(imag!=0)){
        angle=((atan(imag/real)));
        if ((real>0)&&(imag>0)) //first quadrant
        {ans=angle;}
        else if((real<0)&&(imag>0)) //second quadrant
        {ans=PI+angle;}
        else if((real<0)&&(imag<0)) //third quadrant
        {ans=angle-PI;}
        else if((real>0)&&(imag<0)) //fourth quadrant
        {ans=angle;}
    }else{
        if(real==0)
        {
            if (imag<0)
            { ans=-PI/2;}
            else
            {ans=PI/2;}
        }else if(imag==0)
        {
            if(real<0)
            {ans=PI;}
            else
            {ans=0;}
        }else
        {ans=0;}
    }
```

```
    return (ans);
}


void CCSHBDoc::OnAnalyseCalculateresults()
{

    CalculateResults();
}



BOOL CCSHBDoc::PropagateWave(int data1_n,double data[],double dt, int data2_n,double H[],int sign,
 double x)
{
    // This function corrects the measured wave for dispersion and attenuation.  The wave is corre
cted for frequencies
    // up to m_dDisperion Filter.  The rest of the data is filled with zeros.  When sign=-1 the wa
ve is propagting from
    // the gauge towards the interface, if sign=1 it is propagting from the interface to the gauge
.  x is the distance
    // from the gauge to the interface.
    // data MUST BE IN POLAR FORM.  Not checked for.
    int i=0;
    int N=0;
    double fo;
    fo=1/(data1_n*dt); //base frequency


    N=int(m_dFrequencyFilter/fo);//number of point to filter to
    //this should be needless as the filter shoud be less than the number of points in the TF
    if(2*N>data1_n)
    {   N=data1_n/2;}
    else if (2*N>data2_n)
    {   N=data2_n/2;}


    for (i=0;i<(2*N);i+=2)              //apply the dispersion correction to data (replaces data)
    {

        data[i]=data[i]*pow(e,(sign*H[i]*x));
        data[i+1]=data[i+1]+sign*H[i+1]*x;
    }

    for(i=2*N;i<data1_n;i+=2)       //fill the rest of the data with zeros
    {   data[i]=0;
        data[i+1]=0;
    }




    return TRUE;
}



BOOL CCSHBDoc::ChangeForm(int N,double data[],BOOL *polar)
{
    double r=0,modulus=0,angle=0;
    int i=0,a=0;

    if(!*polar)     //Change rectangular polar coordinates
    {
        for (i=0;i<N;i+=2)
        {
            if (fabs(data[i])>=fabs(data[i+1]))
            {   if(data[i]==0)
                {
                    modulus=fabs(data[i+1]);
                }else {
                    r=data[i+1]/data[i];
                    modulus=fabs(data[i])*sqrt(1+r*r);
                }
            }else
```

```
            {   if(data[i+1]==0)
                {
                    modulus=fabs(data[i]);
                }else{
                    r=data[i]/data[i+1];
                    modulus=fabs(data[i+1])*sqrt(1+r*r);
                }
            }


            angle=arctan(data[i+1],data[i]);
            data[i]=modulus;
            data[i+1]=angle;
        }

        a=0;
//unwraps phase spectra
        if(data[1]>PI)
        {   do{
                a--;
            }while ((data[1]+a*2*PI)>PI);
        }
        else if(data[1]<-PI)
        {
            do{
                a++;
            }while ((data[1]+a*2*PI)>PI);
        }
        data[1]+=a*2*PI;

        for(i=3;i<N;i+=2)
        {
            if (fabs((data[i]+a*2*PI)-data[i-2])>PI)
            {   if((data[i]+a*2*PI)<data[i-2])
                {   do{
                        a++;
                    }while(fabs((data[i]+a*2*PI)-data[i-2])>PI);
                }
                else
                {   do{
                        a--;
                    }while(fabs((data[i]+a*2*PI)-data[i-2])>PI);
                }
            }
            data[i]+=a*2*PI;

        }

        *polar=TRUE;
    }else{              //change polar into rectangular

        for (i=0;i<N;i+=2)
        {
            modulus=data[i];
            angle=data[i+1];
            data[i]=modulus*cos(angle);
            data[i+1]=modulus*sin(angle);
        }
        *polar=FALSE;
    }
    return TRUE;
}

BOOL CCSHBDoc::CalculateVelocity()
{
// This routine calculates the velocity at the interface of both the incident and transmitted
// bars.  The strain data must be in rectangular form
    double num_real=0,num_imag=0,sum=0,w=0;
    int i=0;

    if(m_nFourierAnalysis==0)
    {
        if((flag.m_bIncidentStrain)&&(flag.m_bReflectedStrain)
            &&(!reflected.m_bPolar) &&(!incident.m_bPolar)&&
            (!incidentbar.Velocity.m_bCalculated))
        {
```

```
            w=0;
            sum=0;
            for (i=2;i<incidentbar.m_dVelocity.GetSize();i+=2)
            {
                w+=incidentbar.m_dDf*2*PI;
                Cmul(incidentbar.m_dVelocity[i],incidentbar.m_dVelocity[i+1],0,w,
                    &num_real,&num_imag);
                Cdiv(num_real,num_imag,incidentbar.m_dG[i],incidentbar.m_dG[i+1],
                    &incidentbar.m_dVelocity[i],&incidentbar.m_dVelocity[i+1]);
                sum+=incidentbar.m_dVelocity[i];

            }
            incidentbar.m_dVelocity[0]=-2*sum; //DC Term = sum of all others
            incidentbar.m_dVelocity[1]=0;
            incidentbar.Velocity.m_bCalculated=TRUE;
            incidentbar.Velocity.m_bTime=FALSE;

        }

        if((flag.m_bTransmittedStrain)&&(!transmitted.m_bPolar)&&
            (!transmittedbar.Velocity.m_bCalculated))
        {

            w=0;
            sum=0;
            for (i=2;i<transmittedbar.m_dVelocity.GetSize();i+=2)
            {
                w+=transmittedbar.m_dDf*2*PI;
                Cmul(transmittedbar.m_dVelocity[i],transmittedbar.m_dVelocity[i+1],0,w,
                    &num_real,&num_imag);
                Cdiv(num_real,num_imag,transmittedbar.m_dG[i],transmittedbar.m_dG[i+1],
                    &transmittedbar.m_dVelocity[i],&transmittedbar.m_dVelocity[i+1]);
                sum+=transmittedbar.m_dVelocity[i];

            }
            transmittedbar.m_dVelocity[0]=-2*sum; //DC Term = sum of all others
            transmittedbar.m_dVelocity[1]=0;
            transmittedbar.Velocity.m_bCalculated=TRUE;
            transmittedbar.Velocity.m_bTime=FALSE;
        }
    }else{

        if((flag.m_bIncidentStrain)&&(flag.m_bReflectedStrain)&&
            (!incidentbar.Velocity.m_bCalculated))
        {
            for (i=0;i<incidentbar.m_dVelocity.GetSize();i++)
            {
                incidentbar.m_dVelocity[i]*=sqrt(incidentbar.m_dE/incidentbar.m_dDensity);
            }
            incidentbar.Velocity.m_bCalculated=TRUE;
            incidentbar.Velocity.m_bTime=TRUE;

        }

        if((flag.m_bTransmittedStrain)&&
            (!transmittedbar.Velocity.m_bCalculated))
        {

            for (i=0;i<transmittedbar.m_dVelocity.GetSize();i++)
            {
                transmittedbar.m_dVelocity[i]*=sqrt(transmittedbar.m_dE/transmittedbar.m_dDensity)
;
            }
            transmittedbar.Velocity.m_bCalculated=TRUE;
            transmittedbar.Velocity.m_bTime=TRUE;
        }
    }
    return TRUE;

}
BOOL CCSHBDoc::CalculateDisplacement()
{
    double integral=0;
    int i=0;
    integral=0;
    incidentbar.m_dDisplacement.RemoveAll();
```

```
    transmittedbar.m_dDisplacement.RemoveAll();
    incidentbar.Displacement.m_dmax=-100000000000000;
    incidentbar.Displacement.m_dmin=100000000000;
    transmittedbar.Displacement.m_dmax=-100000000000000;
    transmittedbar.Displacement.m_dmin=100000000000;


    if ((incidentbar.Velocity.m_bCalculated)&&(incidentbar.Velocity.m_bTime))
    {
        incidentbar.m_dDisplacement.Add(0);
        for(i=1;i<incidentbar.m_dVelocity.GetSize();i++)
        {
            integral+=(data1.m_dDT)*            //trapeziodal integration
                (incidentbar.m_dVelocity[i]+incidentbar.m_dVelocity[i-1])/2;

            incidentbar.m_dDisplacement.Add(integral);
            if (integral>incidentbar.Displacement.m_dmax)
            {incidentbar.Displacement.m_dmax=integral;}
            if (integral<incidentbar.Displacement.m_dmin)
            {incidentbar.Displacement.m_dmin=integral;}
        }
        incidentbar.Displacement.m_bCalculated=TRUE;
        incidentbar.Displacement.m_bTime=TRUE;
    }
    integral=0;
    if ((transmittedbar.Velocity.m_bCalculated)&&(transmittedbar.Velocity.m_bTime))
    {
        transmittedbar.m_dDisplacement.Add(0);
        for(i=1;i<transmittedbar.m_dVelocity.GetSize();i++)
        {
            integral+=(data2.m_dDT)*            //trapeziodal integration
                (transmittedbar.m_dVelocity[i]+transmittedbar.m_dVelocity[i-1])/2;

            transmittedbar.m_dDisplacement.Add(integral);
            if (integral>transmittedbar.Displacement.m_dmax)
            {transmittedbar.Displacement.m_dmax=integral;}
            if (integral<transmittedbar.Displacement.m_dmin)
            {transmittedbar.Displacement.m_dmin=integral;}
        }
        transmittedbar.Displacement.m_bCalculated=TRUE;
        transmittedbar.Displacement.m_bTime=TRUE;
    }

    return TRUE;

}
BOOL CCSHBDoc::CalculateForce()
{
    double num_real=0,num_imag=0,den_real=0,den_imag=0;
    double temp=0,sum=0,w=0;
    int i=0;

    if(m_nFourierAnalysis==0)
    {
        if((flag.m_bIncidentStrain)&&(flag.m_bReflectedStrain)
            &&(!reflected.m_bPolar) &&(!incident.m_bPolar)&&
            (!incidentbar.Force.m_bCalculated))
        {
            w=0;
            sum=0;
            for (i=2;i<incidentbar.m_dForce.GetSize();i+=2)
            {
                w+=incidentbar.m_dDf*2*PI;
                temp=-w*w*incidentbar.m_dArea*incidentbar.m_dDensity;

                Cmul(incidentbar.m_dForce[i],incidentbar.m_dForce[i+1],temp,0,
                    &num_real,&num_imag);

                Cmul(incidentbar.m_dG[i],incidentbar.m_dG[i+1],incidentbar.m_dG[i],
                    incidentbar.m_dG[i+1],&den_real,&den_imag);

                Cdiv(num_real,num_imag,den_real,den_imag,
                    &incidentbar.m_dForce[i],&incidentbar.m_dForce[i+1]);

                sum+=incidentbar.m_dForce[i];
```

```
            }

            incidentbar.m_dForce[0]=-2*sum; //D.C. Term =sum of all others
            incidentbar.m_dForce[1]=0;
            incidentbar.Force.m_bCalculated=TRUE;
            incidentbar.Force.m_bTime=FALSE;
        }


        if((flag.m_bTransmittedStrain)&&(!transmitted.m_bPolar)
            &&(!transmittedbar.Force.m_bCalculated))
        {
            w=0;
            sum=0;
            for (i=2;i<transmittedbar.m_dForce.GetSize();i+=2)
            {
                w+=transmittedbar.m_dDf*2*PI;
                temp=-w*w*transmittedbar.m_dArea*transmittedbar.m_dDensity;
                Cmul(transmittedbar.m_dForce[i],transmittedbar.m_dForce[i+1],temp,0,
                    &num_real,&num_imag);

                Cmul(transmittedbar.m_dG[i],transmittedbar.m_dG[i+1],transmittedbar.m_dG[i],
                    transmittedbar.m_dG[i+1],&den_real,&den_imag);

                Cdiv(num_real,num_imag,den_real,den_imag,
                    &transmittedbar.m_dForce[i],&transmittedbar.m_dForce[i+1]);

                sum+=transmittedbar.m_dForce[i];
            }
            transmittedbar.m_dForce[0]=-2*sum;  //D.C. Term =sum of all others
            transmittedbar.m_dForce[1]=0;
            transmittedbar.Force.m_bCalculated=TRUE;
            transmittedbar.Force.m_bTime=FALSE;


        }
    }else
    {
        if((flag.m_bIncidentStrain)&&(flag.m_bReflectedStrain)&&
        (!incidentbar.Force.m_bCalculated))
        {
            for (i=0;i<incidentbar.m_dForce.GetSize();i++)
            {
                incidentbar.m_dForce[i]*=incidentbar.m_dArea*incidentbar.m_dE;
            }
            incidentbar.Force.m_bCalculated=TRUE;
            incidentbar.Force.m_bTime=TRUE;

        }

        if((flag.m_bTransmittedStrain)&&
            (!transmittedbar.Force.m_bCalculated))
        {

            for (i=0;i<transmittedbar.m_dForce.GetSize();i++)
            {
                transmittedbar.m_dForce[i]*=transmittedbar.m_dE*transmittedbar.m_dArea;
            }
            transmittedbar.Force.m_bCalculated=TRUE;
            transmittedbar.Force.m_bTime=TRUE;
        }
    }
    return TRUE;

}



void CCSHBDoc::InterpolateData(int n,double dt_a,double dt_b,double data[])
{//replaces a set of data sampled at dt_b rate with set sampled at dt_a rate
    int i=0,j=0,step=0;
    double r=0;

    CArray <double,double> interp;
    interp.Add(data[0]);    //assume first point
```

```
    r=dt_a/dt_b;

    if (dt_b>dt_a)
    {   step=int(ceil(1/r));}
    else
    {   step=1; }

    i=1;
    j=step;

    while ((j<n)||(i<n))            //j<n for when adding points i<n when subtracting
    {   while ((dt_b*i)<=(dt_a*j))
        {   //interpolates to find y value of b at x value of a
            interp.Add((data[j]-data[j-1])/dt_a*(i*dt_b-(j-1)*dt_a)+data[j-1]);
            i++;
        }
        j+=step;
    }
    j-=step;

    if ((n*dt_b)!=(n*dt_a))
    {interp.Add((data[j]-data[j-1])/dt_a*(i*dt_b-(j-1)*dt_a)+data[j-1]);}//extrapolate last point


    for (i=0;i<n;i++)
    {   data[i]=interp[i];}

    interp.RemoveAll();

}




void CCSHBDoc::OnUtilitiesFft()
{
    CFFTUtilityDlg dlg;
    dlg.DoModal();
    InitializeValues();

}

void CCSHBDoc::OnUtilitiesFileconversion()
{
    CConvertUtilityDlg dlg;
    dlg.DoModal();
}


void CCSHBDoc::OnTempShit()
{
    double data2[25];
    data2[0]=0;
    data2[1]=1;
    data2[2]=2;
    data2[3]=3;
    data2[4]=2;
    data2[5]=1;
    data2[6]=0;
    data2[7]=-1;



    InterpolateData(8,1.5,1,&data2[0]);

/*  Realft(&data2[0]-1,16,1);
    Realft(&data2[0]-1,16,-1);
//  FFT(&data2[0]-1,8,1);
//  FFT(&data2[0]-1,8,-1);
*/
}


BOOL CCSHBDoc::CalculateResults()
{
```

```
    int i=0,power=0,N=0;
    double w=0;


    CAnalyseDlg dlg;
    CString szMessage;
    //initialize data
    dlg.m_dSampDf=sample.m_dDf*1000;    //*1000 to get into meters
    dlg.m_dSampDo=sample.m_dDo*1000;
    dlg.m_dSampLf=sample.m_dLf*1000;
    dlg.m_dSampLo=sample.m_dLo*1000;
    dlg.m_szSampName=sample.m_szName;

    dlg.m_szData_I=data1.m_szFilename;
    dlg.m_szData_T=data2.m_szFilename;
    dlg.m_szCalib_I=incidentbar.m_szFilename;
    dlg.m_szCalib_T=transmittedbar.m_szFilename;

    dlg.m_bIncidentDispersion=incidentbar.m_bPropCoeffIncluded;
    dlg.m_bTransmittedDispersion=transmittedbar.m_bPropCoeffIncluded;



    if ((!flag.m_bReadSampleData)||((!incidentbar.m_bPropCoeffIncluded)&&(!transmittedbar.m_bPropC
oeffIncluded)))
    {   //Initialize dialog data

        dlg.m_nFourierAnalysis=0;
        if((!incidentbar.m_bPropCoeffIncluded)&&(!transmittedbar.m_bPropCoeffIncluded))
        {
            dlg.m_nNyquist=0;}
        else
        {dlg.m_nNyquist=1;}

    }else if((flag.m_bReadSampleData)&&(incidentbar.m_bPropCoeffIncluded)&&(transmittedbar.m_bProp
CoeffIncluded))
    {
        dlg.m_nFourierAnalysis=0;   //use Fourier analysis
        dlg.m_nNyquist=1;

    }else
    {
        dlg.m_nFourierAnalysis=1;
    }
//set filter to lowest applicable filter
    if(incidentbar.m_bPropCoeffIncluded||transmittedbar.m_bPropCoeffIncluded)
    {
        if(incidentbar.m_bPropCoeffIncluded&&transmittedbar.m_bPropCoeffIncluded)
        {
            if ((incidentbar.m_dFilter>=transmittedbar.m_dFilter) && (transmittedbar.m_dFilter>0))
            {
                m_dFrequencyFilter=transmittedbar.m_dFilter;
            }else if((incidentbar.m_dFilter<transmittedbar.m_dFilter) && (incidentbar.m_dFilter>0)
)
            {
                m_dFrequencyFilter=incidentbar.m_dFilter;
            }else
            {
                m_dFrequencyFilter=10000;
            }
        }else
        {
            if (incidentbar.m_bPropCoeffIncluded)
            {
                m_dFrequencyFilter=incidentbar.m_dFilter;
            }else
            {
                m_dFrequencyFilter=transmittedbar.m_dFilter;
            }

        }
    }
    dlg.m_dFreqFilter=m_dFrequencyFilter;

    if (dlg.DoModal()==IDOK)
    {
```

```
        sample.m_szName=dlg.m_szSampName;
//      sample.m_dDf=dlg.m_dSampDf/1000;    //// 1000 to get into meters
//      sample.m_dDo=dlg.m_dSampDo/1000;
//      sample.m_dLf=dlg.m_dSampLf/1000;
//      sample.m_dLo=dlg.m_dSampLo/1000;

        m_nFourierAnalysis=dlg.m_nFourierAnalysis;


        if ((!flag.m_bSeparatedIncident)&&(!flag.m_bSeparatedTransmitted))
        {   AfxMessageBox("Raw Data Not Separated.\nUnable to process results.",MB_ICONERROR);
            return 0;
        }


        if  ((!flag.m_bSeparatedIncident&&!flag.m_bReadTransmittedBar)||
            (!flag.m_bSeparatedTransmitted&&!flag.m_bReadIncidentBar)||
            (!flag.m_bReadIncidentBar&&!flag.m_bReadTransmittedBar)||
            (!flag.m_bSeparatedTransmitted&&!flag.m_bSeparatedIncident))

        {   AfxMessageBox("Bar Data Not Read.\nUnable to process results.",MB_ICONERROR);
            return 0;
        }
        if ((fabs(incidentbar.m_dDf-transmittedbar.m_dDf)>5e-6)&&(incidentbar.m_bPropCoeffIncluded
)
            &&(transmittedbar.m_bPropCoeffIncluded))
        {
            AfxMessageBox("Incident and transmitter bar propagation coefficients have different in
crements in frequency.\nUnable to process results.",MB_ICONERROR);
            return 0;
        }
        if(!flag.m_bReadSampleData)
        {AfxMessageBox("Sample Data Not Read.\nOnly force and velocity data will be calculated.",M
B_ICONINFORMATION);}



        BeginWaitCursor();


//      data1.m_szComment="";
//      data2.m_szComment="";
        incident.m_dstrain.RemoveAll();
        reflected.m_dstrain.RemoveAll();
        transmitted.m_dstrain.RemoveAll();
        incidentbar.m_dVelocity.RemoveAll();
        transmittedbar.m_dVelocity.RemoveAll();
        incidentbar.m_dForce.RemoveAll();
        transmittedbar.m_dForce.RemoveAll();
        incidentbar.Force.m_bCalculated=FALSE;
        incidentbar.Velocity.m_bCalculated=FALSE;
        transmittedbar.Force.m_bCalculated=FALSE;
        transmittedbar.Velocity.m_bCalculated=FALSE;
        sample.m_bCalculatedStrainRate=FALSE;
        sample.m_bCalculatedStrain=FALSE;
        sample.m_bCalculatedStress=FALSE;
        flag.m_bIncidentStrain=FALSE;
        flag.m_bReflectedStrain=FALSE;
        flag.m_bTransmittedStrain=FALSE;

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////// Incident Bar Data Calculations
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////

        if((flag.m_bReadIncident)&&(flag.m_bReadIncidentBar)&&(flag.m_bSeparatedIncident))
        {
            data1.m_szComment="";
            if(m_nFourierAnalysis==0){  //include dispersion
                if(((fabs(data1.m_dDT-incidentbar.m_dDT)>5e-8)||
                    (data1.m_dvolts.GetSize()!=incidentbar.m_dG.GetSize()))
                    &&
                    (incidentbar.m_bPropCoeffIncluded))
```

```
{
    //compensate indices for new points
    incident.m_nbegin=int(incident.m_nbegin*data1.m_dDT/incidentbar.m_dDT);
    incident.m_nend=int(incident.m_nend*data1.m_dDT/incidentbar.m_dDT);
    reflected.m_nbegin=int(reflected.m_nbegin*data1.m_dDT/incidentbar.m_dDT);
    reflected.m_nend=int(reflected.m_nend*data1.m_dDT/incidentbar.m_dDT);
    // some error checking
    if (incident.m_nbegin>incidentbar.m_dG.GetSize())
    {   incident.m_nbegin=incidentbar.m_dG.GetSize();}

    if (reflected.m_nbegin>incidentbar.m_dG.GetSize())
    {   reflected.m_nbegin=incidentbar.m_dG.GetSize();}

    if (incident.m_nend>incidentbar.m_dG.GetSize())
    {   incident.m_nend=incidentbar.m_dG.GetSize();}

    if (reflected.m_nend>incidentbar.m_dG.GetSize())
    {   reflected.m_nend=incidentbar.m_dG.GetSize();}

    if(incidentbar.m_dDT>data1.m_dDT)
    {   N=int(ceil(incidentbar.m_dG.GetSize()*incidentbar.m_dDT/data1.m_dDT));
    if(N>data1.m_dvolts.GetSize())
    {   //in case you want more points than you have are needed
        data1.m_dvolts.InsertAt(data1.m_dvolts.GetUpperBound(),0,
            N-data1.m_dvolts.GetSize());
        data1.m_szComment+="had points added; ";
    }

    N=incidentbar.m_dG.GetSize();
    if (fabs(data1.m_dDT-incidentbar.m_dDT)>1e-8)
    {InterpolateData(N,data1.m_dDT,incidentbar.m_dDT,
    &data1.m_dvolts[0]);
    data1.m_szComment+="was changed to a slower sample rate; ";
    }

    //remove points
    if ((N-data1.m_dvolts.GetSize())>0)
    {
        data1.m_dvolts.RemoveAt(N,abs(N-data1.m_dvolts.GetSize()));
        data1.m_dvolts.FreeExtra();
        data1.m_szComment+="had points removed; ";
    }
    data1.m_dDT=incidentbar.m_dDT;//new sample rate
    data1.m_szComment+="to match bar dispersion data; ";

    }else
    {//addpoints
        N=incidentbar.m_dG.GetSize();
        if(N>data1.m_dvolts.GetSize())
            //here it is necessary since adding points
        {
            data1.m_dvolts.InsertAt(data1.m_dvolts.GetUpperBound(),0,
                N-data1.m_dvolts.GetSize());
            data1.m_szComment+="had points added; ";
        }

        if (fabs(data1.m_dDT-incidentbar.m_dDT)>1e-8)
        {
            InterpolateData(N,data1.m_dDT,incidentbar.m_dDT,&data1.m_dvolts[0]);
            data1.m_szComment+="was changed to a faster sample rate; ";
        }
        data1.m_dDT=incidentbar.m_dDT;
        if(data1.m_dvolts.GetSize()>incidentbar.m_dG.GetSize())
        {
            data1.m_dvolts.RemoveAt(incidentbar.m_dG.GetSize(),
                data1.m_dvolts.GetSize()-incidentbar.m_dG.GetSize());
            data1.m_dvolts.FreeExtra();
            data1.m_szComment+="had points removed; ";
        }
        data1.m_szComment+="to match bar dispersion data; ";
    }

}else if    (incidentbar.m_bPropCoeffIncluded)
{
    data1.m_szComment+="matched bar dispersion data; ";
}
```

```
N=data1.m_dvolts.GetSize(); //number of points
power=int((log(N)/log(2)));
if (N>int(pow(2,power)))//if the number of points is not a power of 2
{power++;}
N=int(pow(2,power));

incident.m_dstrain.InsertAt(0,0,N);
reflected.m_dstrain.InsertAt(0,0,N);

incident.m_dDf=1/(incident.m_dstrain.GetSize()*data1.m_dDT);
reflected.m_dDf=1/(reflected.m_dstrain.GetSize()*data1.m_dDT);


if(!incidentbar.m_bPropCoeffIncluded)   //if dispersion is not included propragate
waves with C
{
    incidentbar.m_dDf=incident.m_dDf;
    incidentbar.m_dDT=data1.m_dDT;
    incidentbar.m_dG.RemoveAll();
    w=incident.m_dDf*2*PI;
    incidentbar.m_dG.Add(0);
    incidentbar.m_dG.Add(0);
    w=0;

    for(i=2;i<incident.m_dstrain.GetSize();i+=2)
    {
        w+=incident.m_dDf*2*PI;
        incidentbar.m_dG.Add(0);
        incidentbar.m_dG.Add(w/sqrt(incidentbar.m_dE/incidentbar.m_dDensity));
    }
}


Volts_to_Strain(&data1.m_dvolts[incident.m_nbegin],
    &incident.m_dstrain[incident.m_nbegin],
    (incident.m_nend-incident.m_nbegin),incidentbar.m_dSlope,
    &incident.m_dmaxstrain,&incident.m_dminstrain);
Volts_to_Strain(&data1.m_dvolts[reflected.m_nbegin],
    &reflected.m_dstrain[reflected.m_nbegin],
    (reflected.m_nend-reflected.m_nbegin),incidentbar.m_dSlope,
    &incident.m_dmaxstrain,&incident.m_dminstrain);
incident.m_bTime=TRUE;
flag.m_bIncidentStrain=TRUE;
reflected.m_bTime=TRUE;
flag.m_bReflectedStrain=TRUE;

Realft(&incident.m_dstrain[0]-1,incident.m_dstrain.GetSize()
    ,&incident.m_bTime);
Realft(&reflected.m_dstrain[0]-1,reflected.m_dstrain.GetSize()
    ,&reflected.m_bTime);

incident.m_bPolar=FALSE;
reflected.m_bPolar=FALSE;

//Change to polar form for Dispersion Correction
if(dlg.m_nNyquist!=0)
{
    m_dFrequencyFilter=dlg.m_dFreqFilter;
}
else
{   //calculate values up to the nyquist frequency
    //Propagate wave checks values so don't do it here
    m_dFrequencyFilter=1/(2*incidentbar.m_dDT);  //Nyquist Critical Frequency
}
ChangeForm(incident.m_dstrain.GetSize(),
    &incident.m_dstrain[0],&incident.m_bPolar);
ChangeForm(reflected.m_dstrain.GetSize(),
    &reflected.m_dstrain[0],&reflected.m_bPolar);
PropagateWave(incident.m_dstrain.GetSize(),&incident.m_dstrain[0],
    data1.m_dDT,incidentbar.m_dG.GetSize(),
    &incidentbar.m_dG[0],-1,incidentbar.m_dDistance);
PropagateWave(reflected.m_dstrain.GetSize(),&reflected.m_dstrain[0],
    data1.m_dDT,incidentbar.m_dG.GetSize(),
    &incidentbar.m_dG[0],1,incidentbar.m_dDistance);
//Calculate force and velocity for each bar at interface
```

```
            //change into rectangular form
            ChangeForm(incident.m_dstrain.GetSize(),
                &incident.m_dstrain[0],&incident.m_bPolar);
            ChangeForm(reflected.m_dstrain.GetSize(),
                &reflected.m_dstrain[0],&reflected.m_bPolar);

            for (i=0;i<incident.m_dstrain.GetSize();i++) //initialize arrays
            {
                incidentbar.m_dVelocity.Add(incident.m_dstrain[i]-reflected.m_dstrain[i]);
                incidentbar.m_dForce.Add(incident.m_dstrain[i]+reflected.m_dstrain[i]);
            }

        }else{ //conventional analysis

            if ((incident.m_nend-incident.m_nbegin)>(reflected.m_nend-reflected.m_nbegin))
            {   N=(incident.m_nend-incident.m_nbegin);
            }else
            {   N=(reflected.m_nend-reflected.m_nbegin);
            }

            if(flag.m_bReadTransmitted)
            {   if((transmitted.m_nend-transmitted.m_nbegin)>N)
            {N=(transmitted.m_nend-transmitted.m_nbegin);}
            }


            incident.m_dstrain.InsertAt(0,0,N);
            reflected.m_dstrain.InsertAt(0,0,N);

            Volts_to_Strain(&data1.m_dvolts[incident.m_nbegin],
                &incident.m_dstrain[0],
                (incident.m_nend-incident.m_nbegin),
                incidentbar.m_dSlope,&incident.m_dmaxstrain,&incident.m_dminstrain);
            Volts_to_Strain(&data1.m_dvolts[reflected.m_nbegin],
                &reflected.m_dstrain[0],
                (reflected.m_nend-reflected.m_nbegin),
                incidentbar.m_dSlope,&incident.m_dmaxstrain,&incident.m_dminstrain);

            incident.m_bTime=TRUE;
            flag.m_bIncidentStrain=TRUE;
            reflected.m_bTime=TRUE;
            flag.m_bReflectedStrain=TRUE;

            for (i=0;i<incident.m_dstrain.GetSize();i++) //initialize arrays
            {   incidentbar.m_dVelocity.Add(incident.m_dstrain[i]-reflected.m_dstrain[i]);
            incidentbar.m_dForce.Add(incident.m_dstrain[i]+reflected.m_dstrain[i]);
            }


        }//endif (m_nFourierAnalysis==0)
    }//end if(flag.m_bReadIncident)


////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
//////// Transmitted Bar Data Calculation
////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
    if((flag.m_bReadTransmitted)&&(flag.m_bReadTransmittedBar)&&
        (flag.m_bSeparatedTransmitted))
    {
        if(m_nFourierAnalysis==0){  //include dispersion
            if(((fabs(data2.m_dDT-transmittedbar.m_dDT)>5e-8)||
                (data2.m_dvolts.GetSize()!=transmittedbar.m_dG.GetSize()))
                &&(transmittedbar.m_bPropCoeffIncluded))
            {
                //compensate indices for new points
                transmitted.m_nbegin=int(transmitted.m_nbegin*data2.m_dDT/transmittedbar.m_dDT
);
                transmitted.m_nend=int(transmitted.m_nend*data2.m_dDT/transmittedbar.m_dDT);

                if(transmitted.m_nbegin>transmittedbar.m_dG.GetSize())
                {transmitted.m_nbegin=transmittedbar.m_dG.GetSize();}

                if(transmitted.m_nend>transmittedbar.m_dG.GetSize())
                {transmitted.m_nend=transmittedbar.m_dG.GetSize();}
```

```
                    if(transmittedbar.m_dDT>data2.m_dDT)
                    {   N=int(ceil(transmittedbar.m_dG.GetSize()*transmittedbar.m_dDT/data2.m_dDT)
);
                    if(N>data2.m_dvolts.GetSize())
                    {
                        data2.m_dvolts.InsertAt(data2.m_dvolts.GetUpperBound(),0,
                            N-data2.m_dvolts.GetSize());
                        data2.m_szComment+="had points added; ";
                    }
                    N=transmittedbar.m_dG.GetSize();
                    if (fabs(transmittedbar.m_dDT-data2.m_dDT)>1e-8)
                    {
                        InterpolateData(N,data2.m_dDT,transmittedbar.m_dDT,&data2.m_dvolts[0]);
                        data2.m_szComment+="was changed to a slower sample rate; ";
                    }
                    if ((N-data2.m_dvolts.GetSize())>0)
                    {//remove points
                        data2.m_dvolts.RemoveAt(N,abs(N-data2.m_dvolts.GetSize()));
                        data2.m_dvolts.FreeExtra();
                        data2.m_szComment+="had points removed; ";
                    }
                    data2.m_dDT=transmittedbar.m_dDT;//new sample rate
                    data2.m_szComment+="to match bar dispersion data; ";
                    }
                    else
                    {   ///addpoints
                        N=transmittedbar.m_dG.GetSize();
                        if(N>data2.m_dvolts.GetSize())  //here it is necessary since adding points
                        {   data2.m_dvolts.InsertAt(data2.m_dvolts.GetUpperBound(),0,
                        N-data2.m_dvolts.GetSize());
                        data2.m_szComment+="had points added; ";
                        }

                        if (fabs(transmittedbar.m_dDT-data2.m_dDT)>1e-8)
                        {
                            InterpolateData(N,data2.m_dDT,transmittedbar.m_dDT,&data2.m_dvolts[0])
;
                            data2.m_szComment+="was changed to a faster sample rate; ";
                        }
                        data2.m_dDT=transmittedbar.m_dDT;

                        if(data2.m_dvolts.GetSize()>transmittedbar.m_dG.GetSize())
                        {
                            data2.m_dvolts.RemoveAt(transmittedbar.m_dG.GetSize(),
                                data2.m_dvolts.GetSize()-transmittedbar.m_dG.GetSize());
                            data2.m_dvolts.FreeExtra();
                            data2.m_szComment+="had points removed; ";
                        }


                        data2.m_szComment+="to match bar dispersion data;";
                    }
                }else if(transmittedbar.m_bPropCoeffIncluded)
                {
                    data2.m_szComment+="matched bar dispersion data; ";
                }

                N=data2.m_dvolts.GetSize(); //number of points
                power=int((log(N)/log(2)));
                if (N>int(pow(2,power)))//if the number of points is not a power of 2
                {power++;}
                N=int(pow(2,power));

                transmitted.m_dstrain.InsertAt(0,0,N);

                transmitted.m_dDf=1/(transmitted.m_dstrain.GetSize()*data2.m_dDT);
                if(!transmittedbar.m_bPropCoeffIncluded)    //if dispersion is not included propra
gate waves with C
                {   transmittedbar.m_dDf=transmitted.m_dDf;
                    transmittedbar.m_dDT=data2.m_dDT;
                    transmittedbar.m_dG.RemoveAll();
                    w=transmitted.m_dDf*2*PI;
                    transmittedbar.m_dG.Add(0);
                    transmittedbar.m_dG.Add(0);
```

```
                    w=0;
                    for(i=2;i<transmitted.m_dstrain.GetSize();i+=2)
                    {   w+=transmitted.m_dDf*2*PI;
                        transmittedbar.m_dG.Add(0);
                        transmittedbar.m_dG.Add(w/sqrt(transmittedbar.m_dE/transmittedbar.m_dDensi
ty));
                    }
                }


                Volts_to_Strain(&data2.m_dvolts[transmitted.m_nbegin],
                    &transmitted.m_dstrain[transmitted.m_nbegin],
                    (transmitted.m_nend-transmitted.m_nbegin),
                    transmittedbar.m_dSlope,&transmitted.m_dmaxstrain,&transmitted.m_dminstrain);
                transmitted.m_bTime=TRUE;
                flag.m_bTransmittedStrain=TRUE;


                Realft(&transmitted.m_dstrain[0]-1,transmitted.m_dstrain.GetSize()
                    ,&transmitted.m_bTime);
                transmitted.m_bPolar=FALSE;

                //Change to polar form for Dispersion Correction
                if(dlg.m_nNyquist!=0)
                {   m_dFrequencyFilter=dlg.m_dFreqFilter;}
                else
                {   //calculate values up to the nyquist frequency
                    //Propagate wave checks values so don't do it here
                    m_dFrequencyFilter=1/(2*transmittedbar.m_dDT);  //Nyquist Critical Frequency
                }

                ChangeForm(transmitted.m_dstrain.GetSize(),
                    &transmitted.m_dstrain[0],&transmitted.m_bPolar);
                PropagateWave(transmitted.m_dstrain.GetSize(),&transmitted.m_dstrain[0],
                    data2.m_dDT,transmittedbar.m_dG.GetSize(),
                    &transmittedbar.m_dG[0],1,transmittedbar.m_dDistance);

                //Calculate force and velocity for each bar at interface
                //change into rectangular form
                ChangeForm(transmitted.m_dstrain.GetSize(),
                    &transmitted.m_dstrain[0],&transmitted.m_bPolar);


                for (i=0;i<transmitted.m_dstrain.GetSize();i++) //initialize arrays
                {   transmittedbar.m_dVelocity.Add(transmitted.m_dstrain[i]);
                    transmittedbar.m_dForce.Add(transmitted.m_dstrain[i]);
                }

            }else{ //conventional analysis

                N=(transmitted.m_nend-transmitted.m_nbegin);
                if(flag.m_bReadIncident)
                {   if((transmitted.m_nend-transmitted.m_nbegin)>N)
                    {   if ((incident.m_nend-incident.m_nbegin)>(reflected.m_nend-reflected.m_nbeg
in))
                        {   N=(incident.m_nend-incident.m_nbegin);
                        }else
                        {   N=(reflected.m_nend-reflected.m_nbegin);}
                    }
                }

                transmitted.m_dstrain.InsertAt(0,0,N);
                Volts_to_Strain(&data2.m_dvolts[transmitted.m_nbegin],
                    &transmitted.m_dstrain[0],
                    (transmitted.m_nend-transmitted.m_nbegin),
                    transmittedbar.m_dSlope,&transmitted.m_dmaxstrain,&transmitted.m_dminstrain);
                transmitted.m_bTime=TRUE;
                flag.m_bTransmittedStrain=TRUE;

                for (i=0;i<transmitted.m_dstrain.GetSize();i++) //initialize arrays
                {   transmittedbar.m_dVelocity.Add(transmitted.m_dstrain[i]);
                    transmittedbar.m_dForce.Add(transmitted.m_dstrain[i]);
                }
            }// Include Dispersion

        }//end if (flag.m_bReadTransmitted)
```

```
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////// Bar Data Calculation -> Velocity and Force
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////


        CalculateVelocity();
        CalculateForce();



////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////// Sample Data Calculation -> Stress Strain StrainRate
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
    if ( flag.m_bTransmittedStrain && flag.m_bIncidentStrain &&
        flag.m_bReflectedStrain && flag.m_bReadSampleData)
    {
        CalculateStrainRate();
        CalculateStrain();
        CalculateStress();
    }
    if ((sample.m_bCalculatedStress)&&(sample.m_bCalculatedStrain)&&
        (sample.m_bCalculatedStrainRate))
    {   flag.m_bProcessedResults=TRUE;
    }

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
////////Change Force and Velocity into time domain and find their max and mins for plotting
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////

        if(incidentbar.Velocity.m_bCalculated){
            if (!incidentbar.Velocity.m_bTime)
            {
                Realft(&incidentbar.m_dVelocity[0]-1,incidentbar.m_dVelocity.GetSize(),
                    &incidentbar.Velocity.m_bTime);
            }
            if(!incidentbar.Force.m_bTime)
            {   Realft(&incidentbar.m_dForce[0]-1,incidentbar.m_dForce.GetSize(),
                    &incidentbar.Force.m_bTime);
            }

                incidentbar.Velocity.m_dmax=-1e20;
                incidentbar.Velocity.m_dmin=1e-20;
                incidentbar.Force.m_dmax=-1e20;
                incidentbar.Force.m_dmin=1e20;

                for (i=0;i<incidentbar.m_dForce.GetSize();i++)
                    {if (incidentbar.m_dVelocity[i]>incidentbar.Velocity.m_dmax)
                    {   incidentbar.Velocity.m_dmax=incidentbar.m_dVelocity[i];}
                    if (incidentbar.m_dVelocity[i]<incidentbar.Velocity.m_dmin)
                    {   incidentbar.Velocity.m_dmin=incidentbar.m_dVelocity[i];}

                    if (incidentbar.m_dForce[i]>incidentbar.Force.m_dmax)
                    {   incidentbar.Force.m_dmax=incidentbar.m_dForce[i];}
                    if (incidentbar.m_dForce[i]<incidentbar.Force.m_dmin)
                    {   incidentbar.Force.m_dmin=incidentbar.m_dForce[i];}
                }//end for

        }//end if

        if(transmittedbar.Force.m_bCalculated){
            if(!transmittedbar.Velocity.m_bTime){
                Realft(&transmittedbar.m_dVelocity[0]-1,transmittedbar.m_dVelocity.GetSize(),
                    &transmittedbar.Velocity.m_bTime);
            }
            if(!transmittedbar.Force.m_bTime)
            {
                Realft(&transmittedbar.m_dForce[0]-1,transmittedbar.m_dForce.GetSize(),
                    &transmittedbar.Force.m_bTime);
            }
```

```
                transmittedbar.Velocity.m_dmax=-1e20;
                transmittedbar.Velocity.m_dmin=1e-20;
                transmittedbar.Force.m_dmax=-1e20;
                transmittedbar.Force.m_dmin=1e20;

                for (i=0;i<transmittedbar.m_dForce.GetSize();i++)
                    {if (transmittedbar.m_dVelocity[i]>transmittedbar.Velocity.m_dmax)
                    {   transmittedbar.Velocity.m_dmax=transmittedbar.m_dVelocity[i];}
                    if (transmittedbar.m_dVelocity[i]<transmittedbar.Velocity.m_dmin)
                    {   transmittedbar.Velocity.m_dmin=transmittedbar.m_dVelocity[i];}

                    if (transmittedbar.m_dForce[i]>transmittedbar.Force.m_dmax)
                    {   transmittedbar.Force.m_dmax=transmittedbar.m_dForce[i];}
                    if (transmittedbar.m_dForce[i]<transmittedbar.Force.m_dmin)
                    {   transmittedbar.Force.m_dmin=transmittedbar.m_dForce[i];}
                }
        }//endif

        CalculateDisplacement(); //calcuate the displacement of the bar ends from velocities
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
///////Change propagated waves into time domain and find their max and mins for plotting
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////


        if(flag.m_bIncidentStrain)
        {   if (!incident.m_bTime)
            {   if (incident.m_bPolar)
                {ChangeForm(incident.m_dstrain.GetSize(),&incident.m_dstrain[0],&incident.m_bPolar
);}

                Realft(&incident.m_dstrain[0]-1,incident.m_dstrain.GetSize(),
                    &incident.m_bTime);
            }

            for (i=0;i<incident.m_dstrain.GetSize();i++)
            {
                if(incident.m_dstrain[i]>incident.m_dmaxstrain)
                {incident.m_dmaxstrain=incident.m_dstrain[i];}
                if(incident.m_dstrain[i]<incident.m_dminstrain)
                {incident.m_dminstrain=incident.m_dstrain[i];}
            }
        }

        if(flag.m_bReflectedStrain)
        {   if (!reflected.m_bTime)
            {   if (reflected.m_bPolar)
                {ChangeForm(reflected.m_dstrain.GetSize(),&reflected.m_dstrain[0],&reflected.m_bPo
lar);}

                Realft(&reflected.m_dstrain[0]-1,reflected.m_dstrain.GetSize(),
                    &reflected.m_bTime);
            }

            for (i=0;i<reflected.m_dstrain.GetSize();i++)
            {
                if(reflected.m_dstrain[i]>reflected.m_dmaxstrain)
                {reflected.m_dmaxstrain=reflected.m_dstrain[i];}
                if(reflected.m_dstrain[i]<reflected.m_dminstrain)
                {reflected.m_dminstrain=reflected.m_dstrain[i];}
            }
        }

        if(flag.m_bTransmittedStrain)
        {   if (!transmitted.m_bTime)
            {   if (transmitted.m_bPolar)
                {ChangeForm(transmitted.m_dstrain.GetSize(),&transmitted.m_dstrain[0],&transmitted
.m_bPolar);}

                Realft(&transmitted.m_dstrain[0]-1,transmitted.m_dstrain.GetSize(),
                    &transmitted.m_bTime);
            }
```

```
            for (i=0;i<transmitted.m_dstrain.GetSize();i++)
            {
                if(transmitted.m_dstrain[i]>transmitted.m_dmaxstrain)
                {transmitted.m_dmaxstrain=transmitted.m_dstrain[i];}
                if(transmitted.m_dstrain[i]<transmitted.m_dminstrain)
                {transmitted.m_dminstrain=transmitted.m_dstrain[i];}
            }
        }

        EndWaitCursor();
    }//endif DoModal
    if(!flag.m_bRecalculate)
    {   UpdateAllViews(NULL);}

    flag.m_bRecalculate=FALSE;
    return TRUE;
}

void CCSHBDoc::OnFileNewsample()
{
    //reintialize some of the data values
        sample.m_dDo=0;
        sample.m_dLo=0;
        sample.m_szName="";
        sample.m_dArea=0;
        sample.m_dVelocity=0;
        sample.m_dPressure=0;


        data1.m_dvolts.RemoveAll();
        data2.m_dvolts.RemoveAll();
        data1.m_szFilename="";
        data2.m_szFilename="";
        data1.m_dzero=0;
        data2.m_dzero=0;
        data1.m_szComment="";
        data2.m_szComment="";

        incident.m_dstrain.RemoveAll();
        reflected.m_dstrain.RemoveAll();
        transmitted.m_dstrain.RemoveAll();
        flag.m_bPropagateUtility=FALSE;
        flag.m_bReadIncident=FALSE;
        flag.m_bReadTransmitted=FALSE;
        flag.m_bSeparatedIncident=FALSE;
        flag.m_bSeparatedTransmitted=FALSE;
        flag.m_bProcessedResults=FALSE;
        flag.m_bIncidentStrain=FALSE;
        flag.m_bTransmittedStrain=FALSE;
        flag.m_bReflectedStrain=FALSE;
//      flag.m_bData1Interpolated=FALSE;
//      flag.m_bData2Interpolated=FALSE;
        flag.m_bReadSampleData=FALSE;
        flag.m_bRecalculate=FALSE;

        incidentbar.Velocity.m_bCalculated=FALSE;
        incidentbar.Force.m_bCalculated=FALSE;
        incidentbar.Displacement.m_bCalculated=FALSE;
        transmittedbar.Force.m_bCalculated=FALSE;
        transmittedbar.Velocity.m_bCalculated=FALSE;
        transmittedbar.Displacement.m_bCalculated=FALSE;


        incidentbar.m_dVelocity.RemoveAll();
        incidentbar.m_dForce.RemoveAll();
        transmittedbar.m_dVelocity.RemoveAll();
        transmittedbar.m_dForce.RemoveAll();

        sample.m_bCalculatedStrainRate=FALSE;
        sample.m_bCalculatedStrain=FALSE;
        sample.m_bCalculatedStress=FALSE;
        sample.m_dDf=0;
        sample.m_dLf=0;
        sample.m_dengstress.RemoveAll();
        sample.m_dengstrain.RemoveAll();
        sample.m_dengstrainrate.RemoveAll();
```

```
        trigger.m_dthreshold1_value=0.2;
        trigger.m_dthreshold2_value=0.5;
        trigger.m_dtolerance_value=0.005;
        incident.m_bTime=TRUE;
        reflected.m_bTime=TRUE;
        transmitted.m_bTime=TRUE;

        incident.m_bPolar=FALSE;
        reflected.m_bPolar=FALSE;
        transmitted.m_bPolar=FALSE;

        incident.m_dDf=0;
        reflected.m_dDf=0;
        transmitted.m_dDf=0;

        incident.m_nbegin=0;
        incident.m_nend=0;
        reflected.m_nbegin=0;
        reflected.m_nend=0;
        transmitted.m_nbegin=0;
        transmitted.m_nend=0;

        incident.m_dmaxstrain=0;
        incident.m_dminstrain=0;
        reflected.m_dmaxstrain=0;
        reflected.m_dminstrain=0;
        transmitted.m_dmaxstrain=0;
        transmitted.m_dminstrain=0;


        m_nFourierAnalysis=1;


        UpdateAllViews(NULL);

}

BOOL CCSHBDoc::Export()
{
    CExportDataDlg dlg;
    CString szExportFilename;
    CString szTitle,szUnits,szObject;
    CArray <double,double> dOutput;
    COleDateTime m_dtToday;
    int i=0,N=0,nsize=0,k=0;
    double dmax_stress=0,dmax_strain=0;
    m_dtToday= COleDateTime::GetCurrentTime();

    dlg.m_szExportFileName=".exp"; //set outputfile name to sample name

    //dlg.m_szExportDirectoryName="d:\\export\\";
    //  dlg.m_szFileName=sample.m_szName+".exp"; //set outputfile name to sample name

    dlg.m_bHeader=TRUE;
    if((incident.m_bTime)||(reflected.m_bTime)||(transmitted.m_bTime))
    {   dlg.m_bStrainsAllowed=TRUE; }

    if((sample.m_bCalculatedStrain)&&(sample.m_bCalculatedStress)&&
            (sample.m_bCalculatedStrainRate))
    {
//      dlg.m_bEngineeringData=TRUE;
        dlg.m_bEngineeringData=FALSE;
        dlg.m_bEngineeringAllowed=TRUE;
        dlg.m_bTrueData=FALSE;

    }

    if(((incidentbar.Force.m_bTime)&&(incidentbar.Force.m_bCalculated))||
            ((transmittedbar.Force.m_bTime)&&(transmittedbar.Force.m_bCalculated)))
    {   dlg.m_bForcesAllowed=TRUE;
        dlg.m_bForces=FALSE;
    }
```

```
    if(((incidentbar.Velocity.m_bTime)&&(incidentbar.Velocity.m_bCalculated))||
            ((transmittedbar.Velocity.m_bTime)&&(transmittedbar.Velocity.m_bCalculated)))
    {   dlg.m_bVelocitiesAllowed=TRUE;
        dlg.m_bVelocities=FALSE;
    }




    if (dlg.DoModal()==IDOK)
    {   BeginWaitCursor();
        szExportFilename=dlg.m_szExportFileName;
        dOutput.RemoveAll();
        ofstream expfile(szExportFilename,ios::out);

        if (!expfile)
        {   AfxMessageBox("Can't Create File",MB_ICONHAND);
            return FALSE;
        }

        N=0;
        nsize=0;
        szObject="\t";
        szTitle="Time\t";
        szUnits="s\t";

        if(dlg.m_bTrueData)
        {
            if((sample.m_bCalculatedStrain)&&(sample.m_bCalculatedStress)&&
                (sample.m_bCalculatedStrainRate))
            {   N+=3;
                szObject+="True Data\t\t\t";
                szTitle+="Strain Rate\tTrue Strain\tTrue Stress\t";
                szUnits+="s^-1\tmm/mm\tMPa\t";
                //finds true stress strain curves
                dOutput.Append(sample.m_dengstrainrate);
                nsize=(nsize>sample.m_dengstrainrate.GetSize())?nsize:sample.m_dengstrainrate.GetS
ize();

                dmax_strain=-1e20; //change to true value
                dmax_stress=-1e20; //change to true value

                for (i=0;i<sample.m_dengstrain.GetSize();i++)
                {   dOutput.Add(log(1+sample.m_dengstrain[i]));
                    if(log(1+sample.m_dengstrain[i])>dmax_strain)
                    {dmax_strain=log(1+sample.m_dengstrain[i]);} //change to true value

                }
                for (i=0;i<sample.m_dengstrain.GetSize();i++)
                {   dOutput.Add((sample.m_dengstress[i]*log(1+sample.m_dengstrain[i])/pow(10,6)));

                    if ((sample.m_dengstress[i]*log(1+sample.m_dengstrain[i]))>dmax_stress)
                    {dmax_stress=(sample.m_dengstress[i]*log(1+sample.m_dengstrain[i]));} //change
 to true value

                }
            }
        }

        if(dlg.m_bEngineeringData)
        {
            if((sample.m_bCalculatedStrain)&&(sample.m_bCalculatedStress)&&
                (sample.m_bCalculatedStrainRate))
            {   N+=3;
                szObject+="Engineering Data\t\t\t";
                szTitle+="Strain Rate\tStrain\tStress\t";
                szUnits+="s^-1\tmm/mm\tMPa\t";
                nsize=(nsize>sample.m_dengstrainrate.GetSize())?nsize:sample.m_dengstrainrate.GetS
ize();
                dOutput.Append(sample.m_dengstrainrate);
                dOutput.Append(sample.m_dengstrain);
                for (i=0;i<sample.m_dengstress.GetSize();i++)
                {dOutput.Add(sample.m_dengstress[i]/pow(10,6));}
            }
        }
```

```
        if(dlg.m_bStrains)
        {   if(((incident.m_bTime)&&(reflected.m_bTime)&&(flag.m_bSeparatedIncident))
            &&
            ((transmitted.m_bTime)&&(flag.m_bSeparatedTransmitted)))

        {   szObject+="Strain Waveforms\t\t\t";
                szTitle+="Incident\tReflected\tTransmitted\t";
                szUnits+="mm/mm\tmm/mm\tmm/mm\t";
                N+=3;
                nsize=(nsize>incident.m_dstrain.GetSize())?nsize:incident.m_dstrain.GetSize();
                dOutput.Append( incident.m_dstrain);
                dOutput.Append(reflected.m_dstrain);
                dOutput.Append(transmitted.m_dstrain);
            }else if((incident.m_bTime)&&(reflected.m_bTime)&&(flag.m_bSeparatedIncident))
            {   szObject+="Strain Waveforms\t\t";
                szTitle+="Incident\tReflected\t";
                szUnits+="mm/mm\tmm/mm\t";
                N+=2;
                nsize=(nsize>incident.m_dstrain.GetSize())?nsize:incident.m_dstrain.GetSize();
                dOutput.Append(incident.m_dstrain);
                dOutput.Append(reflected.m_dstrain);
            }else if ((transmitted.m_bTime)&&(flag.m_bSeparatedTransmitted))
            {   szObject+="Strain Waveforms\t";
                szTitle+="Transmitted\t";
                szUnits+="mm/mm\t";
                N++;
                nsize=(nsize>transmitted.m_dstrain.GetSize())?nsize:transmitted.m_dstrain.GetSize(
);
                dOutput.Append(transmitted.m_dstrain);
            }
        }

        if(dlg.m_bForces)
        {
            if((incidentbar.Force.m_bTime)&&(incidentbar.Force.m_bCalculated)&&
            (transmittedbar.Force.m_bTime)&&(transmittedbar.Force.m_bCalculated))
            {   N+=2;
                szObject+="Forces\t\t";
                szTitle+="Incidentbar\tTransmittedbar\t";
                szUnits+="kN\tkN\t";
                nsize=(nsize>incidentbar.m_dForce.GetSize())?nsize:incidentbar.m_dForce.GetSize();
                dOutput.Append(incidentbar.m_dForce);
                dOutput.Append(transmittedbar.m_dForce);

            }else if((incidentbar.Force.m_bTime)&&
                (incidentbar.Force.m_bCalculated))
            {   N++;
                szObject+="Forces\t";
                szTitle+="Incidentbar\t";
                szUnits+="kN\t";
                nsize=(nsize>incidentbar.m_dForce.GetSize())?nsize:incidentbar.m_dForce.GetSize();
                dOutput.Append(incidentbar.m_dForce);

            }else if ((transmittedbar.Force.m_bTime)&&
                (transmittedbar.Force.m_bCalculated))
            {   N++;

                szObject+="Forces\t";
                szTitle+="Transmittedbar\t";
                szUnits+="kN\t";
                nsize=(nsize>transmittedbar.m_dForce.GetSize())?nsize:transmittedbar.m_dForce.GetS
ize();
                dOutput.Append(transmittedbar.m_dForce);


            }
        }

        if(dlg.m_bVelocities)
        {
            if((incidentbar.Velocity.m_bTime)&&(incidentbar.Velocity.m_bCalculated)&&
            (transmittedbar.Velocity.m_bTime)&&(transmittedbar.Velocity.m_bCalculated))
            {   N+=2;
                szObject+="Velocities\t\t";
                szTitle+="Incidentbar\tTransmittedbar\t";
```

```
                szUnits+="m/s\tm/s\t";
                nsize=(nsize>incidentbar.m_dVelocity.GetSize())?nsize:incidentbar.m_dVelocity.GetS
ize();
                dOutput.Append(incidentbar.m_dVelocity);
                dOutput.Append(transmittedbar.m_dVelocity);

            }else if((incidentbar.Velocity.m_bTime)&&
                (incidentbar.Velocity.m_bCalculated))
            {   N++;
                szObject+="Velocity\t";
                szTitle+="Incidentbar\t";
                szUnits+="m/s\t";
                nsize=(nsize>incidentbar.m_dVelocity.GetSize())?nsize:incidentbar.m_dVelocity.GetS
ize();
                dOutput.Append(incidentbar.m_dVelocity);

            }else if ((transmittedbar.Velocity.m_bTime)&&
                (transmittedbar.Velocity.m_bCalculated))
            {   N++;
                szObject+="Velocity\t";
                nsize=(nsize>transmittedbar.m_dVelocity.GetSize())?nsize:transmittedbar.m_dVelocit
y.GetSize();
                szTitle+="Transmittedbar\t";
                szUnits+="m/s\t";
                nsize=(nsize>incidentbar.m_dVelocity.GetSize())?nsize:incidentbar.m_dVelocity.GetS
ize();
                dOutput.Append(transmittedbar.m_dVelocity);

            }
        }

        if(dlg.m_bDisplacements)
        {
            if((incidentbar.Displacement.m_bTime)&&(incidentbar.Displacement.m_bCalculated)&&
            (transmittedbar.Displacement.m_bTime)&&(transmittedbar.Displacement.m_bCalculated))
            {   N+=2;
                szObject+="Displacements\t\t";
                szTitle+="Incidentbar\tTransmittedbar\t";
                szUnits+="m\tm\t";
                nsize=(nsize>incidentbar.m_dDisplacement.GetSize())?nsize:incidentbar.m_dDisplacem
ent.GetSize();
                dOutput.Append(incidentbar.m_dDisplacement);
                dOutput.Append(transmittedbar.m_dDisplacement);

            }else if((incidentbar.Displacement.m_bTime)&&
                (incidentbar.Displacement.m_bCalculated))
            {   N++;
                szObject+="Displacement\t";
                szTitle+="Incidentbar\t";
                szUnits+="m\t";
                nsize=(nsize>incidentbar.m_dDisplacement.GetSize())?nsize:incidentbar.m_dDisplacem
ent.GetSize();
                dOutput.Append(incidentbar.m_dDisplacement);

            }else if ((transmittedbar.Displacement.m_bTime)&&
                (transmittedbar.Displacement.m_bCalculated))
            {   N++;
                szObject+="Displacement\t";
                nsize=(nsize>transmittedbar.m_dDisplacement.GetSize())?nsize:transmittedbar.m_dDis
placement.GetSize();
                szTitle+="Transmittedbar\t";
                szUnits+="m\t";
                nsize=(nsize>incidentbar.m_dDisplacement.GetSize())?nsize:incidentbar.m_dDisplacem
ent.GetSize();
                dOutput.Append(transmittedbar.m_dDisplacement);

            }
        }
        if(dlg.m_bHeader)
        {   expfile<<sample.m_szName<<"\tProperties"<<"\t"<<endl;

            expfile<<"Do=\t"<<sample.m_dDo*1000<<"\tmm Lo=\t"<<sample.m_dLo*1000<<"\tmm"<<endl;
            expfile<<"Df=\t"<<sample.m_dDf*1000<<"\tmm Lf=\t"<<sample.m_dLf*1000<<"\tmm"<<endl;

            expfile<<"Test Date:\t";
            expfile<<sample.m_dtDate.GetMonth()<<"/"<<sample.m_dtDate.GetDay()<<"/";
```

```
            expfile<<sample.m_dtDate.GetYear()<<"\t";
            expfile.width(2);
            expfile.fill('0');
            expfile<<sample.m_dtDate.GetHour()<<":";
            expfile.width(2);
            expfile<<sample.m_dtDate.GetMinute()<<":";
            expfile.width(2);
            expfile<<sample.m_dtDate.GetSecond()<<endl;


            expfile<<"Processing Date:\t";
            expfile<<m_dtToday.GetMonth()<<"/"<<m_dtToday.GetDay()<<"/"<<m_dtToday.GetYear()<<"\t"
;
            expfile.width(2);
            expfile.fill('0');
            expfile<<m_dtToday.GetHour()<<":";
            expfile.width(2);
            expfile<<m_dtToday.GetMinute()<<":";
            expfile.width(2);
            expfile<<m_dtToday.GetSecond()<<endl;
            expfile<<"Input Conditions:\tStrikerVelocity:\t"<<sample.m_dVelocity<<
                "\tm/s\tStriker Length:\t"<<sample.m_dStrikerLength<<
                "\tm\tGun Pressure\t"<<sample.m_dPressure<<"\tpsi"<<endl;

            expfile<<"Data Files"<<endl;
            expfile<<"Incident\t"<<data1.m_szFilename<<"\t"<<data1.m_szComment<<endl;
            expfile<<"Transmitted\t"<<data2.m_szFilename<<"\t"<<data2.m_szComment<<endl;

            expfile<<"Calibration Files"<<endl;
            expfile<<"Incident bar\t"<<incidentbar.m_szFilename<<endl;
            expfile<<"Transmitted bar\t"<<transmittedbar.m_szFilename<<endl;

            expfile<<"Indices\tStart\tEnd\t\tCalculated Maximums"<<endl;
            expfile<<"Incident\t"<<incident.m_nbegin*data1.m_dDT<<"\t"<<incident.m_nend*data1.m_dD
T<<"\t\tMax Strain:\t"<<
                ((dlg.m_bTrueData)?dmax_strain:sample.m_dmax_strain)<<endl;
            expfile<<"Reflected\t"<<reflected.m_nbegin*data1.m_dDT<<"\t"<<reflected.m_nend*data1.m
_dDT<<"\t\tMax Stress:\t"<<
                ((dlg.m_bTrueData)?dmax_stress/pow(10,6):sample.m_dmax_stress/pow(10,6))<<endl;
            expfile<<"Transmitted\t"<<transmitted.m_nbegin*data2.m_dDT<<"\t"<<transmitted.m_nend*d
ata2.m_dDT<<"\t\tMax Strain Rate:\t"<<sample.m_dmax_strainrate<<endl;

        }
        expfile<<szObject<<endl;
        expfile<<szTitle<<endl;
        expfile<<szUnits<<endl;

        //N is the number of columns of output
        //nsize is the number of elements of the columns, assumes all arrays appended to
        //dOutput are of the same size

        for (i=0;i<nsize;i++)
        {   expfile<<i*data1.m_dDT<<"\t";
            for (k=0;k<N;k++)
            {
                expfile<<dOutput[i+k*nsize]<<"\t";
            }
            expfile<<endl;
        }
        dOutput.RemoveAll();

        expfile.close();
        AfxMessageBox("Data Sucessfully Exported",MB_OK);
        EndWaitCursor();
    }// end Do Modal
    return TRUE;
}


CCSHBDoc* CCSHBDoc::GetDoc()
{
   CMDIChildWnd * pChild =
          ((CMDIFrameWnd*)(AfxGetApp()->m_pMainWnd))->MDIGetActive();

      if ( !pChild )
          return NULL;
```

```
      CDocument * pDoc = pChild->GetActiveDocument();

      if ( !pDoc )
         return NULL;

      // Fail if doc is of wrong kind
      if ( ! pDoc->IsKindOf( RUNTIME_CLASS(CCSHBDoc) ) )
         return NULL;

      return (CCSHBDoc *) pDoc;

}

void CCSHBDoc::OnSeperatewave()
{
    int i=0,bi=0,ei=0,br=0,er=0,bt=0,et=0,a=0,b=0;

    CSeparateValuesDlg dlg;

    dlg.m_dthreshold1=trigger.m_dthreshold1_value;
    dlg.m_dthreshold2=trigger.m_dthreshold2_value;
    dlg.m_dtolerance=trigger.m_dtolerance_value;

    if(flag.m_bReadIncident==TRUE)
    {   dlg.m_bSeparateIncident=TRUE;}
    else{
        dlg.m_bSeparateIncident=FALSE;}

    if(flag.m_bReadTransmitted)
    {   dlg.m_bSeparateTransmitted=TRUE;}
    else{
        dlg.m_bSeparateTransmitted=FALSE;}



    if (dlg.DoModal()==IDOK)
    {   if((flag.m_bProcessedResults)||(incidentbar.Velocity.m_bCalculated)||(transmittedbar.Veloc
ity.m_bCalculated))
        {flag.m_bRecalculate=TRUE;}
// Retrieve Dialog Data
        trigger.m_dthreshold1_value=dlg.m_dthreshold1;
        trigger.m_dthreshold2_value=dlg.m_dthreshold2;
        trigger.m_dtolerance_value=dlg.m_dtolerance;
//  Find Incident Wave
        if((dlg.m_bSeparateIncident)&&(flag.m_bReadIncident))
        {   GetBeginEnd(0,&data1.m_dvolts[0],data1.m_dmaxy,data1.m_dminy,
                data1.m_dzero,data1.m_dvolts.GetSize(),&incident.m_nbegin,&incident.m_nend,
                &reflected.m_nbegin,&reflected.m_nend,dlg.m_nIntercept);
        }


        if((dlg.m_bSeparateTransmitted)&&(flag.m_bReadTransmitted))
        {
            GetBeginEnd(1,&data2.m_dvolts[0],data2.m_dmaxy,data2.m_dminy,
                data2.m_dzero,data2.m_dvolts.GetSize(),&transmitted.m_nbegin,&transmitted.m_nend
                ,&a,&b,dlg.m_nIntercept);


        }
//a and b are dummy variables and have no effect on anything


        if(dlg.m_bSeparateIncident)
        {flag.m_bSeparatedIncident=TRUE;}
        if(dlg.m_bSeparateTransmitted)
        {flag.m_bSeparatedTransmitted=TRUE;}

        UpdateAllViews(NULL);

    }//end if
}

void CCSHBDoc::OnUpdateCalculateResults(CCmdUI* pCmdUI)
{
        if  ((!flag.m_bSeparatedIncident&&!flag.m_bReadTransmittedBar)||
```

```
            (!flag.m_bSeparatedTransmitted&&!flag.m_bReadIncidentBar)||
            (!flag.m_bReadIncidentBar&&!flag.m_bReadTransmittedBar)||
            (!flag.m_bSeparatedTransmitted&&!flag.m_bSeparatedIncident))
        {pCmdUI->Enable(FALSE);}
        else
        {pCmdUI->Enable(TRUE);}
}



void CCSHBDoc::OnAnalysePropagationCalculation()
{
    CalculatePropagationCoefficient();
}

void CCSHBDoc::OnUpdateAnalysePropcalc(CCmdUI* pCmdUI)
{
        if  ((flag.m_bSeparatedIncident&&flag.m_bReadIncidentBar)||
        (flag.m_bSeparatedTransmitted&&flag.m_bReadTransmittedBar))
        {pCmdUI->Enable(TRUE);}
        else
        {pCmdUI->Enable(FALSE);}
}

void CCSHBDoc::OnUpdatePreSeperatewave(CCmdUI* pCmdUI)
{
        if(flag.m_bReadIncident||flag.m_bReadTransmitted)
        {pCmdUI->Enable(TRUE);}
        else
        {pCmdUI->Enable(FALSE);}

}

void CCSHBDoc::OnUtilitiesPropagatewave()
{
    CPropagateWaveUtilityDlg dlg;
    double w=0;
    int N=0,power=0,i=0;
    dlg.m_dIncidentDistance=0;
    dlg.m_dTransmittedDistance=0;


    dlg.m_dFreqFilter=m_dFrequencyFilter;
    if((!incidentbar.m_bPropCoeffIncluded)&&(!transmittedbar.m_bPropCoeffIncluded))
    {
        dlg.m_nNyquist=0;
    }
    else
    {
        dlg.m_nNyquist=1;
    }



    if(dlg.DoModal()==IDOK)
    {

        incident.m_dstrain.RemoveAll();
        reflected.m_dstrain.RemoveAll();
        transmitted.m_dstrain.RemoveAll();
        incidentbar.m_dVelocity.RemoveAll();
        transmittedbar.m_dVelocity.RemoveAll();
        incidentbar.m_dForce.RemoveAll();
        transmittedbar.m_dForce.RemoveAll();
        incidentbar.Force.m_bCalculated=FALSE;
        incidentbar.Velocity.m_bCalculated=FALSE;
        transmittedbar.Force.m_bCalculated=FALSE;
        transmittedbar.Velocity.m_bCalculated=FALSE;
        sample.m_bCalculatedStrainRate=FALSE;
        sample.m_bCalculatedStrain=FALSE;
        sample.m_bCalculatedStress=FALSE;
        flag.m_bIncidentStrain=FALSE;
        flag.m_bReflectedStrain=FALSE;
        flag.m_bTransmittedStrain=FALSE;
        BeginWaitCursor();
        if((flag.m_bReadIncident)&&(flag.m_bReadIncidentBar)&&(flag.m_bSeparatedIncident))
```

```
{
    if(((fabs(data1.m_dDT-incidentbar.m_dDT)>5e-8)||
            (data1.m_dvolts.GetSize()!=incidentbar.m_dG.GetSize()))
            &&
            (incidentbar.m_bPropCoeffIncluded))
        {
            //compensate indices for new points
            incident.m_nbegin=int(incident.m_nbegin*data1.m_dDT/incidentbar.m_dDT);
            incident.m_nend=int(incident.m_nend*data1.m_dDT/incidentbar.m_dDT);
            reflected.m_nbegin=int(reflected.m_nbegin*data1.m_dDT/incidentbar.m_dDT);
            reflected.m_nend=int(reflected.m_nend*data1.m_dDT/incidentbar.m_dDT);
            // some error checking
            if (incident.m_nbegin>incidentbar.m_dG.GetSize())
            {   incident.m_nbegin=incidentbar.m_dG.GetSize();}

            if (reflected.m_nbegin>incidentbar.m_dG.GetSize())
            {   reflected.m_nbegin=incidentbar.m_dG.GetSize();}

            if (incident.m_nend>incidentbar.m_dG.GetSize())
            {   incident.m_nend=incidentbar.m_dG.GetSize();}

            if (reflected.m_nend>incidentbar.m_dG.GetSize())
            {   reflected.m_nend=incidentbar.m_dG.GetSize();}

            if(incidentbar.m_dDT>data1.m_dDT)
            {   N=int(ceil(incidentbar.m_dG.GetSize()*incidentbar.m_dDT/data1.m_dDT));
            if(N>data1.m_dvolts.GetSize())
            {   //in case you want more points than you have are needed
                data1.m_dvolts.InsertAt(data1.m_dvolts.GetUpperBound(),0,
                    N-data1.m_dvolts.GetSize());
                data1.m_szComment+="had points added; ";
            }

            N=incidentbar.m_dG.GetSize();
            if (fabs(data1.m_dDT-incidentbar.m_dDT)>1e-8)
            {InterpolateData(N,data1.m_dDT,incidentbar.m_dDT,
            &data1.m_dvolts[0]);
            data1.m_szComment+="was changed to a slower sample rate; ";
            }

            //remove points
            if ((N-data1.m_dvolts.GetSize())>0)
            {
                data1.m_dvolts.RemoveAt(N,abs(N-data1.m_dvolts.GetSize()));
                data1.m_dvolts.FreeExtra();
                data1.m_szComment+="had points removed; ";
            }
            data1.m_dDT=incidentbar.m_dDT;//new sample rate
            data1.m_szComment+="to match bar dispersion data; ";

            }else
            {//addpoints
                N=incidentbar.m_dG.GetSize();
                if(N>data1.m_dvolts.GetSize())
                    //here it is necessary since adding points
                {
                    data1.m_dvolts.InsertAt(data1.m_dvolts.GetUpperBound(),0,
                        N-data1.m_dvolts.GetSize());
                    data1.m_szComment+="had points added; ";
                }

                if (fabs(data1.m_dDT-incidentbar.m_dDT)>1e-8)
                {
                    InterpolateData(N,data1.m_dDT,incidentbar.m_dDT,&data1.m_dvolts[0]);
                    data1.m_szComment+="was changed to a faster sample rate; ";
                }
                data1.m_dDT=incidentbar.m_dDT;
                if(data1.m_dvolts.GetSize()>incidentbar.m_dG.GetSize())
                {
                    data1.m_dvolts.RemoveAt(incidentbar.m_dG.GetSize(),
                        data1.m_dvolts.GetSize()-incidentbar.m_dG.GetSize());
                    data1.m_dvolts.FreeExtra();
                    data1.m_szComment+="had points removed; ";
                }
                data1.m_szComment+="to match bar dispersion data; ";
            }
```

```
}else if    (incidentbar.m_bPropCoeffIncluded)
{
    data1.m_szComment+="matched bar dispersion data; ";
}

N=data1.m_dvolts.GetSize(); //number of points
power=int((log(N)/log(2)));
if (N>int(pow(2,power)))//if the number of points is not a power of 2
{power++;}
N=int(pow(2,power));

incident.m_dstrain.InsertAt(0,0,N);
reflected.m_dstrain.InsertAt(0,0,N);

incident.m_dDf=1/(incident.m_dstrain.GetSize()*data1.m_dDT);
reflected.m_dDf=1/(reflected.m_dstrain.GetSize()*data1.m_dDT);


if(!incidentbar.m_bPropCoeffIncluded)   //if dispersion is not included propragate
waves with C
{
    incidentbar.m_dDf=incident.m_dDf;
    incidentbar.m_dDT=data1.m_dDT;
    incidentbar.m_dG.RemoveAll();
    w=incident.m_dDf*2*PI;
    incidentbar.m_dG.Add(0);
    incidentbar.m_dG.Add(0);
    w=0;

    for(i=2;i<incident.m_dstrain.GetSize();i+=2)
    {
        w+=incident.m_dDf*2*PI;
        incidentbar.m_dG.Add(0);
        incidentbar.m_dG.Add(w/sqrt(incidentbar.m_dE/incidentbar.m_dDensity));
    }
}


Volts_to_Strain(&data1.m_dvolts[incident.m_nbegin],
    &incident.m_dstrain[incident.m_nbegin],
    (incident.m_nend-incident.m_nbegin),incidentbar.m_dSlope,
    &incident.m_dmaxstrain,&incident.m_dminstrain);
Volts_to_Strain(&data1.m_dvolts[reflected.m_nbegin],
    &reflected.m_dstrain[reflected.m_nbegin],
    (reflected.m_nend-reflected.m_nbegin),incidentbar.m_dSlope,
    &incident.m_dmaxstrain,&incident.m_dminstrain);
incident.m_bTime=TRUE;
flag.m_bIncidentStrain=TRUE;
reflected.m_bTime=TRUE;
flag.m_bReflectedStrain=TRUE;

Realft(&incident.m_dstrain[0]-1,incident.m_dstrain.GetSize()
    ,&incident.m_bTime);
Realft(&reflected.m_dstrain[0]-1,reflected.m_dstrain.GetSize()
    ,&reflected.m_bTime);

incident.m_bPolar=FALSE;
reflected.m_bPolar=FALSE;

//Change to polar form for Dispersion Correction
if(dlg.m_nNyquist!=0)
{
    m_dFrequencyFilter=dlg.m_dFreqFilter;
}
else
{   //calculate values up to the nyquist frequency
    //Propagate wave checks values so don't do it here
    m_dFrequencyFilter=1/(2*incidentbar.m_dDT);  //Nyquist Critical Frequency
}
ChangeForm(incident.m_dstrain.GetSize(),
    &incident.m_dstrain[0],&incident.m_bPolar);
ChangeForm(reflected.m_dstrain.GetSize(),
    &reflected.m_dstrain[0],&reflected.m_bPolar);
PropagateWave(incident.m_dstrain.GetSize(),&incident.m_dstrain[0],
    data1.m_dDT,incidentbar.m_dG.GetSize(),
```

```
                    &incidentbar.m_dG[0],-1,dlg.m_dIncidentDistance);
                PropagateWave(reflected.m_dstrain.GetSize(),&reflected.m_dstrain[0],
                    data1.m_dDT,incidentbar.m_dG.GetSize(),
                    &incidentbar.m_dG[0],-1,dlg.m_dReflectedDistance);
                //Calculate force and velocity for each bar at interface
                //change into rectangular form
                ChangeForm(incident.m_dstrain.GetSize(),
                    &incident.m_dstrain[0],&incident.m_bPolar);
                ChangeForm(reflected.m_dstrain.GetSize(),
                    &reflected.m_dstrain[0],&reflected.m_bPolar);

            flag.m_bPropagateUtility=TRUE;
            //flag.m_bProcessedResults=TRUE;

        }
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
////////////  Change Back Into Time Domain
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
        if((flag.m_bReadTransmitted)&&(flag.m_bReadTransmittedBar)&&(flag.m_bSeparatedTransmitted)
)
        {

            if(((fabs(data2.m_dDT-transmittedbar.m_dDT)>5e-8)||
                (data2.m_dvolts.GetSize()!=transmittedbar.m_dG.GetSize()))
                &&(transmittedbar.m_bPropCoeffIncluded))
            {
                //compensate indices for new points
                transmitted.m_nbegin=int(transmitted.m_nbegin*data2.m_dDT/transmittedbar.m_dDT);
                transmitted.m_nend=int(transmitted.m_nend*data2.m_dDT/transmittedbar.m_dDT);

                if(transmitted.m_nbegin>transmittedbar.m_dG.GetSize())
                {transmitted.m_nbegin=transmittedbar.m_dG.GetSize();}

                if(transmitted.m_nend>transmittedbar.m_dG.GetSize())
                {transmitted.m_nend=transmittedbar.m_dG.GetSize();}


                if(transmittedbar.m_dDT>data2.m_dDT)
                {   N=int(ceil(transmittedbar.m_dG.GetSize()*transmittedbar.m_dDT/data2.m_dDT));
                if(N>data2.m_dvolts.GetSize())
                {
                    data2.m_dvolts.InsertAt(data2.m_dvolts.GetUpperBound(),0,
                        N-data2.m_dvolts.GetSize());
                    data2.m_szComment+="had points added; ";
                }
                N=transmittedbar.m_dG.GetSize();
                if (fabs(transmittedbar.m_dDT-data2.m_dDT)>1e-8)
                {
                    InterpolateData(N,data2.m_dDT,transmittedbar.m_dDT,&data2.m_dvolts[0]);
                    data2.m_szComment+="was changed to a slower sample rate; ";
                }
                if ((N-data2.m_dvolts.GetSize())>0)
                {//remove points
                    data2.m_dvolts.RemoveAt(N,abs(N-data2.m_dvolts.GetSize()));
                    data2.m_dvolts.FreeExtra();
                    data2.m_szComment+="had points removed; ";
                }
                data2.m_dDT=transmittedbar.m_dDT;//new sample rate
                data2.m_szComment+="to match bar dispersion data; ";
                }
                else
                {   ///addpoints
                    N=transmittedbar.m_dG.GetSize();
                    if(N>data2.m_dvolts.GetSize())  //here it is necessary since adding points
                    {   data2.m_dvolts.InsertAt(data2.m_dvolts.GetUpperBound(),0,
                    N-data2.m_dvolts.GetSize());
                    data2.m_szComment+="had points added; ";
                    }

                    if (fabs(transmittedbar.m_dDT-data2.m_dDT)>1e-8)
                    {
                        InterpolateData(N,data2.m_dDT,transmittedbar.m_dDT,&data2.m_dvolts[0]);
                        data2.m_szComment+="was changed to a faster sample rate; ";
                    }
```

```
            data2.m_dDT=transmittedbar.m_dDT;

            if(data2.m_dvolts.GetSize()>transmittedbar.m_dG.GetSize())
            {
                data2.m_dvolts.RemoveAt(transmittedbar.m_dG.GetSize(),
                    data2.m_dvolts.GetSize()-transmittedbar.m_dG.GetSize());
                data2.m_dvolts.FreeExtra();
                data2.m_szComment+="had points removed; ";
            }


            data2.m_szComment+="to match bar dispersion data;";
        }
    }else if(transmittedbar.m_bPropCoeffIncluded)
    {
        data2.m_szComment+="matched bar dispersion data; ";
    }

    N=data2.m_dvolts.GetSize(); //number of points
    power=int((log(N)/log(2)));
    if (N>int(pow(2,power)))//if the number of points is not a power of 2
    {power++;}
    N=int(pow(2,power));

    transmitted.m_dstrain.InsertAt(0,0,N);

    transmitted.m_dDf=1/(transmitted.m_dstrain.GetSize()*data2.m_dDT);
    if(!transmittedbar.m_bPropCoeffIncluded)    //if dispersion is not included propragate
waves with C
    {   transmittedbar.m_dDf=transmitted.m_dDf;
    transmittedbar.m_dDT=data2.m_dDT;
    transmittedbar.m_dG.RemoveAll();
    w=transmitted.m_dDf*2*PI;
    transmittedbar.m_dG.Add(0);
    transmittedbar.m_dG.Add(0);
    w=0;
    for(i=2;i<transmitted.m_dstrain.GetSize();i+=2)
    {
        w+=transmitted.m_dDf*2*PI;
        transmittedbar.m_dG.Add(0);
        transmittedbar.m_dG.Add(w/sqrt(transmittedbar.m_dE/transmittedbar.m_dDensity));

    }
    }


    Volts_to_Strain(&data2.m_dvolts[transmitted.m_nbegin],
        &transmitted.m_dstrain[transmitted.m_nbegin],
        (transmitted.m_nend-transmitted.m_nbegin),
        transmittedbar.m_dSlope,&transmitted.m_dmaxstrain,&transmitted.m_dminstrain);
    transmitted.m_bTime=TRUE;
    flag.m_bTransmittedStrain=TRUE;


    Realft(&transmitted.m_dstrain[0]-1,transmitted.m_dstrain.GetSize()
        ,&transmitted.m_bTime);
    transmitted.m_bPolar=FALSE;

    //Change to polar form for Dispersion Correction
    if(dlg.m_nNyquist!=0)
    {   m_dFrequencyFilter=dlg.m_dFreqFilter;}
    else
    {   //calculate values up to the nyquist frequency
        //Propagate wave checks values so don't do it here
        m_dFrequencyFilter=1/(2*transmittedbar.m_dDT);  //Nyquist Critical Frequency
    }

    ChangeForm(transmitted.m_dstrain.GetSize(),
        &transmitted.m_dstrain[0],&transmitted.m_bPolar);
    PropagateWave(transmitted.m_dstrain.GetSize(),&transmitted.m_dstrain[0],
        data2.m_dDT,transmittedbar.m_dG.GetSize(),
        &transmittedbar.m_dG[0],1,dlg.m_dTransmittedDistance);

    //Calculate force and velocity for each bar at interface
    //change into rectangular form
    ChangeForm(transmitted.m_dstrain.GetSize(),
```

```
                &transmitted.m_dstrain[0],&transmitted.m_bPolar);
            flag.m_bPropagateUtility=TRUE;
        }
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
////////////  Change Back Into Time Domain
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

    if(flag.m_bIncidentStrain)
        {   if (!incident.m_bTime)
            {   if (incident.m_bPolar)
                {ChangeForm(incident.m_dstrain.GetSize(),&incident.m_dstrain[0],&incident.m_bPolar
);}

                Realft(&incident.m_dstrain[0]-1,incident.m_dstrain.GetSize(),
                    &incident.m_bTime);
            }

            for (i=0;i<incident.m_dstrain.GetSize();i++)
            {
                if(incident.m_dstrain[i]>incident.m_dmaxstrain)
                {incident.m_dmaxstrain=incident.m_dstrain[i];}
                if(incident.m_dstrain[i]<incident.m_dminstrain)
                {incident.m_dminstrain=incident.m_dstrain[i];}
            }
        }
        if(flag.m_bReflectedStrain)
        {   if (!reflected.m_bTime)
            {   if (reflected.m_bPolar)
                {ChangeForm(reflected.m_dstrain.GetSize(),&reflected.m_dstrain[0],&reflected.m_bPo
lar);}

                Realft(&reflected.m_dstrain[0]-1,reflected.m_dstrain.GetSize(),
                    &reflected.m_bTime);
            }

            for (i=0;i<reflected.m_dstrain.GetSize();i++)
            {
                if(reflected.m_dstrain[i]>reflected.m_dmaxstrain)
                {reflected.m_dmaxstrain=reflected.m_dstrain[i];}
                if(reflected.m_dstrain[i]<reflected.m_dminstrain)
                {reflected.m_dminstrain=reflected.m_dstrain[i];}
            }
        }

        if(flag.m_bTransmittedStrain)
        {   if (!transmitted.m_bTime)
            {   if (transmitted.m_bPolar)
                {ChangeForm(transmitted.m_dstrain.GetSize(),&transmitted.m_dstrain[0],&transmitted
.m_bPolar);}

                Realft(&transmitted.m_dstrain[0]-1,transmitted.m_dstrain.GetSize(),
                    &transmitted.m_bTime);
            }

            for (i=0;i<transmitted.m_dstrain.GetSize();i++)
            {
                if(transmitted.m_dstrain[i]>transmitted.m_dmaxstrain)
                {transmitted.m_dmaxstrain=transmitted.m_dstrain[i];}
                if(transmitted.m_dstrain[i]<transmitted.m_dminstrain)
                {transmitted.m_dminstrain=transmitted.m_dstrain[i];}
            }
        }







        EndWaitCursor();
    }

    UpdateAllViews(NULL);
}
```

```
void CCSHBDoc::OnUpdateUtilitiesPropagatewave(CCmdUI* pCmdUI)
{
    if  ((flag.m_bSeparatedIncident&&flag.m_bReadIncidentBar)||
        (flag.m_bSeparatedTransmitted&&flag.m_bReadTransmittedBar))
        {pCmdUI->Enable(TRUE);}
        else
        {pCmdUI->Enable(FALSE);}
}

BOOL CCSHBDoc::CalculatePropagationCoefficient()
{
    int i=0,power=0,N=0;
    double fo=0,phasev=0,den_r=0,den_i=0,w=0,dnum=0;
    double num_real=0,num_imag=0,sum=0;
    BOOL bE_Time,bG_Polar;
    CPropCoeffDlg dlg;
    CArray <double,double> dG,dE_Cmplx;



    //initialize data

    dlg.m_szCalibrationFile=incidentbar.m_szFilename;
    dlg.m_szDataFile=data1.m_szFilename;
    dlg.m_bUpdateFile=FALSE;
    dlg.m_dFilter=20000;
//Note error checking for valid input data is done in OnUpdateAnalysePropcalc

    if (dlg.DoModal()==IDOK)
    {
        if ((incidentbar.m_bPropCoeffIncluded)&&(dlg.m_bUpdateFile))
        {
            AfxMessageBox("Propagation Coefficient Data Already Included in Calibration File!\nThe
 Old Data Must Be Erased and Calibration Files Re-inputted Before Reprocessing!\nCalculation Will
Not Continue!",MB_ICONERROR);
            return FALSE;
        }


        BeginWaitCursor();
        dG.RemoveAll();
        dE_Cmplx.RemoveAll();
        incident.m_dstrain.RemoveAll();
        reflected.m_dstrain.RemoveAll();

        //initialize strain arrays
        N=data1.m_dvolts.GetSize(); //number of points
        power=int((log(N)/log(2)));
        if (N>int(pow(2,power)))//if the number of points is not a power of 2
        {power++;}
        N=int(pow(2,power));

        incident.m_dstrain.InsertAt(0,0,N);
        reflected.m_dstrain.InsertAt(0,0,N);

        Volts_to_Strain(&data1.m_dvolts[incident.m_nbegin],&incident.m_dstrain[incident.m_nbegin],
            incident.m_nend-incident.m_nbegin,incidentbar.m_dSlope,
            &incident.m_dmaxstrain,&incident.m_dminstrain);
        Volts_to_Strain(&data1.m_dvolts[reflected.m_nbegin],&reflected.m_dstrain[reflected.m_nbegi
n],
            reflected.m_nend-reflected.m_nbegin,incidentbar.m_dSlope,
            &reflected.m_dmaxstrain,&reflected.m_dminstrain);

        flag.m_bIncidentStrain=TRUE;
        incident.m_bTime=TRUE;
        flag.m_bReflectedStrain=TRUE;
        reflected.m_bTime=TRUE;

        Realft(&incident.m_dstrain[0]-1,incident.m_dstrain.GetSize(),&incident.m_bTime);
        incident.m_bPolar=FALSE;

        Realft(&reflected.m_dstrain[0]-1,reflected.m_dstrain.GetSize(),&reflected.m_bTime);
        reflected.m_bPolar=FALSE;
//-reflected strain
        for(i=0;i<reflected.m_dstrain.GetSize();i+=2)
        {
```

```
            Cmul(reflected.m_dstrain[i],reflected.m_dstrain[i+1],-1,0,
                &reflected.m_dstrain[i],&reflected.m_dstrain[i+1]);
        }

        ChangeForm(incident.m_dstrain.GetSize(),&incident.m_dstrain[0],&incident.m_bPolar);
        ChangeForm(reflected.m_dstrain.GetSize(),&reflected.m_dstrain[0],&reflected.m_bPolar);


//reflected/incidentstrain
        for(i=0;i<incident.m_dstrain.GetSize();i+=2)
        {
            dG.Add(log(reflected.m_dstrain[i]/incident.m_dstrain[i])/(-2*incidentbar.m_dDistance))
;
            dG.Add((reflected.m_dstrain[i+1]-incident.m_dstrain[i+1])/(-2*incidentbar.m_dDistance)
);

        }

        bG_Polar=TRUE;


        EndWaitCursor();

        fo=1/(N*data1.m_dDT);       //base frequency

        if( dlg.m_bUpdateFile)
        {
            ofstream appfile(incidentbar.m_szFilename,ios::app|ios::nocreate);
            BeginWaitCursor();
            appfile<<"\n*DISPERSION"<<endl;
            for (i=0;i<dG.GetSize();i+=2)
            {
                appfile<<fo*(i/2)<<"\t"<<dG[i]<<"\t"<<dG[i+1]<<endl;
            }
            appfile.close();
            EndWaitCursor();
        }


        N=int(dlg.m_dFilter/fo);//hz//filter
        if(dlg.m_bComplexModulus)
        {   dE_Cmplx.InsertAt(0,0,dG.GetSize());//initialize array
            ChangeForm(dG.GetSize(),&dG[0],&bG_Polar);
            for (i=2;i<N;i+=2)
            {   w=2*PI*fo*i/2;
                dnum=incidentbar.m_dDensity*w*w;
                Cmul(dG[i],dG[i+1],dG[i],dG[i+1],&den_r,&den_i);
                Cdiv(dnum,0,den_r,den_i,&dE_Cmplx[i],&dE_Cmplx[i+1]);
                sum+=dE_Cmplx[i];

            }

            dE_Cmplx[0]=-2*sum; //D.C. Term =sum of all others
            dE_Cmplx[1]=0;

            bE_Time=FALSE;
        //  Realft(&dE_Cmplx[0]-1,dE_Cmplx.GetSize(),&bE_Time); //change into time domain
            ChangeForm(dG.GetSize(),&dG[0],&bG_Polar);

        }

        if ((dlg.m_bExportFile)||(dlg.m_bComplexModulus))
        {
                bool stopflag = false;
                OPENFILENAME SaveFileName;
                TCHAR szFile[256000] = "\0";    // File list
                static char szFilter[] =  "All Files (*.*)\0*.*\0\0";

                SaveFileName.lStructSize = sizeof(OPENFILENAME);
                SaveFileName.hwndOwner = NULL;
                SaveFileName.hInstance = NULL;
                SaveFileName.lpstrFilter = szFilter;
                SaveFileName.lpstrCustomFilter = NULL;
                SaveFileName.nMaxCustFilter = 0;
                SaveFileName.nFilterIndex = 0;
                SaveFileName.lpstrFile = szFile;
```

```
                SaveFileName.nMaxFile = sizeof(szFile);
                SaveFileName.lpstrFileTitle = NULL;
                SaveFileName.nMaxFileTitle = 0;
                SaveFileName.lpstrInitialDir =NULL;
                SaveFileName.lpstrTitle = "Output File";
                SaveFileName.nFileOffset = 0;
                SaveFileName.nFileExtension = 0;
                SaveFileName.lpstrDefExt = "";
                SaveFileName.lpfnHook = NULL;
                SaveFileName.lpTemplateName = NULL;
                SaveFileName.Flags = OFN_EXPLORER |OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT;

                // Open dialog box
                GetSaveFileName(&SaveFileName);
                ofstream savefile(szFile,ios::out);
                if(!savefile)
                {
                    //AfxMessageBox("Could Not Create File",MB_ERROR);
                }

                savefile<<"\n*DISPERSION\n"<<endl;
                savefile<<"Frequency\tAttenuation\tWave Number\tPhase Velocity\n";
                savefile<<"Hz\t1/m\t\tm/s\n";

                savefile<<0<<"\t"<<dG[0]<<"\t"<<dG[1]<<"\t"<<fo*2*PI/dG[3]<<endl;
                for (i=2;i<dG.GetSize();i+=2)
                {
                    phasev=((i/2*fo)*2*PI)/dG[i+1];
                    savefile<<(i/2)*fo<<"\t"<<dG[i]<<"\t"<<dG[i+1]<<"\t"<<phasev<<endl;
                }
                if(dlg.m_bComplexModulus)
                {
                    savefile<<"\nComplex Modulus\n"<<endl;
                    savefile<<"Frequency\tE*\n";
                    savefile<<"Hz\tReal\tImaginary\n";

                    for (i=0;i<dE_Cmplx.GetSize();i+=2)
                    {
                        savefile<<(i/2)*fo<<
                            "\t"<<dE_Cmplx[i]<<"\t"<<dE_Cmplx[i+1]<<endl;
                    }
                }


                savefile.close();



            }
            EndWaitCursor();
            AfxMessageBox("Calculation Completed",MB_OK);
            InitializeValues();
            UpdateAllViews(NULL);
        }
        return TRUE;

    }
```

```
// CSHBView.cpp : implementation of the CCSHBView class
//

#include "stdafx.h"
#include "CSHB.h"

#include "CSHBDoc.h"
#include "CSHBView.h"
#include "Data.h"
#include "math.h"
#include "IndexDlg.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCSHBView

IMPLEMENT_DYNCREATE(CCSHBView, CView)

BEGIN_MESSAGE_MAP(CCSHBView, CView)
    //{{AFX_MSG_MAP(CCSHBView)
    ON_WM_SIZE()
    ON_WM_MOUSEMOVE()
    ON_BN_CLICKED(IDC_VIEWFREQ, OnViewfreq)
    ON_WM_SETCURSOR()
    ON_WM_CANCELMODE()
    ON_COMMAND(ID_PRE_VIEWINDICES, OnPreViewindices)
    ON_WM_LBUTTONDOWN()
    ON_COMMAND(ID_POST_VIEWVEL, OnViewVelocity)
    ON_COMMAND(ID_POST_VWFOR, OnViewForce)
    ON_COMMAND(ID_POST_VWSTRN, OnViewStrain)
    ON_UPDATE_COMMAND_UI(ID_POST_VIEWVEL, OnUpdateViewVelocity)
    ON_UPDATE_COMMAND_UI(ID_POST_VWFOR, OnUpdateViewForce)
    ON_UPDATE_COMMAND_UI(ID_POST_VWSTRN, OnUpdateViewStrain)
    ON_COMMAND(ID_POST_VWDISP, OnViewDisplacement)
    ON_UPDATE_COMMAND_UI(ID_POST_VWDISP, OnUpdateViewDisplacement)
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCSHBView construction/destruction

CCSHBView::CCSHBView()
{
    // TODO: add construction code here
    m_nViewExtra=0;

}

CCSHBView::~CCSHBView()
{
}

BOOL CCSHBView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CCSHBView drawing
CIndexDlg * g_pIndex; //Declare Global


void CCSHBView::OnDraw(CDC* pDC)
```

```
{
    CCSHBDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    double m_dXscale=1,m_dYscale=1;
    int number=0,xo=0,yo=0,xL=0,yL=0;
    double x_data_length=0,y_data_length=0 ;


    CFont fnTitle;
    CFont* pOldFont;


    if (pDoc->flag.m_bRecalculate)
    {
        AfxMessageBox("The input data has been changed.\n You should Recalculate.",MB_ICONWARNING)
;

        pDoc->flag.m_bRecalculate=FALSE;

    }

    fnTitle.CreateFont(24,0,0,0,FW_BOLD,FALSE,FALSE,FALSE,ANSI_CHARSET,
        OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH+FF_DONTCARE,
        "Arial");

    CRect rcClient;

/*/Printing stuff
    pDC->SetMapMode(MM_LOMETRIC);
    if(pDC->IsPrinting())
    {
//      pDC->SetMapMode(MM_LOENGLISH);
        rcClient=m_rcPrintRect;
    }else
    {
        GetClientRect(&rcClient);
    }
    pDC->DPtoLP(&rcClient);
*/
    if((pDoc->flag.m_bProcessedResults)) {

        xo=int(m_cxClient*.05);         //voltage data
        yo=int(m_cyClient*0.225);       //set origins for axes
        xL=int(m_cxClient*.5*0.7);
        yL=int(-m_cyClient*.5*0.7*0.5); //set length of axes
        DrawAxis(pDC,xo,yo,xL,yL,"","Time (s)","Voltage (V)");

        yL=int(m_cyClient*.5*0.7*0.5);
        DrawAxis(pDC,xo,yo,xL,yL,"Raw Data","","");

        xo=int(m_cxClient*.55);         //strain data
        yo=int(m_cyClient*0.4);
        xL=int(m_cxClient*.5*0.7);
        yL=int(m_cyClient*.5*0.7);
        switch(m_nViewExtra)
        {
        case 0:
            yo=int(m_cyClient*0.225);
            yL=-int(m_cyClient*.5*0.7*0.5);;
            DrawAxis(pDC,xo,yo,xL,yL,"","Time (s)","Strain");
            yL=int(m_cyClient*.5*0.7*0.5);
            DrawAxis(pDC,xo,yo,xL,yL,"Strain Data","","");
            break;
        case 1:
            DrawAxis(pDC,xo,yo,xL,yL,"Velocity Data","Time (s)","Velocity (m/s)");
            break;
        case 2:
            DrawAxis(pDC,xo,yo,xL,yL,"Force Data","Time (s)","Force (N)");
            break;
        case 3:
            DrawAxis(pDC,xo,yo,xL,yL,"Displacement Data","Time (s)","Displacement (m)");
            break;
        }
```

```
        yL=int(m_cyClient*.5*0.7);
        xo=int(m_cxClient*.05);     //stress strain data
        yo=int(m_cyClient*.9);
        DrawAxis(pDC,xo,yo,xL,yL,"Stress Strain","Strain","Stress (MPa)");

        xo=int(m_cxClient*.55);     //strain rate data
        yo=int(m_cyClient*.9);
        DrawAxis(pDC,xo,yo,xL,yL,"Strain Rate","Time (s)","Strain Rate");

        x_data_length=xL*0.9;//0.75;

        pDC->SetTextAlign(TA_CENTER+TA_BASELINE);
        pOldFont=pDC->SelectObject(&fnTitle);
        pDC->SetViewportOrg(0,0);
        if(pDoc->sample.m_szName!="")
        {
            pDC->TextOut(int(0.5*m_cxClient),int(0.5*m_cyClient),pDoc->sample.m_szName);}
        pDC->SelectObject(pOldFont);


    }else if((pDoc->incidentbar.Force.m_bCalculated)||
        (pDoc->transmittedbar.Force.m_bCalculated))
    {
        xo=int(m_cxClient*.05);         //voltage data
        yo=int(m_cyClient*0.225);
        xL=int(m_cxClient*.5*0.7);
        yL=int(-m_cyClient*.5*0.7*0.5);
        DrawAxis(pDC,xo,yo,xL,yL,"","Time (s)","Voltage (V)");
        yL=int(m_cyClient*.5*0.7*0.5);
        DrawAxis(pDC,xo,yo,xL,yL,"Raw Data","","");

        xo=int(m_cxClient*.55);         //right top quad data
        yo=int(m_cyClient*0.225);
        xL=int(m_cxClient*.5*0.7);
        yL=int(-m_cyClient*.5*0.7*0.5);

        switch (m_nViewExtra)
        {
            case 3:
                yo=int(m_cyClient*0.4);
                yL=int(m_cyClient*.5*0.7);
                DrawAxis(pDC,xo,yo,xL,yL,"Displacement Data","Time (s)","Displacement (m)");
                break;
            default:
                DrawAxis(pDC,xo,yo,xL,yL,"","Time (s)","Strain");
                yL=int(m_cyClient*.5*0.7*0.5);
                DrawAxis(pDC,xo,yo,xL,yL,"Strain Data","","");
                break;

        }

        xo=int(m_cxClient*.05);         //Velocity data
        yo=int(m_cyClient*0.9);
        xL=int(m_cxClient*.5*0.7);
        yL=int(m_cyClient*.5*0.7);
        DrawAxis(pDC,xo,yo,xL,yL,"Velocity Data","Time (s)","Velocity (m/s)");

        xo=int(m_cxClient*.55);         //Force data
        yo=int(m_cyClient*0.9);
        xL=int(m_cxClient*.5*0.7);
        yL=int(m_cyClient*.5*0.7);
        DrawAxis(pDC,xo,yo,xL,yL,"Force Data","Time (s)","Force (N)");

        x_data_length=xL*0.9;

        pDC->SetTextAlign(TA_CENTER+TA_BASELINE);
        pOldFont=pDC->SelectObject(&fnTitle);
        if(pDoc->sample.m_szName!="")
        {   pDC->TextOut(int(0.5*m_cxClient),int(0.5*m_cyClient),pDoc->sample.m_szName);}
        pDC->SelectObject(pOldFont);
    }else if (pDoc->flag.m_bPropagateUtility)
    {
////////////////////////////////////////////////////////////
        xo=int(m_cxClient*.05);         //voltage data
        yo=int(m_cyClient*0.225);
        xL=int(m_cxClient*0.9);
```

```
        yL=int(-m_cyClient*.5*0.7*0.5);
        DrawAxis(pDC,xo,yo,xL,yL,"","Time (s)","Voltage (V)");
        yL=int(m_cyClient*.5*0.7*0.5);
        DrawAxis(pDC,xo,yo,xL,yL,"Raw Data","","");

        x_data_length=xL*.9;

        xo=int(m_cxClient*.05);
        yo=int(m_cyClient*0.725);
        yL=int(-m_cyClient*.5*0.7*0.5);
        DrawAxis(pDC,xo,yo,xL,yL,"","Time (s)","Strain");
        yL=int(m_cyClient*.5*0.7*0.5);
        DrawAxis(pDC,xo,yo,xL,yL,"Propagated Strain Data","","");




    }
    else
    {
        xo=int(m_cxClient*.05); //just voltage data
        yo=int(m_cyClient*0.5);
        xL=int(m_cxClient*0.9);
        yL=int(-m_cyClient*.5*0.9);
        DrawAxis(pDC,xo,yo,xL,yL,"","Time (s)","Voltage (V)");
        yL=int(m_cyClient*.5*0.9);
        DrawAxis(pDC,xo,yo,xL,yL,"Raw Data","","");
        x_data_length=xL*.9;            //draw 75% of x
        y_data_length=-yL*.9;       //draw 75% of y,-ve to draw upright
    }



////////////////////////////////////////////////////////////////////////////
///////////////     Graph Raw Data
////////////////////////////////////////////////////////////////////////////
    if ((pDoc->incidentbar.Force.m_bCalculated)||
        (pDoc->transmittedbar.Force.m_bCalculated)||
        (pDoc->flag.m_bProcessedResults)||pDoc->flag.m_bPropagateUtility)
    {
        xo=int(m_cxClient*.05);         //voltage data
        yo=int(m_cyClient*0.225);
        yL=int(m_cyClient*.5*0.7*0.5);
        y_data_length=-yL*.9;
    }else
    {
        xo=int(m_cxClient*.05);
        yo=int(m_cyClient*0.5);
    }

    if((pDoc->flag.m_bReadIncident)&&(pDoc->incident.m_bTime)){

        //scaling factor for x and y
        m_dXscale=(x_data_length/(pDoc->data1.m_dDT*(pDoc->data1.m_dvolts.GetSize())));

        if (fabs(pDoc->data1.m_dmaxy)>fabs(pDoc->data1.m_dminy))
        {   m_dYscale=y_data_length/pDoc->data1.m_dmaxy;
        }else{
            m_dYscale=y_data_length/fabs(pDoc->data1.m_dminy);}
        pDoc->scales.time=m_dXscale;
        pDoc->scales.volts=m_dYscale;


        GraphData(TRUE,pDoc->data1.m_dDT,0,0,&pDoc->data1.m_dvolts[0],
            pDoc->data1.m_dvolts.GetSize()
            ,m_dXscale,m_dYscale,BLACK,1,xo,yo);
        GraphTick(xo,yo,pDoc->data1.m_dDT*pDoc->data1.m_dvolts.GetSize(),
            y_data_length/pDoc->scales.volts,pDoc->scales.time,pDoc->scales.volts);


    if((pDoc->flag.m_bSeparatedIncident)) {

            GraphData(TRUE,pDoc->data1.m_dDT,pDoc->incident.m_nbegin,0,
                &pDoc->data1.m_dvolts[pDoc->incident.m_nbegin],
                (pDoc->incident.m_nend-pDoc->incident.m_nbegin)
```

```
                    ,m_dXscale,m_dYscale,RED,2,xo,yo);
            GraphData(TRUE,pDoc->data1.m_dDT,pDoc->reflected.m_nbegin,0,
                &pDoc->data1.m_dvolts[pDoc->reflected.m_nbegin],
                (pDoc->reflected.m_nend-pDoc->reflected.m_nbegin)
                    ,m_dXscale,m_dYscale,BLUE,2,xo,yo);


        }


    }
    if((pDoc->flag.m_bReadTransmitted)&&(pDoc->transmitted.m_bTime)) {

        if(!pDoc->flag.m_bReadIncident)
        {   m_dXscale=(x_data_length/(pDoc->data2.m_dDT*(pDoc->data2.m_dvolts.GetSize())));
            if (fabs(pDoc->data2.m_dmaxy)>fabs(pDoc->data2.m_dminy))
            {m_dYscale=y_data_length/pDoc->data2.m_dmaxy;
            }else{
            m_dYscale=y_data_length/fabs(pDoc->data2.m_dminy);}
            pDoc->scales.time=m_dXscale;
            pDoc->scales.volts=m_dYscale;
        }
        GraphData(TRUE,pDoc->data2.m_dDT,0,0,
            &pDoc->data2.m_dvolts[0],pDoc->data2.m_dvolts.GetSize()
            ,m_dXscale,m_dYscale,BLACK,1,xo,yo);
        GraphTick(xo,yo,pDoc->data2.m_dDT*pDoc->data2.m_dvolts.GetSize(),
            y_data_length/pDoc->scales.volts,pDoc->scales.time,pDoc->scales.volts);


        if((pDoc->flag.m_bSeparatedTransmitted)) {
            GraphData(TRUE,pDoc->data2.m_dDT,
                pDoc->transmitted.m_nbegin,0,
                &pDoc->data2.m_dvolts[pDoc->transmitted.m_nbegin],
                (pDoc->transmitted.m_nend-pDoc->transmitted.m_nbegin)
                    ,m_dXscale,m_dYscale,GREEN,2,xo,yo);
        }
    }
////////////////////////////////////////////////////////////////////////////
//////////////     Graph Strain Data
////////////////////////////////////////////////////////////////////////////

    if (((m_nViewExtra==0)&&
        (pDoc->incidentbar.Force.m_bCalculated ||
        pDoc->transmittedbar.Force.m_bCalculated ||
        pDoc->flag.m_bProcessedResults))||(pDoc->flag.m_bPropagateUtility))
    {

        if ((pDoc->incidentbar.Force.m_bCalculated)||
            (pDoc->transmittedbar.Force.m_bCalculated)||
            (pDoc->flag.m_bProcessedResults))
        {

            xo=int(m_cxClient*.55);         //voltage data
            yo=int(m_cyClient*0.225);
            yL=int(m_cyClient*.5*0.7*0.5);
            y_data_length=-yL*.9;
        }else
        {
            xo=int(m_cxClient*.05);
            yo=int(m_cyClient*0.725);
        }

        if((pDoc->flag.m_bIncidentStrain)&&(pDoc->incident.m_bTime)) {
            m_dXscale=(x_data_length/(pDoc->data1.m_dDT*(pDoc->incident.m_dstrain.GetSize())));
            if (fabs(pDoc->incident.m_dmaxstrain)>fabs(pDoc->incident.m_dminstrain))
            {m_dYscale=y_data_length/pDoc->incident.m_dmaxstrain;
            }else{
                m_dYscale=y_data_length/fabs(pDoc->incident.m_dminstrain);
            }
            pDoc->scales.strain=m_dYscale;
            pDoc->scales.straintime=m_dXscale;
            GraphData(TRUE,pDoc->data1.m_dDT,0,0,
                &pDoc->incident.m_dstrain[0],pDoc->incident.m_dstrain.GetSize()
                ,m_dXscale,m_dYscale,RED,1,xo,yo);
            GraphTick(xo,yo,pDoc->data1.m_dDT*pDoc->incident.m_dstrain.GetSize(),
                y_data_length/pDoc->scales.strain,pDoc->scales.straintime,pDoc->scales.strain);
```

```
        }

        if((pDoc->flag.m_bReflectedStrain)&&(pDoc->reflected.m_bTime)) {

            if(!pDoc->flag.m_bIncidentStrain)
            {
                m_dXscale=(x_data_length/(pDoc->data2.m_dDT*(pDoc->data1.m_dvolts.GetSize())));
                if (fabs(pDoc->reflected.m_dmaxstrain)>fabs(pDoc->reflected.m_dminstrain))
                {
                    m_dYscale=y_data_length/pDoc->reflected.m_dmaxstrain;
                }else{
                    m_dYscale=y_data_length/fabs(pDoc->reflected.m_dminstrain);
                }
                pDoc->scales.strain=m_dYscale;
                pDoc->scales.straintime=m_dXscale;
                GraphTick(xo,yo,pDoc->data1.m_dDT*pDoc->reflected.m_dstrain.GetSize(),
                    y_data_length/pDoc->scales.strain,pDoc->scales.time,pDoc->scales.strain);

            }

            GraphData(TRUE,pDoc->data1.m_dDT,0,0,
                &pDoc->reflected.m_dstrain[0],pDoc->reflected.m_dstrain.GetSize()
                ,m_dXscale,m_dYscale,BLUE,1,xo,yo);

        }

        if((pDoc->flag.m_bTransmittedStrain)&&(pDoc->transmitted.m_bTime)) {

            if(!pDoc->flag.m_bIncidentStrain)
            {
                m_dXscale=(x_data_length/(pDoc->data2.m_dDT*(pDoc->data2.m_dvolts.GetSize())));
                if (fabs(pDoc->transmitted.m_dmaxstrain)>fabs(pDoc->transmitted.m_dminstrain))
                {
                    m_dYscale=y_data_length/pDoc->transmitted.m_dmaxstrain;
                }else{
                    m_dYscale=y_data_length/fabs(pDoc->transmitted.m_dminstrain);
                }
                pDoc->scales.strain=m_dYscale;
                GraphTick(xo,yo,pDoc->data2.m_dDT*pDoc->transmitted.m_dstrain.GetSize(),
                    y_data_length/pDoc->scales.strain,pDoc->scales.time,pDoc->scales.strain);
            }

            GraphData(TRUE,pDoc->data2.m_dDT,0,0,
                &pDoc->transmitted.m_dstrain[0],pDoc->transmitted.m_dstrain.GetSize()
                ,m_dXscale,m_dYscale,GREEN,1,xo,yo);

        }
    }
////////////////////////////////////////////////////////////////////////////
///////////////     Graph Velocity Data
////////////////////////////////////////////////////////////////////////////
    if((m_nViewExtra==1)||
        ((pDoc->incidentbar.Velocity.m_bCalculated ||
        pDoc->transmittedbar.Velocity.m_bCalculated) &&
        !pDoc->flag.m_bProcessedResults))
    {
        if(!pDoc->flag.m_bProcessedResults)
        {
            xo=int(m_cxClient*.05);
            yo=int(m_cyClient*0.9);
        }else
        {
            xo=int(m_cxClient*.55);
            yo=int(m_cyClient*0.4);
        }
        yL=int(m_cyClient*.5*0.7);
        y_data_length=-yL*.9;


        if((pDoc->incidentbar.Velocity.m_bTime)&&
            (pDoc->incidentbar.Velocity.m_bCalculated))
        {
            m_dXscale=(x_data_length/(pDoc->data1.m_dDT*(pDoc->incidentbar.m_dVelocity.GetSize()))
```

```
);
            if (fabs(pDoc->incidentbar.Velocity.m_dmax)>fabs(pDoc->incidentbar.Velocity.m_dmin))
            {m_dYscale=y_data_length/pDoc->incidentbar.Velocity.m_dmax;
            }else{
                m_dYscale=y_data_length/fabs(pDoc->incidentbar.Velocity.m_dmin);
            }
            pDoc->scales.velocity=m_dYscale;
            GraphData(TRUE,pDoc->data1.m_dDT,0,0,
                &pDoc->incidentbar.m_dVelocity[0],pDoc->incidentbar.m_dVelocity.GetSize()
                ,m_dXscale,m_dYscale,BLUE,1,xo,yo);
            GraphTick(xo,yo,pDoc->data1.m_dDT*pDoc->incidentbar.m_dVelocity.GetSize(),
                y_data_length/pDoc->scales.velocity,pDoc->scales.time,
                pDoc->scales.velocity);

        }

        if((pDoc->transmittedbar.Velocity.m_bTime)&&
            (pDoc->transmittedbar.Velocity.m_bCalculated))
        {
            if(!pDoc->incidentbar.Force.m_bCalculated)
            {
                m_dXscale=(x_data_length/(pDoc->data2.m_dDT*(pDoc->transmittedbar.m_dVelocity.GetS
ize())));
                if (fabs(pDoc->transmittedbar.Velocity.m_dmax)>fabs(pDoc->transmittedbar.Velocity.
m_dmin))
                {
                    m_dYscale=y_data_length/pDoc->transmittedbar.Velocity.m_dmax;
                }else{
                    m_dYscale=y_data_length/fabs(pDoc->transmittedbar.Velocity.m_dmin);
                }
                pDoc->scales.velocity=m_dYscale;

                GraphTick(xo,yo,pDoc->data2.m_dDT*pDoc->transmittedbar.m_dVelocity.GetSize(),
                    y_data_length/pDoc->scales.velocity,pDoc->scales.time,
                    pDoc->scales.velocity);

            }

            GraphData(TRUE,pDoc->data2.m_dDT,0,0,
                &pDoc->transmittedbar.m_dVelocity[0],pDoc->transmittedbar.m_dVelocity.GetSize()
                ,m_dXscale,m_dYscale,GREEN,1,xo,yo);
        }
    }
////////////////////////////////////////////////////////////////////////////
///////////////     Graph Displacement Data
////////////////////////////////////////////////////////////////////////////
    if((m_nViewExtra==3)&&
        (pDoc->incidentbar.Displacement.m_bCalculated ||
        pDoc->transmittedbar.Displacement.m_bCalculated))
    {
        xo=int(m_cxClient*.55);
        yo=int(m_cyClient*0.4);

        yL=int(m_cyClient*.5*0.7);
        y_data_length=-yL*.9;


        if((pDoc->incidentbar.Displacement.m_bTime)&&
            (pDoc->incidentbar.Displacement.m_bCalculated))
        {
            m_dXscale=(x_data_length/(pDoc->data1.m_dDT*(pDoc->incidentbar.m_dDisplacement.GetSize
())));
            if (fabs(pDoc->incidentbar.Displacement.m_dmax)>fabs(pDoc->incidentbar.Displacement.m_
dmin))
            {m_dYscale=y_data_length/pDoc->incidentbar.Displacement.m_dmax;
            }else{
                m_dYscale=y_data_length/fabs(pDoc->incidentbar.Displacement.m_dmin);
            }
            pDoc->scales.displacement=m_dYscale;
            GraphData(TRUE,pDoc->data1.m_dDT,0,0,
                &pDoc->incidentbar.m_dDisplacement[0],pDoc->incidentbar.m_dDisplacement.GetSize()
                ,m_dXscale,m_dYscale,BLUE,1,xo,yo);
            GraphTick(xo,yo,pDoc->data1.m_dDT*pDoc->incidentbar.m_dDisplacement.GetSize(),
                y_data_length/pDoc->scales.displacement,pDoc->scales.time,
                pDoc->scales.displacement);
```

```
        }

        if((pDoc->transmittedbar.Displacement.m_bTime)&&
            (pDoc->transmittedbar.Displacement.m_bCalculated))
        {
            if(!pDoc->incidentbar.Force.m_bCalculated)
            {
                m_dXscale=(x_data_length/(pDoc->data2.m_dDT*(pDoc->transmittedbar.m_dDisplacement.
GetSize())));
                if (fabs(pDoc->transmittedbar.Displacement.m_dmax)>fabs(pDoc->transmittedbar.Displ
acement.m_dmin))
                {
                    m_dYscale=y_data_length/pDoc->transmittedbar.Displacement.m_dmax;
                }else{
                    m_dYscale=y_data_length/fabs(pDoc->transmittedbar.Displacement.m_dmin);
                }
                pDoc->scales.displacement=m_dYscale;

                GraphTick(xo,yo,pDoc->data2.m_dDT*pDoc->transmittedbar.m_dDisplacement.GetSize(),
                    y_data_length/pDoc->scales.displacement,pDoc->scales.time,
                    pDoc->scales.displacement);

            }

            GraphData(TRUE,pDoc->data2.m_dDT,0,0,
                &pDoc->transmittedbar.m_dDisplacement[0],pDoc->transmittedbar.m_dDisplacement.GetS
ize()
                ,m_dXscale,m_dYscale,GREEN,1,xo,yo);
        }
    }
////////////////////////////////////////////////////////////////////////////
//////////////      Graph Force Data
////////////////////////////////////////////////////////////////////////////
    if((m_nViewExtra==2)||
        ((pDoc->incidentbar.Force.m_bCalculated||
        pDoc->transmittedbar.Force.m_bCalculated)&&
        !pDoc->flag.m_bProcessedResults))
    {
        if(!pDoc->flag.m_bProcessedResults)
        {
            xo=int(m_cxClient*.55);
            yo=int(m_cyClient*0.9);

        }else
        {
            xo=int(m_cxClient*.55);
            yo=int(m_cyClient*0.4);
        }
        yL=int(m_cyClient*.5*0.7);
        y_data_length=-yL*.9;

        if((pDoc->incidentbar.Force.m_bTime)&&
            (pDoc->incidentbar.Force.m_bCalculated))
        {
            m_dXscale=(x_data_length/(pDoc->data1.m_dDT*(pDoc->incidentbar.m_dForce.GetSize())));
            if (fabs(pDoc->incidentbar.Force.m_dmax)>fabs(pDoc->incidentbar.Force.m_dmin))
            {m_dYscale=y_data_length/pDoc->incidentbar.Force.m_dmax;
            }else{
                m_dYscale=y_data_length/fabs(pDoc->incidentbar.Force.m_dmin);
            }
            pDoc->scales.force=m_dYscale;

            GraphData(TRUE,pDoc->data1.m_dDT,0,0,
                &pDoc->incidentbar.m_dForce[0],pDoc->incidentbar.m_dForce.GetSize()
                ,m_dXscale,m_dYscale,BLUE,1,xo,yo);

            GraphTick(xo,yo,pDoc->data1.m_dDT*pDoc->incidentbar.m_dForce.GetSize(),
                    y_data_length/pDoc->scales.force,pDoc->scales.time,
                    pDoc->scales.force);

        }

        if((pDoc->transmittedbar.Force.m_bTime)&&
            (pDoc->transmittedbar.Force.m_bCalculated))
        {
            if(!pDoc->incidentbar.Force.m_bCalculated)
```

```
            {
                m_dXscale=(x_data_length/(pDoc->data2.m_dDT*(pDoc->transmittedbar.m_dForce.GetSize
()));
                if (fabs(pDoc->transmittedbar.Force.m_dmax)>fabs(pDoc->transmittedbar.Force.m_dmin
))
                {m_dYscale=y_data_length/pDoc->transmittedbar.Force.m_dmax;
                }else{
                    m_dYscale=y_data_length/fabs(pDoc->transmittedbar.Force.m_dmin);
                }
                pDoc->scales.force=m_dYscale;

                GraphTick(xo,yo,pDoc->data2.m_dDT*pDoc->transmittedbar.m_dForce.GetSize(),
                    y_data_length/pDoc->scales.force,pDoc->scales.time,
                    pDoc->scales.force);

            }

            GraphData(TRUE,pDoc->data2.m_dDT,0,0,
                &pDoc->transmittedbar.m_dForce[0],pDoc->transmittedbar.m_dForce.GetSize()
                ,m_dXscale,m_dYscale,GREEN,1,xo,yo);
        }
    }
////////////////////////////////////////////////////////////////////////////
///////////////     Graph Processed Data
////////////////////////////////////////////////////////////////////////////

    if((pDoc->flag.m_bProcessedResults)) {
        xL=int(m_cxClient*.5*0.7);
        yL=int(-m_cyClient*.5*0.7);
        y_data_length=yL*0.9;

        if (fabs(pDoc->sample.m_dmax_strain)>fabs(pDoc->sample.m_dmin_strain))
        {   m_dXscale=x_data_length/pDoc->sample.m_dmax_strain;
        }else{
            m_dXscale=x_data_length/fabs(pDoc->sample.m_dmin_strain);}
        pDoc->scales.strain=m_dXscale;

        if (fabs(pDoc->sample.m_dmax_stress)>fabs(pDoc->sample.m_dmin_stress))
        {   m_dYscale=y_data_length/pDoc->sample.m_dmax_stress;
        }else{
            m_dYscale=y_data_length/fabs(pDoc->sample.m_dmin_stress);}

        pDoc->scales.stress=m_dYscale;

        if  (pDoc->sample.m_dengstrain.GetSize()>pDoc->sample.m_dengstress.GetSize())
            {number=pDoc->sample.m_dengstress.GetSize();
        }else{
            number=pDoc->sample.m_dengstrain.GetSize();}
    //draw stress strain in magenta
        xo=int(m_cxClient*.05);     //stress strain data
        yo=int(m_cyClient*.9);

        GraphData(FALSE,0,0,&pDoc->sample.m_dengstrain[0],
            &pDoc->sample.m_dengstress[0],number,m_dXscale,m_dYscale,
            MAGENTA,1,xo,yo);

        GraphTick(xo,yo,pDoc->sample.m_dmax_strain,
                y_data_length/pDoc->scales.stress,pDoc->scales.strain,
                pDoc->scales.stress);

        if (fabs(pDoc->sample.m_dmax_strainrate)>fabs(pDoc->sample.m_dmin_strainrate))
        {m_dYscale=y_data_length/pDoc->sample.m_dmax_strainrate;
        }else{
         m_dYscale=y_data_length/fabs(pDoc->sample.m_dmin_strainrate);}
        pDoc->scales.strainrate=m_dYscale;

        m_dXscale=(x_data_length/(pDoc->data1.m_dDT*(pDoc->sample.m_dengstrainrate.GetSize())));
        pDoc->scales.strainratetime=m_dXscale;
        //draw strainrate in orange
        xo=int(m_cxClient*.55);     //strain rate data
        yo=int(m_cyClient*.9);

        GraphData(TRUE,pDoc->data1.m_dDT,0,0,
            &pDoc->sample.m_dengstrainrate[0],pDoc->sample.m_dengstrainrate.GetSize()

            ,m_dXscale,m_dYscale,ORANGE,1,xo,yo);
```

```
        GraphTick(xo,yo,pDoc->data1.m_dDT*pDoc->sample.m_dengstrainrate.GetSize(),
                    y_data_length/pDoc->scales.strainrate,pDoc->scales.strainratetime,
                    pDoc->scales.strainrate);


    }

}

/////////////////////////////////////////////////////////////////////////////
// CCSHBView printing

BOOL CCSHBView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CCSHBView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CCSHBView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CCSHBView diagnostics

#ifdef _DEBUG
void CCSHBView::AssertValid() const
{
    CView::AssertValid();
}

void CCSHBView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CCSHBDoc* CCSHBView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CCSHBDoc)));
    return (CCSHBDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CCSHBView message handlers

void CCSHBView::OnSize(UINT nType, int cx, int cy)
{

    CView::OnSize(nType, cx, cy);

    m_cxClient=cx;
    m_cyClient=cy;
}


BOOL CCSHBView::DrawAxis(CDC *pDC, int x_origin,int y_origin,int x_length,int y_length,
                         CString strTitle, CString strXAxis, CString strYAxis)
{
    CPen newpen;
    CPen *pOldPen;
    CFont fnYaxis,fnXAxis,fnTitle;
    CFont* pOldFont;


    if (!newpen.CreatePen(PS_SOLID,1,RGB(0,0,0)))
        return FALSE;
    pOldPen=pDC->SelectObject(&newpen);
```

```
    pDC->SetViewportExt(m_cxClient,-m_cyClient);
    pDC->SetViewportOrg(x_origin,y_origin);


    fnTitle.CreateFont(20,0,0,0,FW_BOLD,FALSE,FALSE,FALSE,ANSI_CHARSET,
        OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH+FF_DONTCARE,
        "Arial");
    fnXAxis.CreateFont(18,0,0,0,FW_BOLD,FALSE,FALSE,FALSE,ANSI_CHARSET,
        OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH+FF_DONTCARE,
        "Arial");
    fnYaxis.CreateFont(17,0,900,0,FW_BOLD,FALSE,FALSE,FALSE,ANSI_CHARSET,
        OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,PROOF_QUALITY,DEFAULT_PITCH+FF_DONTCARE,
        "Arial");

    pOldFont=pDC->SelectObject(&fnTitle);
    pDC->SetTextAlign(TA_CENTER+TA_BOTTOM);
    if(strTitle!="")
    {pDC->TextOut(int(x_length/2),int(-y_length),strTitle);}

    pDC->SelectObject(&fnXAxis);
    pDC->SetTextAlign(TA_CENTER+TA_TOP);
    if(strXAxis!="")
    {
        if(y_length<0)
        {
            pDC->TextOut(int(x_length/2),int(-y_length),strXAxis);
        }else{
            pDC->TextOut(int(x_length/2),0,strXAxis);
        }
    }


    pDC->SelectObject(&fnYaxis);
    if (strYAxis!="")
    {
        if(y_length<0)
        {
            pDC->SetTextAlign(TA_CENTER+TA_BOTTOM);
            pDC->TextOut(0,0,strYAxis);
        }else{
            pDC->SetTextAlign(TA_CENTER+TA_BOTTOM);
            pDC->TextOut(0,int(-y_length/2),strYAxis);
        }
    }
    pDC->MoveTo(0,0);
    pDC->LineTo(x_length,0);
    pDC->MoveTo(0,0);
    pDC->LineTo(0,-y_length);


    pDC->SelectObject(pOldFont);
    pDC->SelectObject(pOldPen);
    return TRUE;
}

BOOL CCSHBView::GraphData(BOOL y_vs_t,double dx,double xo,double x[], double y[], int n, double sc
ale_x, double scale_y,DWORD colour, int size,int x_origin,int y_origin)
{

    CClientDC dc(this);
    CPen newpen;
//  CRect rcClient;

    if (!newpen.CreatePen(PS_SOLID,size,colour))
        return FALSE;
    CPen *pOldPen=dc.SelectObject(&newpen);

/*
    GetClientRect(&rcClient);
    dc.SetViewPortExt(cSize
    dc.SetViewportOrg(x_origin,y_origin);

    dc.SetMapMode(MM_ANISOTROPIC);

    dc.SetWindowExt(x_scale,y_scale);
    dc.DPtoLP(&rcClient);
```

```
*/
    dc.SetViewportOrg(x_origin,y_origin);
    if(y_vs_t)      //plot y versus time
    {   dc.MoveTo(int (xo*scale_x*dx),int(y[0]*scale_y));
        for(int a=1;a<n;a++)
        {
            dc.LineTo(int((a+xo)*dx*scale_x),int(y[a]*(scale_y)));
        }
    }else                   //plot y versus x
    {
        dc.MoveTo(int (scale_x*x[0]),int(y[0]*scale_y));
        for(int a=1;a<n;a++)
        {
            dc.LineTo(int((x[a])*scale_x),int(y[a]*(scale_y)));
        }
    }

    dc.SelectObject(pOldPen);
    return TRUE;
}



BOOL CCSHBView::GraphFrequency(BOOL y_vs_t,double dx,double xo,double x[], double y[], int n, doub
le scale_x, double scale_y,DWORD colour, int size)
{
/*  CClientDC dc(this);
    CPen newpen;
    CRect rcClient;


    if (!newpen.CreatePen(PS_SOLID,size,colour))
        return FALSE;
    CPen *pOldPen=dc.SelectObject(&newpen);
    dc.SetViewportOrg(m_cxClient/20,m_cyClient/2);

    //plot real data
    dc.MoveTo(int (xo*scale_x*dx),int(y[0]*scale_y));
    for(int a=2;a<n;a+=2)
    {
        //dc.LineTo(int((a+xo)*dx*scale_x),int(y[a]*(scale_y)));
        dc.LineTo(int((a+xo)*dx),int(y[a]));
    }


    dc.SelectObject(pOldPen);
    */
    return TRUE;
}

void CCSHBView::OnMouseMove(UINT nFlags, CPoint point)
{

    CString strMessage,strpoint;
    CCSHBDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    if(g_pIndex)
    {
        if (g_pIndex->m_nIndex!=0)
        {
            strMessage="Please Choose ";
            switch (g_pIndex->m_nIndex)
            {
            case 1:
                strMessage+="Incident Start Point";
                break;
            case(2):
                strMessage+="Reflected Start Point";
                break;
            case 3:
                strMessage+="Transmitted Start Point";
                break;
            case 4:
                strMessage+="Incident End Point";
```

```
            break;
        case 5:
            strMessage+="Reflected End Point";
            break;
        case 6:
            strMessage+="Transmitted End Point";
            break;
        default:
            strMessage="Error!";
            break;
        }
        if ((pDoc->data1.m_dDT!=0)&&
            (!pDoc->incidentbar.Force.m_bCalculated &&
            !pDoc->transmittedbar.Force.m_bCalculated))

        {   strpoint.Format(" Index: %i, Time: %8.6f",
                int((point.x-m_cxClient*.05)/(pDoc->scales.time*pDoc->data1.m_dDT)),
                ((point.x-m_cxClient*.05)/(pDoc->scales.time)));
            strMessage+=strpoint;
        }else if ((pDoc->data2.m_dDT!=0)&&
            (!pDoc->incidentbar.Force.m_bCalculated &&
            !pDoc->transmittedbar.Force.m_bCalculated))

        {
            strpoint.Format(" Index: %i, Time: %8.6f",
                int((point.x-m_cxClient*.05)/(pDoc->scales.time*pDoc->data2.m_dDT)),
                ((point.x-m_cxClient*.05)/(pDoc->scales.time)));
            strMessage+=strpoint;
        }else
        {//small graph in left corner
            strpoint.Format(" Index: %i, Time: %8.6f",
                int((point.x-m_cxClient*.05)/(pDoc->scales.time*pDoc->data2.m_dDT)),
                ((point.x-m_cxClient*.05)/(pDoc->scales.time)));
            strMessage+=strpoint;
        }

    }

}else if
    ((pDoc->incidentbar.Force.m_bCalculated||
    pDoc->transmittedbar.Force.m_bCalculated)&&
    !pDoc->flag.m_bProcessedResults)
{
    if (point.x>0.5*m_cxClient)
    { // in right half
        if (point.y>0.5*m_cyClient)
        {//in bottom right quadrant Force
            strMessage.Format("Time: %8.6f s Force: %8.2f N",
                ((point.x-m_cxClient*.55)/(pDoc->scales.time)),
                ((point.y-m_cyClient*.9)/(pDoc->scales.force)));

        }else
        {//top right quadrant strain data
            if(m_nViewExtra==3)
            {
                strMessage.Format("Time: %8.5f s Displacement: %8.6fmm",
                    (point.x-m_cxClient*.55)/(pDoc->scales.time),
                    ((point.y-m_cyClient*0.4)/(pDoc->scales.displacement))*1000);
            }else
            {
                strMessage.Format("Time: %8.5f s Strain: %8.6f",
                    (point.x-m_cxClient*.55)/(pDoc->scales.time),
                    ((point.y-m_cyClient*0.225)/(pDoc->scales.strain)));
            }

        }
    }else
    {//left half
        if (point.y>0.5*m_cyClient)
        {//in bottom left quadrant velocity
            strMessage.Format("Time: %8.5f s Velocity: %8.2f m/s",
                (point.x-m_cxClient*.05)/(pDoc->scales.time),
                ((point.y-m_cyClient*0.9)/(pDoc->scales.velocity)));

        }else
        {//top left quadrant voltage time
```

```
                strMessage.Format("Time: %8.6f s Voltage: %8.6f V",
                    ((point.x-m_cxClient*.05)/(pDoc->scales.time)),
                    ((point.y-m_cyClient*0.225)/(pDoc->scales.volts)));
            }
        }
    }else if    (pDoc->flag.m_bProcessedResults)
    {
        if (point.x>0.5*m_cxClient)
        { // in right half
            if (point.y>0.5*m_cyClient)
            {//in bottom right quadrant strain rate
                strMessage.Format("Time: %8.6f s StrainRate: %6.0f s^-1",
                    ((point.x-m_cxClient*.55)/(pDoc->scales.strainratetime)),
                    ((point.y-m_cyClient*.9)/(pDoc->scales.strainrate)));

            }else
            {//top right quadrant strain data
                switch (m_nViewExtra)
                {
                    case 0: //Quad has strain data
                        strMessage.Format("Time: %8.5f s Strain: %8.6f",
                            (point.x-m_cxClient*.55)/(pDoc->scales.straintime),
                            ((point.y-m_cyClient*0.225)/(pDoc->scales.strain)));
                        break;
                    case 1://Quad has Velocity Data
                        strMessage.Format("Time: %8.5f s Velocity: %8.2fm/s",
                            (point.x-m_cxClient*.55)/(pDoc->scales.time),
                            ((point.y-m_cyClient*0.4)/(pDoc->scales.velocity)));
                        break;
                    case 2://Quad has Force Data
                        strMessage.Format("Time: %8.5f s Force: %8.2fN",
                            (point.x-m_cxClient*.55)/(pDoc->scales.time),
                            ((point.y-m_cyClient*0.4)/(pDoc->scales.force)));
                        break;
                    case 3:
                        strMessage.Format("Time: %8.5f s Displacement: %8.6fmm",
                            (point.x-m_cxClient*.55)/(pDoc->scales.time),
                            ((point.y-m_cyClient*0.4)/(pDoc->scales.displacement))*1000);
                        break;
                }//end switch
            }
        }else
        {//left half
            if (point.y>0.5*m_cyClient)
            {//in bottom left quadrant stress strain
                strMessage.Format("Strain: %8.5f Stress: %8.2f MPa",
                    (point.x-m_cxClient*.05)/(pDoc->scales.strain),
                    ((point.y-m_cyClient*.9)/(pDoc->scales.stress))/pow(10,6));

            }else
            {//top left quadrant voltage time
                strMessage.Format("Time: %8.6f s Voltage: %8.6f V",
                    ((point.x-m_cxClient*.05)/(pDoc->scales.time)),
                    ((point.y-m_cyClient*0.225)/(pDoc->scales.volts)));
            }
        }
    }else if ((pDoc->flag.m_bReadIncident)||(pDoc->flag.m_bReadTransmitted)){

        strMessage.Format("Time: %8.6fs Volts: %8.6fV",((point.x-m_cxClient*0.05)/(pDoc->scales.ti
me)),
            ((point.y-m_cyClient*.5)/(pDoc->scales.volts)));
    }

    GetDocument()->SetTitle(strMessage);


}

void CCSHBView::OnViewfreq()
{
/*  CCSHBDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    double dw=0,yscale=1,xscale=1;
    double max=-100000000,min=1000000000000;
    int i=0;
```

```
    if((pDoc->flag.m_bReadIncident)&&(!pDoc->incident.m_bTime)) {
        //scaling factor for x and y
    //w(n)=2*Pi*f(n)=2*Pi*n/(N*delta_time)
        dw=2*3.14159/(pDoc->data1.m_dDT*pDoc->incident.m_dstrain.GetSize());

        xscale=((m_cxClient*3/4)/(dw*(pDoc->incident.m_dstrain.GetSize())));

        for (i=0;i<pDoc->incident.m_dstrain.GetSize();i+=2)
        {
            if (pDoc->incident.m_dstrain[i]>max)
            {   max=pDoc->incident.m_dstrain[i];
            }else if(pDoc->incident.m_dstrain[i]<min)
            {   min=pDoc->incident.m_dstrain[i];
            }
        }

        for (i=0;i<pDoc->reflected.m_dstrain.GetSize();i+=2)
        {
            if (pDoc->reflected.m_dstrain[i]>max)
            {   max=pDoc->reflected.m_dstrain[i];
            }else if(pDoc->reflected.m_dstrain[i]<min)
            {   min=pDoc->reflected.m_dstrain[i];
            }
        }

        if (fabs(max)>fabs(min))
        {   yscale=-y_data_length/max;
        }else{
            yscale=-y_data_length/fabs(min);}

        GraphFrequency(TRUE,dw,0,0,&pDoc->incident.m_dstrain[0],
            pDoc->incident.m_dstrain.GetSize()
            ,xscale,yscale,ORANGE,1);

        GraphFrequency(TRUE,dw,0,0,&pDoc->reflected.m_dstrain[0],
            pDoc->reflected.m_dstrain.GetSize()
            ,xscale,yscale,BLUE,1);



    }
    */
}

//DEL void CCSHBView::OnDrawprocesseddata()
//DEL {
//DEL   // TODO: Add your command handler code here
//DEL
//DEL }

void CCSHBView::AllDoneModeless()
{
    //Set Global pointer to Null so that
    //it can be recalled

    g_pIndex = NULL;
}

BOOL CCSHBView::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
    HCURSOR hCursor;

    hCursor = AfxGetApp()->LoadCursor(IDC_CROSSHAIR);
    SetCursor(hCursor);
    return TRUE;
//  return CView::OnSetCursor(pWnd, nHitTest, message);
}

void CCSHBView::OnCancelMode()
{
    CView::OnCancelMode();

    // TODO: Add your message handler code here

}
```

```
void CCSHBView::OnPreViewindices()
{
    //  index.m_nEnd1=int(dlg.m_dEnd1);
    //  index.m_nStart1=int(dlg.m_dStart1);
    //  flag.m_bChoosenPoints=TRUE;
    CCSHBDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    if (!g_pIndex)
    {g_pIndex=new CIndexDlg();
        g_pIndex->m_nData1Size=pDoc->data1.m_dvolts.GetSize();
        g_pIndex->m_nData2Size=pDoc->data2.m_dvolts.GetSize();
        g_pIndex->m_dIDT=pDoc->data1.m_dDT;
        g_pIndex->m_dTDT=pDoc->data2.m_dDT;
        g_pIndex->m_dIs=pDoc->incident.m_nbegin;
        g_pIndex->m_dIe=pDoc->incident.m_nend;
        g_pIndex->m_dRs=pDoc->reflected.m_nbegin;
        g_pIndex->m_dRe=pDoc->reflected.m_nend;
        g_pIndex->m_dTs=pDoc->transmitted.m_nbegin;
        g_pIndex->m_dTe=pDoc->transmitted.m_nend;
        g_pIndex->UpdateData(FALSE);
    }


}

void CCSHBView::OnLButtonDown(UINT nFlags, CPoint point)
{

    CCSHBDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);


    if(g_pIndex)
    {
        if (g_pIndex->m_nIndex!=0)
        {
            switch (g_pIndex->m_nIndex)
            {
            case 1:
                g_pIndex->m_dIs=int((point.x-m_cxClient*.05)/(pDoc->data1.m_dDT*pDoc->scales.time)
);
                break;
            case(2):
                g_pIndex->m_dRs=int((point.x-m_cxClient*.05)/(pDoc->data1.m_dDT*pDoc->scales.time)
);
                break;
            case 3:
                g_pIndex->m_dTs=int((point.x-m_cxClient*.05)/(pDoc->data1.m_dDT*pDoc->scales.time)
);
                break;
            case 4:
                g_pIndex->m_dIe=int((point.x-m_cxClient*.05)/(pDoc->data1.m_dDT*pDoc->scales.time)
);
                break;
            case 5:
                g_pIndex->m_dRe=int((point.x-m_cxClient*.05)/(pDoc->data1.m_dDT*pDoc->scales.time)
);
                break;
            case 6:
                g_pIndex->m_dTe=int((point.x-m_cxClient*.05)/(pDoc->data1.m_dDT*pDoc->scales.time)
);
                break;
            default:
                g_pIndex->m_nIndex=0;
                break;
            }
        }
        g_pIndex->m_nIndex=0;
        g_pIndex->ValidateData();
        g_pIndex->UpdateData(FALSE);
        g_pIndex->ShowWindow(SW_SHOW);

    }
```

```
    CView::OnLButtonDown(nFlags, point);
}



void CCSHBView::OnViewVelocity()
{

    m_nViewExtra=1;//for velocity
    Invalidate(TRUE);


}

void CCSHBView::OnViewForce()
{
    m_nViewExtra=2;//for force
    Invalidate(TRUE);

}

void CCSHBView::OnViewStrain()
{

    m_nViewExtra=0;//for strain
    Invalidate(TRUE);

}
void CCSHBView::OnViewDisplacement()
{
    m_nViewExtra=3;//for strain
    Invalidate(TRUE);

}

void CCSHBView::OnUpdateViewDisplacement(CCmdUI* pCmdUI)
{
    if (m_nViewExtra==3)
    {pCmdUI->SetCheck(TRUE);}
    else
    {pCmdUI->SetCheck(FALSE);}


}

void CCSHBView::OnUpdateViewVelocity(CCmdUI* pCmdUI)
{
    if (m_nViewExtra==1)
    {pCmdUI->SetCheck(TRUE);}
    else
    {pCmdUI->SetCheck(FALSE);}

}

void CCSHBView::OnUpdateViewForce(CCmdUI* pCmdUI)
{
    if (m_nViewExtra==2)
    {pCmdUI->SetCheck(TRUE);}
    else
    {pCmdUI->SetCheck(FALSE);}

}

void CCSHBView::OnUpdateViewStrain(CCmdUI* pCmdUI)
{
    if (m_nViewExtra==0)
    {pCmdUI->SetCheck(TRUE);}
    else
    {pCmdUI->SetCheck(FALSE);}

}


BOOL CCSHBView::GraphTick(int x_origin,int y_origin,double maxx,double maxy,double scalex,double s
```

```
caley)
{
    CClientDC dc(this);
    double ticklength=.02;
    CPen newpen;
    CString strMessg;
    CFont fnXTick,fnYTick;
    CFont* pOldFont;

//  CRect rcClient;

    if (!newpen.CreatePen(PS_SOLID,1,BLACK))
        return FALSE;
    CPen *pOldPen=dc.SelectObject(&newpen);

    dc.SetViewportOrg(x_origin,y_origin);

    dc.MoveTo(int(maxx*scalex),0);
    dc.LineTo(int(maxx*scalex),int(ticklength*m_cyClient));

    dc.MoveTo(0,int(maxy*scaley));
    dc.LineTo(-int(ticklength*m_cxClient),int(maxy*scaley));

    fnXTick.CreateFont(14,0,0,0,FW_LIGHT,FALSE,FALSE,FALSE,ANSI_CHARSET,
        OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH+FF_DONTCARE,
        "Arial");

    fnYTick.CreateFont(12,0,900,0,FW_LIGHT,FALSE,FALSE,FALSE,ANSI_CHARSET,
        OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH+FF_DONTCARE,
        "Arial");

    if((maxx<1)&&(maxx>-1))
    {   strMessg.Format("%8.6f",maxx);
    }else
    {
            strMessg.Format("%8.2f",maxx);
    }

    dc.SetTextAlign(TA_CENTER+TA_TOP);
    pOldFont=dc.SelectObject(&fnXTick);
    dc.TextOut(int(maxx*scalex),int(ticklength*m_cyClient),strMessg);

    if((maxy<1)&&(maxy>-1))
    {   strMessg.Format("%8.6f",maxy);
    }else if((maxy>10000)||(maxy<-10000))
    {   strMessg.Format("%8.2f",maxy/pow(10,6)); //assume >10k should only be Stress
    }else
    {   strMessg.Format("%8.2f",maxy);
    }
    dc.SetTextAlign(TA_CENTER+TA_BOTTOM);
    dc.SelectObject(&fnYTick);
    dc.TextOut(-int(ticklength*m_cxClient),int(maxy*scaley),strMessg);

    dc.SelectObject(pOldPen);
    dc.SelectObject(pOldFont);



    return TRUE;
}

void CCSHBView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
{

//  if (pInfo) m_rcPrintRect=pInfo->m_rectDraw;
    pDC->SetWindowOrg(pInfo->m_rectDraw.left,-pInfo->m_rectDraw.top);
    CView::OnPrint(pDC, pInfo);
}

void CCSHBView::OnInitialUpdate()
{
//  SetScrollSizes( MM_LOENGLISH, GetDocument()->GetDocSize() );


    CView::OnInitialUpdate();
```

```
    // TODO: Add your specialized code here and/or call the base class
}
```

```
// ExportDataDlg.cpp : implementation file
//

#include "stdafx.h"
#include "CSHB.h"
#include "ExportDataDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CExportDataDlg dialog


CExportDataDlg::CExportDataDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CExportDataDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CExportDataDlg)
    m_bEngineeringData = FALSE;
    m_bHeader = FALSE;
    m_bStrains = FALSE;
    m_bTrueData = FALSE;
    m_bForces = FALSE;
    m_bVelocities = FALSE;
    m_szExportFileName = _T("");
    m_bDisplacements = FALSE;
    //}}AFX_DATA_INIT

}


void CExportDataDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CExportDataDlg)
    DDX_Check(pDX, IDC_EXP_ENGDATA, m_bEngineeringData);
    DDX_Check(pDX, IDC_EXP_HEADER, m_bHeader);
    DDX_Check(pDX, IDC_EXP_STRAINS, m_bStrains);
    DDX_Check(pDX, IDC_EXP_TRUEDATA, m_bTrueData);
    DDX_Check(pDX, IDC_FORCES, m_bForces);
    DDX_Check(pDX, IDC_VELOCITIES, m_bVelocities);
    DDX_Text(pDX, IDC_EXPORTFILE, m_szExportFileName);
    DDX_Check(pDX, IDC_DISPLACEMENTS, m_bDisplacements);
    //}}AFX_DATA_MAP

}


BEGIN_MESSAGE_MAP(CExportDataDlg, CDialog)
    //{{AFX_MSG_MAP(CExportDataDlg)
    ON_BN_CLICKED(IDC_BROWSE, OnBrowse)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CExportDataDlg message handlers



void CExportDataDlg::SetAllowed()
{
    if (m_bEngineeringAllowed==FALSE)
    {   GetDlgItem(IDC_EXP_ENGDATA)->EnableWindow(FALSE);
        GetDlgItem(IDC_EXP_TRUEDATA)->EnableWindow(FALSE);
    }
    if (m_bForcesAllowed==FALSE)
    {   GetDlgItem(IDC_FORCES)->EnableWindow(FALSE);}

    if (m_bStrainsAllowed==FALSE)
    {   GetDlgItem(IDC_EXP_STRAINS)->EnableWindow(FALSE);}

    if (m_bVelocitiesAllowed==FALSE)
    {   GetDlgItem(IDC_VELOCITIES)->EnableWindow(FALSE);}
```

```
}



void CExportDataDlg::OnBrowse()
{
    bool stopflag = false;
    OPENFILENAME SaveFileName;
    TCHAR szFile[256000] = "\0";    // File list
    static char szFilter[] =  "Export Files (*.exp)\0*.exp\0"
                              "All Files (*.*)\0*.*\0\0";

    //LPTSTR szFile = new TCHAR[m_szExportFileName.GetLength()+1];
    //_tcscpy(szFile, m_szExportFileName);


        SaveFileName.lStructSize = sizeof(OPENFILENAME);
        SaveFileName.hwndOwner = NULL;
        SaveFileName.hInstance = NULL;
        SaveFileName.lpstrFilter = szFilter;
        SaveFileName.lpstrCustomFilter = NULL;
        SaveFileName.nMaxCustFilter = 0;
        SaveFileName.nFilterIndex = 0;
        SaveFileName.lpstrFile = szFile;
        SaveFileName.nMaxFile = sizeof(szFile);
        SaveFileName.lpstrFileTitle = NULL;
        SaveFileName.nMaxFileTitle = 0;
        SaveFileName.lpstrInitialDir =NULL;
        SaveFileName.lpstrTitle = "Export File";
        SaveFileName.nFileOffset = 0;
        SaveFileName.nFileExtension = 0;
        SaveFileName.lpstrDefExt = "";
        SaveFileName.lpfnHook = NULL;
        SaveFileName.lpTemplateName = NULL;
        SaveFileName.Flags = OFN_EXPLORER |OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT;

    // Open dialog box
        GetSaveFileName(&SaveFileName);
        m_szExportFileName=szFile;
        CDialog::UpdateData(FALSE);



}

void CExportDataDlg::OnOK()
{
    CDialog::OnOK();
}

void CExportDataDlg::OnCancel()
{
    // TODO: Add extra cleanup here

    CDialog::OnCancel();
}
```

```
// FFTUtilityDlg.cpp : implementation file
//

#include "stdafx.h"
#include "cshb.h"
#include "FFTUtilityDlg.h"
#include "CSHBDoc.h"
#include <fstream.h>
#include <math.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CFFTUtilityDlg dialog


CFFTUtilityDlg::CFFTUtilityDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CFFTUtilityDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CFFTUtilityDlg)
    m_szInputFile = _T("");
    m_szOutputFile = _T("");
    m_nTransform = -1;
    //}}AFX_DATA_INIT
    m_nTransform=0;
}


void CFFTUtilityDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CFFTUtilityDlg)
    DDX_Text(pDX, IDC_INPUTNAME, m_szInputFile);
    DDX_Text(pDX, IDC_OUTPUTNAME, m_szOutputFile);
    DDX_Radio(pDX, IDC_FFT, m_nTransform);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CFFTUtilityDlg, CDialog)
    //{{AFX_MSG_MAP(CFFTUtilityDlg)
    ON_BN_CLICKED(IDC_BROWSE_IN, OnBrowseIn)
    ON_BN_CLICKED(IDC_BROWSE_OUT, OnBrowseOut)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFFTUtilityDlg message handlers

void CFFTUtilityDlg::OnBrowseIn()
{
    CDialog::UpdateData(TRUE);

    bool stopflag = false;


    OPENFILENAME OpenFileName;
    TCHAR szFile[256000] = "\0";    // File list
    static char szFilter[] = "All Files (*.*)\0*.*\0\0";

    OpenFileName.lStructSize = sizeof(OPENFILENAME);
    OpenFileName.hwndOwner = NULL;
    OpenFileName.hInstance = NULL;
    OpenFileName.lpstrFilter = szFilter;
    OpenFileName.lpstrCustomFilter = NULL;
    OpenFileName.nMaxCustFilter = 0;
    OpenFileName.nFilterIndex = 0;
    OpenFileName.lpstrFile = szFile;
    OpenFileName.nMaxFile = sizeof(szFile);
    OpenFileName.lpstrFileTitle = NULL;
    OpenFileName.nMaxFileTitle = 0;
    OpenFileName.lpstrInitialDir = m_szInputFile;
    OpenFileName.lpstrTitle = "Select Input File";
```

```
    OpenFileName.nFileOffset = 0;
    OpenFileName.nFileExtension = 0;
    OpenFileName.lpstrDefExt = "";
    OpenFileName.lpfnHook = NULL;
    OpenFileName.lpTemplateName = NULL;
    OpenFileName.Flags = OFN_EXPLORER | OFN_FILEMUSTEXIST;

    // Open dialog box
    GetOpenFileName(&OpenFileName);
    // Filenames are stored in szFile;
    // Assign the value of szFile to m_FileList
    m_szInputFile = szFile;
    // Update the information in the textbox to show the current files selected


    CDialog::UpdateData(FALSE);


}

void CFFTUtilityDlg::OnBrowseOut()
{

    bool stopflag = false;
    OPENFILENAME SaveFileName;
    TCHAR szFile[256000] = "\0";    // File list
    static char szFilter[] =  "All Files (*.*)\0*.*\0\0";



        SaveFileName.lStructSize = sizeof(OPENFILENAME);
        SaveFileName.hwndOwner = NULL;
        SaveFileName.hInstance = NULL;
        SaveFileName.lpstrFilter = szFilter;
        SaveFileName.lpstrCustomFilter = NULL;
        SaveFileName.nMaxCustFilter = 0;
        SaveFileName.nFilterIndex = 0;
        SaveFileName.lpstrFile = szFile;
        SaveFileName.nMaxFile = sizeof(szFile);
        SaveFileName.lpstrFileTitle = NULL;
        SaveFileName.nMaxFileTitle = 0;
        SaveFileName.lpstrInitialDir =NULL;
        SaveFileName.lpstrTitle = "Output File";
        SaveFileName.nFileOffset = 0;
        SaveFileName.nFileExtension = 0;
        SaveFileName.lpstrDefExt = "";
        SaveFileName.lpfnHook = NULL;
        SaveFileName.lpTemplateName = NULL;
        SaveFileName.Flags = OFN_EXPLORER |OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT;

    // Open dialog box
        GetSaveFileName(&SaveFileName);
        m_szOutputFile=szFile;
        CDialog::UpdateData(FALSE);


}

BOOL CFFTUtilityDlg::GetInputData()
{
//  CCSHBDoc  P;
    char string[1000];
    char seps[]   = " ,\t";
    int i=0,power=0,N=0,a=0;
    BOOL bTime;

    m_dData1.RemoveAll();
    m_dData2.RemoveAll();

    ifstream ifsin(m_szInputFile,ios::in|ios::nocreate);
    if (!ifsin)
        return FALSE;


    if (m_nTransform==0)                            //Read in Time and Value
    {   ifsin.getline(string,1000,'\n');
```

```
        i=0;
        do{
            if ((string[0] != '"')&&(string[0]!='\0'))
            {
                m_dData1.Add((atof( strtok( string, seps ))));
                m_dData2.Add((atof( strtok( NULL, seps ))));
                i++;
            }
            ifsin.getline(string,1000,'\n');

        }   while(ifsin!=0);

        //makes sure the number of points is a power of 2
        N=m_dData2.GetSize();
        power=int((log(N)/log(2)));
        if (N>int(pow(2,power)))
        {power++;}
        N=int(pow(2,power));
        for (a=i;a<N;a++)       //add points
        {m_dData2.Add(0);}
        bTime=TRUE;
        CCSHBDoc::Realft(&m_dData2[0]-1,m_dData2.GetSize(),&bTime); //Perform FFT on data

    }
    else
    {                                   //Read in Freq, Real Imag
        ifsin.getline(string,1000,'\n');
        i=0;

        do{
            if ((string[0] != '"')&&(string[0]!='\0'))
            {
                m_dData1.Add((atof( strtok( string, seps ))); //Freq
                m_dData2.Add((atof( strtok( NULL, seps ))));   //real
                m_dData2.Add((atof( strtok( NULL, seps ))));    //imag
                i+=2;
            }
            ifsin.getline(string,1000,'\n');
        }   while(ifsin!=0);

        //strip out Nyquist value and put into complex part of first set of terms
        //needed for Realft
        m_dData2[1]=m_dData2[m_dData2.GetSize()-1];
        m_dData2.RemoveAt(m_dData2.GetUpperBound()-2,2);
        N=m_dData2.GetSize();
        power=int((log(N)/log(2)));
        if (N>int(pow(2,power)))//if the number of points is not a power of 2
        {power++;}
        N=int(pow(2,power));
        for (a=i;a<N;a++)
        {m_dData2.Add(0);}
        bTime=FALSE;
        CCSHBDoc::Realft(&m_dData2[0]-1,m_dData2.GetSize(),&bTime);
    }

    ifsin.close();



    return TRUE;
}

void CFFTUtilityDlg::OnOK()
{
    BeginWaitCursor();
    UpdateData();

    GetInputData();
    OutputData();
    EndWaitCursor();
    CDialog::OnOK();
}

BOOL CFFTUtilityDlg::OutputData()
{
```

```
    double r=0,modulus=0,angle=0,frequency,time=0,amplitude=0,dx=0;
    int i=0,N=0;
    double dNyquist=0;
    CArray <double,double> dPolar;
    BOOL bPolar=FALSE;

    dPolar.RemoveAll();

    ofstream outfile(m_szOutputFile,ios::out); //create output file
    outfile<<"\"Input from:"<<m_szInputFile<<endl;


    dx=m_dData1[1]-m_dData1[0];
    N=m_dData2.GetSize();
    if(m_nTransform==0)
    {

        //get Nyquist Value add to end
        m_dData2.Add(m_dData2[1]);
        m_dData2.Add(0);
        m_dData2[1]=0;
        dPolar.Append(m_dData2);
        bPolar=FALSE;
        CCSHBDoc::ChangeForm(dPolar.GetSize(),&dPolar[0],&bPolar);
        //output freq real imag mod ang
        outfile<<"\"\tFrequency(Hz)\tReal\tImaginary\tModulus\tAngle(radians)"<<endl;
        for (i=0;i<m_dData2.GetSize();i+=2)
        {


            frequency=(i/2)/(N*dx);

            outfile<<frequency<<"\t"<<m_dData2[i]<<"\t"<<m_dData2[i+1]<<"\t"<<
                dPolar[i]<<"\t"<<dPolar[i+1]<<endl;
        }
        //output NyquistValue
//      i++;
//      frequency=(i/2)/(N*dx);
//      outfile<<frequency<<"\t"<<dNyquist<<"\t"<<0<<"\t"<<
//              dNyquist<<"\t"<<0<<endl;
    }
    else
    {
        outfile<<"\"\tTime(s)\tValue"<<endl;

        for (i=0;i<N;i++)
        {

            outfile<<i*((1/dx)/N)<<"\t"<<m_dData2[i]<<endl;
        }
    }


    return TRUE;

}



void CFFTUtilityDlg::OnCancel()
{
    // TODO: Add extra cleanup here

    CDialog::OnCancel();
}
```

```
// IndexDlg.cpp : implementation file
//

#include "stdafx.h"
#include "cshb.h"
#include "IndexDlg.h"
#include "CSHBDoc.h"
#include "CSHBView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CIndexDlg dialog


CIndexDlg::CIndexDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CIndexDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CIndexDlg)
    m_dIe = 0.0;
    m_dIs = 0.0;
    m_dRe = 0.0;
    m_dRs = 0.0;
    m_dTe = 0.0;
    m_dTs = 0.0;
    //}}AFX_DATA_INIT
    //modeless data box
    if (Create(CIndexDlg::IDD,pParent))
    {       ShowWindow(SW_SHOW);
    }
    m_nIndex=0;
    m_bTime=FALSE;
}


void CIndexDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CIndexDlg)
    DDX_Text(pDX, IDC_IE, m_dIe);
    DDV_MinMaxDouble(pDX, m_dIe, 0., 1000000.);
    DDX_Text(pDX, IDC_IS, m_dIs);
    DDV_MinMaxDouble(pDX, m_dIs, 0., 100000.);
    DDX_Text(pDX, IDC_RE, m_dRe);
    DDV_MinMaxDouble(pDX, m_dRe, 0., 100000.);
    DDX_Text(pDX, IDC_RS, m_dRs);
    DDV_MinMaxDouble(pDX, m_dRs, 0., 100000.);
    DDX_Text(pDX, IDC_TE, m_dTe);
    DDV_MinMaxDouble(pDX, m_dTe, 0., 1000000.);
    DDX_Text(pDX, IDC_TS, m_dTs);
    DDV_MinMaxDouble(pDX, m_dTs, 0., 1000000.);
    //}}AFX_DATA_MAP

}


BEGIN_MESSAGE_MAP(CIndexDlg, CDialog)
    //{{AFX_MSG_MAP(CIndexDlg)
    ON_BN_CLICKED(IDC_VIEWTIME, OnViewtime)
    ON_BN_CLICKED(IDC_CHOOSE_IS, OnChooseIs)
    ON_BN_CLICKED(IDC_CHOOSE_RS, OnChooseRs)
    ON_BN_CLICKED(IDC_CHOOSE_TS, OnChooseTs)
    ON_BN_CLICKED(IDC_CHOOSE_IE, OnChooseIe)
    ON_BN_CLICKED(IDC_CHOOSE_RE, OnChooseRe)
    ON_BN_CLICKED(IDC_CHOOSE_TE, OnChooseTe)
    ON_WM_DESTROY()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CIndexDlg message handlers
```

```
/*  if((m_dIDT!=0))
    {
        if (m_bTime==TRUE)
        {
            if((m_dIDT!=0))
            {   m_dIe =int( m_dIe/m_dIDT);
                m_dIs =int(m_dIs/m_dIDT);
                m_dRe =int(m_dRe/m_dIDT);
                m_dRs =int(m_dRs/m_dIDT);
            }
            if (m_dTDT!=0)
            {   m_dTe =int(m_dTe/m_dTDT);
                m_dTs =int(m_dTs/m_dIDT);
            }

            m_bTime=FALSE;

        }else{
            m_dIe =(m_dIe*m_dIDT);
            m_dIs =(m_dIs*m_dIDT);
            m_dRe =(m_dRe*m_dIDT);
            m_dRs =(m_dRs*m_dIDT);
            m_dTe =(m_dTe*m_dTDT);
            m_dTs =(m_dTs*m_dIDT);

            m_bTime=TRUE;
        }
    }
    CDialog::UpdateData(FALSE);
*/
void CIndexDlg::OnOK()
{
    CCSHBDoc* pDoc=CCSHBDoc::GetDoc(); //get pointer to current document
    UpdateData();


    ValidateData();

    //CElvisDoc::UpdateAllViews(NULL);
    pDoc->incident.m_nbegin=int(m_dIs);
    pDoc->incident.m_nend=int(m_dIe);
    pDoc->reflected.m_nbegin=int(m_dRs);
    pDoc->reflected.m_nend=int(m_dRe);
    pDoc->transmitted.m_nbegin=int(m_dTs);
    pDoc->transmitted.m_nend=int(m_dTe);
    if ((m_dIe!=0)&&(m_dRe!=0))
    {pDoc->flag.m_bSeparatedIncident=TRUE;}
    if(m_dTe!=0)
    {pDoc->flag.m_bSeparatedTransmitted=TRUE;}
    if((pDoc->flag.m_bProcessedResults)||(pDoc->incidentbar.Velocity.m_bCalculated)||
        (pDoc->transmittedbar.Velocity.m_bCalculated))
    {pDoc->flag.m_bRecalculate=TRUE;}

    DestroyWindow();
    ((CCSHBView*)m_pParent)->AllDoneModeless();


//  CDialog::OnOK();
}

void CIndexDlg::OnViewtime()
{
    UpdateData();
    if((m_dIDT!=0)||(m_dTDT!=0))
    {
        if (m_bTime==TRUE)
        {
            if((m_dIDT!=0))
            {   m_dIe =int( m_dIe/m_dIDT);
                m_dIs =int(m_dIs/m_dIDT);
                m_dRe =int(m_dRe/m_dIDT);
                m_dRs =int(m_dRs/m_dIDT);
```

```
            }
            if (m_dTDT!=0)
            {   m_dTe =int(m_dTe/m_dTDT);
                m_dTs =int(m_dTs/m_dTDT);
            }

            m_bTime=FALSE;

        }else{
            m_dIe =(m_dIe*m_dIDT);
            m_dIs =(m_dIs*m_dIDT);
            m_dRe =(m_dRe*m_dIDT);
            m_dRs =(m_dRs*m_dIDT);
            m_dTe =(m_dTe*m_dTDT);
            m_dTs =(m_dTs*m_dTDT);

            m_bTime=TRUE;
        }
        GetDlgItem(IDC_VIEWTIME)->SetWindowText(m_bTime?"View Index":"View Time");
        GetDlgItem(IDC_START)->SetWindowText(m_bTime?"Start Time":"Start Index");
        GetDlgItem(IDC_END)->SetWindowText(m_bTime?"End Time":"End Index");
        CDialog::UpdateData(FALSE);

    }


}

void CIndexDlg::OnChooseIs()
{
    m_nIndex=1;
    CheckTime();
    ShowWindow(SW_HIDE);

}

void CIndexDlg::OnChooseRs()
{
    m_nIndex=2;
    CheckTime();
    ShowWindow(SW_HIDE);

}

void CIndexDlg::OnChooseTs()
{
    m_nIndex=3;
    CheckTime();
    ShowWindow(SW_HIDE);

}

void CIndexDlg::OnChooseIe()
{
    m_nIndex=4;
    CheckTime();
    ShowWindow(SW_HIDE);

}

void CIndexDlg::OnChooseRe()
{
    m_nIndex=5;
    CheckTime();
    ShowWindow(SW_HIDE);

}

void CIndexDlg::OnChooseTe()
{
    m_nIndex=6;
    CheckTime();//makes sure in index form
    ShowWindow(SW_HIDE);

}
```

```
void CIndexDlg::PostNcDestroy()
{
    delete this;
    CDialog::PostNcDestroy();
}

void CIndexDlg::OnDestroy()
{
    CDialog::OnDestroy();
    //redraws window to show new indices
    GetWindow(GW_OWNER)->RedrawWindow(NULL,NULL,RDW_ERASENOW+RDW_ALLCHILDREN+RDW_INTERNALPAINT+RDW
_INVALIDATE);
}

void CIndexDlg::OnCancel()
{
    DestroyWindow();
    ((CCSHBView*)m_pParent)->AllDoneModeless();


//  CDialog::OnCancel();
}

BOOL CIndexDlg::ValidateData()
{
    CheckTime();//Make sure
//checks to make sure time is less than total time and that the end index is not less than
// the original index.
    if(m_dIs>m_nData1Size)
    {m_dIs=m_nData1Size;}
    else if(m_dIs<0)
    {m_dIs=0;}

    if(m_dIe>m_nData1Size)
    {m_dIe=m_nData1Size;}
    else if(m_dIe<m_dIs)
    {m_dIe=m_dIs;}

    if(m_dRs>m_nData1Size)
    {m_dRs=m_nData1Size;}
    else if(m_dRs<0)
    {m_dRs=0;}


    if(m_dRe>m_nData1Size)
    {m_dRs=m_nData1Size;}
    else if(m_dRe<m_dRs)
    {m_dRe=m_dRs;}

    if(m_dTs>m_nData2Size)
    {m_dTs=m_nData2Size;}
    else if(m_dTs<0)
    {m_dTs=0;}


    if(m_dTe>m_nData2Size)
    {m_dTs=m_nData2Size;}
    else if(m_dRe<m_dRs)
    {m_dTe=m_dTs;}
    return TRUE;
}

BOOL CIndexDlg::CheckTime()
{   if (m_bTime==TRUE)
    {
        if((m_dIDT!=0))
        {   m_dIe =int( m_dIe/m_dIDT);
            m_dIs =int(m_dIs/m_dIDT);
            m_dRe =int(m_dRe/m_dIDT);
            m_dRs =int(m_dRs/m_dIDT);
        }
        if (m_dTDT!=0)
        {   m_dTe =int(m_dTe/m_dTDT);
            m_dTs =int(m_dTs/m_dTDT);
        }
        m_bTime=FALSE;
```

```
    }
    GetDlgItem(IDC_VIEWTIME)->SetWindowText(m_bTime?"View Index":"View Time");
    GetDlgItem(IDC_START)->SetWindowText(m_bTime?"Start Time":"Start Index");
    GetDlgItem(IDC_END)->SetWindowText(m_bTime?"End Time":"End Index");
    return TRUE;
}
```

```
// InputFileDlg.cpp : implementation file
//

#include "stdafx.h"
#include "cshb.h"
#include "InputFileDlg.h"
#include "math.h"
#include "fstream.h"
#include "AmplitudePropDlg.h"
#include <afxdisp.h>  //For COleDateTime

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CInputFileDlg dialog


CInputFileDlg::CInputFileDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CInputFileDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CInputFileDlg)
    m_nBinary = -1;
    m_szFileName = _T("");
    m_bAmplitudeShift = FALSE;
    //}}AFX_DATA_INIT
    m_nBinary=0;
    m_bAmplitudeShift=TRUE;
    m_nZeroPoints=20;
}


void CInputFileDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CInputFileDlg)
    DDX_Radio(pDX, IDC_BINARY, m_nBinary);
    DDX_Text(pDX, IDC_FILENAME, m_szFileName);
    DDX_Check(pDX, IDC_AMPSHIFT, m_bAmplitudeShift);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CInputFileDlg, CDialog)
    //{{AFX_MSG_MAP(CInputFileDlg)
    ON_BN_CLICKED(IDC_BROWSE, OnBrowse)
    ON_BN_CLICKED(IDC_ASCII, OnAscii)
    ON_BN_CLICKED(IDC_BINARY, OnBinary)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CInputFileDlg message handlers

void CInputFileDlg::OnBrowse()
{
    bool stopflag = false;
    OPENFILENAME OpenFileName;
    TCHAR szFile[256000] = "\0";    // File list

    char szFilter[]="WFT Files (*.wft)\0*.wft\0"
                            "AD Files (*.ad)\0*.ad\0"
                            "Text Files (*.txt)\0*.txt\0"
                            "Data Files (*.dat)\0*.dat\0"
                            "All Files (*.*)\0*.*\0\0";


    OpenFileName.lStructSize = sizeof(OPENFILENAME);
    OpenFileName.hwndOwner = NULL;
    OpenFileName.hInstance = NULL;
    if (m_nBinary==0){
    OpenFileName.lpstrFilter = //szFilter;
                            "WFT Files (*.wft)\0*.wft\0"
```

```
                            "AD Files (*.ad)\0*.ad\0"
                            "Text Files (*.txt)\0*.txt\0"
                            "Data Files (*.dat)\0*.dat\0"
                            "All Files (*.*)\0*.*\0\0";
    }else
    {OpenFileName.lpstrFilter = //szFilter;
                            "AD Files (*.ad)\0*.ad\0"
                            "WFT Files (*.wft)\0*.wft\0"
                            "Text Files (*.txt)\0*.txt\0"
                            "Data Files (*.dat)\0*.dat\0"
                            "All Files (*.*)\0*.*\0\0";
    }
    OpenFileName.lpstrCustomFilter = NULL;
    OpenFileName.nMaxCustFilter = 0;
    OpenFileName.nFilterIndex = 0;
    OpenFileName.lpstrFile = szFile;
    OpenFileName.nMaxFile = sizeof(szFile);
    OpenFileName.lpstrFileTitle = NULL;
    OpenFileName.nMaxFileTitle = 0;
    OpenFileName.lpstrInitialDir = NULL;//data1.m_szFilename;
    OpenFileName.lpstrTitle = "Select Data Set";
    OpenFileName.nFileOffset = 0;
    OpenFileName.nFileExtension = 0;
    OpenFileName.lpstrDefExt = "";
    OpenFileName.lpfnHook = NULL;
    OpenFileName.lpTemplateName = NULL;
    OpenFileName.Flags = OFN_EXPLORER | OFN_FILEMUSTEXIST;

    // Open dialog box
    GetOpenFileName(&OpenFileName);
    m_szFileName= szFile;

    CDialog::UpdateData(FALSE);
}

void CInputFileDlg::OnOK()
{
    CAmplitudePropDlg dlg;
    // TODO: Add extra validation here
    UpdateData();

    if (m_bAmplitudeShift)
    {
        dlg.m_nPoints=m_nZeroPoints;
        if (dlg.DoModal()==IDOK)
        {
            m_nZeroPoints=dlg.m_nPoints;
            GetData();
        }
    }else
    {
        m_nZeroPoints=0;
        GetData();
    }



//  CDialog::OnOK();
}

BOOL CInputFileDlg::GetData()
{
    int i=0;
    double dzero=0;

    if(m_nBinary!=0){
            char string[1000];
            char seps[]   = " ,\t";

            ifstream ifsin(m_szFileName,ios::in|ios::nocreate);

            if (!ifsin)
            {   MessageBox("Can't Open Input File","Error",MB_ICONERROR);
                return FALSE;
            }
```

```
            Data.m_dtime.RemoveAll();
            Data.m_dvolts.RemoveAll();
            ifsin.getline(string,1000,'\n');
            do{
                if ((string[0] != '"')&&(string[0]!='\0'))
                {
                    Data.m_dtime.Add((atof( strtok( string, seps ))));
                    Data.m_dvolts.Add((atof( strtok( NULL, seps ))));
                }
                ifsin.getline(string,1000,'\n');

            }   while(ifsin!=0);
            ifsin.close();
            Data.m_dDT=Data.m_dtime[1]-Data.m_dtime[0];

    }else
    {
        struct file_info{
            int year,month,day,time,hour,minute,second;
            int Header_size,File_size,data_count,Vertical_zero;
            double Vertical_norm,User_vertical_norm,User_vertical_zero;
            double User_horizontal_zero,User_horizontal_norm;
            CString Title,User_vertical_label,User_horizontal_label;
            CString User_notes,Audit,Nicolet_Digitizer_Type;
            int Bytes_per_data_point,Resolution,Process_flag,Data_compression;
            int Number_of_Segments,Length_of_Segments,Number_timebases,Length_Zone1;
            double Horz_norm_Zone1,Horz_zero_Zone1;
            int Time_Domain;
        }file;


        char string[1750];
        int i=0;
        char highbyte[1],lowbyte[1];
        CArray <double,double> xpoint;
        CArray <double,double> ypoint;
        CString strMessage;
        int number;
        double dzero=0;



        ifstream binary(m_szFileName,ios::nocreate | ios::binary);

        if (!binary)
        {
            strMessage="Could Not Open Input File";
            MessageBox(strMessage,"Error",MB_ICONERROR);
            return 0;
        }

        binary.getline(string,1750,'\n');

        file.Time_Domain=atoi(GetField(string,4,2));
        file.Header_size=atoi(GetField(string,8,12));
        file.File_size=atoi(GetField(string,20,12));

        file.year=atoi(GetField(string,125,3));
        file.month=atoi(GetField(string,128,3));
        file.day=atoi(GetField(string,131,3));
        file.time=atoi(GetField(string,134,12));//divide by 1e3 to get into seconds
        file.hour=int(file.time/3600e3);
        file.minute=int((file.time-file.hour*3600e3)/60e3);
        file.second=int((file.time-file.hour*3600e3-file.minute*60e3)/1e3);
        if(file.year<50)
        {file.year+=2000;}
        else{
            file.year+=1900;}
        //put year in correct format.  I above 50 assume 1950-1999 if below 2000-2049
        //I doubt that this program will be running in 2050 and if it is REWRITE it lazy bums
        Data.m_dtDate.SetDateTime(file.year,file.month,file.day,file.hour,file.minute,file.second)
;

        file.data_count=atoi(GetField(string,146,12));

        file.Vertical_zero=atoi(GetField(string,158,12));
```

```
        file.Vertical_norm=atof(GetField(string,170,24));
        file.User_vertical_zero=atof(GetField(string,194,24));
        file.User_vertical_norm=atof(GetField(string,218,24));
        file.User_vertical_label=GetField(string,242,11);

        file.Horz_zero_Zone1=atof(GetField(string,1060,24));
        file.Horz_norm_Zone1=atof(GetField(string,1036,24));
        file.User_horizontal_zero=atof(GetField(string,253,24));
        file.User_horizontal_norm=atof(GetField(string,277,24));
        file.User_horizontal_label=GetField(string,301,11);

        file.Title=GetField(string,44,81);

        file.User_notes=GetField(string,312,129);
        file.Audit=GetField(string,441,196);
        file.Nicolet_Digitizer_Type=GetField(string,637,21);

        file.Bytes_per_data_point=atoi(GetField(string,658,3));
        file.Resolution=atoi(GetField(string,661,3));

        file.Process_flag=atoi(GetField(string,826,3));
        file.Data_compression=atoi(GetField(string,829,3));
        file.Number_of_Segments=atoi(GetField(string,832,12));
        file.Length_of_Segments=atoi(GetField(string,844,12));
        file.Number_timebases=atoi(GetField(string,856,12));
        file.Length_Zone1=atoi(GetField(string,1024,12));

        if(file.Time_Domain!=1)
        {
            strMessage="Data Not in ASCII Text or Not in Time Domain";
            MessageBox(strMessage,"Error",MB_ICONERROR);

            return 0;
        }

        if(file.Data_compression!=0)
        {
            strMessage="Data Has Been Compressed";
            MessageBox(strMessage,"Warning",MB_ICONEXCLAMATION );
            return 0;
        }
        if(file.Number_of_Segments!=1)
        {
            strMessage="Will Only Convert Segment 1";
            MessageBox(strMessage,"Warning",MB_ICONEXCLAMATION );
        }
        if(file.Process_flag!=0)
        {
            strMessage="Warning:Data Has Been Processed on Oscilliscope";
            MessageBox(strMessage,"Warning",MB_ICONEXCLAMATION );
        }

        binary.seekg(file.Header_size);




        binary.read(lowbyte, 1);
        binary.read(highbyte, 1);
        i=0;
        do{
            number=0x00;

            number=(BYTE(highbyte[0])<<8)+BYTE(lowbyte[0]);
            if (number>=int(pow(2,15)))
            {
                number-=int(pow(2,16));
            }
//      m_dData.Add(((i*file.Horz_norm_Zone1)+file.Horz_zero_Zone1)*file.User_horizontal_norm+
//      file.User_horizontal_zero);
            Data.m_dvolts.Add(((number-file.Vertical_zero)*file.Vertical_norm)*file.User_vertical_
norm+
            file.User_vertical_zero);

            binary.read(lowbyte, 1);
            binary.read(highbyte, 1);
```

```
            i++;

        }while (i<file.Length_Zone1);

        Data.m_dDT=(file.Horz_norm_Zone1)*file.User_horizontal_norm;

        binary.close();

    }

    for (i=0;i<m_nZeroPoints;i++)
    {
        dzero+=Data.m_dvolts[i];
    }

    if(m_nZeroPoints!=0)
    {dzero/=m_nZeroPoints;}
    else{
        dzero=0;}

    for(i=0;i<Data.m_dvolts.GetSize();i++)
    {
        Data.m_dvolts[i]-=dzero;
    }
    CDialog::OnOK();
    return TRUE;
}



CString CInputFileDlg::GetField(char *string, int offset, int size)
{
    CString combine;
    int i=0;

    for (i=offset;i<(offset+size);i++)
    {
        combine+=string[i];
    }

    return combine;
}

void CInputFileDlg::OnAscii()
{
        m_nBinary=1;    //this is cheesy but I couldn't figure out a better way
}

void CInputFileDlg::OnBinary()
{
    m_nBinary=0;        //this is cheesy but I couldn't figure out a better way
}



void CInputFileDlg::OnCancel()
{
    // TODO: Add extra cleanup here

    CDialog::OnCancel();
}
```

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "CSHB.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    ON_WM_CREATE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP
        | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // fail to create
    }

    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
```

```
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CMDIFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CMDIFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
```

```
// PropagateWaveUtilityDlg.cpp : implementation file
//

#include "stdafx.h"
#include "cshb.h"
#include "PropagateWaveUtilityDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPropagateWaveUtilityDlg dialog


CPropagateWaveUtilityDlg::CPropagateWaveUtilityDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CPropagateWaveUtilityDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CPropagateWaveUtilityDlg)
    m_dTransmittedDistance = 0.0;
    m_dIncidentDistance = 0.0;
    m_nNyquist = -1;
    m_dFreqFilter = 0.0;
    m_dReflectedDistance = 0.0;
    //}}AFX_DATA_INIT
}


void CPropagateWaveUtilityDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPropagateWaveUtilityDlg)
    DDX_Text(pDX, IDC_TRANSMITTEDDIST, m_dTransmittedDistance);
    DDX_Text(pDX, IDC_INCIDENTDIST, m_dIncidentDistance);
    DDX_Radio(pDX, IDC_NYQUIST, m_nNyquist);
    DDX_Text(pDX, IDC_DISPFILTER2, m_dFreqFilter);
    DDX_Text(pDX, IDC_REFLECTEDDIST, m_dReflectedDistance);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CPropagateWaveUtilityDlg, CDialog)
    //{{AFX_MSG_MAP(CPropagateWaveUtilityDlg)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPropagateWaveUtilityDlg message handlers
```

```
// PropCoeffDlg.cpp : implementation file
//

#include "stdafx.h"
#include "cshb.h"
#include "PropCoeffDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPropCoeffDlg dialog


CPropCoeffDlg::CPropCoeffDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CPropCoeffDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CPropCoeffDlg)
    m_szCalibrationFile = _T("");
    m_szDataFile = _T("");
    m_bUpdateFile = FALSE;
    m_bExportFile = FALSE;
    m_bComplexModulus = FALSE;
    m_dFilter = 0.0;
    //}}AFX_DATA_INIT
}


void CPropCoeffDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPropCoeffDlg)
    DDX_Text(pDX, IDC_CALIB_I, m_szCalibrationFile);
    DDX_Text(pDX, IDC_DATA_I, m_szDataFile);
    DDX_Check(pDX, IDC_UPDATEFILE, m_bUpdateFile);
    DDX_Check(pDX, IDC_EXPFILE, m_bExportFile);
    DDX_Check(pDX, IDC_COMPLEXMOD, m_bComplexModulus);
    DDX_Text(pDX, IDC_FILTER, m_dFilter);
    DDV_MinMaxDouble(pDX, m_dFilter, 0., 100000000000000);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CPropCoeffDlg, CDialog)
    //{{AFX_MSG_MAP(CPropCoeffDlg)
    ON_BN_CLICKED(IDC_DETERMINECOEFF, OnDeterminecoeff)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPropCoeffDlg message handlers

void CPropCoeffDlg::OnDeterminecoeff()
{
    // TODO: Add your control notification handler code here

}

void CPropCoeffDlg::OnCancel()
{
    // TODO: Add extra cleanup here

    CDialog::OnCancel();
}
```

```
// SampleDataDlg.cpp : implementation file
//

#include "stdafx.h"
#include "CSHB.h"
#include "SampleDataDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSampleDataDlg dialog


CSampleDataDlg::CSampleDataDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CSampleDataDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CSampleDataDlg)
    m_dDf = 0.0;
    m_dDo = 0.0;
    m_dLf = 0.0;
    m_dLo = 0.0;
    m_szName = _T("");
    m_dtIDate = COleDateTime::GetCurrentTime();
    m_dtTDate = COleDateTime::GetCurrentTime();
    m_dPressure = 0.0;
    m_dVelocity = 0.0;
    m_dStrikerLength = 0.0;
    //}}AFX_DATA_INIT
}


void CSampleDataDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CSampleDataDlg)
    DDX_Control(pDX, IDC_DATE, m_dctrlDate);
    DDX_Text(pDX, IDC_Df, m_dDf);
    DDV_MinMaxDouble(pDX, m_dDf, 0., 100000.);
    DDX_Text(pDX, IDC_Do, m_dDo);
    DDV_MinMaxDouble(pDX, m_dDo, 0., 10000.);
    DDX_Text(pDX, IDC_Lf, m_dLf);
    DDV_MinMaxDouble(pDX, m_dLf, 0., 10000.);
    DDX_Text(pDX, IDC_Lo, m_dLo);
    DDV_MinMaxDouble(pDX, m_dLo, 0., 10000.);
    DDX_Text(pDX, IDC_SAMPLETITLE, m_szName);
    DDV_MaxChars(pDX, m_szName, 30);
    DDX_Text(pDX, IDC_INCDATE, m_dtIDate);
    DDX_Text(pDX, IDC_TRANSDATE, m_dtTDate);
    DDX_Text(pDX, IDC_PRESSURE, m_dPressure);
    DDV_MinMaxDouble(pDX, m_dPressure, 0., 1000000000.);
    DDX_Text(pDX, IDC_VELOCITY, m_dVelocity);
    DDV_MinMaxDouble(pDX, m_dVelocity, 0., 1000000000.);
    DDX_Text(pDX, IDC_STRIKERL, m_dStrikerLength);
    DDV_MinMaxDouble(pDX, m_dStrikerLength, 0., 1000000000.);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CSampleDataDlg, CDialog)
    //{{AFX_MSG_MAP(CSampleDataDlg)
    ON_BN_CLICKED(IDC_SAMPLE_DEFAULT, OnSampleDefault)
    ON_NOTIFY(DTN_DATETIMECHANGE, IDC_DATE, OnDateTimeChange)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSampleDataDlg message handlers

void CSampleDataDlg::OnSampleDefault()
{
    m_dDo=12.7;
    m_dLo=12.7;
```

```
    m_dDf=12.7;
    m_dLf=12.7;

    UpdateData(FALSE);

}

void CSampleDataDlg::OnOK()
{
/*  m_dDf=m_dDf/1000;
    m_dLf=m_dLf/1000;
    m_dDo=m_dDo/1000;
    m_dLo=m_dLo/1000;
*/
    m_dctrlDate.GetTime(m_dtDate);
    CDialog::OnOK();
}

void CSampleDataDlg::OnDateTimeChange(NMHDR* pNMHDR, LRESULT* pResult)
{
    // TODO: Add your control notification handler code here

    *pResult = 0;
}

BOOL CSampleDataDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

//  m_dtIDate.Format("%B %d, %Y %H:%M:%S");
//  m_dtTDate.Format("%B %d,%Y %H:%M:%S");
    m_dctrlDate.SetFormat("MMMM dd, yyyy HH':'mm':'ss");
    m_dctrlDate.SetTime(m_dtDate);
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CSampleDataDlg::OnCancel()
{
    // TODO: Add extra cleanup here

    CDialog::OnCancel();
}
```

```
// SeparateValuesDlg.cpp : implementation file
//

#include "stdafx.h"
#include "cshb.h"
#include "SeparateValuesDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSeparateValuesDlg dialog


CSeparateValuesDlg::CSeparateValuesDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CSeparateValuesDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CSeparateValuesDlg)
    m_dthreshold1 = 0.0;
    m_dthreshold2 = 0.0;
    m_dtolerance = 0.0;
    m_bSeparateTransmitted = FALSE;
    m_bSeparateIncident = FALSE;
    m_nIntercept = -1;
    //}}AFX_DATA_INIT
    m_nIntercept=0;
}


void CSeparateValuesDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CSeparateValuesDlg)
    DDX_Text(pDX, IDC_THRESHOLD1, m_dthreshold1);
    DDV_MinMaxDouble(pDX, m_dthreshold1, 0., 1.);
    DDX_Text(pDX, IDC_THRESHOLD2, m_dthreshold2);
    DDV_MinMaxDouble(pDX, m_dthreshold2, 0., 1.);
    DDX_Text(pDX, IDC_TOLERANCE, m_dtolerance);
    DDV_MinMaxDouble(pDX, m_dtolerance, 0., 1.);
    DDX_Check(pDX, IDC_SEPARATETRANSMITTED, m_bSeparateTransmitted);
    DDX_Check(pDX, IDC_SEPARATEINCIDENT, m_bSeparateIncident);
    DDX_Radio(pDX, IDC_INTERCEPT, m_nIntercept);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CSeparateValuesDlg, CDialog)
    //{{AFX_MSG_MAP(CSeparateValuesDlg)
    ON_BN_CLICKED(IDC_Default, OnDefault)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSeparateValuesDlg message handlers

void CSeparateValuesDlg::OnOK()
{
    // TODO: Add extra validation here
    CButton* ptemp=(CButton*)GetDlgItem(IDC_SEPARATEINCIDENT);


        m_bSeparateIncident=ptemp->GetCheck();

        ptemp=(CButton*)GetDlgItem(IDC_SEPARATETRANSMITTED);
        m_bSeparateTransmitted=ptemp->GetCheck();

    CDialog::OnOK();
}

void CSeparateValuesDlg::OnDefault()
{

    m_dthreshold1=0.2;
```

```
    m_dthreshold2=0.5;
    m_dtolerance=0.01;
    UpdateData();
}

void CSeparateValuesDlg::OnCancel()
{
    // TODO: Add extra cleanup here

    CDialog::OnCancel();
}
```

```
// stdafx.cpp : source file that includes just the standard includes
//  CSHB.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```