University of Alberta

Development of an Experimental Apparatus for Studying the Effects of Acoustic Excitation on Viscosity

by

Marc David Evans

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Engineering Management

Department of Mechanical Engineering

Fall 2012
Edmonton, Alberta

Dedication

To my family

Abstract

An experimental apparatus was developed capable of measuring changes in fluid viscosity occurring due to acoustic stimulation. Controls allowed measurements at simulated oil sand reservoir pressures and temperatures with near real-time data visualization. Calibration was performed using NIST-traceable viscosity standards. Parametric acoustic excitation experiments were performed on bitumen, bentonite slurries, and viscosity standards at 500psi static pressure, 20-80°C temperatures, ±100-400psi acoustic pressures, and 5-20Hz sinusoidal frequencies.

The viscosities of bitumen and NIST standards were unaffected by excitation at any of these amplitudes/frequencies. Bentonite showed viscosity reductions as large as 75% with a positive correlation observed between acoustic excitation amplitude and magnitude of reduction. Frequency variation had minimal to no effect on viscosity. Bentonite viscosities quickly approached minimum values after the start of stimulation but took hours to plateau. Once stimulation ceased, slurries recovered to their pre-stimulated viscosities. Viscometer damage that occurred during testing prevented collection of results for oil sand.

Acknowledgments

This work could not have been accomplished without the support of a great many people so I would like to take this opportunity to briefly mention those who have been such a terrific help along the way.

First off I would like to thank Imperial Oil and the Centre for Oil Sands Innovation for their generous funding of this project without which none of this work would be possible. In addition I would like to single out several people who have been especially helpful: Brian Speirs and Murray Gray for their technical support throughout the project, Keith Draganiuk for his technical assistance, Andrew Page for his assistance with rheometer testing, and Cam McGregor for his advice and commitment to my professional success.

I have received advice and support from too many individuals within the staff of the mechanical engineering department and the faculty of engineering to list all individual contributions so I will be brief. My warmest thanks go out to the staff of the machine shop (especially Bernie Faulkner) for their always helpful design advice, to the office staff (especially Teresa Gray and Gail Dowler) for their cheerful help with all my administrative needs, to Dr. Subir Bhattarjee for the use of his lab facilities, to the IT staff for always keeping me connected, to the faculty of engineering for their scholarship support, to MEGSA for keeping life on campus both fun and interesting, to FGSR Outreach (especially Renee Polziehn) for letting me give something back to the community, and to the many classroom professors (especially Peter Flynn) who have shared their knowledge and life lessons with me. I will always remember these experiences.

Outside of academic life I owe many thanks to my family and friends who have always been there for me. To my Edmonton family for giving me a home away from home, to my immediate family for all their support, to my newfound UofA friends for the many fun times, and to my roommate Steve for joining me in the great Alberta adventure, thank you. Extra special thanks go out to Lindsey for being the most patient and supportive girlfriend a guy could ask for. I could not have come this far without her.

At last I would like to thank my advisors on this project, Dr. Mike Lipsett and Dr. David Nobes. I can't even begin to describe how much I have learned from them these past years. Through thick and thin they have always been supportive of me on both a professional and a personal level and have played a major role in shaping me into the person I am today. I count myself truly lucky to have had such great role models and wish them the best of luck in the future.

To any whom I have forgotten here, thank you!

Table of Contents

List of Tables

Table of Figures

Nomenclature

τ_{yx}	Shear Stress
$\frac{du}{dy}, \dot{\gamma}$	Shear Rate
n	Flow Behaviour Index
k	Consistency Index
μ	Absolute (or Dynamic) Viscosity
η	Apparent Viscosity
η_0	Low-Shear Viscosity
η_∞	Infinite-Shear Viscosity
A, B	Arrhenius Constants
T	Absolute Temperature
K	Spring Constants
m	Lumped Parameter Masses
X, Y	Displacements
V	Volume
β_T	Isothermal Compressibility
p	Pressure
$D_{chamber}$	Test Chamber Inner Diameter
$L_{chamber}$	Test Chamber Axial Length
$L_{piston\ stroke}$	Piston Stroke Length
D_{piston}	Piston Diameter

Chapter 1 Introduction

The natural resource industry is an integral part of Canada's economy. Of the country's resources, perhaps the most discussed in recent years and arguably one of the most important for the country are the oil sands found in Northern Alberta. Vast reserves of a heavy oil known as bitumen sit largely untapped because of the unique challenges associated with recovery and processing of the resource. Increased oil prices paired with advances in technology made over the past five decades however, have turned unconventional oil production from oil sands into a very profitable industry and a major source of employment. A recent publication by the government of Alberta showed that one in six Albertans was directly or indirectly employed by the energy industry (Department of Energy, 2008). This change to profitability has attracted significant investment in recent years from many domestic and international oil companies seeking to establish new operations and expand existing operations in the oil sands.

In the past decade, large amounts of research funding have gone into the development of this industry. The two areas of research that have received the most funding are bitumen upgrading and resource recovery (Heidrick, Bilodeau, & Godin, 2004). The thesis work falls under the latter topic so this introduction is designed to give the reader an understanding of the industry and some of the challenges associated with resource recovery.

1.1 Oil Sand

Oil sand is a naturally occurring mixture of quartz sand (75-80%), silt, clay, water (3-5%), trace minerals, and bitumen (10-12%). Deeply buried oil sands contain high concentrations of dissolved hydrocarbon gases in the pore fluid as well (Agar, Morgenstern, & Scott, 1987). The target fraction, bitumen, is a heavy petroleum (API gravity of 8° to 14°) of residuals and asphaltenes, which once recovered, is refined into a range of lighter petroleum products in a process known as upgrading (Hirsch, 2005). The term API gravity refers to the American Petroleum Institute's measure of how heavy a petroleum substance is relative to water. Oil with a specific gravity of 1 will measure 10° on the API gravity scale, hence, the density of bitumen is comparable to that of water.

Alberta contains the largest concentration of oil sands in the world, the majority of which are found in three main deposits in the Northern half of the province. These three deposits, shown in Figure 1, cover an area of 140,000 km^2 and contain an estimated 1.7 trillion barrels of oil (Allen, 2008).

Figure 1 - Three main oil sand deposits in Alberta: Athabasca, Cold Lake, and Peace River. Image courtesy of (UBC Department of Forestry)

Of these reserves, a fraction are deemed to be "proven" reserves, meaning that they are known to be recoverable using existing technologies (Department of Energy, 2008). Estimates for proven reserves have been reported as low as 173 billion barrels (Department of Energy, 2008) and as high as 178 billion barrels (Hirsch, 2005). Similarly, other sources have put the quantity of unproven reserves as low as 1.5 trillion barrels (Heidrick, Bilodeau, & Godin, 2004) and as high as 2.5 trillion barrels (Hirsch, 2005). Approximately 1.5 trillion barrels of the oil sand are located underground in what are known as "in-situ" reserves and the remaining 200 billion barrels are located close to the surface at depths of less than 75 meters. Figure 2 compares Canada's proven reserves to the proven reserves in the world's other major oil producing countries.

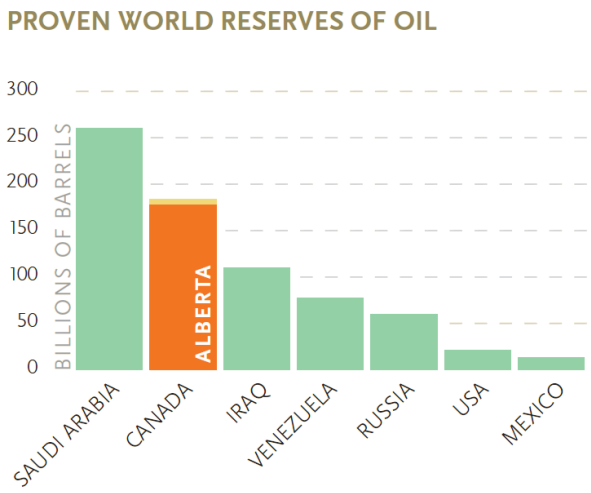


Figure 2 - Comparison of proven oil reserves by country. Image from (Department of Energy, 2008)

1.2 Production Technologies

The fact that proven reserves make up only 10% of the total available resource in Alberta emphasizes one of the major challenges in the industry, developing effective methods for bitumen production. Bitumen is rigidly attached to the other oil sand constituents so the challenge in production is finding a technology capable of isolating the bitumen.

Three main production strategies are currently in use for producing bitumen from oil sand. A brief introduction to each process is given below.

1.2.1 Mechanical Separation + Clark Hot Water Process

The most widely employed process at the moment involves open-pit truck and shovel mining of oil sand at shallow depths. The oil sand is transported to extraction plants which recover the bitumen using variants of the Clark hot water process (Clark, 1931). This production method accounts for 65% of current bitumen production and has received the most negative publicity in recent years due to the large quantities of tailings that are produced during extraction. Approximately $1m^3$ of oil sand is required to produce 1 barrel of oil using this method (Allen, 2008).

1.2.2 In-Situ Thermal Separation

In-situ thermal production methods are the next most widely used in Alberta. These processes act by injecting heat into underground reservoirs in order to reduce the viscosity of the bitumen fraction. The mobile bitumen is then pumped to the surface through either the same well that delivered the thermal stimulation or another nearby well. Thermal processes are highly subject to geological conditions in the reservoir (e.g. ground porosity, availability of cap rock, etc.) as they can have a large effect on the heating efficiency. Two proven processes of this type are:

i. Steam Assisted Gravity Drainage (SAGD)
   - Two parallel horizontal wells are drilled into a reservoir so that one is directly above the other. Steam is delivered through the top well, heating the reservoir and reducing the viscosity of the bitumen, which then flows down into the production well under the force of gravity.

ii. Cyclic Steam Stimulation (CSS)
   - A vertical well is drilled into a deposit and over several weeks, steam is pumped into the reservoir at high pressure and left to soak. The bitumen fraction undergoes a decrease in viscosity due to the heat, and radial cracks are formed as a result of the high pressure. The mobile bitumen is pumped back through the cracks out the initial well.

1.2.3 In-Situ Chemical Separation

Similar to the thermal processes, the chemical processes rely on an in-situ reduction of bitumen viscosity. The reduction results from the introduction of a chemical solvent through the well allowing the bitumen fraction to be pumped to the surface. As in the thermal processes, specific geological conditions are required for efficient delivery of the viscosity reduction stimulation (Hirsch, 2005). One process of this type is:

i. Vaporized Extraction (VAPEX)

- Two parallel horizontal wells are drilled into a reservoir so that one is directly above the other. A vaporized hydrocarbon solvent is injected into the reservoir through the upper well which reduces the viscosity of the surrounding bitumen and allows it to flow into the production well under the force of gravity.

The following table summarizes the three main strategies for bitumen recovery and provides some operational data for each. Summarized from (Hirsch, 2005), (Isaacs, 2005).

Table 1 - Overview of bitumen production strategies

Production Strategy	Target Reservoir Characteristics	Operating Cost ($/barrel)	% Bitumen Recovery	Extraction Mechanism
Mechanical Separation	Depth: less than 30-75 m Thickness: greater than 3 m Proven Volume: 9.4B m^3	$6-10	>90%	(After Mining) Clark Hot Water Process
Thermal Separation	Depth: greater than 75-80 m Thickness: greater than 10 m Proven Volume: 69B m^3	$8-14	SAGD: 50-60% CSS: 20-25%	Viscosity Reduction
Chemical Separation	Not Available	Not Available	Not Available	Viscosity Reduction

1.3 Inaccessible Oil Sand Reserves

A review published in 2004 by the Alberta Energy Research Institute classified three major reservoir types that required advances in production technology before they could be exploited. These reservoir types are listed in Table 2 below. Reproduced from (Heidrick, Bilodeau, & Godin, 2004).

Table 2 - Characteristics of inaccessible oil sand reservoirs

	Reservoir Type		
Reservoir Characteristic	**Thinner than Current SAGD and CSS**	**Shallower Depth than Current SAGD and CSS**	**Bitumen in Carbonates**
Depth	> 75-80 m	40 to 80 m	Bitumen in Carbonate Reservoirs
Thickness	10 to 1.5 m	> 10 m	
Saturation	> 8-10 % mass	> 6 % mass	
In Place Volume	~ 12 billion m^3	~ 4.4 billion m^3	71.1 billion m^3
Current Recovery	0 %	0%	0 %
Opportunities	New Exploitation Methods	New Exploitation Methods	New Exploitation Methods

The "Shallower Depth than Current SAGD and CSS" reservoirs are of interest in this thesis. Using the median reported value of proven deposits, this shallow oil sand, if exploited, would increase the quantity of proven reserves by approximately 16%.

1.4 Motivation

As evidenced from Table 1, production methods that rely on the reduction of bitumen viscosity have been the most widely adopted for in-situ recovery. With the long term aim of developing a new production technology for exploiting the shallow reservoirs mentioned in Table 2, this work investigates an alternative method for reducing the viscosity of bitumen.

Chapter 2 Relevant Theory and Review of the Literature

The following review is broken into three main sections. The first deals with the fundamentals of viscosity and rheology theory and methods for viscosity measurement. This is followed by a discussion of the effects of physical properties such as temperature, pressure, and acoustic excitation on the viscosity of fluids. The chapter ends with a review of industry experiments and anecdotal evidence pertaining to the stimulation of oil production reservoirs.

2.1 Viscosity and Rheology Theory and Terminology

This section introduces the theory behind some of the concepts of viscosity and rheology used in later discussion. The main sources for this information were (Mezger, 2006), (Bair, 2007), and (Fox, McDonald, & Pritchard, 2006).

2.1.1 Basic Viscosity Definitions

At a fundamental mathematical level, viscosity is defined as the relationship between the shear stress applied to a fluid and the resulting deformation of the fluid (expressed as shear rate). This definition encompasses two distinct classes of fluids, those that exhibit a proportional relationship between shear stress and shear rate, called **Newtonian fluids**, and those that do not, called **non-Newtonian fluids**. For one-dimensional flow, both categories of fluids may be described using the power law model below.

$$\tau_{yx} = k\left|\frac{du}{dy}\right|^{n-1}\frac{du}{dy}$$

Equation 1 - Power law relationship for fluid viscosity

Equation 1 is a modified form of Newton's law of viscosity. In this expression n is the flow behaviour index, k is the consistency index, τ_{yx} is the shear stress and $\frac{du}{dy}$ is the shear rate (commonly denoted $\dot{\gamma}$). For Newtonian fluids (where shear stress is proportional to shear rate), the flow behaviour index is equal to 1 and the expression is reduced to that shown below.

$$\tau_{yx} = k\frac{du}{dy}$$

Equation 2 - Power law relationship for Newtonian fluids

In this case, "k" is the coefficient of proportionality between the shear stress and shear rate. It is customary to then represent it by the symbol "μ", which is called the **absolute** (or **dynamic**) **viscosity** of the fluid.

Equation 1 is commonly restructured for non-Newtonian fluids. By replacing the first terms by the symbol "η", Equation 1 takes on a similar form to Equation 2.

$$\tau_{yx} = \eta\frac{du}{dy} \quad \text{where } \eta = k\left|\frac{du}{dy}\right|^{n-1}$$

Equation 3 - Power law relationship for non-Newtonian fluids

In Equation 3, known as the **Power Law** or **Ostwald de Waele Equation** (Krishnan & Aghijit, 2010), the coefficient "η" (called the **apparent viscosity)** relates the shear stress to the shear rate. The study of this shear rate-dependent viscosity and the time effects surrounding it is known as **rheology**.

The apparent viscosity of a non-Newtonian fluid may exhibit a positive or a negative correlation with shear rate and as such defines two sub-classes of non-Newtonian fluids. The first type, known as a **shear thinning** or **pseudoplastic** fluid, is one where the apparent viscosity decreases as shear rate increases (i.e. where n <1). These types of fluids flow more easily when sheared. The second

type, known as a **shear thickening** or **dilatant** fluid, is one where the apparent viscosity increases as shear rate increases (i.e. where n >1). These types of fluids show an increased resistance to flow when sheared. These behaviours are summarized in Figure 3 below.

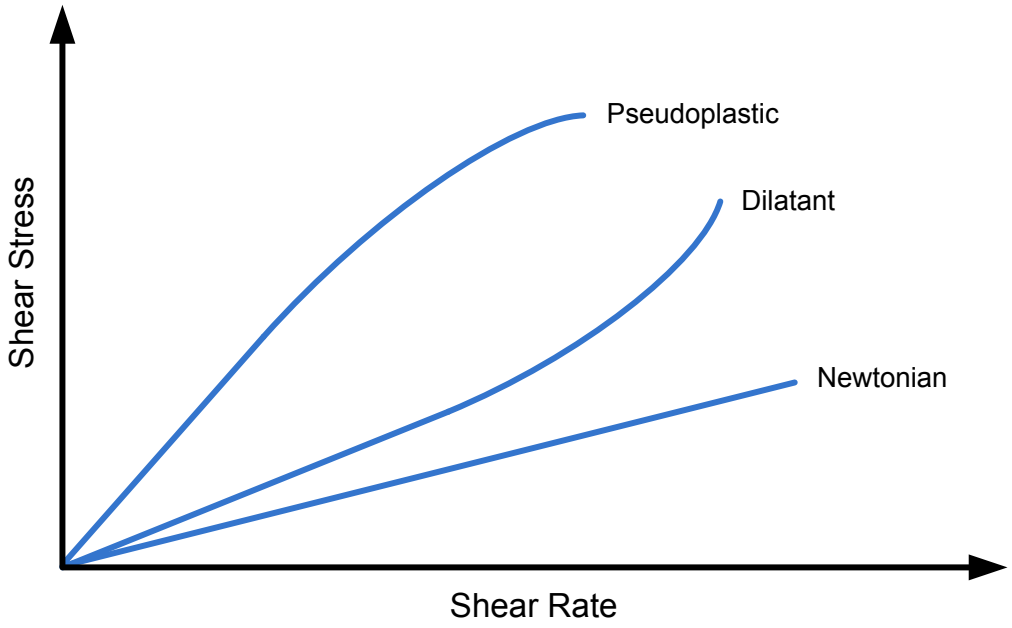


Figure 3 – Relationship between shear stress and shear rate for Newtonian, pseudoplastic, and dilatant fluids. Figure reproduced from (Fox, McDonald, & Pritchard, 2006)

2.1.2 Basic Rheology Definitions

Figure 4 provides a useful reference for describing a number of the rheology terms and behaviors observed in pseudoplastic fluids. This graph shows the results of a typical rheological experiment wherein a fluid at constant temperature and pressure is subjected to a gradually increasing shear rate whilst measuring its apparent viscosity.

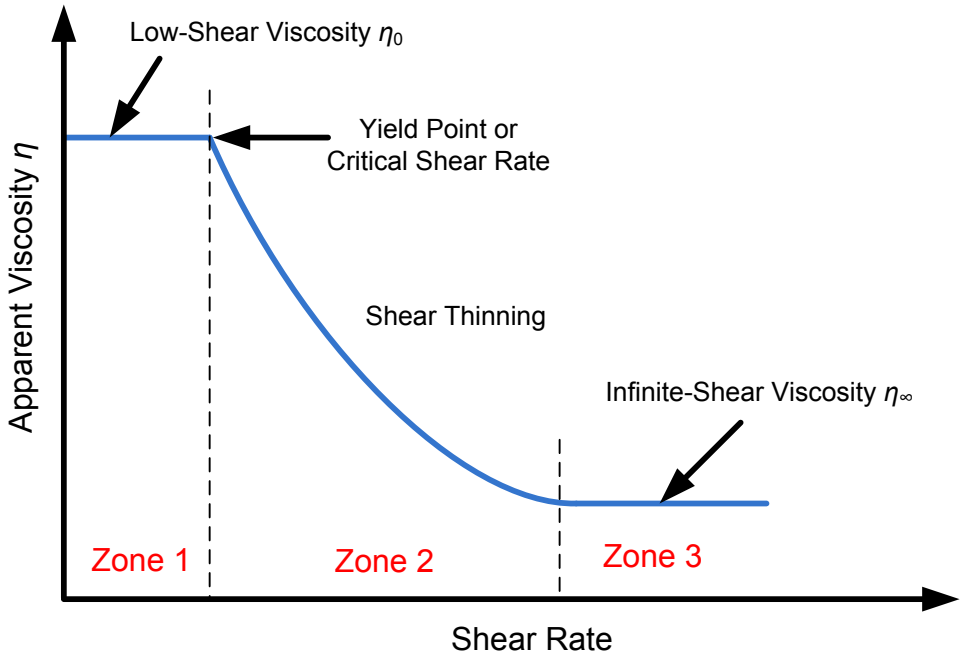


Figure 4 - Viscous behaviour of a yielding pseudoplastic fluid under constant temperature and pressure subjected to a varying shear rate

Three distinct zones are noted on the graph. The first, occurring at low shear rates, is known as the **low-shear viscosity** (denoted η_0). When subjected to sufficiently low shear rates (when the fluid is effectively undisturbed) many pseudoplastic fluids exhibit a constant viscosity value. As the shear rate is increased past a critical value (sometimes described as the **yield stress**), the fluid enters the second zone and begins to exhibit observable shear-thinning behavior. Finally, as shear rate is increased further, the viscosity plateaus at a minimum value known as the **infinite-shear viscosity** (denoted η_∞). In this third zone, shear is sufficiently high that any macro-scale particles in the fluid do not have enough time to return to a state where they significantly interfere with each other. Resistance to flow occurs solely from molecules rubbing against one another and intermolecular forces.

Many dispersions exhibit this characteristic type of behavior. In the first zone, the degree of interaction between particles in the fluid is sufficiently high as to provide a high resistance to flow. As shear rate is increased beyond the yield stress, particles begin to orient themselves in the shearing direction resulting in less particle interactions and less resistance to flow. In the third

zone, particles may be completely separated into bands of shear flow such that they undergo minimal interaction and thus exhibit minimum resistance to flow.

In addition to varying with shear rate, the apparent viscosity of a pseudoplastic fluid can vary with time. Figure 5 illustrates this behavior, which is known as **thixotropy**. When the shear stress is removed from a thixotropic fluid, the apparent viscosity does not immediately return to the low-shear value but rather undergoes a period of **thixotropic recovery**.

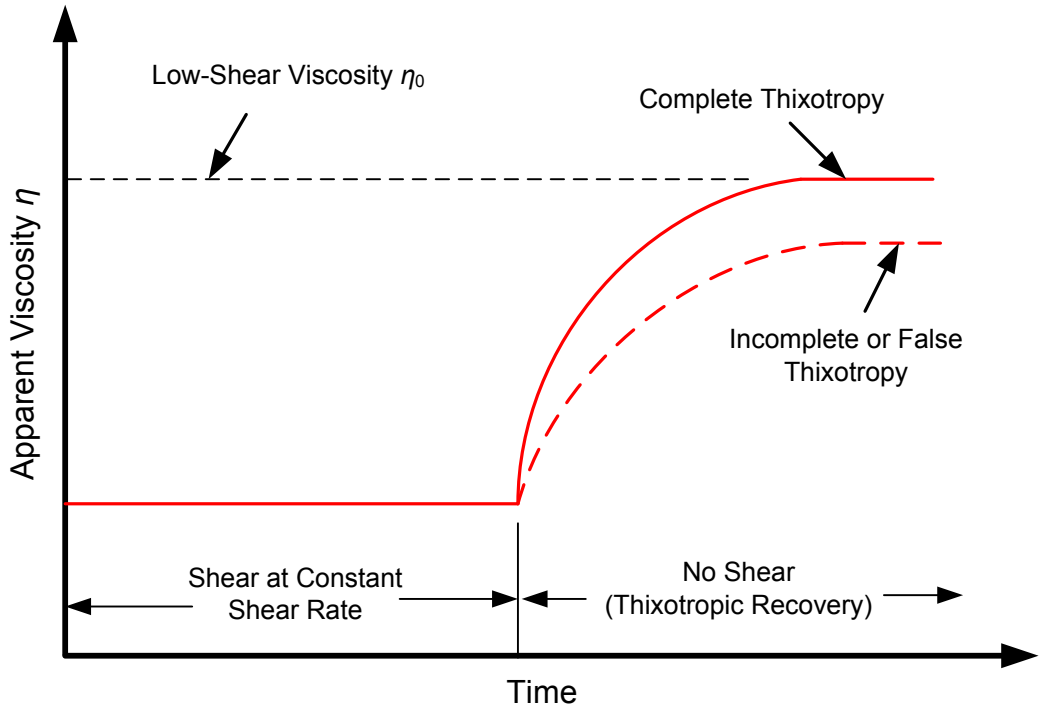


Figure 5 – Complete and incomplete thixotropic behaviour in a pseudoplastic fluid

A true thixotropic fluid will undergo a complete regeneration of its internal structure during thixotropic recovery and will thus return it to its low-shear viscosity over time. Several factors may disrupt this complete regeneration. Sufficiently high shear rates can cause particles in a pseudoplastic fluid to change form, thereby preventing the fluid from ever returning to its low-shear viscosity. This is known as an **incomplete** or **false thixotropy**. Further, high enough shear rates may cause fluids to undergo **molecular degradation** which can cause pseudoplastic fluids to exhibit Newtonian behavior.

Bentonite mixtures are known to display thixotropic behaviour and exhibit increased viscosity with increasing bentonite concentration (Grim & Necip, 1978). Mixtures in excess of 5% concentration by mass will exhibit thixotropy (Haydn, 2006).

Dilatant fluids do not exhibit the same zones of behavior described in Figure 4. The low-shear viscosity is followed by a zone of shear thickening behavior until the point where there is sufficient resistance to flow that the fluid begins to act as a solid. Sufficiently high shear rates will "tear" the fluid and as such the concept of infinite shear viscosity does not apply to dilatant fluids. Figure 6 illustrates this behavior.

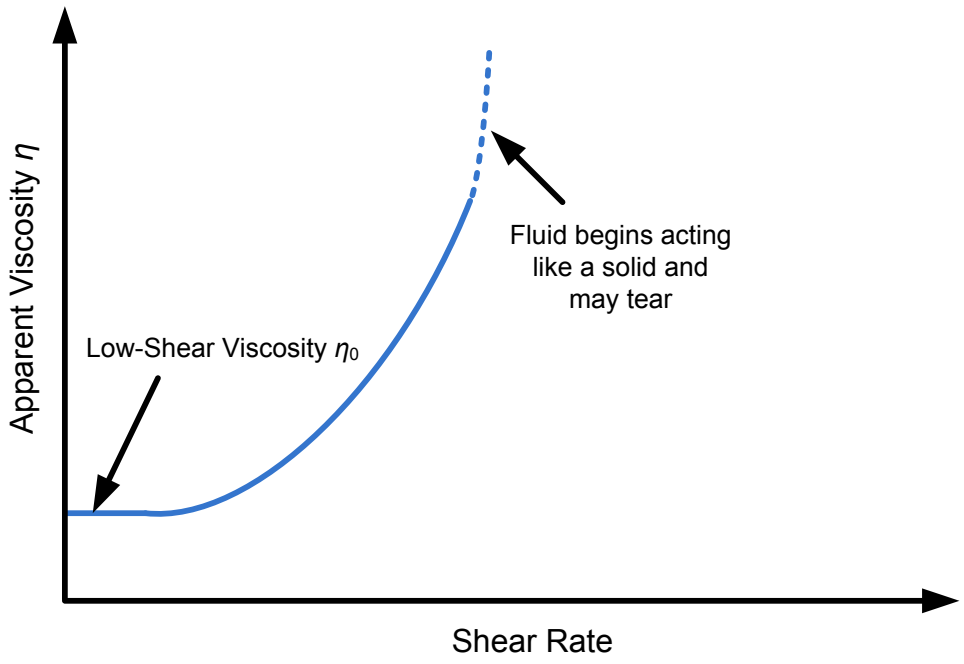


Figure 6 - Viscous behaviour of a dilatant fluid undergoing a significant increase in apparent viscosity

Dilatant fluids do however exhibit time-dependent viscosity behavior similar to thixotropy. This time-dependent behavior is known as **rheopexy** and is characterized by a gradual decrease in viscosity after a period of shearing. Similar to thixotropy, rheopexy is a completely reversible process unless a permanent degradation occurs in the fluid due to excessive shear. Figure 7

illustrates rheopexy in a dilatant fluid that experience a period of shear at a constant rate followed by a period of no shear.

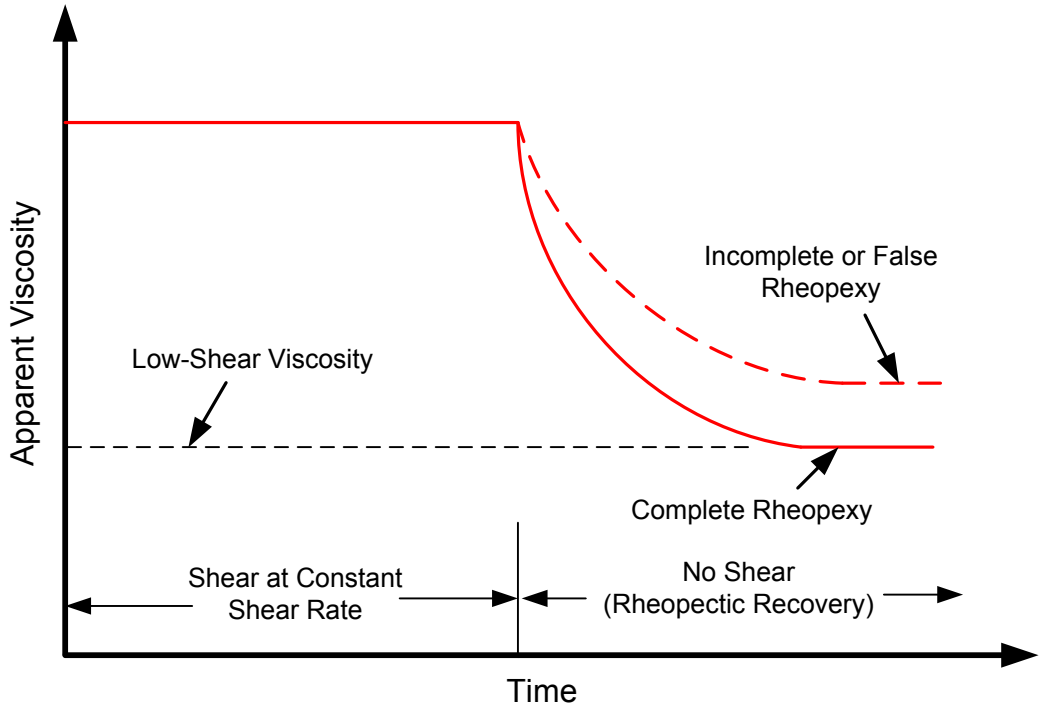


Figure 7 - Complete and incomplete rheopectic behaviour in a dilatant fluid

2.1.3 Viscosity Measurement

Viscosity is an indirect measurement, meaning that it is inferred from other directly measurable properties such as torque or speed (Mezger, 2006). Two types of measurement devices exist in this field:

1. **Rheometers**, which relate stress to fluid deformation and are thus capable of measuring apparent and absolute (shear rate dependent) viscosity, and
2. **Viscometers,** which operate at a single rate or speed and are thus only capable of measuring the absolute viscosity in Newtonian fluids or the Newtonian viscosity plateaus (infinite-shear or low-shear viscosity) of non-Newtonian fluids.

The more commonly used methods of obtaining viscosity measurements are:

- Rotational Couette Rheometers
- Capillary Viscometers
- Dropping Ball and Rolling Ball Viscometers
- Vibrating Wire Viscometers
- Oscillating Viscometers

2.1.3.1 *Rotational Couette Rheometers*

Rotational Couette rheometers infer viscosity by measuring the torque or rotational speed of a cylindrical spindle as it is rotated in a fluid sample. Figure 8 illustrates a typical setup for a conical rheometer. By applying a defined torque and measuring the rotational speed of the spindle or vice-versa, the viscosity (absolute or apparent) of the fluid can be calculated using Newton's law of viscosity. Since the surface areas in contact with the fluid are known, spindle torque can be converted to shear stress. Similarly, since the gap between the spindle and the fluid reservoir is known, the rotational speed of the spindle can be converted to shear rate as long as there is no slip between the fluid and viscometer surfaces. Couette rheometers are commonly used for measuring viscosity at low pressure and they are seldom used for high pressures measurements due to difficulties in sealing the rotating components (Bair, 2007).

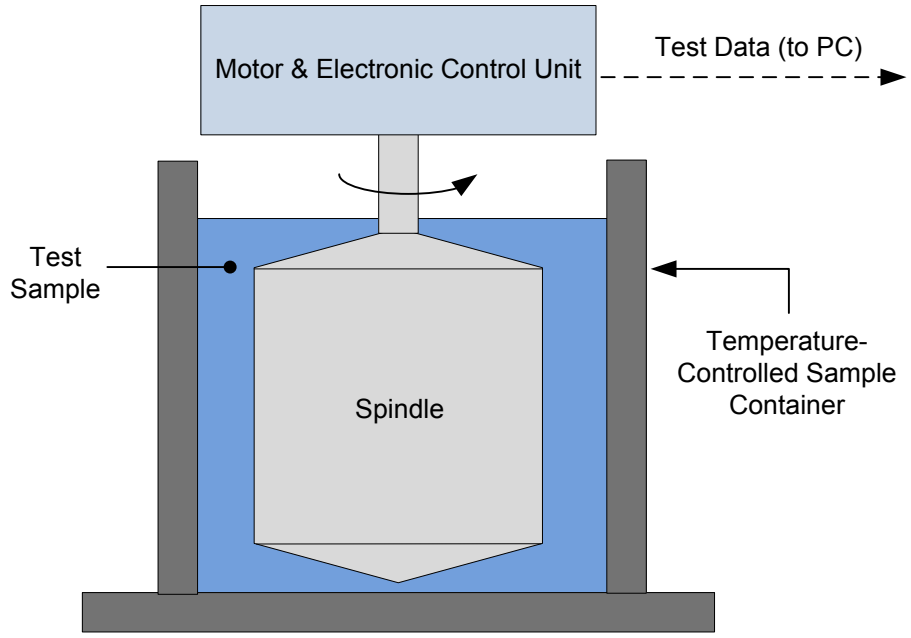


Figure 8 - Schematic of a coned-spindle Couette rheometer

2.1.3.2 *Capillary Viscometers*

Capillary viscometers infer the kinematic viscosity of fluids by measuring the time taken for a known volume of fluid to pass through a narrow section of tube known as the capillary. The sample is drawn up beyond the capillary section by vacuum then released and allowed to flow through the tube. Fluid resistance along the walls of the capillary slows the fluid by a factor related to the viscosity of the fluid. Measuring the time taken for the meniscus to cross between the start and stop line allows for a quick computation of viscosity. Typically made of glass, time measurements can be made by eye using either a stopwatch or more accurately with a video camera or other automatic means. Figure 9 shows the simplest form of capillary viscometer, known as an Ostwald viscometer.

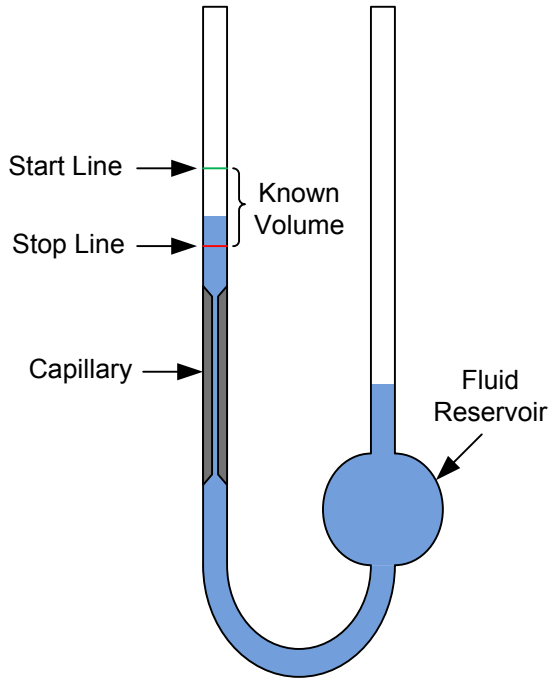


Figure 9 - Schematic of Ostwald capillary viscometer

2.1.3.3 *Dropping Ball and Rolling Ball Viscometers*

Dropping ball or rolling ball viscometers infer viscosity by measuring the time taken for a sphere to fall or roll from one level in a tight-fitting liquid filled tube to another. The time taken to travel the distance at a steady-state velocity is a function of the internal fluid resistance and therefore the viscosity of the fluid. The simple design allows dropping/rolling ball viscometers to be used for high-pressure viscosity measurements, upwards of 8 GPa (1.16×10^6 psi). Since it is important that the sphere be traveling at a steady state velocity as it passes between the different levels, dropping/rolling ball viscometers are often built of transparent materials to facilitate velocity observation. Since high- pressure instruments cannot be built with such materials, inductance coils are placed around the tube to provide a way of partially tracking the sphere's position during the traverse. The accuracy of such instruments is adversely affected by the inability to continuously measure velocity.

2.1.3.4 *Vibrating Wire Viscometers*

Vibrating wire viscometers function by vibrating a tensioned wire near its resonant frequency in a test fluid. A magnetic field is introduced and the frequency dependent voltage induced across the wire is measured. Simultaneous viscosity and density measurements are calculated using the Navier-Stokes solution to the vibrating wire problem. This type of viscometer was only introduced in 2004 and as such limited information is available on its full capabilities. (Bair, 2007) suggests that it is one of the most accurate methods of viscosity measurement but is limited in the range of viscosities that the device can measure.

2.1.3.5 *Vibrational Viscometers*

Vibrational viscometers operate on the principal that the motion of a rotationally oscillating bulb within a fluid undergoes damping as a result of viscous forces. Such viscometers may operate in several different manners to infer viscosity from the damping:

- Continuously vibrating the bulb and measuring the phase lag between the excitation and response signals
- Cutting off the excitation signal to the bulb and measuring the attenuation time (e.g. using the method of logarithmic decrement)
- Measuring the input power needed to keep the bulb vibrating at a given amplitude

Minimal oscillatory motion is required for these measurements so such viscometers are easily sealed for operation at pressure. In addition to this, since the above measurements can be taken in near real-time, vibrational viscometers are often used as process viscometers.

2.2 Physical Properties and their Effects on Viscosity

2.2.1 Viscosity vs. Temperature

Of the thermodynamic properties, temperature is known to have the greatest effect on viscosity. As temperature of a fluid is increased, its viscosity decreases exponentially such that small changes in temperature can result in significant changes in viscosity. This behavior is often described by the Arrhenius relationship of Equation 4 where "A" and "B" are experimentally determined constants of the liquid and "T" is the absolute temperature (Krishnan & Aghijit, 2010).

$$\eta = A^{-\frac{B}{T}}$$

Equation 4 - Arrhenius relationship between temperature and viscosity

High viscosity materials exhibit greater temperature dependence thus, as stated in Chapter 1, elevating temperature is a well-suited method for liquefying the high viscosity bitumen in current in-situ oil sand production technologies.

2.2.2 Viscosity vs. Pressure

In contrast with temperature, pressure has the opposite effect on the viscosity of liquids. As pressure is increased, the intermolecular spacing in the fluid decreases causing increased interaction and thus an increased fluid viscosity. This effect is most pronounced in high molecular weight fluids and in polymers with a high degree of branching (Mezger, 2006). Contrary to temperature, the relationship between viscosity and pressure is linear until high pressures (beyond the glass transition of the fluid, >1.2 GPa (Mezger, 2006) or >2GPa (Bair, 2007)) where it begins to exhibit solid-like behavior.

Figure 10 illustrates the temperature and pressure dependence of viscosity for a gas-free Athabasca bitumen. It should be noted that review of similar graphs prepared by other academic and industry sources, viscous behavior is similar in all cases but with slight variations depending on the location

of the bitumen reservoir from which the test samples were collected. The sample used in the figure below exhibit behavior typical of Peace River bitumen.

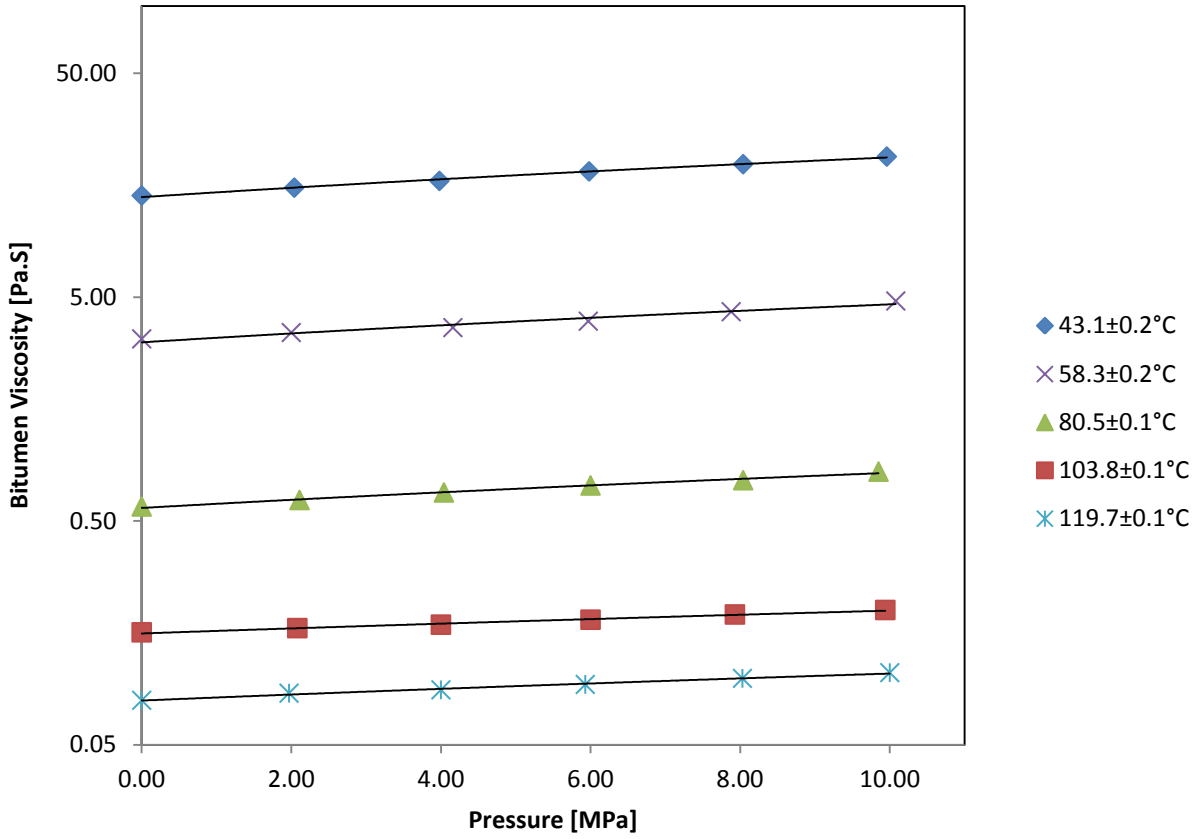


Figure 10 - Graph of viscosity versus pressure and temperature for a gas-free Athabasca (ARC) bitumen. Reproduced from (Mehrotra & Svrcek, 1986)

2.2.3 Viscosity vs. Particle Concentration

While temperature and pressure changes result in either an increase or a decrease in fluid viscosity, a change in the particle properties may cause fluids to exhibit entirely different viscous behavior. If for example, particle concentration is increased beyond a critical value in a Newtonian fluid, it will exhibit non-Newtonian behavior, either shear-thickening or shear-thinning depending on the concentration.

As particle concentration is increased in a pseudoplastic fluid, the onset of shear thinning behaviour will occur at a lower critical shear rate and at an increased rate (Chen, Chen, Wang, & Li, 2006). Still other fluids such as aqueous suspensions of cornstarch may exhibit pseudoplastic behaviour (at low shear rates) and dilatant behaviour (at high shear rates) depending on the particle concentration (Merkt, Robert, & Deegan, 2004). These viscous changes cannot be accurately represented with the power law so empirical models are typically used when dealing with such complex fluid behaviours. Other factors such as particle size, density, and shape are known to have an effect and further compound such modeling.

2.2.4 Viscosity vs. Acoustic Stimulation

Few experimental results are available on the effects of acoustic stimulation on fluid viscosity. Of those found in the literature, the experiments of (Ariadji, 2005) yielded some of the most significant findings. In the study, an oil (0.66-1.1 cP) was subjected to temperatures between 70 and 90°C and static pressures between 1 and 3kpsi whilst measuring viscosity before and during a period of vibration at frequencies between 5 and 40 Hz and unreported values of stimulation amplitude. Viscosity reductions as large as 30% were observed in the test samples with vibration amplitude having a greater effect on the viscosity change than frequency.

The result of the stimulation amplitude study just described, reproduced in Figure 11, indicate that increasing the acoustic stimulation amplitude tended to decrease the viscosity of the oil towards some asymptotic value. In addition, the result of their acoustic excitation frequency study, shown in Figure 12, indicated the presence of an optimal vibration frequency for achieving viscosity reduction. No mechanisms were mentioned or put forward for either result, however, it was postulated that the optimum frequency may have been a local minima and that retesting at a larger range of frequencies may have yielded a sinusoidal relationship. No explanation was offered for this conjecture.

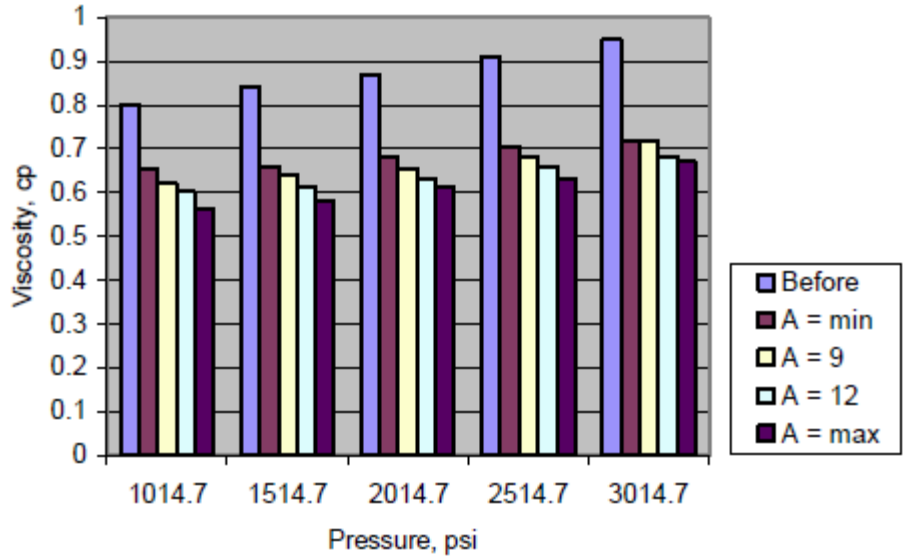


Figure 11 – Graph illustrating increased viscosity reductions with increased stimulation amplitude. Figure from (Ariadji, 2005)

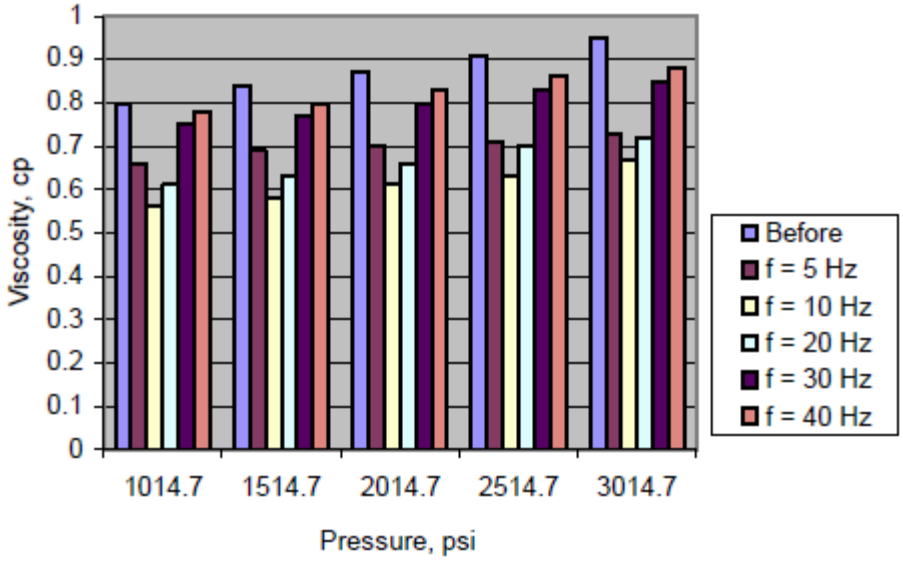


Figure 12 – Graph illustrating the optimum stimulation frequency for achieving the maximum viscosity reduction in an oil. Figure from (Ariadji, 2005)

2.3 The Stimulation of Oil Production Reservoirs

There exist a number of accounts in the literature where oil production in conventional reservoirs seemed to have been affected by vibration, either man-made or naturally-occurring. A selection of these anecdotal accounts is presented here.

2.3.1 Industry Observations and Experiments

It was noted in the extensive review by (Beresnev & Johnson, 1994) that conventional oil reservoirs in Russia saw increased oil production following earthquakes in the nearby vicinity. Though likely the result of large-scale ground movements, the effects were again observed when earthquakes occurred far from the reservoir site such that ground disturbances were of low amplitude. The review by (Huh, 2006) details some of the attempts designed to artificially stimulate other conventional oil reservoirs. The main technologies included: ground-level mechanical (vibroseis) (Kouznetsov, 1998) and electromagnetic (Simonov, 1996) vibration generators meant to deliver low frequency acoustic energy from the surface down into the reservoirs and pressure pulse generating equipment designed to deliver acoustic stresses to the reservoir from within the wellbore, both at sonic (Kuznetsov, 2002), (Dusseault, 1993), (Zhu, Xutao, & Vajjha, 2005), (Bogolyubov & al., 2001) and at ultrasonic frequencies (Duhon & Campbell, 1965). In most instances, production levels in surrounding wells were reported to have increased both during and for a period after stimulation although the physical mechanisms causing the improvements were never identified.

2.3.2 Physical Mechanisms behind the Industry Observations

Many suggestions were put forward to explain the observed production gains such as cavitation in pore fluids (sonocapillary) (Malykh, Petrov, & Sankin, 2003), reduction of capillary forces arising from the destruction of surface films, increased permeability (Roberts, 2005), and peristaltic transport by mechanical vibration (Hamida & Babadaglia, 2005) but few experiments were uncovered to support the ideas. In the review of proposed mechanisms by (Hamida & Babadaglia, 2005), citing the work by (Fairbanks & Chen, 1971), it was postulated that a reduction in fluid

viscosity due to the application of acoustic energy, particularly in the case of thixotropic fluids, could be the reason for the improvement.

Chapter 3 Experimental Methodology

3.1 Introduction

Fully understanding the effects of acoustic stimulation on oil sand viscosity in a reservoir is made complicated by the number of variables that can affect the physics. Short of doing in-situ field experiments in a reservoir, one way to approach such a problem is to reduce the complexity of the reservoir scenario such that it is suitable for study in the laboratory. This chapter outlines the assumptions made in defining the scope of the lab-scale problem, describes the equipment used to simulate reservoir conditions, and details the experiments that were performed.

3.2 Defining the Experimental Variables

3.2.1 Identifying the Most Useful Data for Industry

In industry, a tailored in-situ production strategy is developed for each reservoir that is to be recovered. This strategy is developed using an assortment of properties about the ore body and surrounding ground conditions as well as a fundamental physical understanding of what effect the production strategy will have on the reservoir (e.g. hot steam injection is known to reduce the viscosity of bitumen). Given the amount of site-specific data that is required to develop such a strategy, it was decided that this research would be most generally useful to industry if it was restricted in scope to establishing the fundamental physical understanding of a production strategy that employs acoustic stimulation.

3.2.2 The Model Reservoir

In order to experimentally obtain this fundamental physical understanding, some of the complexities of pressure wave propagation that occur in real reservoirs needed to be removed from

the research problem. By using a model reservoir devoid of these complexities, the resulting experimental data would be most generally useful in describing the basic physical response of the fluid. The first step in developing this general reservoir model was therefore to identify which complexities of a real reservoir could be eliminated in a lab-scale experiment.

A basic understanding of the mechanics of wave propagation was used to identify which reservoir complexities posed the biggest problem to the experimental investigation. Ore proximity to the acoustic stimulation site, soil properties, and reservoir geometry were identified as major factors affecting how a target ore body would be stimulated by an incident pressure wave. Attenuation and superposition of reflected pressure waves were also identified as sources of additional difficulty in understanding what stimulation actually occurs at the ore site. With these factors in mind, a number of assumptions about a model reservoir were developed. These assumptions are listed below and shown pictorially in Figure 13.

Assumption 1 The ore between the acoustic stimulation site and the target ore is homogenous in temperature, density, water content, and chemical composition. This eliminates the ground variations that would normally be encountered in field reservoirs.

Assumption 2 The model reservoir is assumed to be infinite. When combined with Assumption 1, this implies that no pressure waves are reflected and therefore that no superposition of pressure waves occurs in the target ore. This allows better control of how the target ore in the experiment is stimulated.

Assumption 3 The target ore is far enough from the stimulation site that an incident pressure wave is approximately planar. This allows further control of how the target ore in the experiment is stimulated.

Assumption 4 The pressures and temperatures in the model reservoir will be of a similar magnitude to those typically found in the intermediate depth reserves. This ensures that experiments performed in the model reservoir are still subject to the effects of pressure and temperature that are found in intermediate reserves.

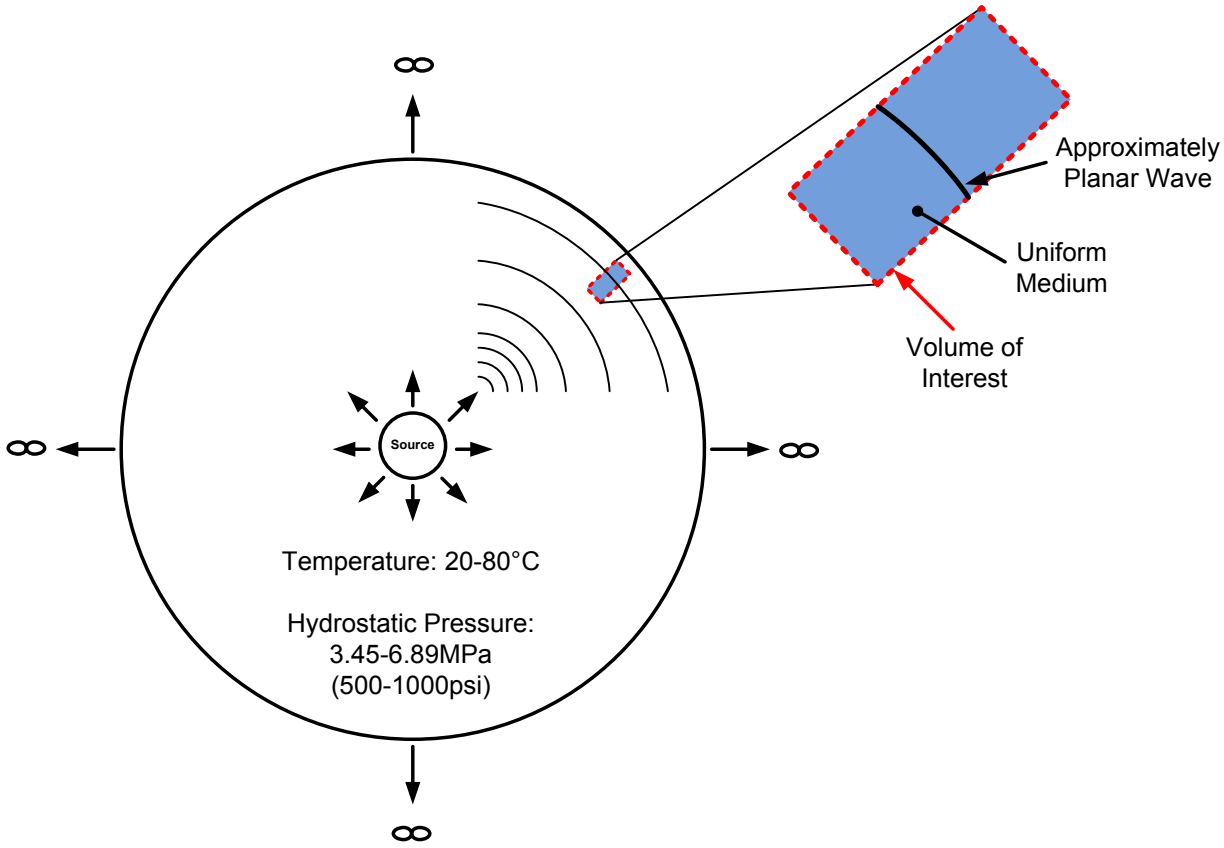


Figure 13 - Pictorial representation of an acoustic stimulation source in a model reservoir at depth.

3.2.3 Experimental Variables

In identifying the variables of interest to the study and determining which were to be controlled and which measured, four distinct groups were identified: those variables needed to quantify the acoustic stimulation, those variables known to affect the viscosity of fluids, those variables needed to validate the model reservoir assumptions, and those variables needed to quantify the viscous response of the fluid. These are detailed below, and where applicable the appropriate variable ranges are given along with a justification for the range.

3.2.3.1 *Group 1 - Variables Needed to Describe the Acoustic Stimulation*

In order to quantify the acoustic stimulation, the properties of the stimulation source needed to be well characterized. Additionally, since the acoustic wave was the primary independent variable in the research, the properties of this acoustic stimulation source needed to be both measurable and controllable. The variables needed to describe the acoustic stimulation source were those needed to characterize any wave, namely:

- Amplitude (Pressure)
- Frequency
- Wave Shape (e.g. Sinusoidal)

The controllable range for these variables was deemed unimportant at the onset of the research since there was limited quantitative evidence of an effect of either dynamic pressure amplitude or frequency on viscosity. The ranges (detailed in Chapter 4) were eventually selected based on the capabilities of the pressure generating equipment.

3.2.3.2 *Group 2 - Variables Known to Affect the Viscosity of Fluids*

From the review of the literature on viscosity theory, a number of different variables were identified which were known to have an effect on the viscosity of fluids. In order to isolate an observed effect on viscosity by any of the Group 1 variables, each of these Group 2 variables needed to be measured and controlled in the experiments. These variables were:

- Temperature
- Pressure
- Properties of Particles Within the Fluid (Size Distribution, Porosity, Shape, Concentration)
- Stimulation Duration (Thixotropic/Rheopectic Effects)
- Shear Rate

In accordance with model reservoir Assumption 4, the measurement ranges for temperature and pressure were selected to encompass the range likely to be encountered in an actual intermediate depth reservoir. For this reason, measurement and control in the range of 0-6.89 MPa (0-1000 psi) for pressure and 0-80°C for temperature were selected. The measurement and control ranges for the other variables in this group were again selected arbitrarily.

3.2.3.3 *Group 3 - Variables Needed to Validate the Model Reservoir Assumptions*

Conveniently, the first, second, and fourth reservoir assumptions: a uniform medium, no superposition of pressure waves, and realistic pressures and temperatures, could be validated by accurate measurement of the first three Group 2 variables: temperature, pressure, and particle properties. The third assumption, a planar incident pressure wave, could be validated by carefully designing the shape of the experimental apparatus test chamber, specifically the distance between the stimulation source and the viscosity measurement

Initially, the distance between the stimulation source and viscosity measurement was selected to help minimize the 3-D curvature of the incident pressure wave (as depicted in Figure 13). As the design of the measurement apparatus evolved, a method for generating planar pressure waves right at the stimulation source was developed and so this model reservoir assumption was deemed accounted for.

3.2.3.4 *Group 4 – Variables Needed to Quantify the Viscous Response of the Fluid*

The fourth group of variables were those needed to quantify the physical response of the fluid to the acoustic stimulation. Of principal interest here were variables used to describe the viscous response. These were:

- Low Shear (or Low Shear) Viscosity
- Shear Rate Dependent Viscosity

The measurement range for each of these variables was selected based on the fluids to be tested. Both low viscosity fluids and oil sand would be tested in the study, each one representing an extreme of the measurement spectrum. Due to the high cost of instruments capable of measuring such a broad range, the measurement range was decreased. In this way, a significant portion of the measurement range was covered and measurements on oil sand could still be performed at high temperatures where the viscosity was at its lowest. Details of the measurement range are available in Table 3.

3.2.3.5 *Test Fluids*

In addition to the aforementioned physical variables, an assortment of fluids was selected for testing. These were selected to help validate the test equipment and to broaden the data so that the results were not only applicable to oil sands but to fluid mechanics in general. These fluids, along with the experimental results sought from each, are listed below.

- N2500 – A NIST traceable viscosity standard for calibrating the viscosity measurement system
- Bitumen – An Athabasca bitumen for testing the effects of acoustic excitation on the target phase within the reservoir
- Oil Sand – A mid grade (~12% bitumen) Athabasca oil sand for testing the effects of acoustic excitation on the multi-phase reservoir ore
- Bentonite/Water – A drilling mud for testing the effects of acoustic excitation on a pseudoplastic fluid (13% mass concentration)
- Cornstarch/Water – A mixture for testing the effects of acoustic excitation on a dilatant fluid (55% mass concentration)

3.3 Experimental Equipment

3.3.1 Design Requirements

With the scope of the experimentation and the key variables identified, the design requirements for an experimental apparatus system concept were developed. At the highest level, each of the system concepts needed to simulate the conditions of the model reservoir while performing acoustic stimulation experiments at the lab scale. At the lower levels, the more specific design requirements were identified using the variable groups listed in the previous section.

Table 3 summarizes the list of variables and the corresponding lower level design requirement that each imposed on the experimental apparatus concepts. Each dependent variable necessitated specific sensor equipment while each independent variable called for either control equipment or consideration when designing the geometry of the test chamber. In addition to these variable specific design requirements, a number of supplementary requirements were identified to ease the operation of the device. These are outlined in Table 4.

Table 3 - Experimental apparatus design requirements identified using experimental variables

	Variable	Independent/ Dependent	Design Requirement	Measurement/Control Range (if applicable)	Justification for Range
Group 1	Amplitude	Independent	Acoustic Stimulation Amplitude Control	Arbitrary Range	-
	Frequency	Independent	Acoustic Stimulation Frequency Control	Arbitrary range	-
Group 2	Temperature	Independent	Feedback temperature control	0 – 80°C	Model Reservoir Assumption 4
	Pressure	Independent	Feedback pressure control	0 - 6.89 MPa (0 – 1000 psi)	Model Reservoir Assumption 4
	Particle Properties	Independent	Must be able to contain a variety of multi-phase fluids and resist abrasion	Range from no particles to coarse sand grains	Oil sand contains coarse sand grains
	Stimulation Duration	Independent	Acoustic stimulation source with precisely controlled duration	Arbitrary Range	-
	Shear Rate	Independent	Control of shear rate in the fluid	Arbitrary Range	-
Group 3	Distance between Stimulation Source and Viscosity Measurement	Independent	Minimize curvature of incident pressure wave	-	-
Group 4	Low Shear Viscosity	Dependent	Viscometer	0 – 10,000 cP	Viscosity range from water to oil sand
	Shear Rate Dependent Viscosity	Dependent	Rheometer	0 – 10,000 cP	Viscosity range from water to oil sand

Table 4 – Supplementary design requirements of the experimental apparatus concept

Design Requirement	Justification
Computer control of all control and data logging operations	A transient heat transfer analysis of the chamber geometry showed that thermal experiments were likely to last for several hours at a time so automation of the instruments freed the researcher from being present for the entire duration
Desired test chamber to be easily disassembled and/or rotated	Facilitated chamber cleaning and material handling operations

3.3.2 Test Chamber Concept

3.3.2.1 *Conceptual Design*

By taking into account all of the design requirements, a number of experimental apparatus system concepts were developed. Though several ideas were presented for satisfying each individual requirement, ultimately, all of the devised system concepts followed a similar test chamber design, illustrated in Figure 14.

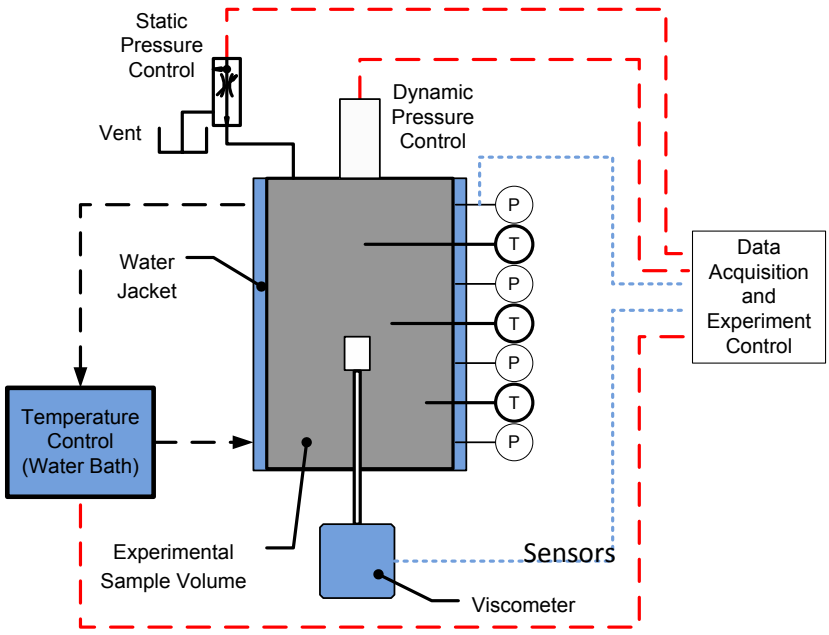


Figure 14 - Experimental apparatus chamber concept

The test chamber concept shown in Figure 14 involved a vertically oriented, flanged pressure vessel perforated by sensors and control equipment as called for by the various instrumentation requirements. This design was amenable to housing a variety of multi-phase test fluids and subjecting them to the various temperature and pressure loads specified in the model reservoir assumptions. Additionally, this design allowed for easy reconfiguration of sensor equipment owing to the flanged ends and multiple mounting locations along the main axis of the chamber.

With the overall concept established, the next step was the selection/design of components for each of the specific measurement and control requirements.

3.3.3 Instrument Selection and Integration

This section presents the engineering considerations for each of the major measurement, control, and sample containment requirements. For each design requirement, a list of the different options considered is followed by a detailed description of the final component selected in the initial build. Where applicable, details about the implementation and installation into the test chamber are given.

3.3.3.1 *Viscosity Measurement*

One of the most critical design decisions was determining the method for measuring the viscosity of the test sample during acoustic excitation. The review of viscosity measurement techniques (Bair, 2007) (briefly summarized in the last chapter) was useful in this matter as it dealt with some of the difficulties of obtaining accurate measurements in devices requiring high-pressure seals. In addition to high-pressure sealing considerations, the chosen device needed to be able to withstand typical reservoir temperatures whilst measuring the wide range of shear rate dependent viscosities encountered with water and oil sand.

Pressure sealing was of primary importance so a vibrational viscometer was chosen because it could be easily mated to a flange and installed on the pressure vessel. Additionally, vibrational models were readily available which could be interfaced with a PC and could measure the extremely wide

viscosity ranges required for the study. A Hydramotion VJ1-100 viscometer was selected as the best fit for the requirements. It was capable of viscosity measurements in the range of 1 to 10,000 cP, at temperatures from -20 to 130°C. As shown in Figure 15 below, sealing was accomplished by means of an o-ring above the device threads. PC monitoring of the measurement was established over an RS-485 connection using the MODBUS communication protocol.

The choice to use a vibrational viscometer was a compromise, as it meant that the device was only able to measure the infinite-shear viscosity. The measurement of shear rate dependent viscosities was therefore only possible at atmospheric pressure using a rate-controlled Brookfield Couette rheometer available in the Oil Sands and Coal Interfacial Engineering Facility at the University of Alberta.

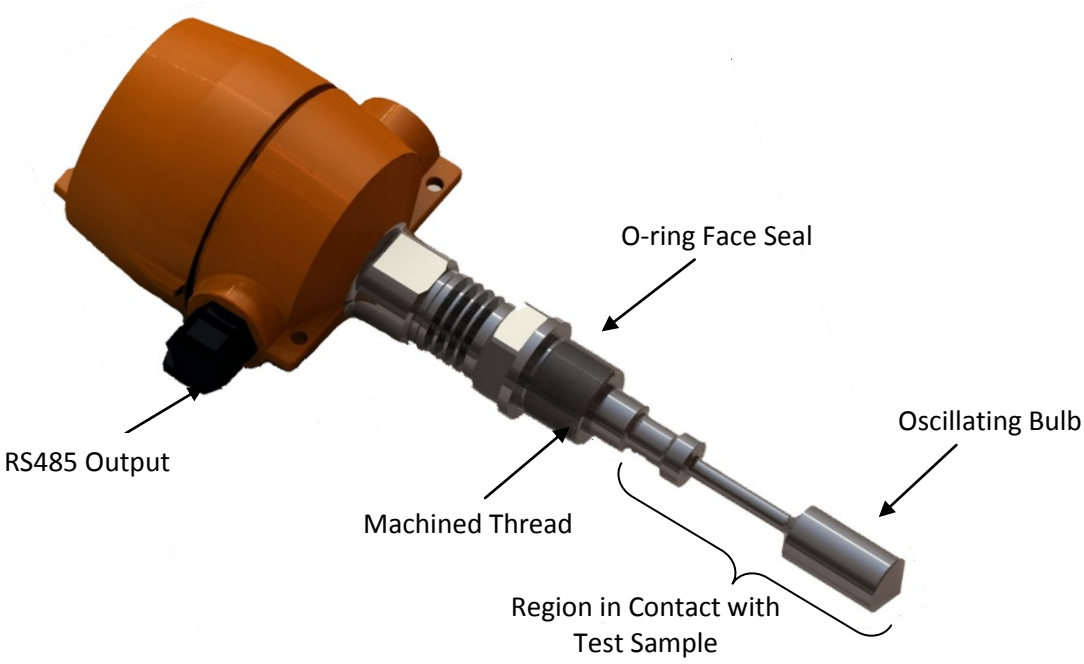


Figure 15 - CAD Rendering of Hydramotion VJ1-100 Viscometer

3.3.3.2 *Temperature Control*

A variety of temperature control strategies were assessed including electrical resistance heating outside the test chamber and a variety of plumbing designs for conductive and convective heating

using a bath and circulating fluid. A combination heating/cooling temperature bath and welded water jacket design was selected because this allowed for a self-contained temperature control system and could easily accommodate the geometric constraints imposed by the sensors positioned around the jacket of the test chamber.

In keeping with the automation design requirement, a bath was selected which could be interfaced with a PC for both monitoring and control purposes. The Cole-Parmer 12101-41 digital heating/cooling bath (see Figure 16) provided this functionality via RS-232 serial connection.

Figure 16 - Cole-Parmer 12101-41 digital heating/cooling circulating bath with onboard digital controller. Note: Fluid inlet and outlet ports as well as serial connection are located at the rear of the device Image Source: www.coleparmer.ca

A 50/50 mixture of ethylene glycol and water was used within the range of -40°C to 105°C and controlled by temperature feedback electronics and heating/cooling elements in the circulating bath. This fluid was pumped from the bath through the fluid jacket around the test sample, transferring and removing heat from the test sample by means of conduction through the chamber walls.

A few modifications were made to the thermal system during the work. The polyethylene tubing initially used for all fluid connections on the water jacket was later replaced by Swagelok tubing when leaking became an issue. Adhesive-backed 25.4mm (1in) foam insulation was added to the outside of the water jacket to improve the heating and cooling performance at the extremes of the

temperature range; and an external RTD was later added to allow the temperature feedback signal to come from the site of viscosity measurement within the fluid.

3.3.3.3 *Data Acquisition System*

A data acquisition system was required that could interface the control software with the pressure and temperature transducers and allow data logging on an attached PC. A National Instruments CompactDAQ system was selected for its ease of integration with the National Instruments LabWindows/CVI programming environment. Specifically, the NI cDAQ-9172 chassis housed eight signal conditioning modules which could be easily configured to allow a variety of wired sensors to be connected to the PC via USB through the chassis. This hardware was also able to make use of the wiring and sensor calibration utilities built into the National Instruments programming environment as well as the DAQmx programming functions, all of which greatly facilitated system commissioning. The chassis is shown below in Figure 17 beside several signal conditioning modules.

Figure 17 - NI cDAQ-9172 chassis with an assortment of signal conditioning modules
Image Source: www.zone.ni.com (Accessed December 2009)

3.3.3.4 *Temperature Measurement*

There were four design considerations when selecting temperature sensors: (1) sensors had to be able to measure temperature within the range of 0 to 80°C, (2) had to be able to withstand

sustained contact with abrasive fluids such as oil sand, (3) had to be installed in a high pressure environment, and (4) had to be interfaced with the NI DAQ system. Off-the-shelf resistance temperature detectors were capable of achieving (1) and (2) and could be mounted in a high pressure environment (3) when installed using Swagelok fittings so they were an ideal choice. The sensors could be interfaced with the NI DAQ chassis through the use of NI 9217 analog RTD input modules.

The sensors selected were 100 Ω Platinum 3-wire RTDs housed in 316 stainless steel sheaths from Aircom Industries (RT4-GP-B-S-3-36-4-T-2-MC-X-LT). The photograph in Figure 18 shows one of these RTDs mounted in an NPT Swagelok fitting.

Figure 18 - 3-wire RTD mounted in an NPT Swagelok fitting.

3.3.3.5 *Pressure Measurement*

When selecting pressure transducers, a number of requirements needed to be considered. Sensors had to be capable of measuring the simulated downhole pressures outlined in Table 3, had to be mountable in the test chamber, had to be able to withstand contact with abrasive test materials such as oil sand, and had to be interfaced with the NI DAQ system.

A series of flush-mounted pressure transducers made by American Sensor Technologies (model: AST4700-A-02000-P-5-R-0-0233) satisfied all of these requirements. The transducers outputted a voltage signal from 0-10 V proportional to pressures between 0 and 13.9MPa (0 and 2000psi) and could be screwed directly into 1/2" NPT holes in the test chamber. Interfacing with the NI Compact

DAQ chassis was accomplished using the 32 channel NI 9205 analog input module (±10V, 16-bit, 250 kS/s) shown in Figure 19 right.

Figure 19 - Left: American Sensor Technologies flush mounted pressure transducer. Right: NI 9205 analog input modules. Image Source: www.zone.ni.com

3.3.3.6 *Static Pressure Control*

As stated in the model reservoir requirements, a method for controlling static pressure in the test chamber was required which could both apply and vent pressures within the range 0-6.89MPa (0-1000psi). The components used for this task evolved several times throughout the project in order to remain compatible with the changing needs of the acoustic stimulation control equipment. Early design iterations employed cylinders of compressed inert gas and computer controlled pressure regulators made by Bronkhorst (model: EL-PRESS P612CV-100A-AAD) and later General Electric (model: GE Pace 5000) to accomplish this. Aside from using the GE Pace 5000 for pressure calibration, the final design discarded these components in favour of the hydraulically actuated piston described in section 3.3.3.7 below, which was capable of controlling both acoustic and static pressure simultaneously.

3.3.3.7 *Acoustic Stimulation Control*

Several methods for supplying and controlling the dynamic pressure amplitude and frequency were evaluated during the apparatus conceptualization. The various designs fell into two categories: those which generated pressure using compressed gas and those which generated pressure by

physically compressing the test fluids. The former were discarded owing to the fact that they would require a very large supply of compressed gas since each pressure pulse would have involved the chamber being pressurized and then vented. Designs utilizing physical compression of the fluid on the other hand only required some form of oscillating piston to compress the test fluid by a precisely controlled amount thus there were minimal material requirements after the initial manufacturing cost.

Using physical compression of the fluid, pulse amplitude was controlled by piston displacement (change in volume) and stimulation frequency by piston frequency. What remained was to precisely control these two motions. Two designs were considered: (1) a slider crank attaching a piston to a speed-controlled motor, and (2) a voltage-controlled linear actuator attached directly to the piston. The linear actuator combination was selected since it allowed for easier control of piston stroke (no need to change a linkage) and it did not require complex dynamic balancing as all components were inline.

After the initial calculations of force and stroke requirements for the acoustic stimulation experiments (provided in Appendix D), a linear piezoelectric actuator and signal generator combination were selected. When passed through an amplifier, the output waveform from a signal generator (Tektronix model AFG 3021B) controlled the position of the actuator, allowing precise control of the acoustic excitation amplitude and frequency. The analysis done when sizing the actuator assumed that the test chamber and test fluid were gas-free during experimentation; however, in practice, removing residual gas proved more difficult to achieve than anticipated. This meant that the test fluids were more compressible than predicted in the analysis. The result was that the actuators sized for the task were unable to provide sufficient stroke to generate the pressures specified in the design requirements.

A new system was then developed to compensate for the compressibility effects of the residual gas. The new system used a larger diameter piston mounted in a hydraulic tensile testing machine which allowed for increased stroke and piston force. The stroke and frequency of this new piston system were again controlled by the Tektronics signal generator which was in turn controlled by an attached PC via USB. Figure 20 below shows a schematic of the volume displaced by the piston as it travels within the chamber. In addition, the new system had sufficient stroke capability to generate

both the static and acoustic components of pressure. A schematic representation of this relationship between piston position and chamber pressure is shown in Figure 21 for clarity. Further information about this relationship between piston stroke and chamber pressure can be found in section 4.4.3, which deals with the commissioning of the acoustic excitation system.

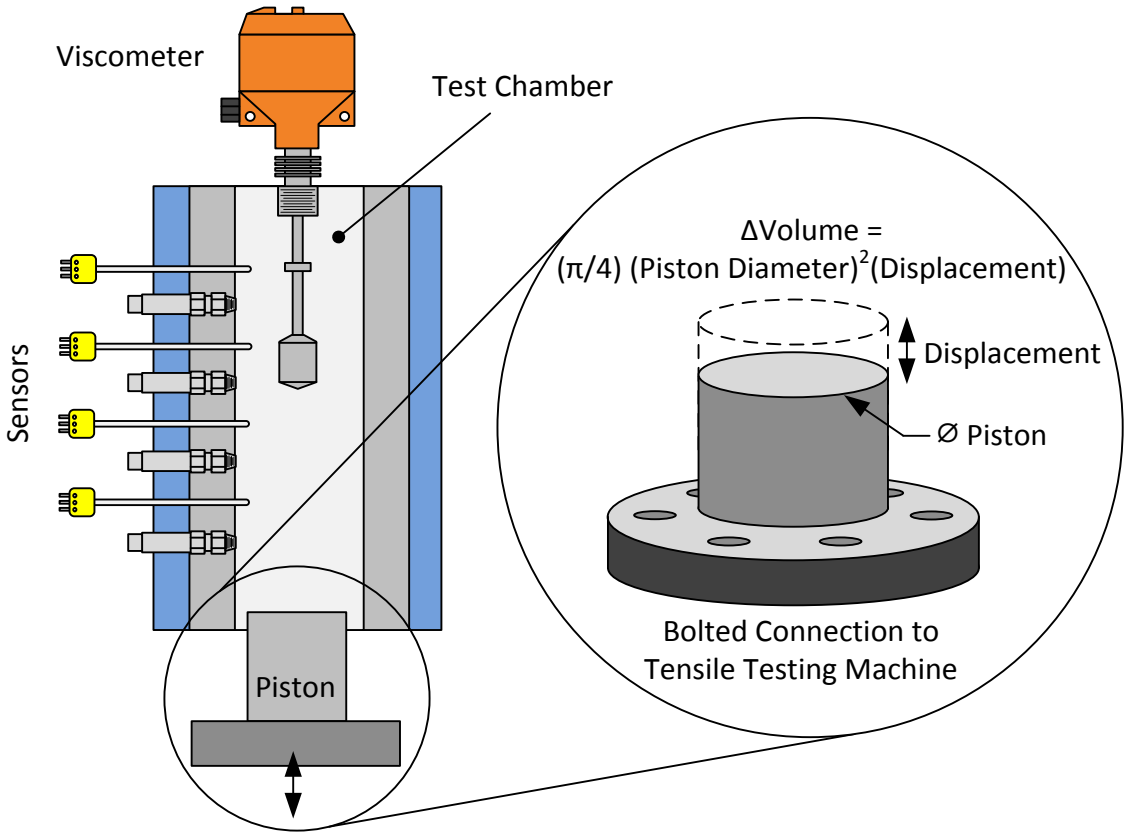


Figure 20 - Hydraulic piston installation at base of test chamber. (O-ring seals not shown)

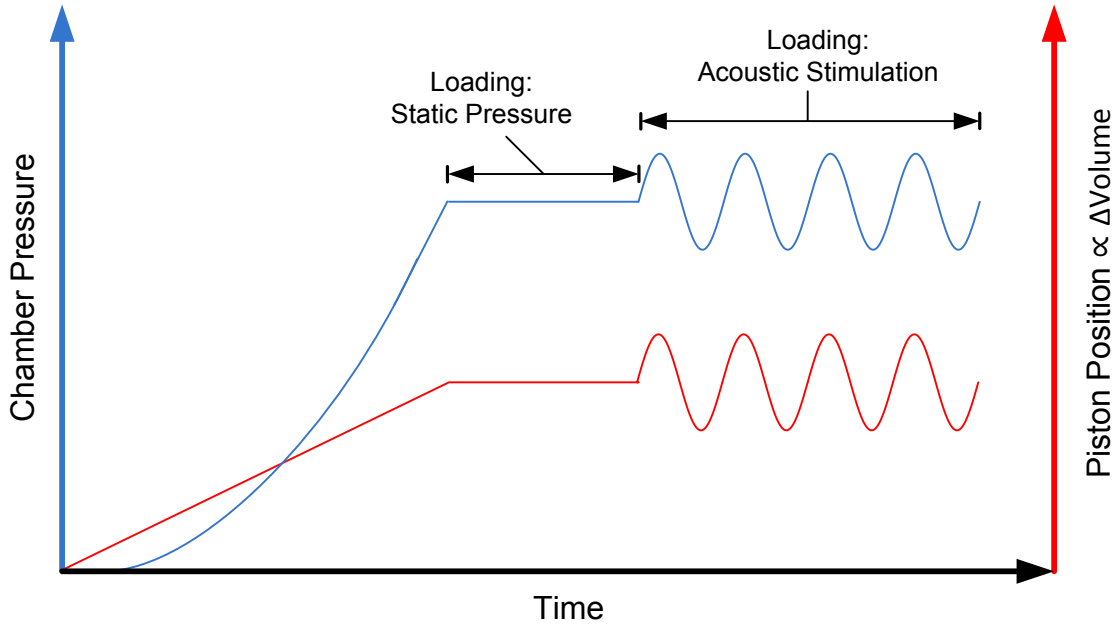


Figure 21 – Plot showing the typical pressure response of a test fluid to compression by the tensile testing machine piston.

3.3.3.8 *Sample Containment*

The specific geometry of the test chamber design was based largely on the mounting requirements for the sensor and control equipment, as well as the design requirements arising from the model reservoir assumptions. These numerous design considerations significantly narrowed down the feasible chamber geometries such that in the end only one design was really considered. The following CAD rendering of the chamber shows the overall shape and identifies some of the major design features.

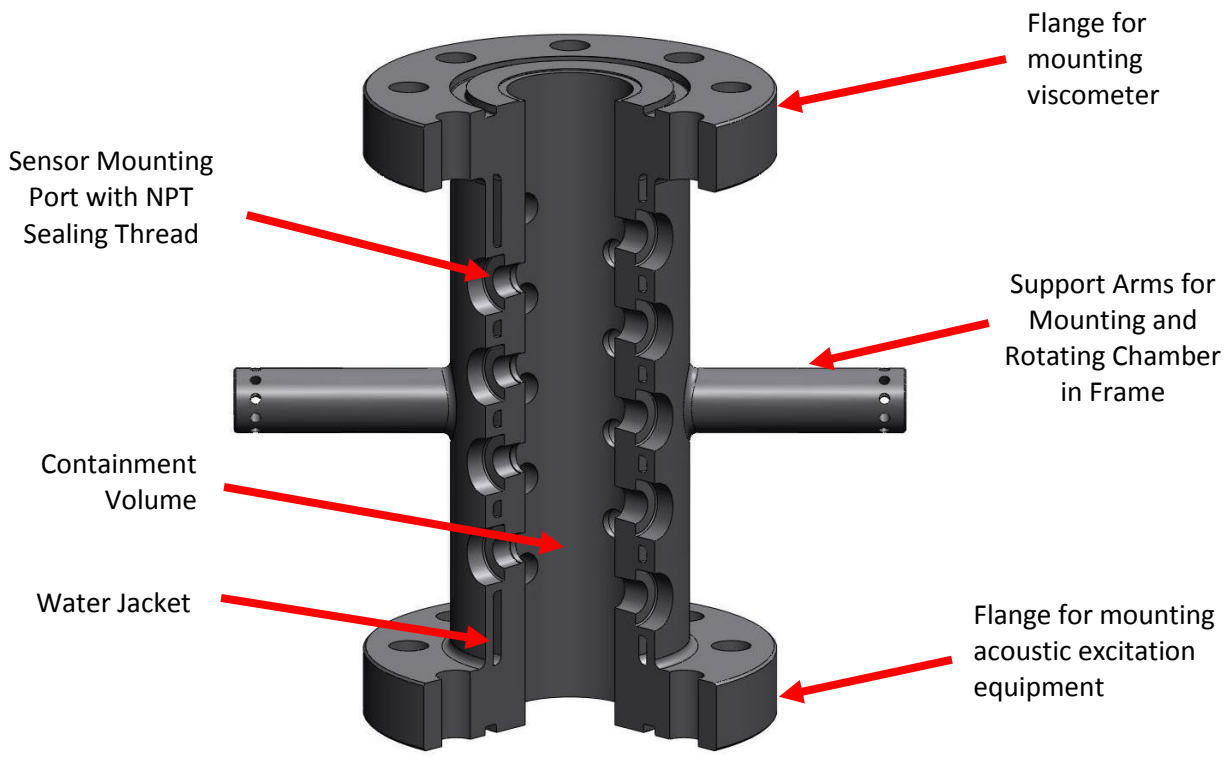


Figure 22 - Sectioned CAD rendering of experimental apparatus chamber

Along with containing the sample in the central cylindrical chamber, this design allowed for:

- Pressure containment up to 17.2 MPa (2500psi)
- The viscometer to be flange mounted and easily removed from one end of the chamber
- The acoustic stimulation source to be flange mounted opposite the viscometer and the two separated by a large enough distance to allow for acoustic pressure waves to be approximately planar
- Modular arrangement of the temperature and pressure sensors in both the radial and axial directions for monitoring temperature and pressure profiles
- An insulated water jacket for circulating the heating and cooling fluids around the sample
- Installation of: a pressure relief valve, vent for priming the chamber, and valve for draining the chamber
- Easy rotation of the chamber in a supporting frame to facilitate cleaning

3.3.3.9 *Chamber Assembly*

Figure 23 shows a sectioned CAD rendering of the final experimental apparatus chamber with all sensor equipment installed. For the sake of highlighting the mechanical system integration, sensor electrical connections have been left out. For details about the electrical system integration, refer to section 3.3.5. Detailed Drawings of all components, subassemblies, and the completed experimental apparatus assembly are available in Appendix F.

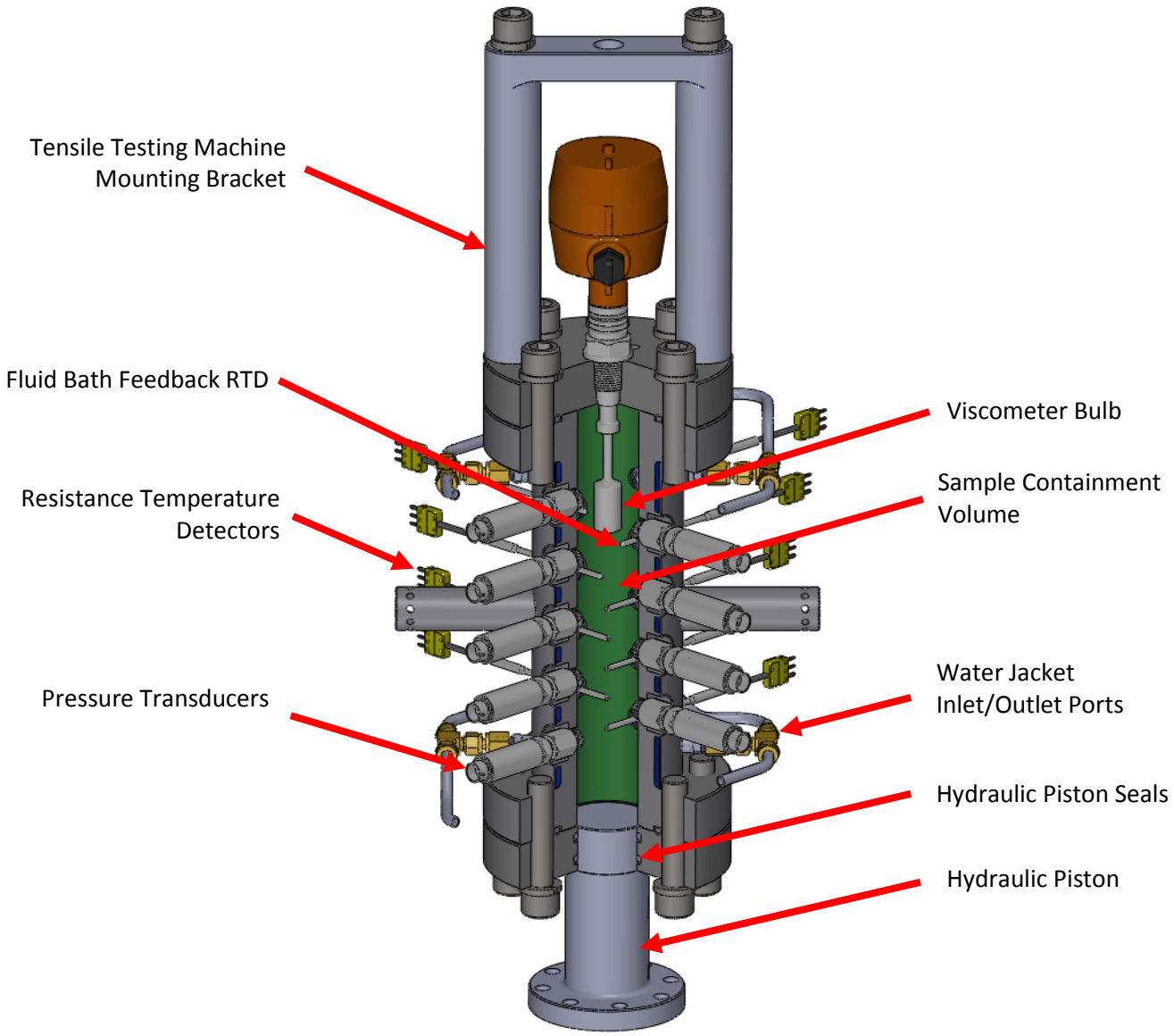


Figure 23 - Sectioned CAD rendering of the final test chamber shown with all sensor equipment installed. Note: Support frame, hydraulic connections, insulation, and electrical connections are not depicted.

3.3.4 Mechanical Design and Analyses

Several important engineering analyses were performed during the design of the experimental apparatus. These included the structural design of the pressure vessel, thermal modeling of chamber's heating performance, a modal analysis of the chamber structure, and analytical development of the constitutive relationship between piston displacement and pressure within the chamber. These analyses are respectively presented in Appendix A, Appendix B, Appendix C, and Appendix D.

3.3.5 System Integration and Control

The following section presents details of how the sensor and control equipment was incorporated into the overall measurement system. Emphasis is placed on the electrical system integration though some minor aspects of the mechanical system integration are presented as well.

3.3.5.1 *System Process and Instrumentation (PI) Diagram*

As specified in the design requirements for the experimental apparatus, it was necessary for the entire system to be controlled by a PC and for all data to be logged electronically. The following process and instrumentation diagram gives an overview of the instrumentation architecture and data flow that enabled this. The process and instrumentation diagram presented in Figure 24 shows the communication pathways between all measurement instruments and where applicable the communication protocol used. Control relationships between instruments are also marked, illustrating how the PC has control of each element in the system.

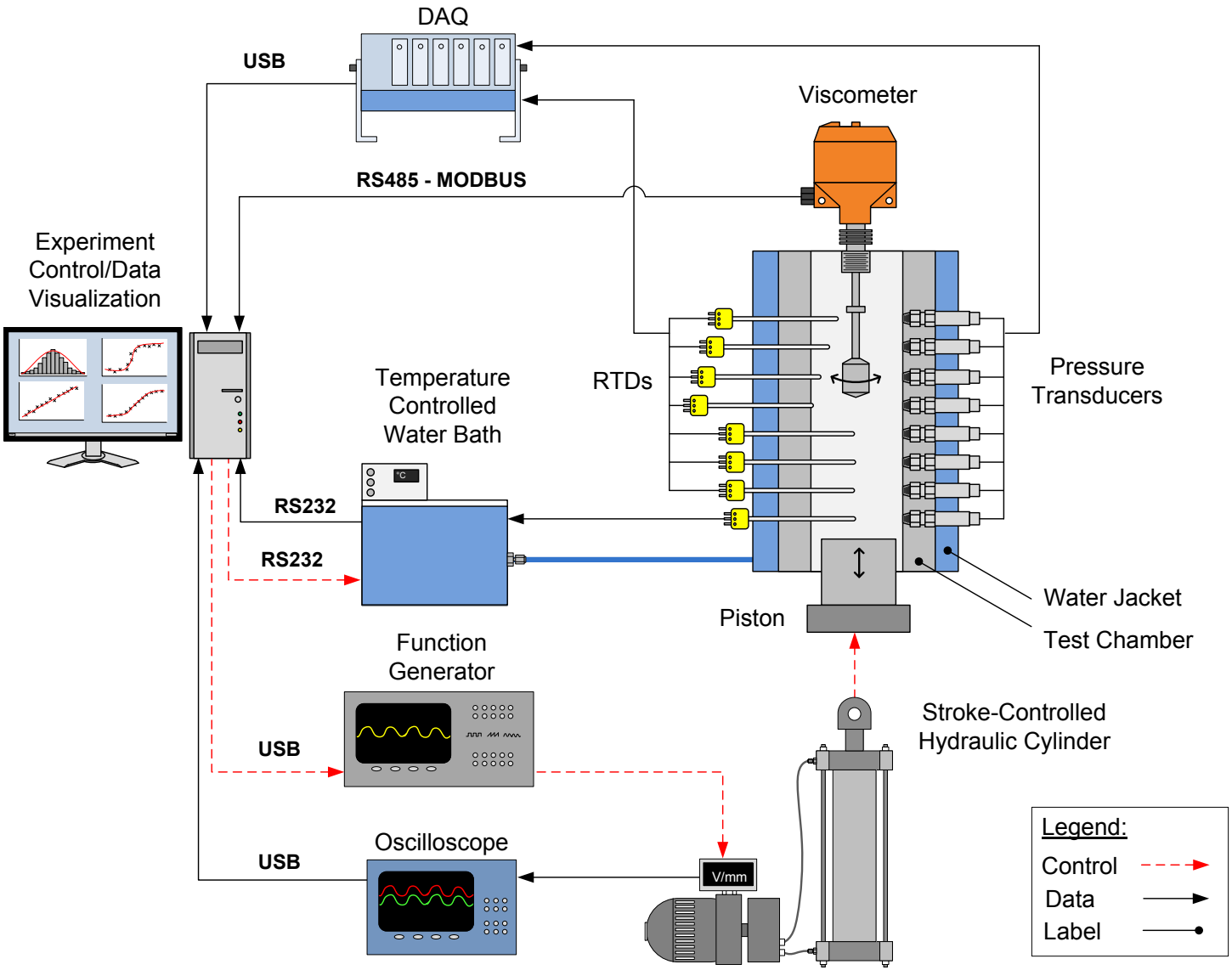


Figure 24 - Process and instrumentation diagram of the experimental equipment setup

3.3.5.2 *Electronics Enclosure*

In addition to simply wiring up the equipment as shown in the process and instrumentation diagram, further circuitry was required to ensure that all sensors received their specified supply voltages and that all sensitive instruments were protected by fuses in the event of a power surge. It was also necessary to protect all exposed circuits from accidental spills since they needed to operate in close proximity to the fluid-filled test chamber. Figure 25 illustrates the layout of the enclosure and shows sample wiring for RTDs, pressure transducers, and the viscometer.

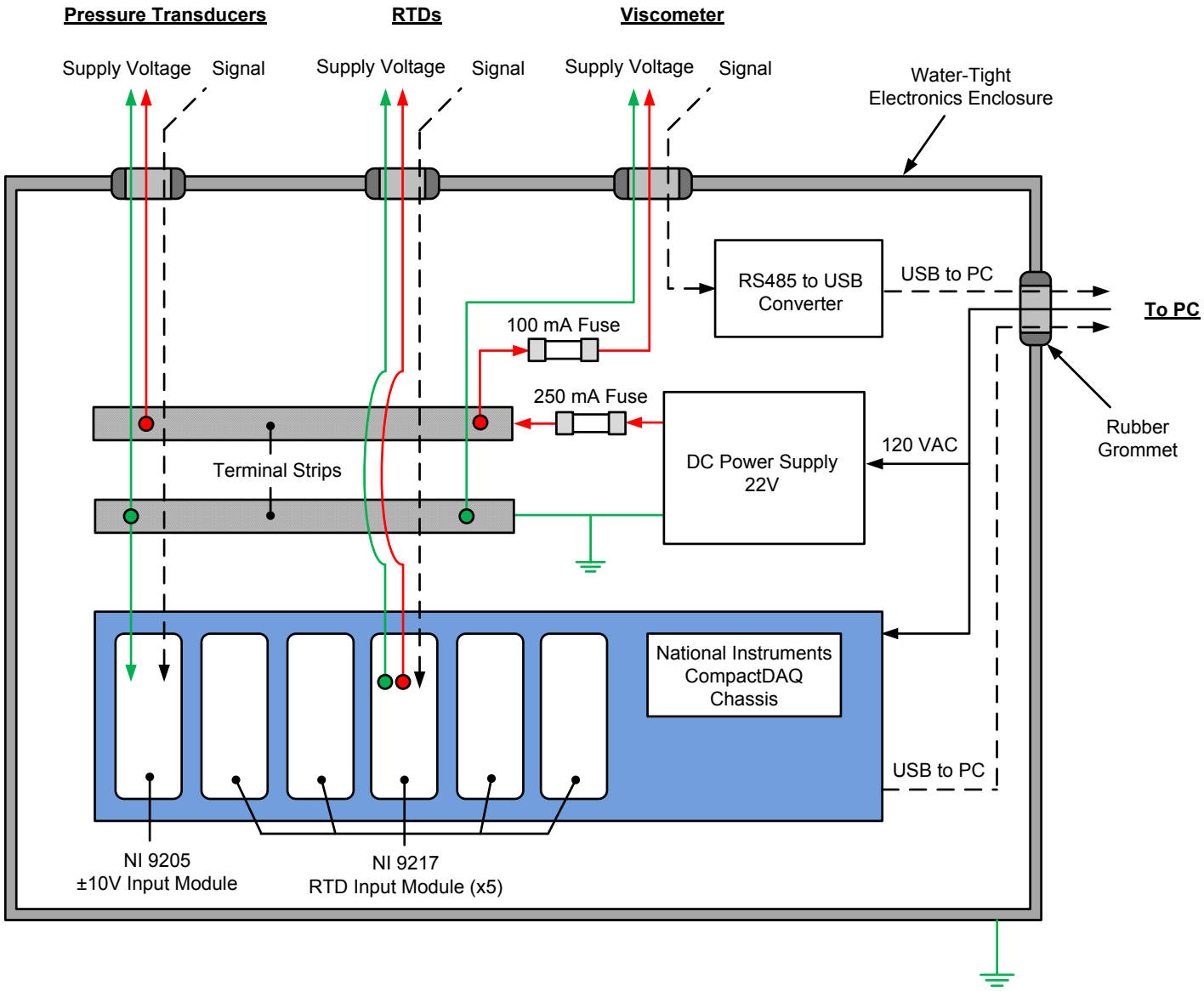
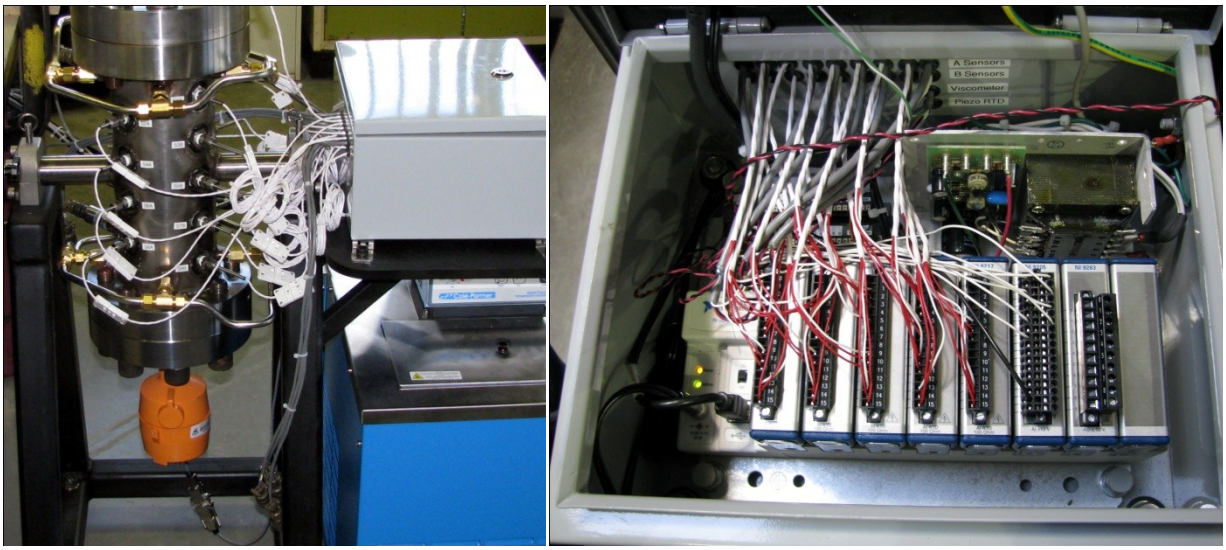


Figure 25 - Schematic of the electronics enclosure showing both the wiring layout and the physical arrangement of components.

A water-tight electronics enclosure was designed to house all sensitive electronic equipment. The hinged enclosure allowed easy access to the data acquisition system as well as to the sensor supply voltage circuitry. The installation of this enclosure relative to the test chamber is shown in Figure 26 alongside a view of the completed instrumentation wiring.



(a) (b)

Figure 26 – (a) Electronics enclosure (grey) mounted to the chamber support frame. (b) Inside view of the electronics enclosure showing the wiring when using all available chamber sensors.

3.3.5.3 *Monitoring and Control Software*

As shown in the PI diagram, control and data logging operations for all instruments were performed by the PC. In order to accomplish these tasks, custom software was developed using National Instruments LabWindows/CVI, a C-based programming environment. The three main requirements of this software were to interface with the equipment, run the experiments, and provide the user with near real-time operational data. The following sections present how these main requirements were met and introduce some of the key features of the program. The high-level flow chart presented in Figure 27 is provided as a reference for the following sections. The main source code has been included in Appendix A for additional reference.

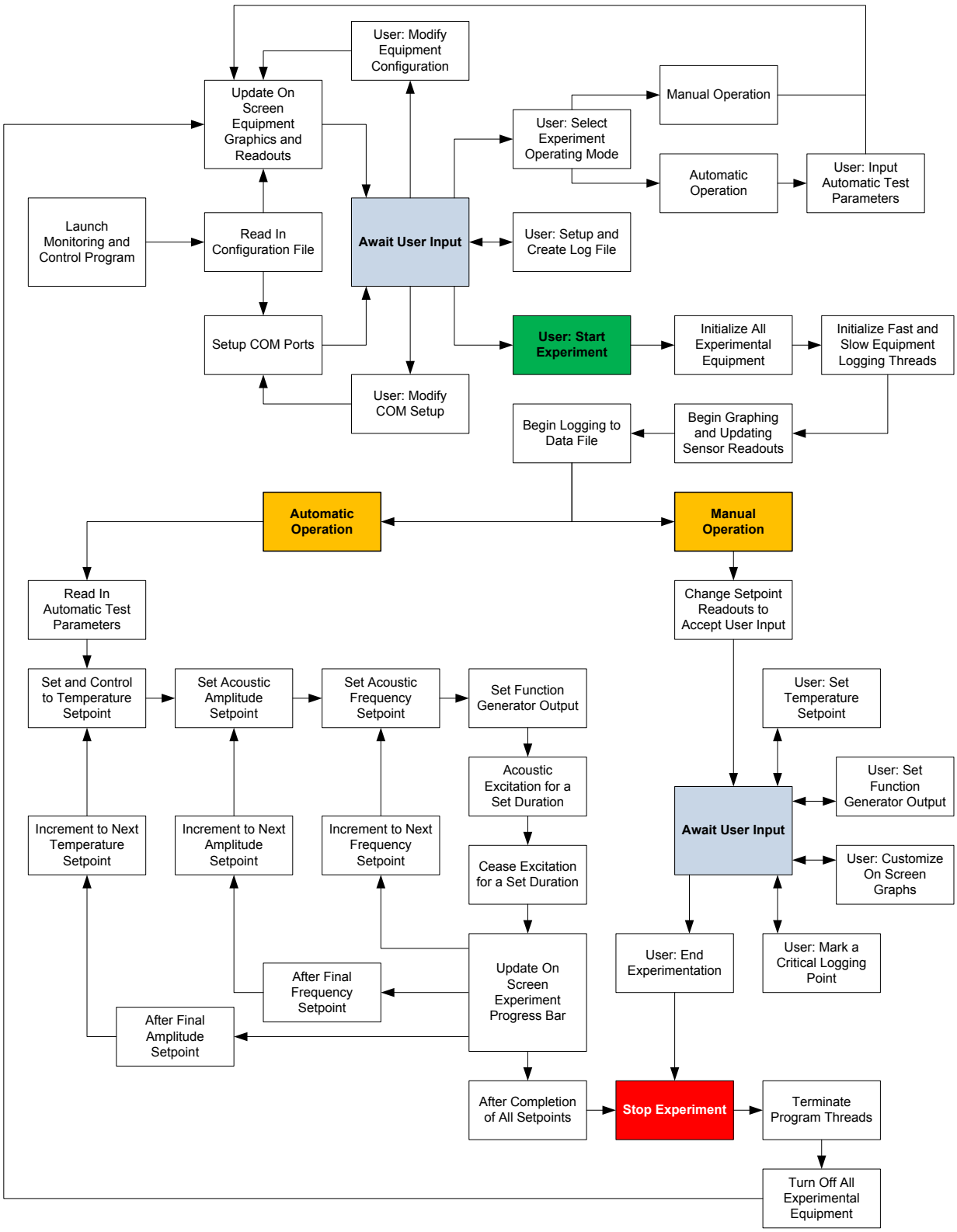


Figure 27 - High-level flow chart of monitoring and control software.

3.3.5.3.1 Interfacing with and Logging from Equipment

As illustrated in the PI diagram, several communication protocols were used to transfer data and commands between the PC and the various instruments. When available, built-in LabWindows/CVI library functions were used; however, function libraries and drivers for several instruments had to be developed separately. Table 5 below summarizes the degree of software development required to communicate with each instrument. The final program build included all data logging and instrument control functions for these instruments.

Table 5 - Programming requirements for each instrument

Instrument Name	Communication Protocol Used	Drivers (Available or Developed)	Function Library (Available or Developed)
Fluid Bath	RS-232	Available	Developed
Data Acquisition System	USB	Available	Available
Function Generator	USB	Available	Available
PACE 5000 Pressure Controller	RS-232	Available	Developed
Viscometer	RS-485 MODBUS	Developed	Developed

In addition to simply establishing communication, the program was responsible for logging data to a file for later post processing. Since the devices had drastically different logging rates (e.g. viscometer @ 0.5Hz vs. data acquisition system @ 1000Hz) multi-threading was incorporated into the program to allow logging from all instruments without the risk of memory access faults. This allowed the program to output a single tab-delimited log file with all sensor and instrument data from experiments.

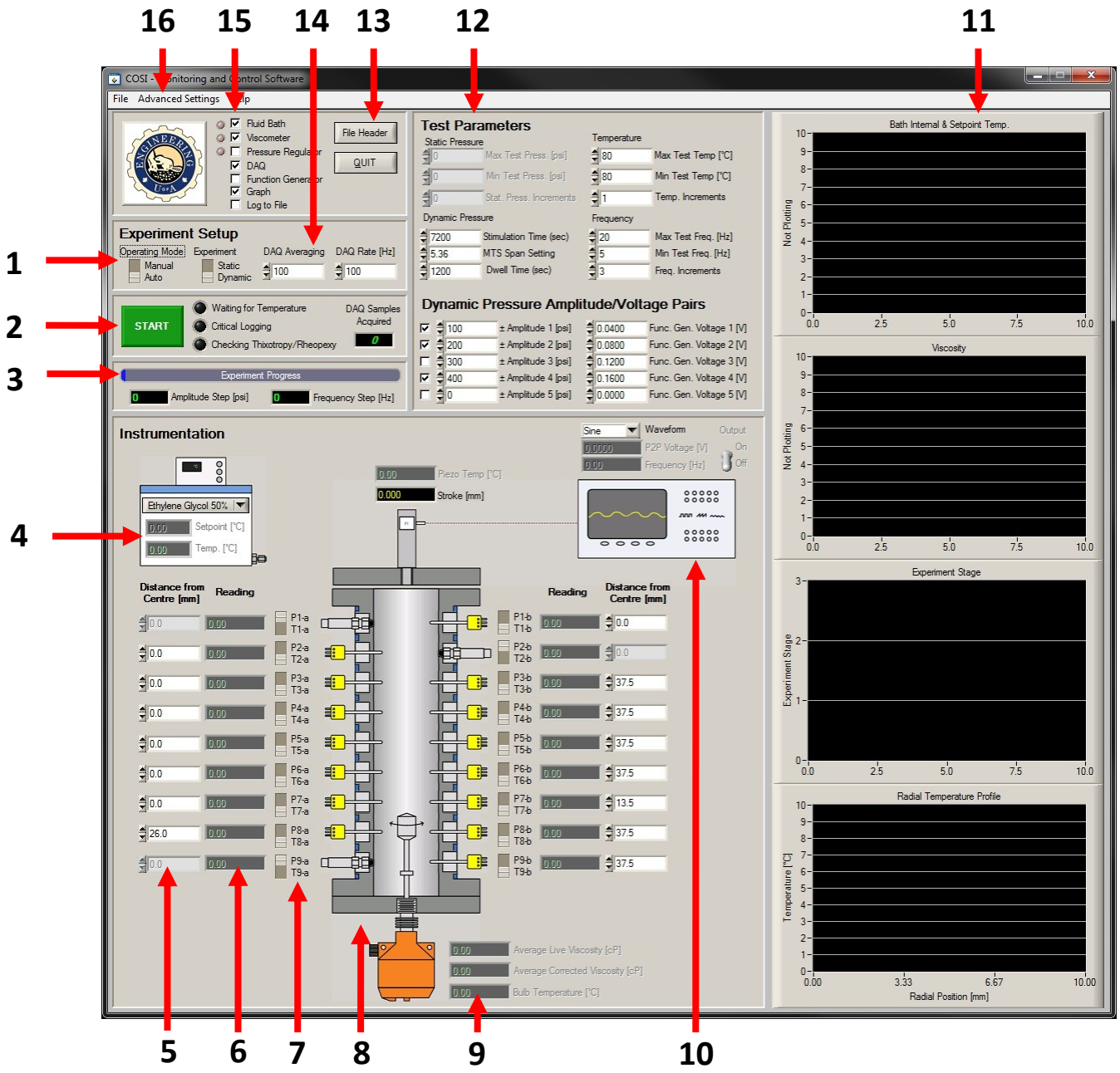
3.3.5.3.2 Programmatic Experiment Control

The monitoring and control program allowed static and acoustic experiments to be performed either manually or automatically. In manual mode, all equipment functions were directly controllable from the GUI where as in the more commonly used automatic mode, the GUI was only used to setup the test parameters and view operational data. After inputting the test parameters and naming the log file, the user could initiate the automatic experiment procedure with an onscreen control. Algorithms stepped through the parametric study according to the user's initial input, while separate feedback control algorithms ensured temperature and pressure setpoints were correctly achieved. In the final build of the program, this automatic testing functionality was successfully used to run stable multi-day experiments without the need for any user input besides the initial configuration.

During automatic operation, safety was a primary concern since the user was not always present to react to any risky situations that arose. Warning alarms and an emergency stop button were incorporated into the program so that, in the unlikely event of a control algorithm failure, the equipment and anyone nearby would be safe. In addition to these features, off-site monitoring and control of all equipment was possible using the Windows remote desktop function complemented by a webcam trained on the equipment. Thanks to these safety features, no accidents occurred during software control of the experiments.

3.3.5.3.3 Operational Feedback

The third main function of the monitoring and control software was to provide the user with near real-time information about the state of the experiment. This was accomplished through a GUI interface. The on-screen data readouts and graphics provided the user with all operational data for the experiment. This included current sensor data, historical sensor data, and information about the equipment configuration. On-screen graphics depicting the equipment configuration and relevant sensor readouts changed automatically according to the user's selected experiment type. Figure 28 below illustrates these and other operational data visible on the main GUI window.



Legend

1. Operating Mode and Experiment Type
2. Experiment Start / Emergency Stop Button
3. Experiment Progress Bar
4. Fluid Bath Readings
5. DAQ Sensor Positions
6. DAQ Sensor Readings
7. Sensor Selection Buttons (Pressure or Temperature)
8. Graphic of Sensor Configuration (Updates to match experiment type and sensor configuration)
9. Viscometer Readings
10. Function Generator Readings/Setpoint
11. Historical Sensor Data (Graphs)
12. Parameter Entry for Automatic Experiments
13. Log File Configuration
14. DAQ Configuration
15. Equipment Configuration
16. COM Port Configuration

Figure 28 – Screenshot of the monitoring and control program GUI showing data readouts

3.3.6 Post-Processing Software

All data post-processing operations were performed in MATLAB using a custom m-file. The processing algorithms written into this script were used to:

- Apply calibration coefficients to raw sensor data
- Apply density and temperature correction coefficients to raw viscosity data
- Perform data averaging operations
- Plot and save all figures

The full source code of the MATLAB script is available in Appendix A for further reference. As the post processing requirements for each data set were dependent on the observed trends in the data, further discussion is reserved for Chapter 5.

3.4 Design of the Acoustic Excitation Experiment

This section outlines the test conditions and basic procedures used in performing the acoustic excitation experiments. A detailed test matrix is also presented.

3.4.1 Parametric Study

The controlled elements of any parametric study are a set of independent variables to be tested individually and a list of control variables that are kept constant in all tests throughout the study. In the case of this investigation, the variables presented in section 3.2.3.1 (properties of an acoustic stimulation source) defined the independent variables to be tested. The variables presented in section 3.2.3.2 (variables previously known to have an effect on viscosity) defined the list of variables to be kept constant through the entire parametric study. An experiment was developed that considered each of these variables.

The high level procedure for the parametric study is shown in Figure 29. As can be seen on this flowchart, the basic premise behind the acoustic excitation experiments was to subject the test fluids to a set of possible reservoir conditions and then step through a variety of acoustic stimulation amplitudes and frequencies, all the while recording the resulting fluid viscosity. In this way, the effect on viscosity of each of the control variables could be tested. The parametric procedure was ordered such that the variables requiring the longest time to stabilize (i.e. temperature, static pressure) were changed less frequently than those requiring little time to stabilize (i.e. stimulation amplitude and frequency) thus minimizing the total time needed to run an experiment. This structured approach allowed much of the experiment to be controlled programmatically by the monitoring and control software and thus required limited supervision.

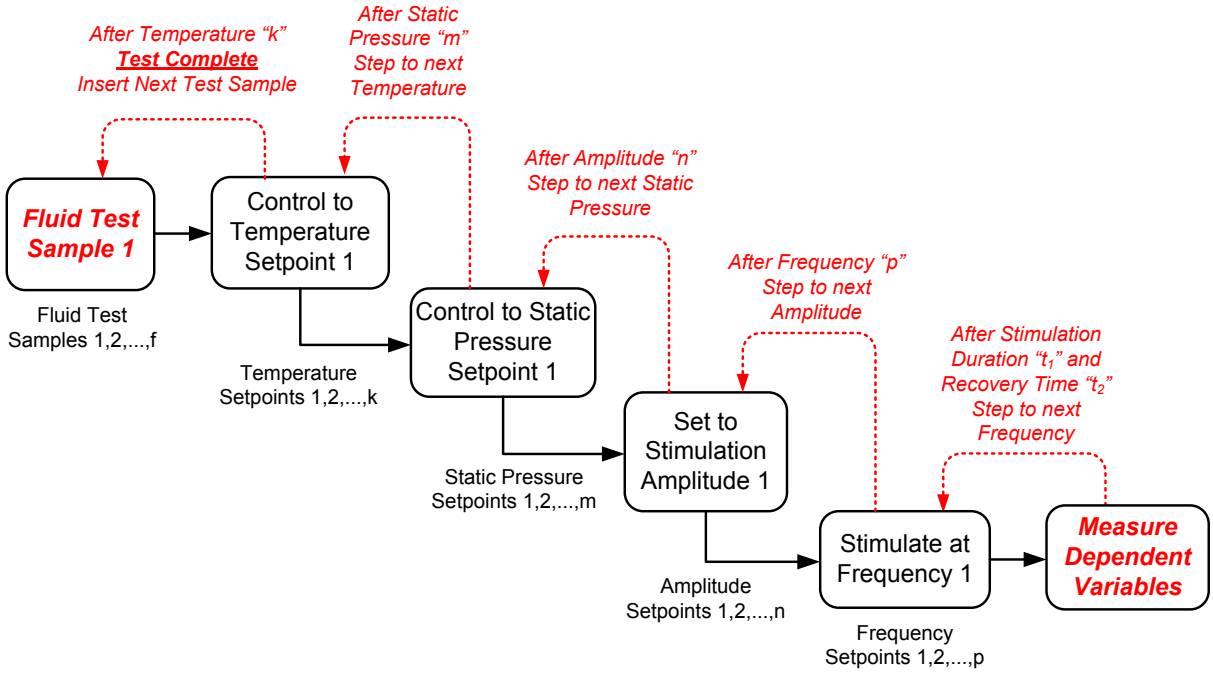


Figure 29 - High level procedure for the acoustic excitation parametric study

Figure 30 is a simulated time trace from a typical acoustic excitation experiment. This snapshot of the experiment illustrates how some of the time dependent elements are controlled in the experiment procedure. In the specific cases of thixotropy and stimulation duration, the experiment does not progress to the next setpoints until the viscosity has stabilized. For stimulation duration,

stimulation is continued until viscosity reaches a stable value. For thixotropy, the experiment does not progress to the next setpoint until the fluid returns to its original viscosity (i.e., that measured prior to acoustic stimulation).

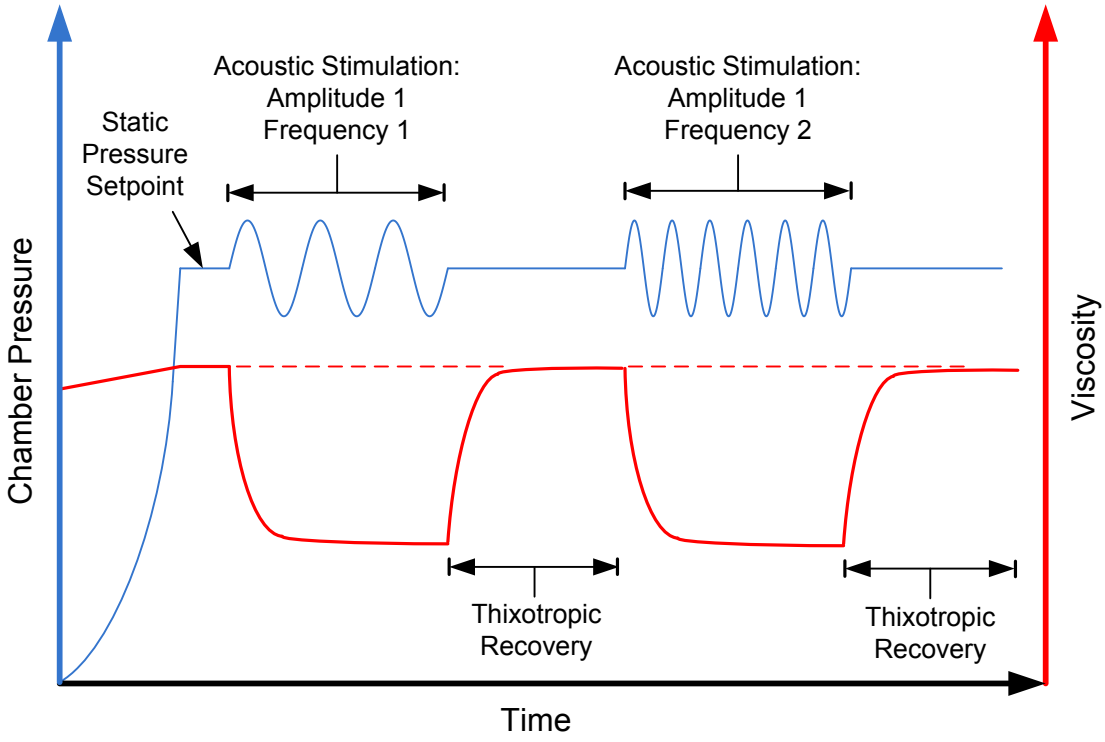


Figure 30 - Simulated result of an acoustic excitation procedure showing two frequency setpoints.

3.4.2 Test Matrix

Table 6 is a test matrix detailing the acoustic excitation parametric study. Each row represents one fluid to be tested and each column represents the variable range for a controlled variable. The user may note this is the first mention of specific acoustic excitation amplitude and frequency setpoints. Since the selection of these two ranges depended on the actual performance of the acoustic excitation system, the explanation for these ranges is presented at the end of Chapter 4 which deals with the commissioning of the apparatus.

Table 6 - Test matrix for acoustic excitation experiments

Test Fluid	Temperatures	Static Pressure	Acoustic Excitation Amplitudes	Acoustic Excitation Frequencies
N2500	20°C	3.45 MPa (500 psi)	0.69, 1.38, 2.76 MPa (100, 200, 400 psi)	5, 10, 15, 20 Hz
Bentonite/Water [13% mass]	20°C	3.45 MPa (500 psi)	0.69, 1.38, 2.76 MPa (100, 200, 400 psi)	5, 10, 15, 20 Hz
Cornstarch/Water [55% mass] (*)	20°C	3.45 MPa (500 psi)	0.69, 1.38, 2.76 MPa (100, 200, 400 psi)	5, 10, 15, 20 Hz
Bitumen	80°C	3.45 MPa (500 psi)	0.69, 1.38, 2.76 MPa (100, 200, 400 psi)	5, 10, 15, 20 Hz
Athabasca Oil Sand (*)	80°C	3.45 MPa (500 psi)	0.69, 1.38, 2.76 MPa (100, 200, 400 psi)	5, 10, 15, 20 Hz

(*) Ultimately, these fluids were not tested. See Section "5.5 Cornstarch and Oil Sand" for the reasoning behind this.

3.5 Summary

This chapter outlined the development of the research methodology from experiment conception to the design of the testing apparatus and experimental procedures. Commissioning and calibration of the aforementioned equipment and test procedures is treated in Chapter 4.

Chapter 4 System Commissioning and Calibration

4.1 Introduction

This chapter presents the results of experimentation performed during the commissioning and calibration of the experimental apparatus as well as preliminary experimentation done to characterize the rheological behavior of the test fluids. The end of this chapter summarizes the actual capabilities of the measurement apparatus and presents the test matrix used in the parametric acoustic experiments.

4.2 Fluid Characterization

The main purpose of the acoustic excitation experiments was to observe changes in viscosity caused by an imposed acoustic field. Since non-Newtonian fluids exhibit changes in viscosity with imposed shear fields, one point of interest for this study was to determine whether or not any observed changes mimicked the non-Newtonian behavior of the test fluids under shear loading. As a preliminary experiment, data was collected about the rheological behavior of the test fluids.

4.2.1 Rheology Experiments

One sample of each test fluid was placed into a Couette rheometer and programmatically stepped through a range of rotational shear rates. The measured torque was analyzed by the device and used to calculate the corresponding shear rate dependent viscosity. Since comparisons between the non-Newtonian viscous behaviour and the acoustic excitation behaviour were intended to be mainly qualitative, the rheology experiments were not exhaustive parametric studies. Rather, each sample was tested over a large enough range of shear rates to observe the onset of any non-Newtonian behaviour.

Figure 31 illustrates the test procedure that was followed during these experiments. Beginning at a low rotational speed, the rheometer spindle was rotated in the test fluid for a sufficient amount of

time to allow the sheared flow to develop. At this point, a series of torque measurements were taken at evenly spaced intervals and converted to viscosity values by the rheometer software. Upon collection of this data the spindle rotational speed was increased and the viscosity measurement repeated until the infinite shear viscosity was detected.

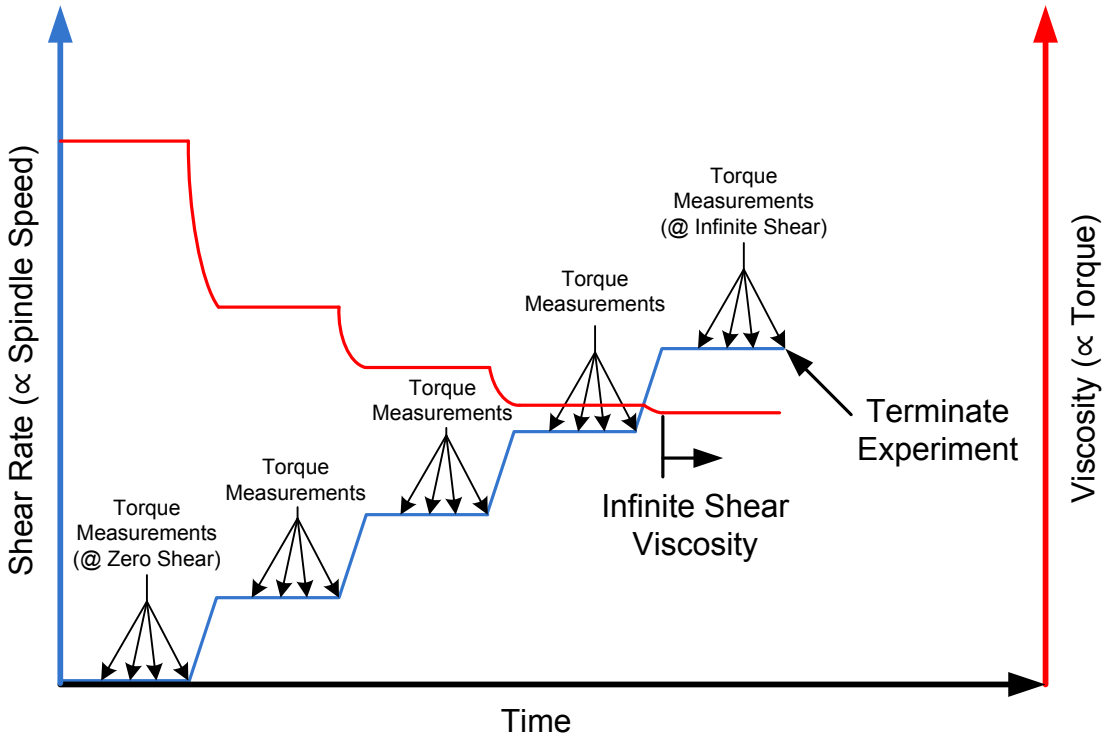


Figure 31 - Time series plot of a Couette rheometer experiment procedure for an example pseudoplastic fluid.

4.2.2 Results

Figure 32 shows a sample of the data output from a Couette rheometer experiment performed on a bentonite slurry. As evidenced by the decrease in viscosity with increased shear rate, this bentonite slurry exhibits pseudoplastic behaviour. The constant viscosity data points at the highest shear rates indicate that the infinite shear viscosity was reached. Table 7 summarizes the qualitative findings of the characterization experiments for all other test fluids.

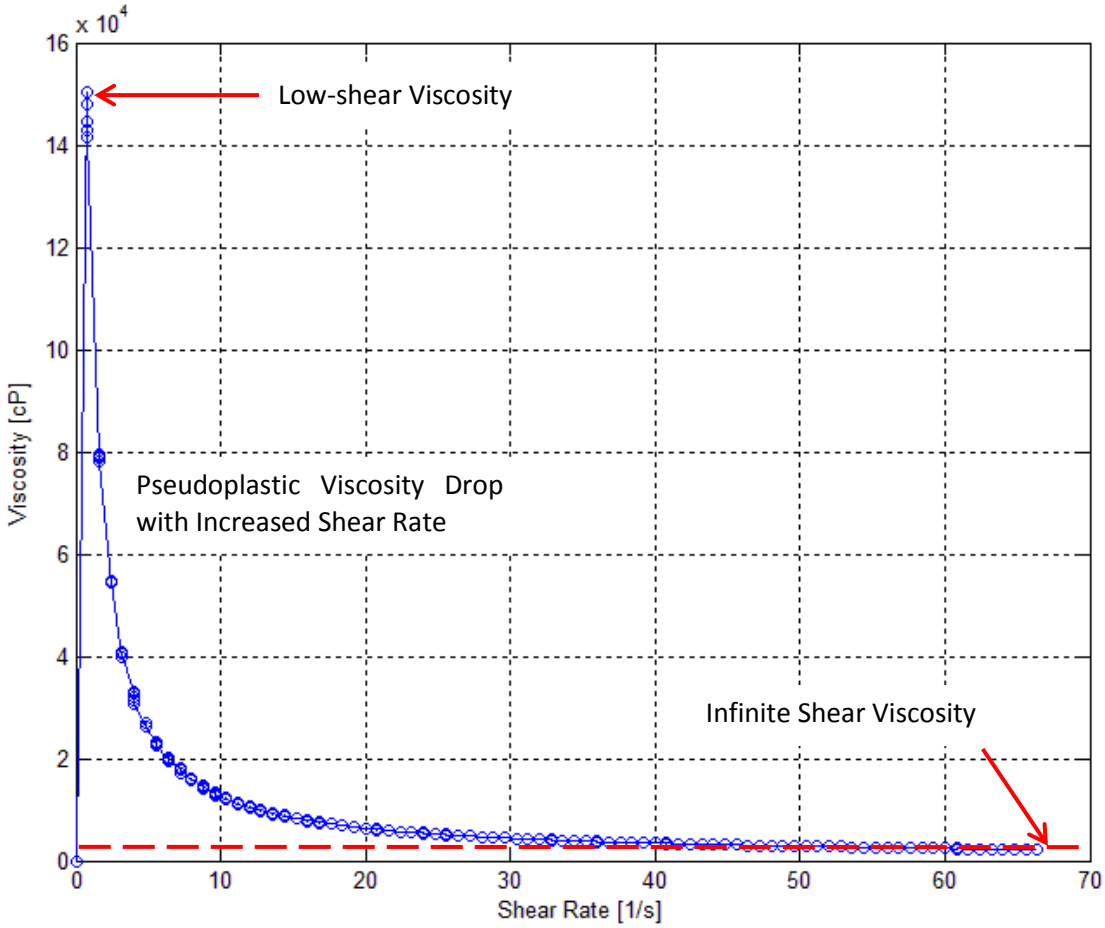


Figure 32 – Shear-rate dependent viscosity result showing pseudoplastic behaviour in a bentonite slurry (17% mass concentration).

Table 7 - Summarized results of the fluid characterization experiments.

Test Fluid	Nature of Non-Newtonian Behaviour
N2500	Pseudoplastic
Bentonite	Pseudoplastic
Cornstarch	Dilatant
Bitumen	Newtonian
Oil Sand	n/a

Bitumen was tested at approximately 80°C in order to bring the viscosity values within the measurement range of the rheometer. A vane rheometer attachment was not available for characterizing the coarse oil sand. As a result, the shear rate dependent viscous behaviour of oil sand at atmospheric pressure could not be accurately characterized using the available equipment.

4.3 Sensor Calibration

4.3.1 Temperature Sensor Calibration

The RTDs were calibrated in the Cole-Parmer fluid bath using the onboard reservoir temperature probe (factory calibrated) as the standard to compare against. Sensors were suspended in the turbulent reservoir and monitored using National Instruments MAX calibration software. The temperature in the reservoir was increased from 10°C to 90°C in 10°C increments and temperature readings (actual temperature vs measured temperature) were monitored by the calibration software.

Slope and offset calibration coefficients were computed by the MAX software. During experimentation these calibration coefficients were read in by the MATLAB post processing software and applied to all raw sensor data prior to plotting.

4.3.2 Pressure Sensor Calibration

Pressure sensors were calibrated by connecting them to the factory calibrated PACE 5000 pressure controller. In a similar procedure to that performed for the RTDs, the PACE 5000 exposed the sensors to Nitrogen pressures from 0 psi to 1400 psi in 200 psi increments while readings of measured vs actual pressure were monitored by the National Instruments MAX calibration software. Again, slope and offset calibration coefficients were computed by MAX for use in the MATLAB post processing software.

4.3.3 Viscometer Calibration

The viscometer arrived from the manufacturer factory calibrated, however, since its accuracy was of prime importance to this research study it was decided that the factory calibration would be verified in the lab. A NIST traceable viscosity calibration fluid (N2500) was ordered that was highly sensitive to changes in temperature. Between 20°C and 80°C the viscosity of this fluid changed enough to cover approximately 70% of the measurement range of the viscometer giving a wide range of temperature versus viscosity points to calibrate the viscometer against.

The viscometer was installed in the test chamber and the chamber filled with N2500. Using the monitoring and control software written for the apparatus, a test procedure was written that simultaneously controlled the temperature setpoint of the calibration fluid within 20°C and 80°C and took measurements from the calibrated RTDs and the viscometer. In addition, the static pressure in the chamber was controlled to observe its effect on the viscosity of the calibration fluid although no NIST data was available to verify the pressure dependence against.

The resulting viscosity versus temperature and pressure data was processed and plotted in custom Matlab post processing software and is shown in Figure 33. The plot shows viscosity measurements for four temperature series plotted against a range of test pressures. The stars indicate the NIST traceable viscosity values for each temperature setpoint measured at zero gauge pressure. The wide spacing between these points is indicative of the high sensitivity of viscosity to temperature (changing 20 times in magnitude over a 60°C span). As can be seen in the plot, there was close agreement between the actual and measured viscosities. The maximum measurement error of 7.9% occurred at 80°C while the minimum error of 2.7% occurred at 20°C. Considering the sensitivity of N2500 viscosity to temperature, these relatively small errors give a high degree of confidence in both the factory calibration of the viscometer and in the test chamber temperature control system. It should be noted that although the commissioning of the temperature control system is described later in this text, it was performed prior to the viscometer calibration experiments.

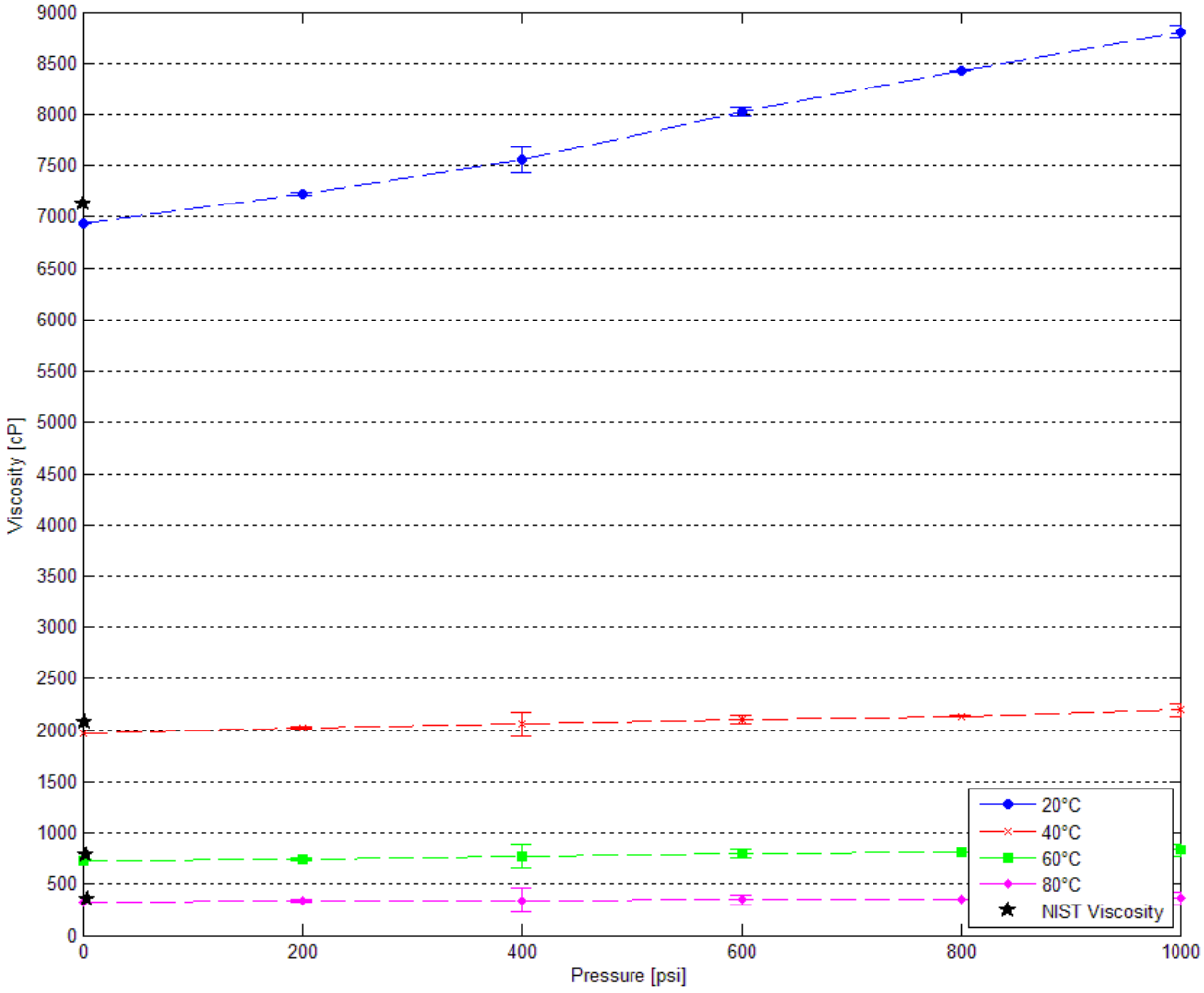


Figure 33 – Plot of NIST traceable viscosity values and viscosity calibration data for N2500.

4.4 System Commissioning

4.4.1 Temperature Control System Commissioning

The temperature control system was the first system in the experimental apparatus to be commissioned. Each thermal commissioning trial involved filling the chamber up with a test fluid and then stepping up the setpoint temperature from 20°C to 80°C in set increments allowing the system to arrive at steady state after each setpoint change. RTDs were positioned at various radial and axial positions in the chamber and data was logged continuously. Several thermal performance metrics were deduced from this data including heating time between setpoints, radial

and axial temperature gradients, and temperature stability. These are defined below followed by a sample data set from a commissioning experiment.

4.4.1.1 *Heating Time*

During the equipment design, heating times were roughly estimated using a radial heat transfer model so as to compare different possible chamber geometries against one another from a thermal standpoint. This is briefly introduced in Appendix B. The thermal commissioning trials provided a measurement of actual heating times. These more accurate values allowed more precise calculation of the estimated duration of later multi-temperature setpoint experiments. When analyzing the data from the thermal commissioning trials, rise time was used to describe the approximate heating time between setpoints.

4.4.1.2 *Temperature Gradient*

The viscometer needed to be exposed to a fluid of homogeneous temperature in order to yield the most accurate viscosity measurements. For this reason, it was important to understand whether any axial or radial temperature gradients existed in the chamber region surrounding the viscometer bulb. Logging from RTDs at a variety of known radial and axial positions during the commissioning trials allowed the magnitude of both gradients to be observed during transient and steady state operation of the device. When analyzing the commissioning trial data, four RTDs were used to observe the radial temperature gradient and eight RTDs for the axial gradient.

4.4.1.3 *Temperature Stability*

Two of the desired outcomes from the acoustic excitation experiments were the observance of any thixotropic effects and the observance of whether sustained acoustic excitation had any effect on fluid viscosity. Both observations were time dependent and thus required that viscosity measurements be taken over prolonged periods of time. During these periods, temperature

stability was critical since changes in viscosity due to temperature fluctuations could lead to a misinterpretation of the time dependent data. Realizing that the test fluids were unlikely to reach perfect steady-state temperatures in the amount of time available for each experiment, it was important to understand the degree of temperature fluctuation that would occur at each setpoint. Observing that the temperature control system was underdamped, maximum overshoot and speed of attenuation were used to gauge the relative temperature stability at each setpoint.

4.4.1.4 *Commissioning Data*

Thermal commissioning trials were performed on a variety of test fluids under a wide range of temperature setpoints and increments. Since this commissioning performance data was primarily used for internal experiment planning purposes and for programmatic control of experiment progression, the full set of plots is not presented here. The previous discussion is instead meant to present some of the considerations that went into mitigating potential errors in viscosity measurement that may have otherwise been introduced by uncertainty in test fluid temperature.

4.4.2 Pressure Vessel Commissioning

Since the test chamber had a total volume of less than 42.5 litres and was being used for experiments conducted in a research facility, as per section 2(2)(n) of the Alberta Pressure Equipment Exemption Order (Alberta Queen's Printer, 2006), it was exempt from the Alberta Pressure Equipment Safety Regulation. As such, it was not tested by a provincial safety inspector prior to use in the lab. In lieu of this inspection, commissioning tests were performed to assess the safety of the chamber under pressure. Three types of tests were performed: static and dynamic testing of the chamber under pressure and impulse testing of the pressure relief valve.

4.4.2.1 *Quasi-Static Pressure Testing*

The first commissioning tests assessed the chamber's ability to both withstand and maintain internal pressure. In order to avoid the risk of an explosion should the chamber undergo a

structural failure, the volume was pressurized with liquid rather than gas. Tap water was used for this purpose.

With all sensors and flanges installed in the chamber and the pressure relief valve removed, the volume was filled with tap water at atmospheric pressure. Once filled, a small piston was manually screwed into one of the end flanges, entering the chamber volume and thereby compressing the internal fluid. The resulting internal pressure was carefully monitored using the data acquisition program to avoid over-pressurization. Gauge pressure was gradually increased from atmospheric pressure to 11.03 MPa (1600 psi) (slightly above the dynamic working pressure of the chamber) stopping periodically to monitor for pressure leaks.

This testing was repeated when leaks were discovered around some of the peripheral sensors. After re-sealing the sensors, the chamber was found to be able to withstand and maintain pressure over the full range of static pressures.

4.4.2.2 *Pressure Relief Valve Testing*

During the dynamic pressure commissioning in the tensile testing machine, it was easy to over-pressurize the chamber. Only a small error in the displacement of the hydraulic piston was required to generate excessive internal pressures. A pressure relief valve was installed to avoid a potential accident. The relief valve was factory calibrated to vent at pressures above 10.3 MPa (1500psi); however, no information was available on its dynamic performance or on its ability to vent high viscosity fluids. To evaluate these and to verify the factory calibration, a short series of over-pressurization tests was performed using bentonite slurry (17% mass concentration).

Each experiment involved priming the test chamber with bentonite slurry and then raising the hydraulic piston into the chamber volume until the pressure relief valve was triggered. Three experiments were performed. The first two involved raising the internal pressure very slowly to verify that the relief valve triggered at approximately 10.3 MPa (1500 psi). The trigger events

observed in both of these trials indicated that the factory calibration was accurate, and gave a basic indication that the valve was not highly susceptible to fouling by high viscosity fluids. The third trial involved a step increase in pressure from atmospheric pressure to 11.0 MPa (1600 psi). The pressure relief triggered and oscilloscope measurements indicated that the internal pressure never reached 11.03MPa (1600psi). With the success of these trials, it was deemed safe to continue with the dynamic pressure commissioning.

4.4.2.3 *Dynamic Pressure Testing*

A good deal of analysis was performed during the chamber design to determine resonant frequencies (see Appendix C). The analysis indicated three resonant frequencies in the radial direction (7, 13, and 22 GHz) and two in the longitudinal direction (23, 48 MHz). Although these frequencies were many orders of magnitude higher than the desired acoustic stimulation frequencies, basic resonance testing was carried out on the system to detect any possible resonant behavior in the operational range.

A series of frequency sweeps were performed from 1-60Hz @ ±0.69 MPa (±100 psi) and the researcher was present to observe/hear any resulting resonant behavior. Over this frequency range, there was no audible or visual indication of excessive vibration. The risk of a structural failure due to resonance was deemed minimal and further tests using accelerometers were not conducted.

4.4.3 Acoustic Excitation System Commissioning

It was important to have an accurate understanding of what dynamic pressure amplitudes and frequencies could be generated within the test chamber. This final set of commissioning trials characterized this dynamic performance of the acoustic excitation system and yielded the variable limits used in the acoustic excitation parametric study. Two types of commissioning experiments were required to gain this understanding of the system behaviour: (1) quasi-static compressibility experiments for determining the precise relationship between hydraulic piston position and

chamber pressure; and (2) dynamic compression experiments for determining the frequency response of the tensile testing machine hydraulics.

4.4.3.1 *Quasi-Static Compressibility Experiments*

The quasi-static compressibility experiments provided the data needed to check the previously derived constitutive relationship between hydraulic piston stroke and internal chamber pressure. The procedure was similar to one performed in the pressure vessel commissioning. The chamber was primed with a test fluid and the hydraulic piston was then slowly raised into the chamber volume, compressing the test fluid and pressurizing the chamber up to 9.65MPa (1400psi). Using a digital oscilloscope, the piston position data was logged from the tensile testing machine's linearly variable differential transformer (LVDT) alongside the pressure data from one of the chamber pressure sensors. The resulting oscilloscope trace from one trial is shown in Figure 34. The orange curve (upper) represents the internal chamber pressure while the blue curve (lower) represents the piston displacement. As can be seen by the phase lag between the two curves, no appreciable pressurization occurs until the piston has moved approximately 50% of its total displacement.

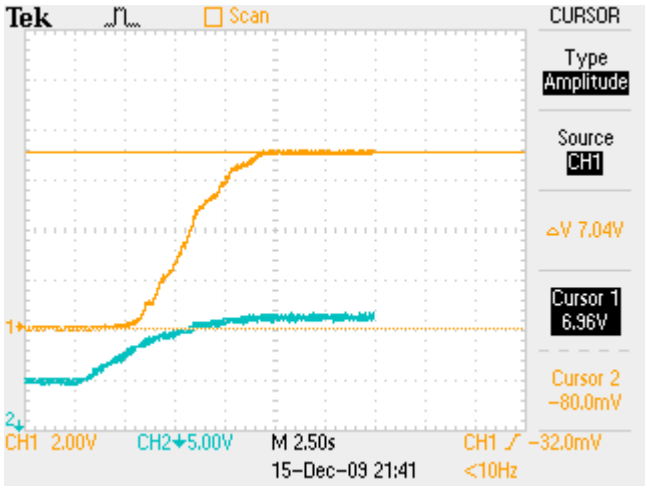


Figure 34 - Oscilloscope trace illustrating non-linear pressurization due to residual gas in the test chamber

As seen in the figure, the relationship between internal pressure and piston position was highly non-linear at the onset of pressurization and only became linear at higher internal pressures.

Subsequent quasi-static compressibility experiments performed on other test fluids showed this same non-linear behavior. This was contrary to the linear constitutive relationship developed in the engineering analysis of fluid compressibility. Further reading on the topic of fluid compressibility (McCain, 1990) revealed that this behavior was likely the result of small amounts of residual gas left in the test chamber, gas that the engineering analysis had assumed was completely removed. The cause is likely that during the initial piston compression (at low pressures), residual gas was compressed and forced back into solution so it was not until after this occurred that compression of the test fluid became linear with piston position. Figure 35 shows this graphically by overlaying an ideal time trace of a piston compression and the resulting linear and non-linear pressurizations within the test chamber.

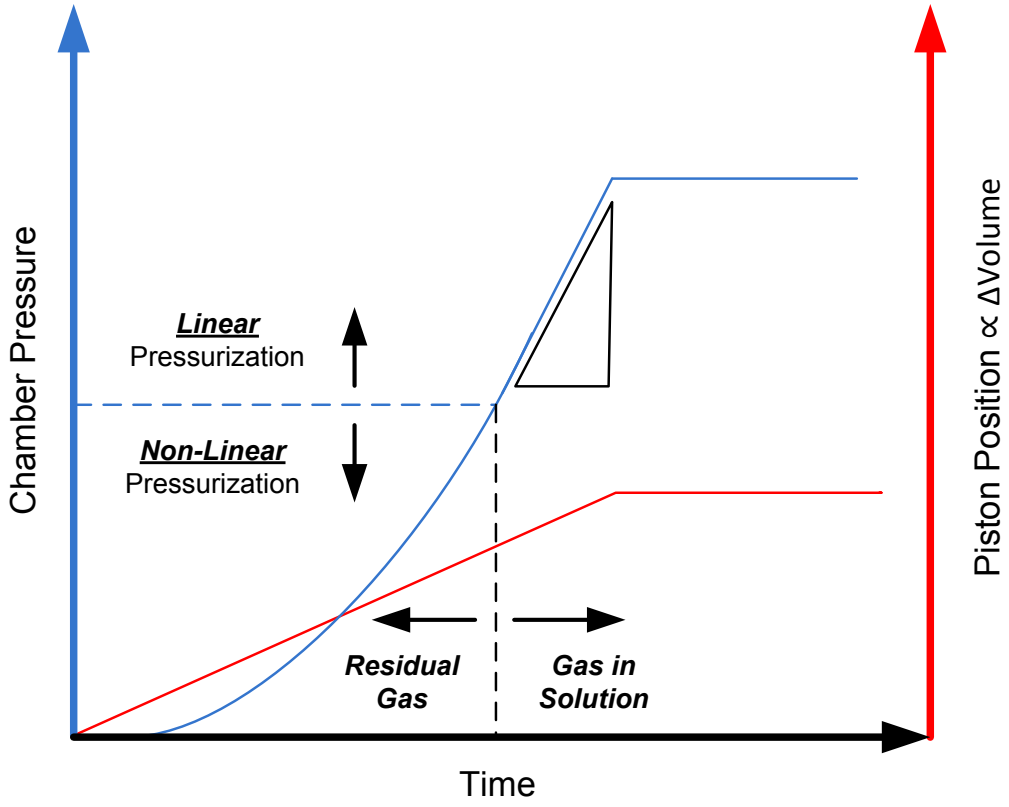


Figure 35 – Plot of piston position against chamber pressure showing linear and non-linear pressurization regions and state of residual gas.

In all test fluids, the linear pressurization region began at pressures near 1.38MPa (200psi) but the acoustic excitation experiments were designed to be performed at estimated reservoir pressures

[i.e. 3.45 MPa (500 psi)] which meant that during experimentation, the test fluids were in the region of linear compressibility. The data collected during these quasi-static compressibility experiments was more accurate for predicting/controlling the pressure amplitude than the analytically derived constitutive relationship and so the empirical relationships were used.

4.4.3.2 *Dynamic Compressibility Experiments*

The constitutive relationships developed from the quasi-static compressibility data allowed precise prediction of the internal chamber pressure; however, information on the frequency response of the hydraulics was still needed to fully understand the dynamic performance of the acoustic excitation system. Pressurization frequency was dependent on piston velocity. The tensile testing machine hydraulics limited which amplitude/frequency pairs were possible since they governed the maximum velocity of the piston at any given stroke length. The aim of the dynamic compressibility experiments was thus to determine the maximum pressurization frequencies achievable at a given set of acoustic pressure amplitudes.

Acoustic pressure amplitudes of ±0.69, 1.38, 2.76 MPa (±100, 200, and 400 psi) were selected for the dynamic testing. Each test consisted of priming the test chamber with N2500 (highly compressible and thus the probable worst case), raising the internal pressure up to 3.45 MPa (500psi) then cycling the piston in a sinusoidal motion to generate the target acoustic excitation amplitude. Piston frequency was gradually increased using the function generator. The chamber pressure and piston LVDT output were monitored with a digital oscilloscope.

The tensile testing machine was wired such that it would sacrifice piston stroke in favor of maintaining the target pressurization frequency. In this configuration, failure of the hydraulic system to produce a desired pressurization frequency/amplitude combination was easily identifiable by a drop in pressure amplitude and piston stroke. Figure 36 illustrates such a scenario for water pressurized at 3.45 MPa (500 psi) static pressure and cycled at ±1.38 MPa (200 psi) acoustic pressure. The oscilloscope trace shows the chamber pressure observed over an upward and then downward frequency sweep. As evidenced by the degraded sine wave at high

frequencies, the hydraulics were unable to accurately sustain this amplitude/frequency combination. By performing the tests at each of the acoustic pressure setpoints and observing the frequency which triggered the onset of a drop in amplitude, the frequency response of the acoustic excitation system was characterized. The resulting amplitude/frequency data are summarized in Table 8.

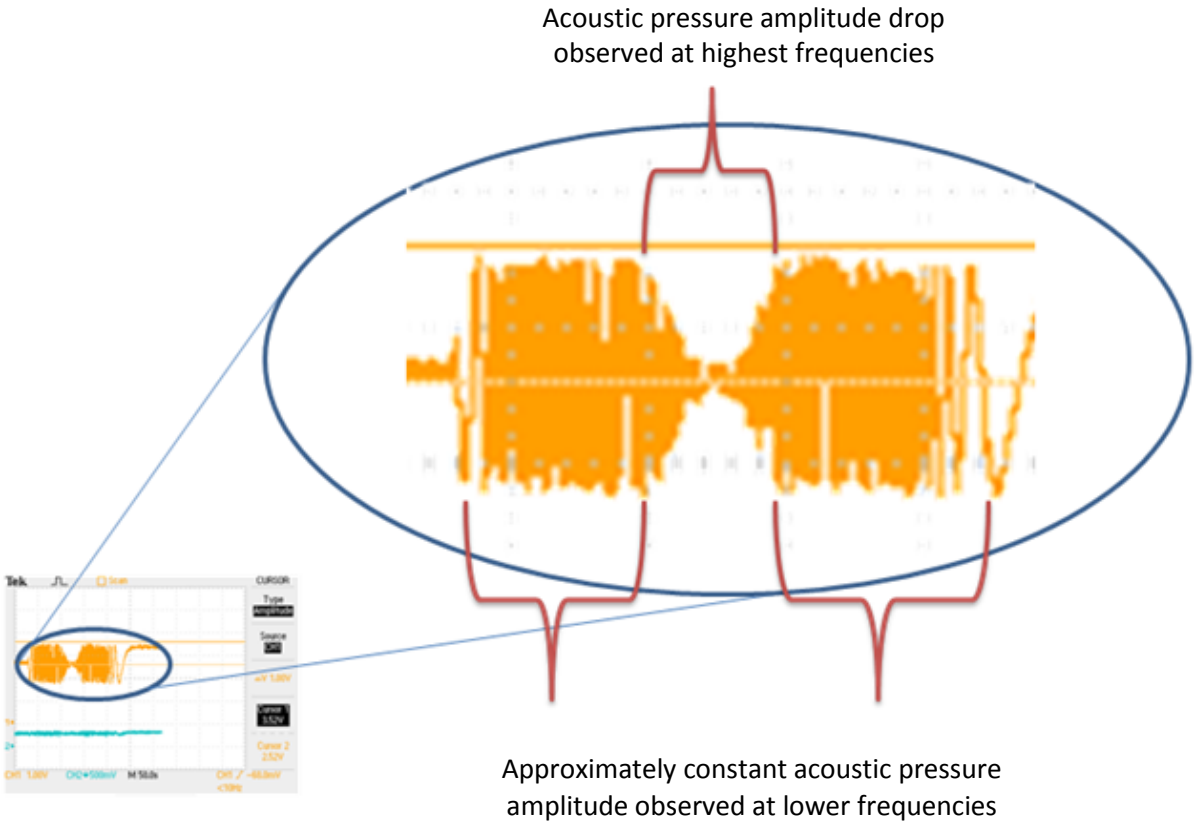


Figure 36 – Zoomed in oscilloscope trace of a dynamic pressurization experiment illustrating the pressure drop seen at high pressurization frequencies

Table 8 - Maximum pressurization frequencies/amplitudes achievable with N2500

Acoustic Excitation Amplitude (MPa) [psi]	±0.69 [100]	±1.38 [200]	±2.76 [400]
Drop-off Frequency (Hz)	115	55	20

As acoustic pressure amplitude increased, the maximum achievable pressurization frequency decreased. This frequency response data was used to determine the frequency setpoints in the acoustic excitation parametric study. In order to test each acoustic pressure amplitude at the same frequencies, the frequency setpoints in the parametric study were capped at the lowest achievable frequency: 20Hz. The amplitude and frequency ranges used in the test matrix were therefore set at ±0.69, 1.38, 2.76 MPa (100, 200, 400 psi) and 5, 10, 15, 20 Hz respectively.

4.5 **Summary**

The experiments performed in the commissioning and calibration of the experimental equipment were presented in this chapter. Calibration of the temperature, pressure, and viscosity sensors was performed and analysis of the commissioning data showed that the vessel was capable of safely controlling and maintaining the pressures and temperatures called for in the model reservoir assumptions. In addition, commissioning trials of the acoustic excitation system indicated that the apparatus was capable of subjecting fluid samples to sinusoidal pressure fluctuations of up to 2.76 MPa (400 psi) at frequencies of up to 20Hz.

Chapter 5 Acoustic Excitation Experiments

5.1 Introduction

Chapter 5 presents the results of the acoustic excitation experiments along with a discussion of potential industrial implications of the findings. The discussion is divided into sections for each test fluid and, where applicable, subsections for each of the two types of data plots: time series plots and amplitude frequency plots. The following discussion will aid the reader in the interpretation of these data plots.

5.1.1 Time Series Plots

Time series plots are used in this chapter to illustrate the time-dependent viscous behavior of test fluids. These include any changes in viscosity that occur during acoustic excitation as well as any thixotropic or rheopectic behaviors that occur after excitation has ceased. All such time series plots are similar to one another in that they each present viscosity on the vertical axis against time on the horizontal axis. Figure 37 is an example of such a plot illustrating time dependent viscous behavior of a fluid which undergoes a viscosity reduction during excitation. The labels on this figure indicate the period where acoustic excitation takes place as well as identify the “baseline viscosity”, “stimulated viscosity”, and “recovered viscosity”, terms used later on in the discussion of results to describe the steady state viscosity readings taken at various points during an experiment. As is evident from this figure, the speed of response to the onset of acoustic excitation and the speed of viscous recovery can also be gauged from a time series plot.

While amplitude frequency plots are presented for each of the test fluids, time series plots are only presented where measurably significant viscosity changes are detected so that a presentation of the time dependent viscous behavior would complement the discussion.

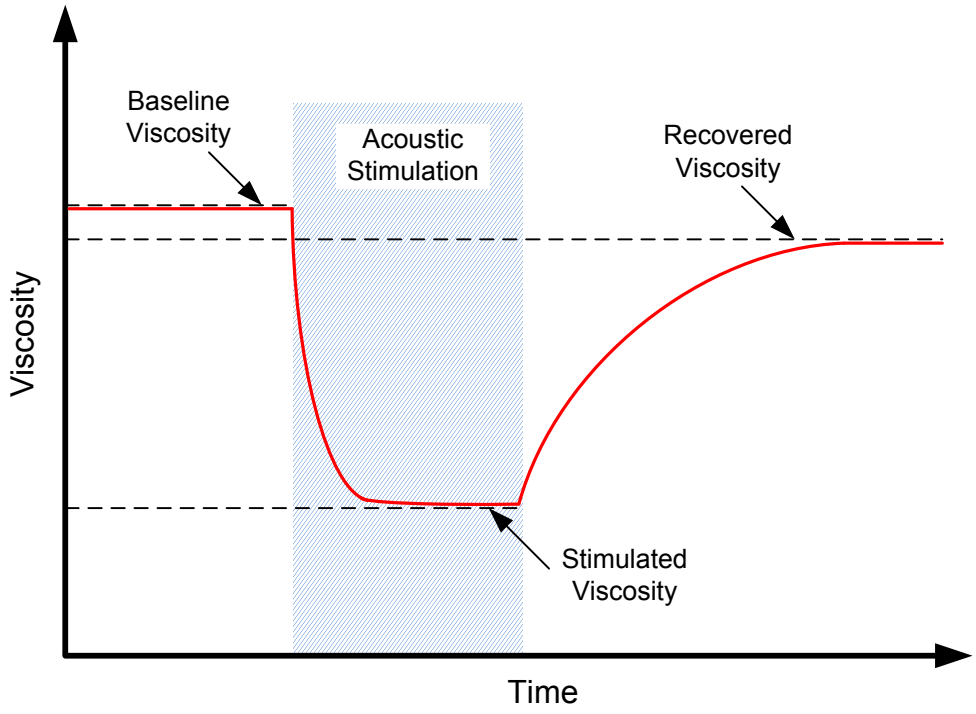


Figure 37 - Schematic representation of time dependent viscous behaviour illustrating the 'baseline', 'stimulated', and 'recovered' viscosities

5.1.2 Amplitude Frequency Plots

Much information can be obtained from time series plots however they are ineffective at illustrating the dependence of viscosity on acoustic excitation parameters. In a potential reservoir production strategy employing acoustic excitation over a large volume for long periods of time, the asymptotic stimulated viscosity would be of more interest than the transient viscous behavior. A different data plotting scheme is more effective for quantifying such data. 'Amplitude frequency plots', as they are referred to in this document, use averaged data taken from time series plots to illustrate the effects of acoustic excitation amplitude and frequency on the stimulated viscosity.

Figure 38 illustrates the amplitude frequency plotting strategy. Each colored data series represents a given acoustic excitation amplitude and each point on the data series represents the steady state viscosity of the fluid measured at a given acoustic excitation frequency. Contrary to time series

plots where stimulated viscosity is shown more qualitatively, amplitude frequency plots show calculated quantitative values for stimulated viscosity. Each point is the result of averaging the final portion of viscosity data collected at the asymptotic stimulated value. These data averaging operations (performed in MATLAB) serve the function of removing the slight viscosity oscillations caused by minor temperature fluctuations centered about the setpoint temperature. By averaging over several temperature fluctuation cycles, this allows for the calculation of single point stimulated viscosity values at the setpoint temperatures.

The solid horizontal black line on the plot indicates the baseline viscosity of the fluid (i.e., under no acoustic excitation) and error bars at ±10% of the measured value have been calculated and plotted for each data point. This 10% error is meant to encompass errors in viscosity measurement (maximum of 9% as seen on Figure 33) and in temperature measurement (estimated at 1% since sensors were calibrated). This plotting strategy reveals the significance of acoustic excitation amplitude and frequency on the viscosity of the test fluids. A data point with error bars overlapping the solid horizontal line indicates that at that particular amplitude and frequency, acoustic excitation does not have a measurably significant effect on the viscosity of the fluid being tested since any deviation from the baseline viscosity could be due to temperature or viscosity measurement error. By the same logic, a data point with error bars lying outside the solid horizontal line indicates a measurably significant effect of acoustic excitation on the stimulated viscosity of the fluid. Examples of both such observations (green series – significant, blue series – insignificant) are illustrated in Figure 38.

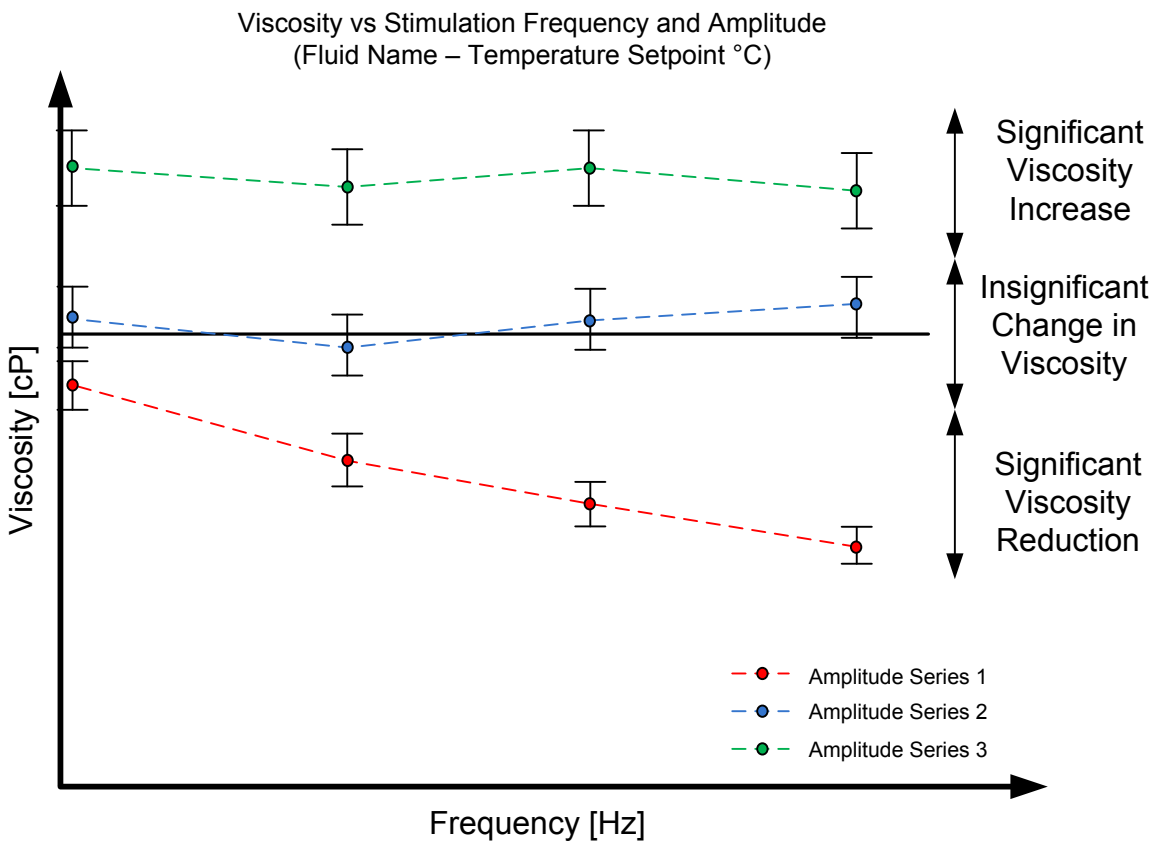


Figure 38 – Sample amplitude frequency plot depicting measurably significant (green) and insignificant (blue) results of an acoustic excitation experiment

In addition to providing insights as to the effect of acoustic excitation on viscosity, this plotting style facilitates future economic analyses of viscosity dependent production technologies. If the viscosity-temperature relationship is well characterized for a fluid, one could use an amplitude frequency plot to estimate the acoustic excitation amplitude and frequency combination required to generate the equivalent change in viscosity. Provided information is available on both the cost of heating the fluid and on the cost of acoustically exciting the fluid to its stimulated viscosity, a cost comparison between the two methods of viscosity modification could be made to determine which viscosity modification technique is most cost effective. This analysis was outside of the scope of the current study.

5.2 N2500 Calibration Standard

5.2.1 Amplitude/Frequency Plot

The amplitude frequency plot for the N2500 viscosity calibration standard is shown in Figure 39. As indicated by the close proximity of the horizontal dashed black lines to the solid black line, there is a relatively small change in the viscosity of N2500 over a 0.25°C temperature range. Despite this cramping of the threshold lines one can see that for all acoustic excitation amplitudes and frequencies tested, the viscosity points overlapped or barely dipped below the lower threshold line. This indicated that at the amplitudes and frequencies tested, acoustic excitation had a negligible effect on the viscosity of N2500.

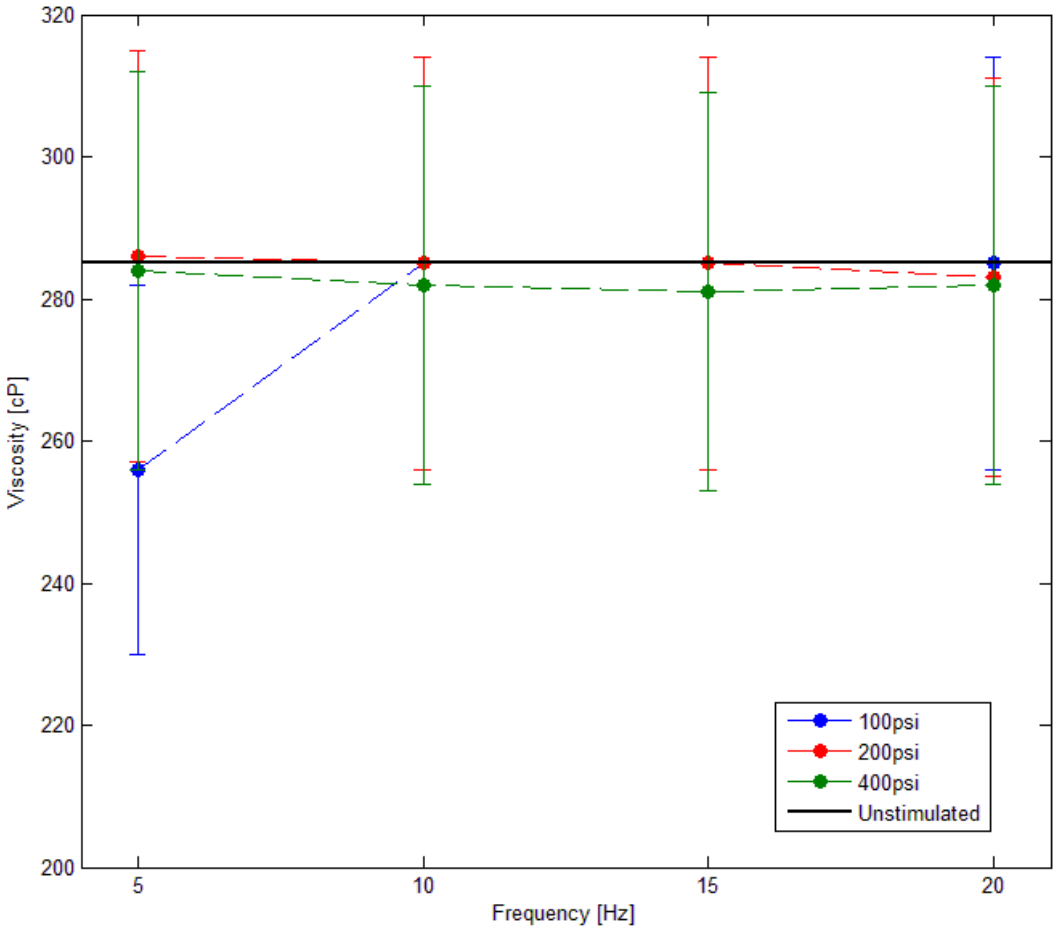


Figure 39 - Amplitude frequency plot for N2500 calibration standard

5.3 Bentonite

5.3.1 Amplitude/Frequency Plot

The amplitude frequency plot for a 13% mass concentration of bentonite is shown in Figure 40. Similar to the plot for N2500, the ±0.25°C threshold lines are in very close proximity to the solid black line and in this case are barely distinguishable. Contrary to the case for N2500 however, the data series show that acoustic excitation causes a significant change in the viscosity of this bentonite mixture.

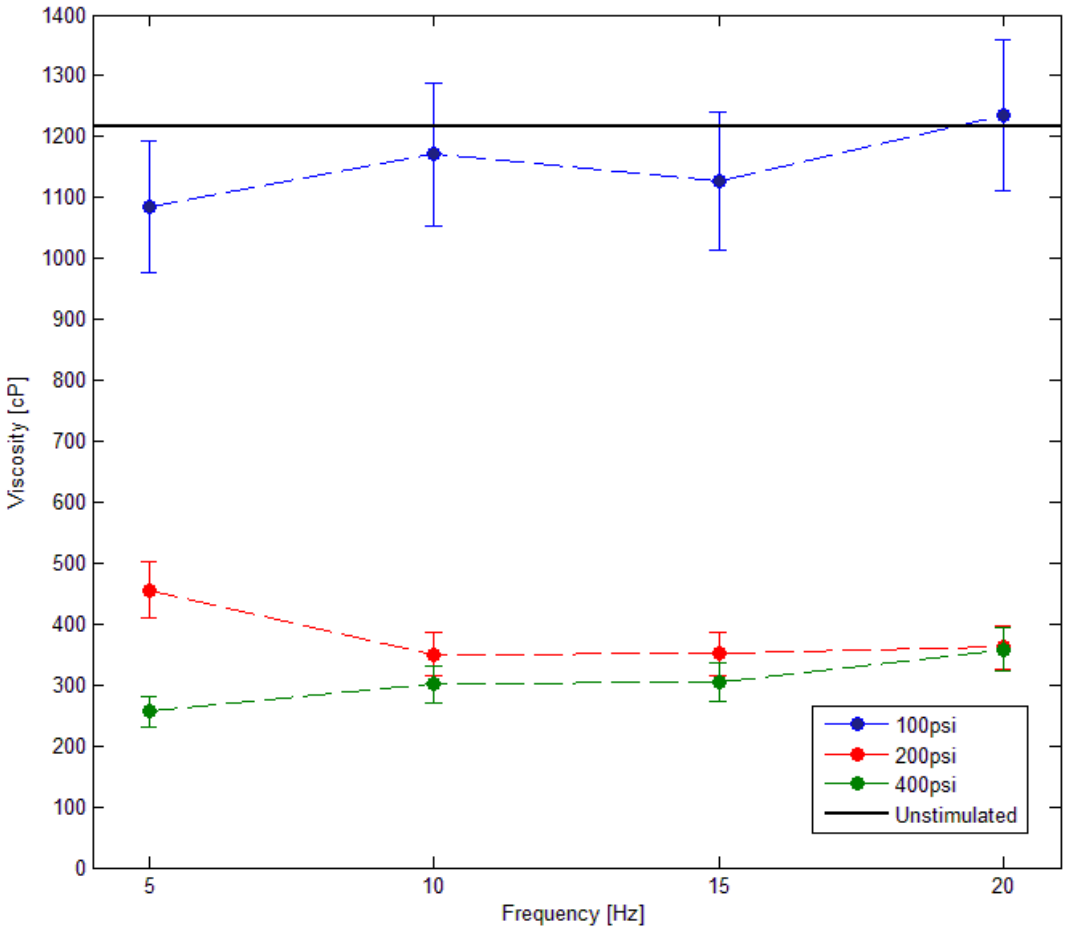


Figure 40 - Amplitude frequency plot for bentonite (13% mass concentration)

The three acoustic excitation amplitude series are well below the lower threshold line which indicates measurably significant viscosity reductions at all acoustic excitation amplitudes. There is a positive correlation between the acoustic excitation amplitude and the magnitude of the viscosity reduction as indicated by the separation between the increasing amplitude series.

The lowest acoustic excitation amplitude series (±100psi - blue data points) showed the smallest viscosity reductions, approximately 5% from ~1220cP down to ~1150cP. Similarly, the largest amplitude series (±400psi – green data points) showed the largest viscosity reduction, approximately 75% from ~1220cP down to ~300cP. The middle amplitude series (±200 psi – red data points) showed a viscosity reduction near in magnitude to the ±400psi series indicating a possible non-linear correlation between the magnitude of the acoustic excitation amplitude and the magnitude of the viscosity reduction. This non-linearity also indicated the possibility that there may be a maximum possible (i.e. asymptotic) reduction in viscosity with increasing acoustic excitation amplitude. A broader range of acoustic excitation amplitude experiments would be required to increase the resolution and make this determination for certain.

The fact that each amplitude series is approximately horizontal indicates that for the range of frequencies tested there is minimal effect if any of acoustic excitation frequency on viscosity. There is however very likely a minimum acoustic excitation frequency below which viscosity is unaffected by excitation. The reasoning behind this rests on the fact that at some very low frequency, the nature of stimulation ceases to be acoustic and is instead quasi-static pressurization. At this point, it is expected that the stimulated viscosity would be the same as the baseline viscosity. The exact frequency where this occurs could be determined by repeating the acoustic excitation experiment at a number of frequencies approaching 0Hz.

5.3.2 Time Series Plots

Figure 41 is a snapshot of the acoustic excitation experiment performed at ±400psi at 5 Hz using a bentonite sample. The shape of this curve was common to all other experiments performed on

bentonite at different stimulation amplitudes and frequencies, the only difference being the magnitude of the viscosity reduction. The onset and termination of acoustic excitation are marked on the figure alongside a number of other points of interest including the baseline, stimulated, and recovered viscosities.

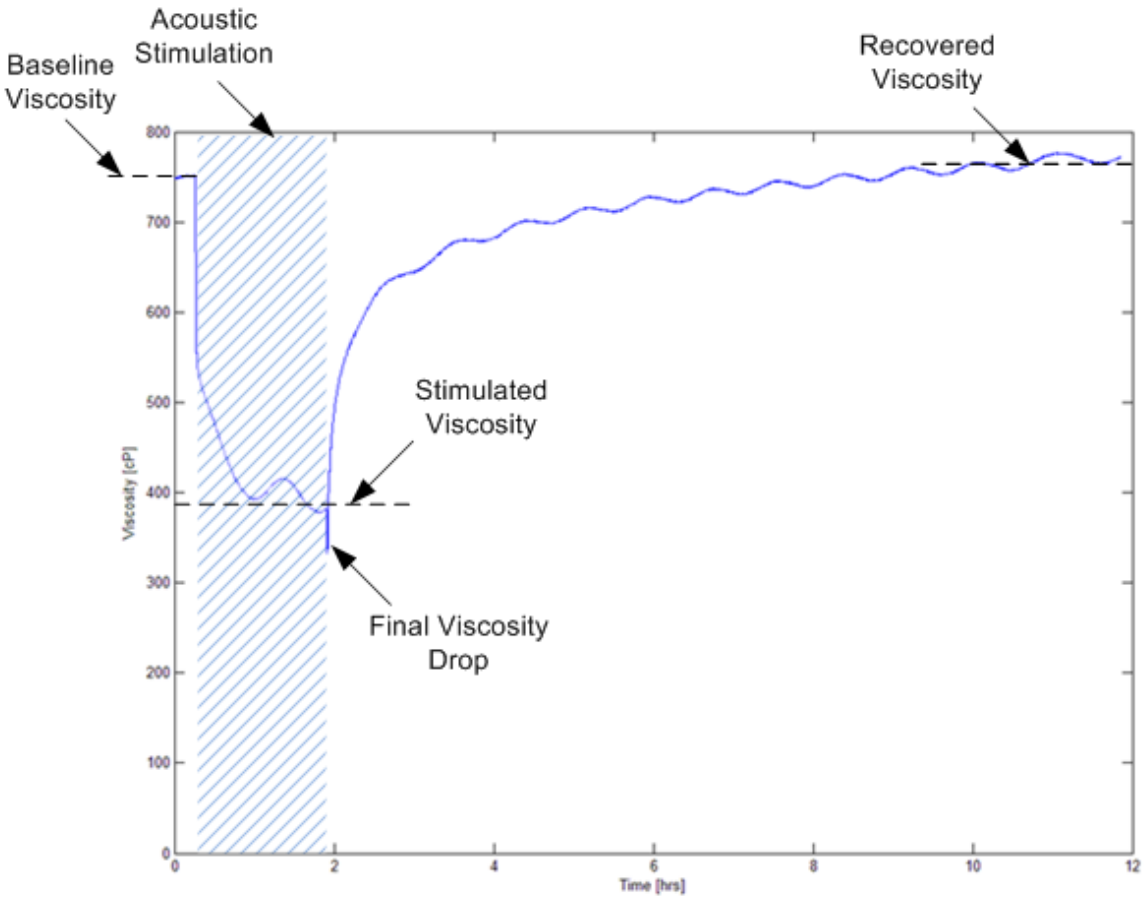


Figure 41 - Time series plot of a bentonite sample stimulated at ±400 psi at 5Hz exhibiting changes in viscosity

As can be seen by the steep negative slope of the viscosity curve following the onset of stimulation, the initial viscosity reduction occurred rapidly. For the curve presented above this amounted to a 27% reduction in viscosity in 82 seconds. Further viscosity reduction occurred at a slower rate until the stimulated viscosity was eventually reached just short of 2 hours into the stimulation cycle. Viscosity curves at other stimulation amplitudes and frequencies showed this same behaviour with

stimulation at ±100psi exhibiting slightly slower initial rates of viscosity reduction (approximately 15% drops in 120 seconds) than the ±200 and ±400psi curves which were similar to one another.

Figure 42 is a time series plot for the ±200 psi experiment (red curve) of Figure 40. As can be seen on this graph, the viscosity of the test sample is reduced to the minimum value of stimulated viscosity shortly after the onset of excitation at the different frequencies. The 10, 15, and 20 Hz tests show almost complete agreement in their behavior whilst the 5Hz test follows a less steep curve and plateaus at a slightly higher minimum viscosity. It is postulated that the discrepancy in curve shapes might be caused by the rate of energy input to the system, which is a function of the frequency. The lower left portion of the graph does indicate that the 20 Hz test reached the stimulated viscosity slightly before the 15 Hz test which in turn reached its stimulated viscosity slightly before the 10 Hz test. No theories are put forward as to why the 5 Hz test plateau was higher than the others.

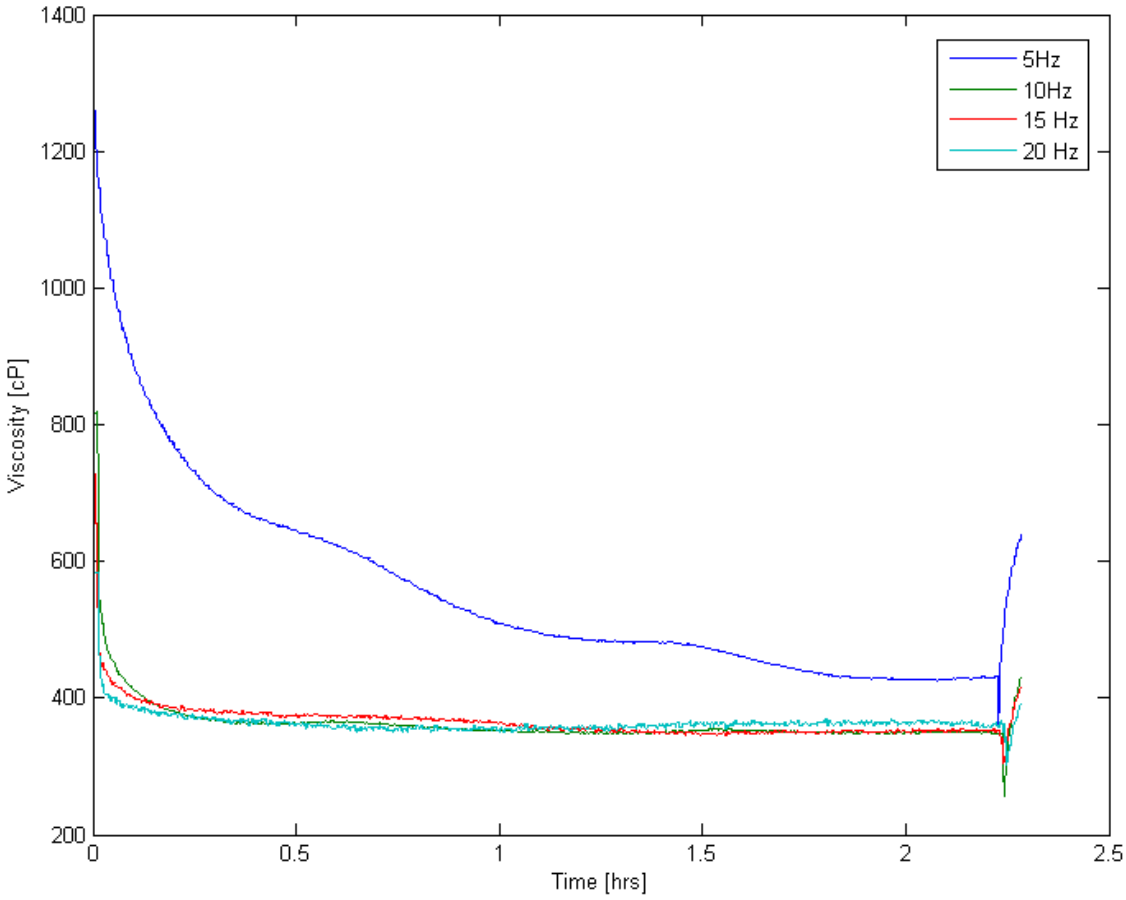


Figure 42 - Time series plots of a bentonite sample stimulated at ±200 psi at 5, 10, 15, and 20 Hz

The final viscosity drop identified on Figure 41 was present in all bentonite trials and occurred immediately after excitation was ceased. It was suggested that this final drop might be caused by the piston returning to its neutral position upon termination of the acoustic excitation cycle. Figure 44 shows an example of the viscosity, temperature, and pressure within the cylinder in the brief period following an acoustic excitation cycle. As expected, there is a marked decrease in chamber pressure as the piston is withdrawn to its neutral position which in turn results in a sharp drop in viscosity.

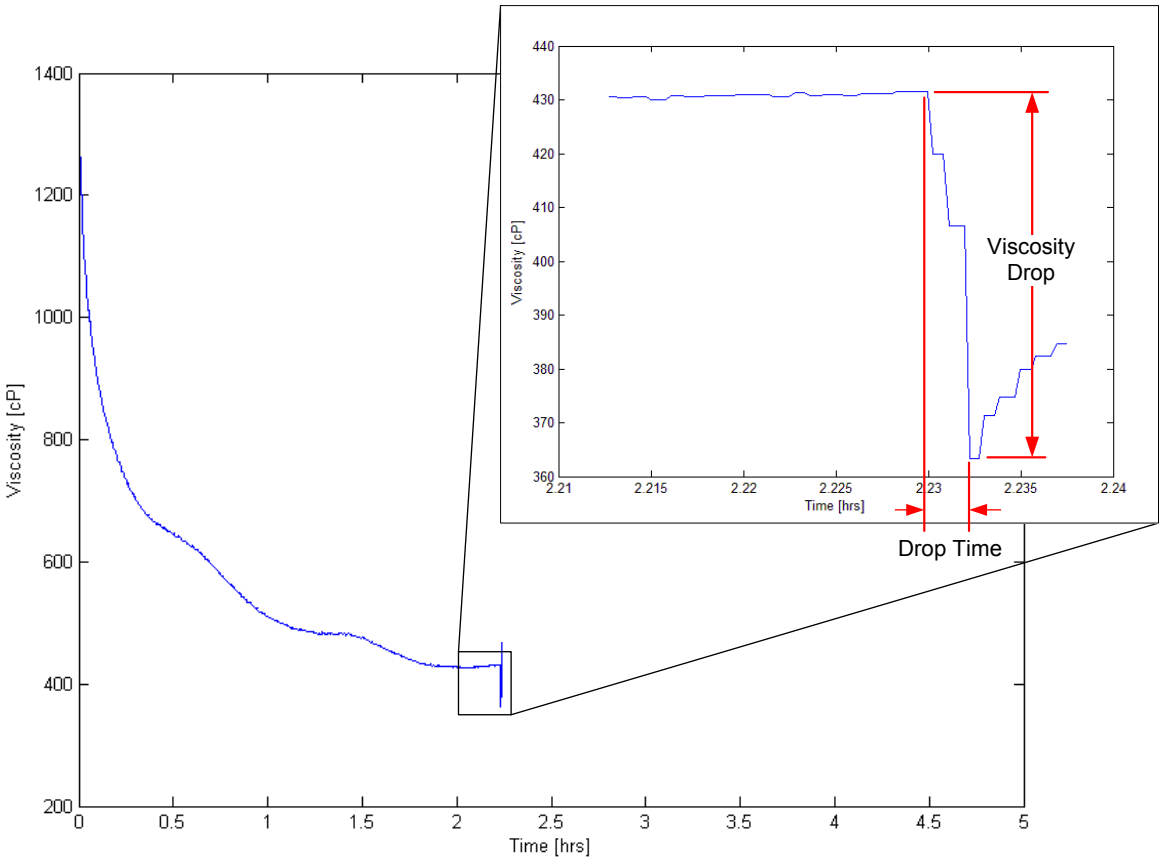


Figure 43 - Enlarged graph of a final viscosity drop illustrating how viscosity drop magnitude and drop time are measured

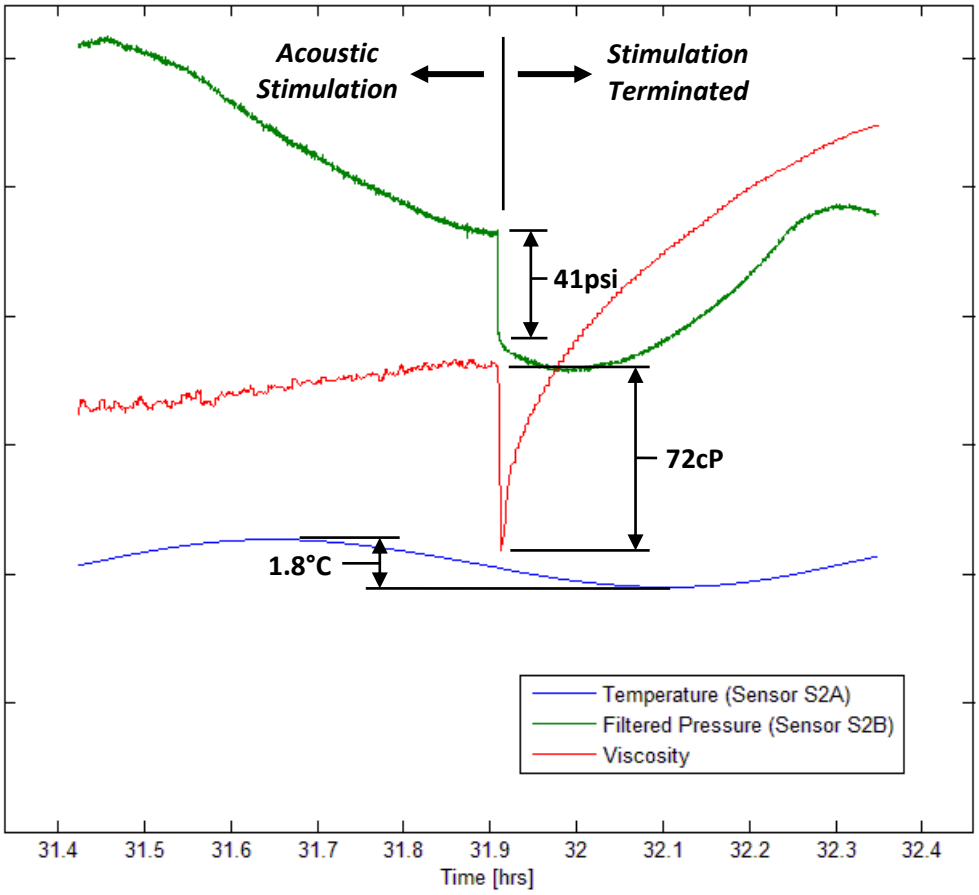


Figure 44 - Plot showing the magnitude and duration of a viscosity and pressure drop observed immediately following the termination of stimulation in a bentonite slurry (±200 psi @ 15Hz).

Another test was performed on a bentonite sample, this time designed to ascertain whether the primary viscosity drops were absolute or relative in magnitude (i.e. whether or not the viscosity at the onset of stimulation affected the magnitude of the stimulated viscosity). A 10% mass concentration bentonite sample was stimulated at ±400 psi at 5 Hz until the stimulated viscosity could be estimated. Stimulation was then stopped for a brief time allowing only a partial viscous recovery to take place before stimulation was restarted. This process was repeated three times and yielded the results shown in Figure 45.

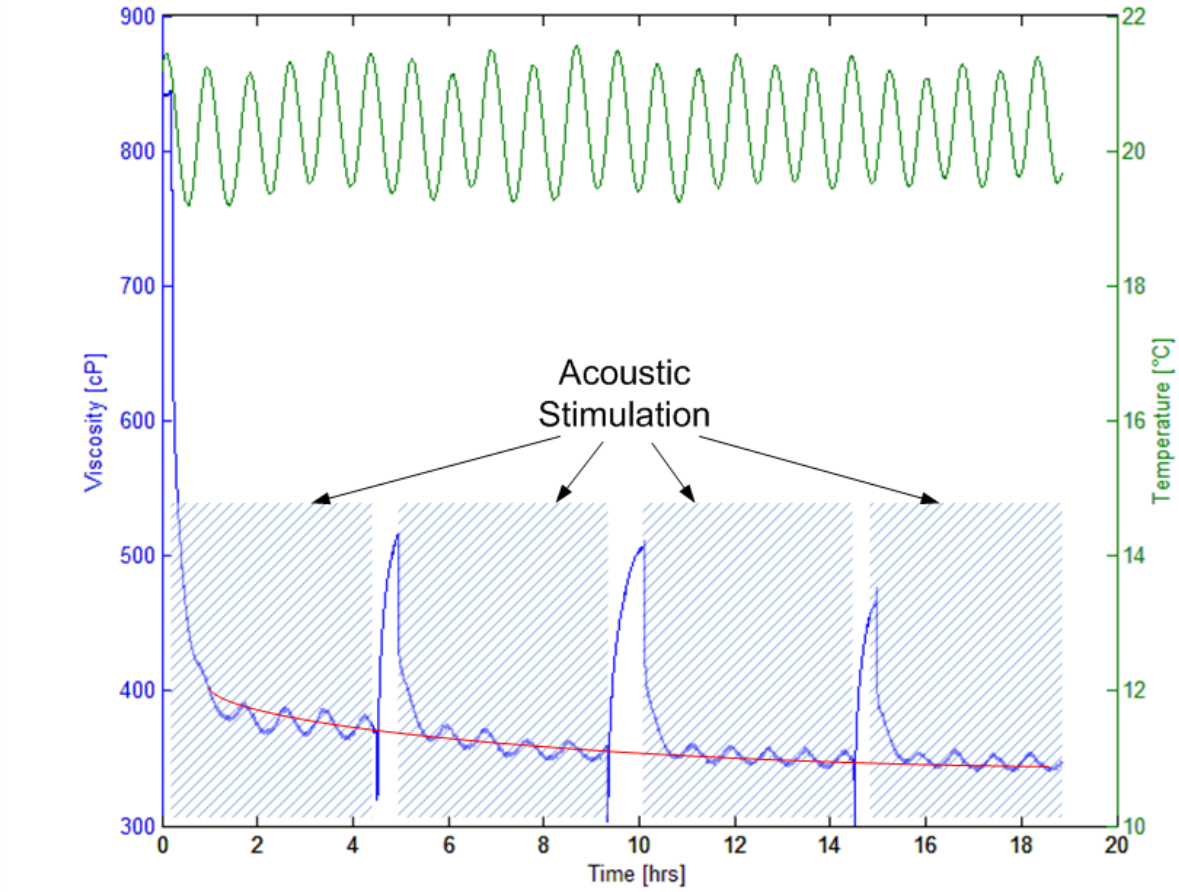


Figure 45 - Time series plot of a bentonite sample illustrating how viscosity drops to a minimum value irrespective of whether recovery is allowed to complete

The blue line represents the viscosity logged over the duration of the experiment and the red line is included to help illustrate the asymptotic approach to a final stimulated viscosity value. The start and termination of each stimulation cycle are marked on the figure. The reader may also observe how the shape of each stimulation cycle and the final viscosity drops observed upon termination of stimulation are similar to those previously discussed in Figure 41.

As can be seen in Figure 45, stimulating the bentonite sample before viscous recovery could complete had no effect on the magnitude of the stimulated viscosity. Each stimulation period approached the same asymptotic stimulated viscosity value. It can therefore be concluded that

there is a minimum stimulated viscosity magnitude that is reached over time regardless of the stimulation history of the bentonite.

It was desired to better understand the mechanism by which changes in bentonite viscosity took place. It had been observed that the consistency of freshly mixed bentonite samples changed dramatically over the first 24 hours, becoming much more gelatinous over time, in a similar way to how stimulated samples underwent thixotropic recovery. It was therefore postulated that the observed changes in viscosity measured during the acoustic excitation experiments may have been caused by a disintegration of the gel structure during stimulation. A thorough investigation of the bentonite gel structure during and after stimulation was beyond the scope of this study however a comparison of thixotropic recovery data and viscosity data from a newly mixed sample was done as an initial test of this theory.

Figure 46 plots two viscosity series on the same time scale. One is the measured viscosity of a bentonite sample immediately following the cessation of stimulation at ±400 psi at 5 Hz. The other is the viscosity of a bentonite sample of a slightly higher mass concentration (16% vs 13%) 10 minutes after it was first mixed. The temperature of both samples was maintained at 20°C. As the figure shows, the viscosity of both samples is almost the same at the onset of the measurements. The viscosity of both samples then gradually increased over time, eventually reaching their respective asymptotic viscosity values. An ideal comparison would have used samples of exactly the same mass concentration however only this data was available at the time of writing. Despite this fact, it is obvious that the curves have a similar shape. If the newly mixed sample were of a lower mass concentration (i.e. 13%), it is conceivable that its viscosity curve would be closer to overlapping that of the recently stimulated sample since it would approach a lower asymptotic viscosity value. Further experimentation observing such conditions is therefore warranted in a thorough investigation of the mechanism of viscosity change in bentonite.

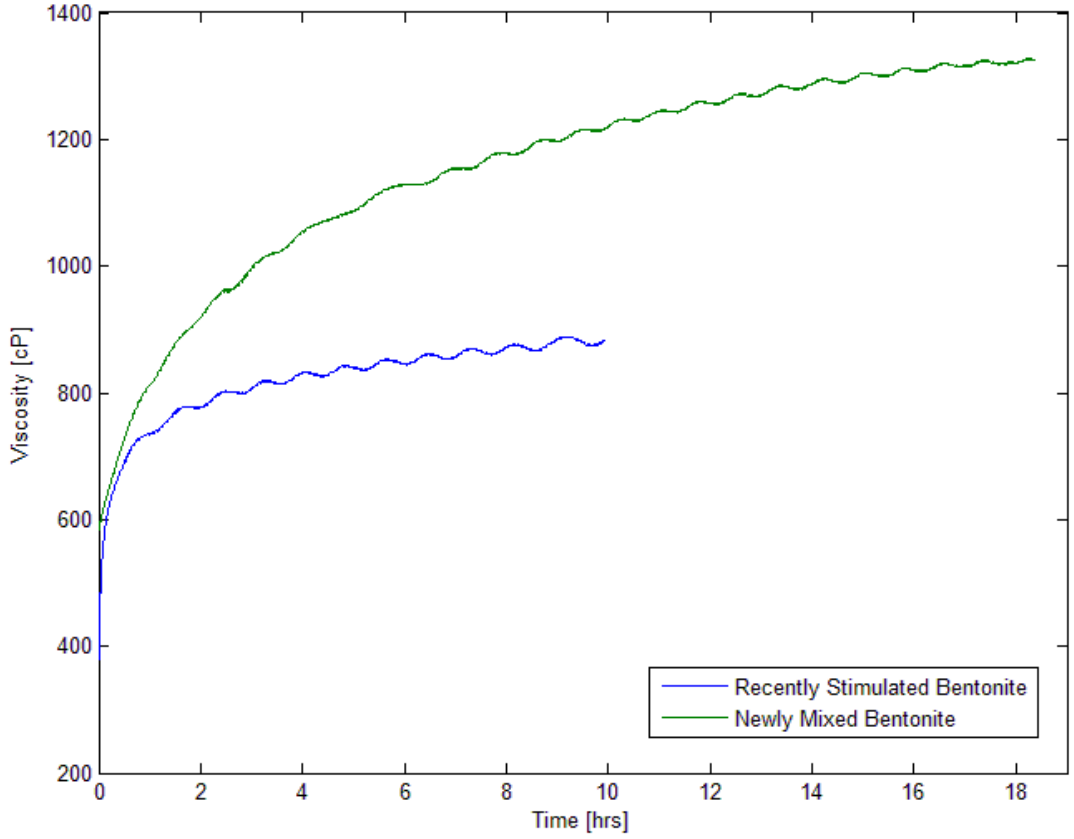


Figure 46 - Graph showing the thixotropic recovery of a recently stimulated bentonite sample against the viscosity of a newly mixed bentonite sample

It is worth noting that the unstimulated viscosity values measured for bentonite using the test chamber were quite different from those quoted in industry at similar temperatures and concentration. Perfect data comparisons were not available but in one rough comparison, a 10% concentration sample at 20°C was estimated to be around 50cP in industry whilst a 13% concentration measured 1200cP in the test chamber. As Figure 32 showed, measurements taken over a wide range of shear rates yielded a viscosity range stretching several orders of magnitude so it is hypothesized that this difference may be the result of measuring at different shear rates (lower shear rate in the test chamber). More details of the industry experiments would be required to confirm this.

5.4 Bitumen

5.4.1 Amplitude/Frequency Plot

The amplitude frequency plot for bitumen is shown in Figure 47. There is a relatively large change in the viscosity of bitumen over 0.25°C as indicated by the distance between the horizontal dashed black line and the solid black line. Similar to the case for the N2500 calibration standard one can see that for all acoustic excitation amplitudes and frequencies tested, the viscosity points were located within the measurably insignificant region between the lower threshold line and the baseline. This indicated that at the amplitudes and frequencies tested, acoustic excitation had a negligible effect on the viscosity of bitumen.

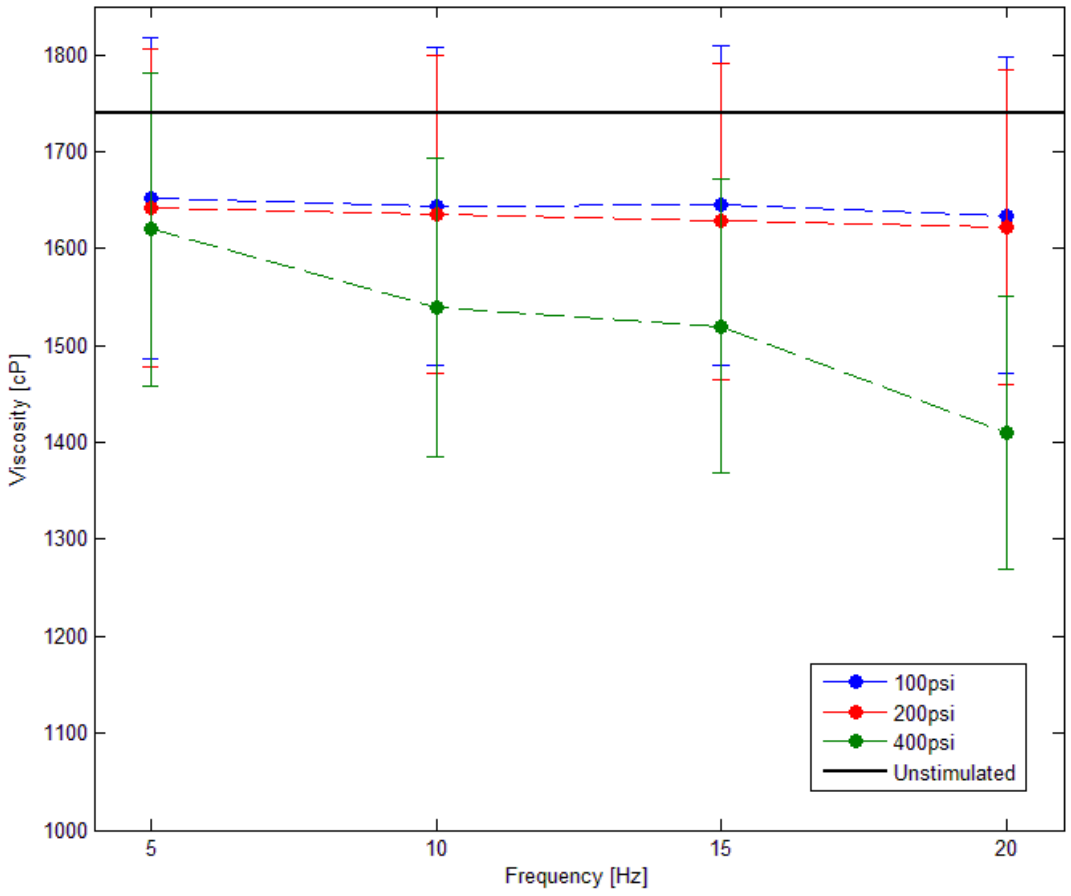


Figure 47 - Amplitude frequency plot for bitumen at 80°C

This result is discouraging for potential in-situ production strategies since no significant change in bitumen viscosity was detected. Further testing at a wide range of amplitudes and frequencies would be required to make a concrete statement on the potential usefulness of acoustic excitation as a production technology however judging from the data collected in this study, the chemical and thermal production technologies discussed in the introduction remain the best in-situ methods for reducing the viscosity of the bitumen fraction in oil sand reservoirs.

5.5 Cornstarch and Oil Sand

In order to meet the deadlines of the industrial sponsor, acoustic excitation experiments on oil sand were accelerated ahead of the cornstarch experiments. As a result of damage that occurred to the viscometer and RTDs during the oil sand experiments and the time and cost involved in sending the device out for repair, the cornstarch investigation was discontinued. It is instead left as point for future study. Appendix E provides a more thorough explanation of how the granular nature of the oil sand resulted in the damage to the viscometer and RTDs.

Chapter 6 Conclusions and Recommendations for Future Work

6.1 Conclusions

6.1.1 Experimental Apparatus Development

A novel experimental apparatus was developed which was capable of studying the effects of acoustic excitation on the viscosity of an enclosed fluid sample. The apparatus was capable of:

- Simulating the temperature and static pressure conditions present in an intermediate depth oil sand reservoir with temperatures ranging from -20 to 95°C and static pressures ranging from 0 to 1500 psi.
- Measuring the temperature and pressure at an array of locations in and around the test sample.
- Measuring the infinite shear viscosity at the centre of the test sample.
- Subjecting the test sample to one-dimensional acoustic excitation at amplitudes ranging from 0 to 400psi and frequencies from 5 to 20 Hz.
- Running parametric experiments whilst logging sensor data in near real-time and populating graphs on a custom design control program with a graphical user interface.

In addition to the physical apparatus and control software, a custom Matlab script was written for post-processing the experimental data. This script was responsible for:

- Applying calibration coefficients to raw experimental data and density correction factors to viscosity data
- Parsing and analyzing the experimental data to detect regions of interest
- Plotting the processed experimental data on customized graphs to reveal trends

Several calibration experiments were performed to ensure the integrity of the data being collected. Sensors were calibrated as follows:

- RTDs were calibrated using a factory-calibrated temperature controlled water bath. Sensors were suspended in the bath and incrementally subjected to prescribed temperatures. Calibration coefficients were calculated using the National Instruments MAX calibration utility.
- Pressure transducers were calibrated using a factory-calibrated air pressure controller. Sensors were installed in the test chamber and incrementally subjected to prescribed static pressures. Again, calibration coefficients were calculated using the National Instruments MAX calibration utility.
- The viscometer was calibrated using a NIST traceable viscosity standard (N2500). The viscometer was suspended in the fluid at atmospheric pressure and the temperature was adjusted incrementally such that the viscosity of the fluid varied over the measurement range of the viscometer.

6.1.2 Experimental Results

Acoustic excitation experiments were performed on a variety of test fluids. The following observations were derived from the resulting data:

- Acoustic excitation at a magnitude of 0 to 400psi and at frequencies of 5 to 20 Hz did not have a measurable effect on the viscosity of N2500 or bitumen
    - Acoustic excitation under these conditions is therefore not suitable for reducing the viscosity of the bitumen fraction in an intermediate depth oil sand production reservoir
- Acoustic excitation under the same conditions had a measurable effect on the viscosity of bentonite and water slurry (tested at a 13% bentonite mass concentration)

- Increased acoustic excitation amplitude resulted in decreased bentonite slurry viscosity (negative correlation).
    - The magnitude of this effect appeared to be asymptotic with the difference in stimulated viscosity between 100 and 200psi being greater than the difference in stimulated viscosity between 200 and 400psi. This echoes the viscous behavior reported in (Ariadji, 2005)
- In the range of 5 to 20Hz, acoustic excitation frequency did not have a measurable effect on the viscosity of bentonite slurries tested (no correlation). This contrasts the results reported in (Ariadji, 2005) where it was observed that excitation frequency had a measurable impact on viscosity. This difference could be the result of the testing being performed on different fluids.
- Changes in bentonite slurry viscosity due to acoustic excitation were time dependent
    - Large viscosity reductions occurred immediately after the onset of the acoustic excitation. Initial viscosity reductions as high as 20% per minute were observed.
    - Viscosity asymptotically approached a minimum value over time. Total viscosity reductions as high as 75% were observed.
    - Immediately following the termination of acoustic excitation, bentonite viscosity dropped sharply from the asymptotic value. The magnitude and duration of this final viscosity reduction increased with increased acoustic excitation amplitude (positive correlation)
    - Thixotropic behavior was observed once acoustic excitation was terminated
- Subjecting bentonite samples to acoustic excitation before thixotropic recovery could complete had no effect on the magnitude of the asymptotic viscosity. Viscosity reductions due to acoustic excitation are therefore independent of the initial viscosity of the bentonite slurry.

6.2 Recommendations for Future Work

As a result of the damage that occurred to the viscometer during the oil sand experiments and time constraints on the project, a number of experiments could not be performed. The following are left as suggested future exercises for persons wishing to continue this work:

- Continue the study of the effects of acoustic excitation on the viscosity of bentonite slurries. Specifically:
    - Performing acoustic excitation experiments at lower frequencies (0 to 5 Hz) may yield a cutoff frequency where stimulation ceases to be acoustic and is instead quasi-static in nature. In place of the large viscosity reductions observed during acoustic excitation, relatively small changes in viscosity are expected under quasi-static pressure loading so the behavior around this cutoff frequency is of interest.
    - Studying the final viscosity drop observed after termination of acoustic excitation could yield insights as to the relationship between the magnitude of the acoustic excitation and the magnitude and duration of this final viscosity reduction. A greater resolution of acoustic excitation amplitudes and a more responsive viscometer would be required for such an investigation.
    - Performing similar experiments using a larger number of acoustic excitation amplitudes could provide insight into whether the negative correlation between viscosity and acoustic excitation amplitude is non-linear, as was hinted at by the current experiments.
    - Studying the effects of the acoustic excitation waveform on the viscous response of bentonite slurries (i.e. square, sawtooth, etc. vs sinusoidal waveform) could result in a different dynamic response due to the different stress patterns acting on the particles as they try to settle.
    - Two of the experiments in the current study were discontinued when the viscosity of the bentonite sample failed to undergo complete thixotropic recovery. This permanent reduction in viscosity occurred after extended periods of excitation at ±400psi. Understanding the mechanism behind this permanent viscosity reduction and the conditions required to produce it may be of interest to members of the

drilling industry since they subject bentonite to cyclic pressurization during slurry pumping operations.

- Investigate the effects of acoustic excitation on the viscosity of other fluids. Specifically:
    - Dilatant fluids such as cornstarch and water mixtures.
    - Oil sand. A study using oil sand would require a vane rheometer or some better method for measuring the viscosity of such a course substance. The study would have limited applicability to an in-situ production technology since oil sand at intermediate depths is effectively a solid.

References

1. Agar, J. G., Morgenstern, N. R., & Scott, J. D. (1987). Shear strength and stress-strain behaviour of Athabasca oil sand at elevated temperatures and pressures. *Canadian Geotechnical Journal* , 1-10.
2. Alberta Queen's Printer. (2006). Pressure Equipment Exemption Order. *Safety Codes Act* .
3. Allen, E. W. (2008). Process water treatment in Canada's oil sands industry: I. Target pollutants and treatment objectives. *Journal of Environment Engineering Science , Vol. 7*, pg.123-128.
4. Ariadji, T. (2005). Effect of vibration on rock and fluid properties: on seeking the vibroseismic technology mechanicsms. *Society of Petroleum Engineering International* .
5. Bair, S. S. (2007). *High-Pressure Rheology for Quantitative Elastohydrodynamics* (Vol. First Edition). Oxford: Elsevier Science.
6. Beresnev, I. A., & Johnson, P. A. (1994). Elastic-wave stimulation of oil production: A review of methods and results. *Geophysics* , 1000-1017.
7. Bogolyubov, B. N., & al., e. (2001). Action of powerful seisomo-acoustic radiation on oil-bearing layers. *International symposium on nonlinear acoustics.* Moscow.
8. Chen, H. S., Chen, D. R., Wang, J. D., & Li, Y. J. (2006). Experimental study on the speacial shear thinning process of a kind of non-Newtonian fluid. *Science in China Series E: Technological Sciences* , 138-143.
9. Clark, K. A. (1931). *Patent No. 1,791,797.* United States.
10. Department of Energy. (2008). *Alberta's Oil Sands. Opportunity. Balance.* Government of Alberta.
11. Duhon, R. D., & Campbell, J. M. (1965). The effect of ultrasonic energy on the flow of fluids in porous media. *Annual Eastern Regional Meeting of SPE/AIME.* Charleston.
12. Dusseault, M. B. (1993). Cold production and enhanced oil recovery. *Journal of Petroleum Technology* , 16-18.
13. Fairbanks, H. V., & Chen, W. I. (1971). Ultrasonic acceleration of liquid flow through porous media. *Chemical Engineering Progress Symposium Series* , 108-116.
14. Fox, R. W., McDonald, A. T., & Pritchard, P. J. (2006). *Introduction to Fluid Mechanics.* Hoboken: John Wiley & Sonds Inc.

15. Grim, R. E., & Necip, G. (1978). *Bentonites - Geology, Mineralogy, Properties, and Uses.* Elsevier.
16. Hamida, T., & Babadaglia, T. (2005). Effect of ultrasonic waves on the capillary-imbibition recovery of oil. *Society of Petroleum Engineering International* .
17. Hamida, T., & Babadaglia, T. (2005). Effects of ultrasonic waves on immiscible and miscible displacement in porous media. *Society of Petroleum Engineering International* .
18. Haydn, M. H. (Ed.). (2006). *Developments in Clay Science, Applied Clay Mineralogy - Occurrences, Processing and Applications of Kaolins, Bentonites, Palygorskite-Sepiolite, and Common Clays* (Vol. 2). Elsevier.
19. Heidrick, T., Bilodeau, V., & Godin, M. (2004). *Oil Sands Research Inventory.* Alberta Energy Research Institute.
20. Hirsch, T. (2005). *Treasure in the Sand: An Overview of Alberta's Oil Sands Resources.* Canada West Foundation.
21. Huh, C. (2006). Improved oil recovery by seismic vibration: A preliminary assessment of possible mechanisms. *Society of Petroleum Engineering International* .
22. Isaacs, E. (2005). *Canadian Oil Sands: Development and Future Outlook.* Alberta Energy Research Institute.
23. Kouznetsov, O. L. (1998). Improved oil recovery by application of vibro-energy to waterflooded sandstones. *Journal of Petroleum Science & Engineering* , 191-200.
24. Krishnan, M. D., & Aghijit, P. K. (2010). *Rheology of Complex Fluids.* New York: Springer.
25. Kuznetsov, O. L. (2002). Seismic techniques of enhanced oil recovery: Experimental and field results. *Energy Sources* , 877-889.
26. Malykh, N., Petrov, V., & Sankin, G. (2003). On sonocapillary effect. *World Congress on Ultrasonics.* Paris.
27. McCain, W. D. (1990). *The Properties of Petroleum Fluids.* Tulsa: Pennwell Books.
28. Mehrotra, A. K., & Svrcek, W. Y. (1986). Viscosity of Compressed Athabasca Bitumen. *Canadian Journal of Chemical Engineering* , 844-847.
29. Merkt, F. S., Robert, D., & Deegan, R. D. (2004). Persistent holes in a fluid. *Physical Review Letters* .
30. Mezger, T. G. (2006). *The Rheology Handbook* (2nd Revised Edition ed.). Hannover: Vincentz.

31. Popov, E. P. (1998). *Engineering Mechanics of Solids* (2nd Edition ed.). Upper Saddle River, New Jersey: Prentice Hall.
32. Roberts, P. M. (2005). Laboratory observations of altered porous fluid-flow behavior in Berea sandstone induced by low-frequency dynamic stress stimulation. *Acoustic Physics* , S140-S148.
33. Simonov, V. F. (1996). Results of experimental oilfield study on enhancing oil recovery by vibroseismic method. *Oilfield Development and Production* , 48-52.
34. UBC Department of Forestry. (n.d.). *University of British Columbia*. Retrieved October 5, 2009, from Oil Sands Moratorium: http://courses.forestry.ubc.ca/cons425/Simulations/OilSandsMoratorium/tabid/3557/Default.aspx
35. Zhu, T., Xutao, H., & Vajjha, P. (2005). Downhole harmonic vibration oil-displacement system: A new IOR tool. *Society of Petroleum Engineering Western Regional Meet.* Irvin.

Appendices

Appendix A Pressure Vessel Design

To ensure safe operation when pressurized, a significant amount of time was spent designing the experimental apparatus chamber to withstand the range of internal pressures needed to simulate downhole reservoir conditions. As the first component of the engineering design, the Alberta Boilers Safety Association (ABSA) documentation governing the design and operation of boilers and pressure vessels was consulted extensively to ensure compliance with provincial legislation. The design and operation of the apparatus chamber was exempted from provincial regulations since the device was of a small volume and was to be used for research. With the legislative requirements met, focus was then shifted to the mechanical design of the chamber.

The initial pressure vessel design was based on the thick-walled pressure vessel equations commonly found in strength of materials textbooks such as (Popov, 1998). These gave insights into the radial, axial, and longitudinal components of stress and thus the combined von Mises stress present in the chamber walls. The analysis was then refined to include stress concentrations at the sensor mounting locations as well as a fatigue analysis since the chamber was to be subjected to cyclical pressure loading. After including a conservative factor of safety, the resulting wall thickness was used in the Solidworks CAD model. As the chamber already needed to be a fairly complex geometry with regards to areas of stress concentration, every attempt was made to avoid welding on the pressure bearing walls. For this reason, the CAD model was designed such that the structural component of the chamber would be machined as a single component.

Once the CAD model was fully developed, including the Class 1500 flanged ends of the chamber and the sensor mounting locations, an FEA analysis was performed using Solidworks COSMOS to get a more accurate picture of the stress distribution. The original calculations proved to be quite accurate in this respect and no further modification of the CAD model was required. Regardless of the close agreement between the two analyses, the chamber was still hydro-tested before being

used in acoustic experiments. The figures below show the graphical result of an FEA analysis and the manufacturing of the main chamber in the CNC mill.

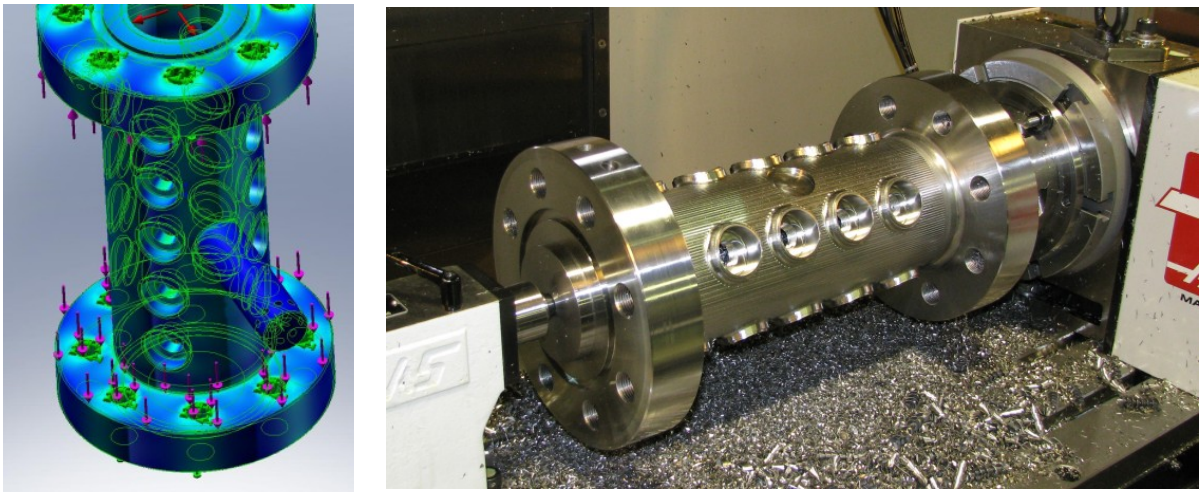


Figure 48- Left: Stress distribution in the chamber structure (from a Solidworks COSMOS FEA internal pressure study). Right: Main body of the chamber being manufactured on the CNC mill (water jacket was later welded on).

Appendix B Thermal Design

It was anticipated that experiments using samples with low thermal diffusivities such as bitumen and oil sand would take several hours to stabilize at the temperature setpoints. In order to gain a better understanding of how long these tests would actually take, a transient heat transfer analysis was performed to estimate the heating times required to bring room temperature samples up to the maximum testing temperature of 80°C.

As a first approximation, heating times were calculated using a radial heat conduction simulation in Solidworks COSMOS. Even distribution of the water jacket inlet and outlet ports around the perimeter of the chamber allowed an assumption of radial symmetry. The analysis also assumed a constant outer wall temperature of 100°C, the maximum temperature of the ethylene glycol mixture in the water bath.

Using bitumen as the worst case scenario with regards to thermal diffusivity, the analysis yielded an upper limit of approximately 8 hours for heating a test sample from 20°C to 80°C. Actual testing using bitumen showed this analysis to be a somewhat conservative estimate since heating times were on the order of 5-6 hours. Since the device outperformed the theoretical predictions, no additional effort was made to explain the discrepancy in the heating times though it was thought that convection within the chamber (which was not included in the simulation) would have contributed to the error.

Appendix C Modal Analysis

Since the test chamber was to be subjected to acoustic excitation, there was a risk that it might inadvertently be stimulated at one of its natural frequencies with potentially dangerous consequences. In order to determine whether or not this was a legitimate concern it was decided to estimate these natural frequencies using a modal analysis.

The modal analysis was performed in a series of key steps for both the radial and longitudinal directions of the test chamber. These consisted of developing material constitutive relationships, generating lumped parameter models, solving the equations of motion, and solving the specific solution based on the actual chamber geometry. The complete analysis was submitted for project credit in the course ENG M 670 – Modeling and Simulation of Engineering Systems in the Spring of 2008. A summary is presented here.

In order to perform the modal analysis, constitutive relationships were needed to relate force to displacement (i.e. $\text{Force} = \text{Stiffness} \times \text{Displacement}$) for the various sections of the test chamber. These relationships were derived from the material properties of the various components. For most sections of the test chamber this involved converting stress-strain relationships to force-displacement relationships while for the bolted connections, joint stiffnesses were used.

Using these stiffnesses, lumped parameter models were developed for both the radial and longitudinal motions of the test chamber. Figure 49 and Figure 50 illustrate the position of the lumped masses as well as the stiffnesses connecting them. Both figures show a radial slice of the test chamber with both top and bottom flanges bolted in place. In the first case, radial motions were assumed axisymmetric so the center vertical axis of the test chamber was treated as a fixed boundary. Similarly, in the second case longitudinal motions were assumed symmetric about the chamber trunnions so the center horizontal axis was treated as a fixed boundary.

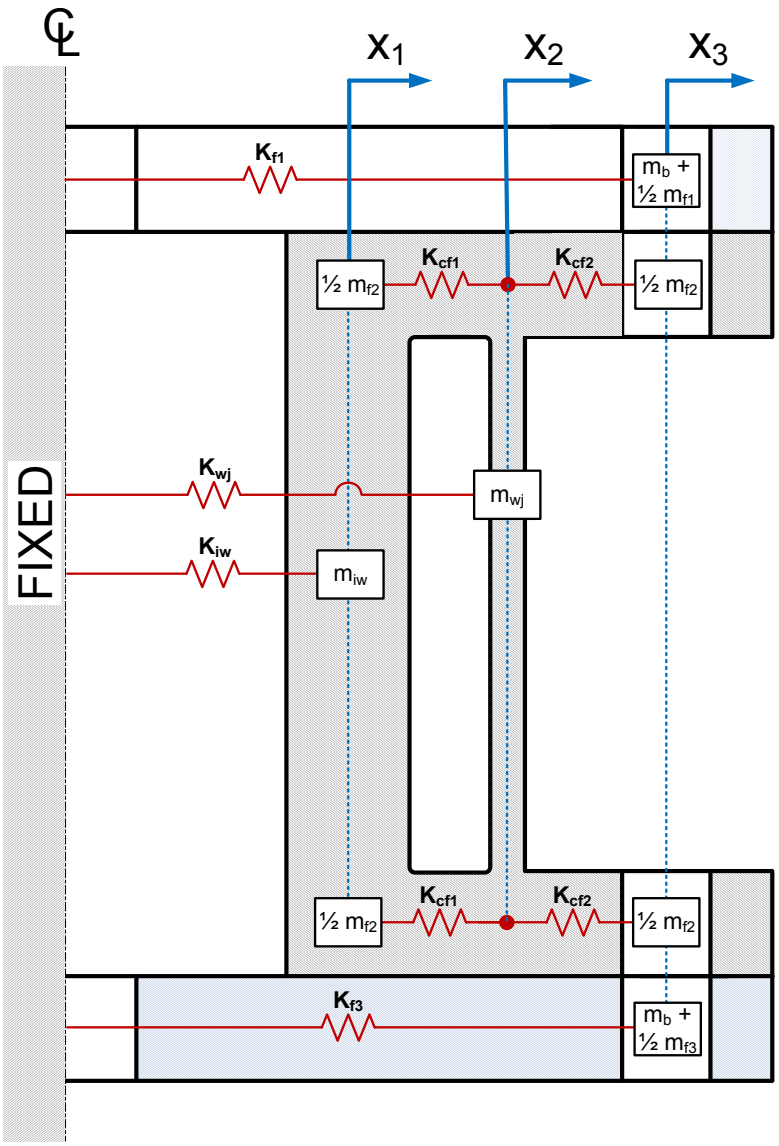


Figure 49 - Lumped parameter model of the radial motion of the test chamber. Note: Sensor ports were not considered in the radial model.

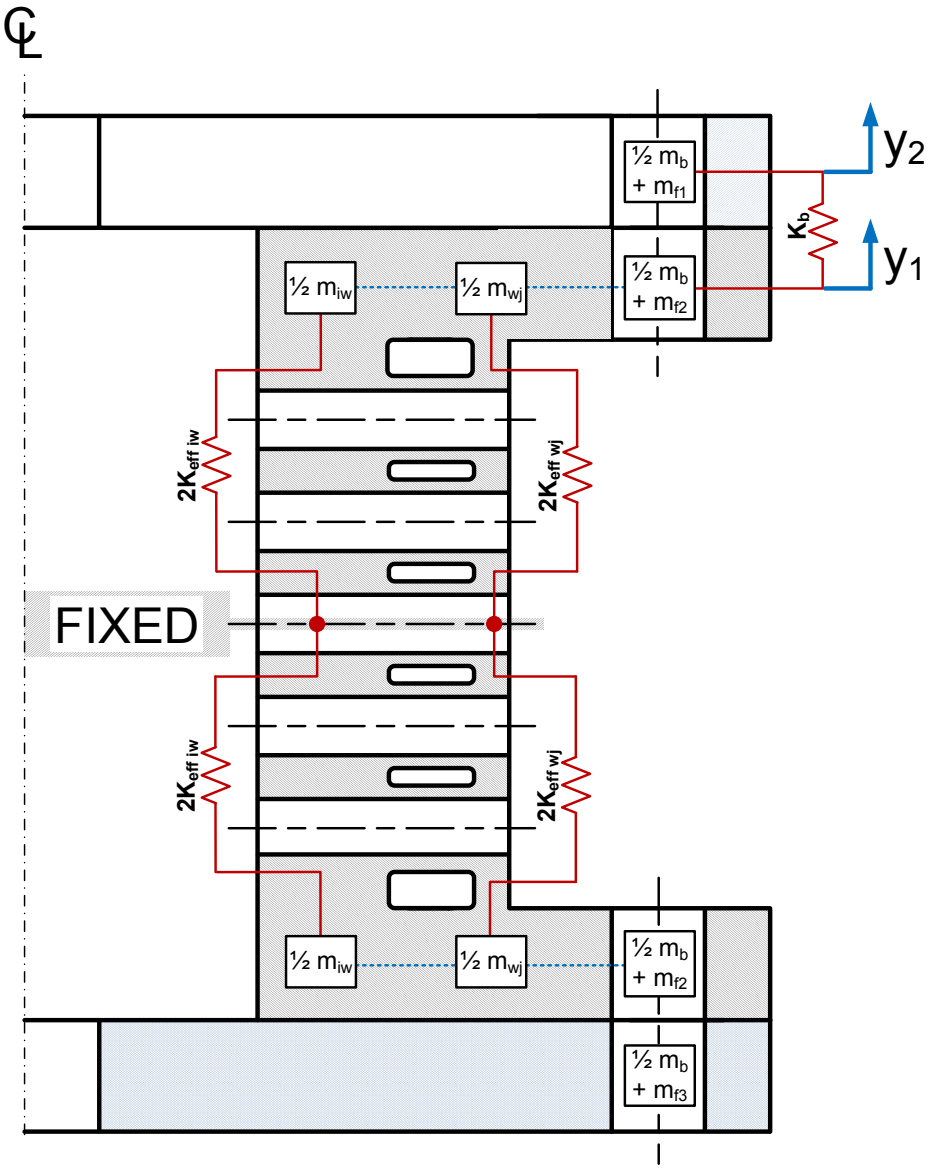


Figure 50 – Lumped parameter model of the longitudinal motion of the test chamber. Note: The water jacket was not considered in the longitudinal model.

These lumped parameter models were converted to network diagrams and in turn to free-body diagrams as shown in Figure 51 and Figure 52. The equations of motion for the lumped masses were derived from these free-body diagrams and solved by assuming an unforced harmonic motion of the masses. The natural frequencies and modes of vibration were extracted from the resulting Eigen value problem using MATLAB.

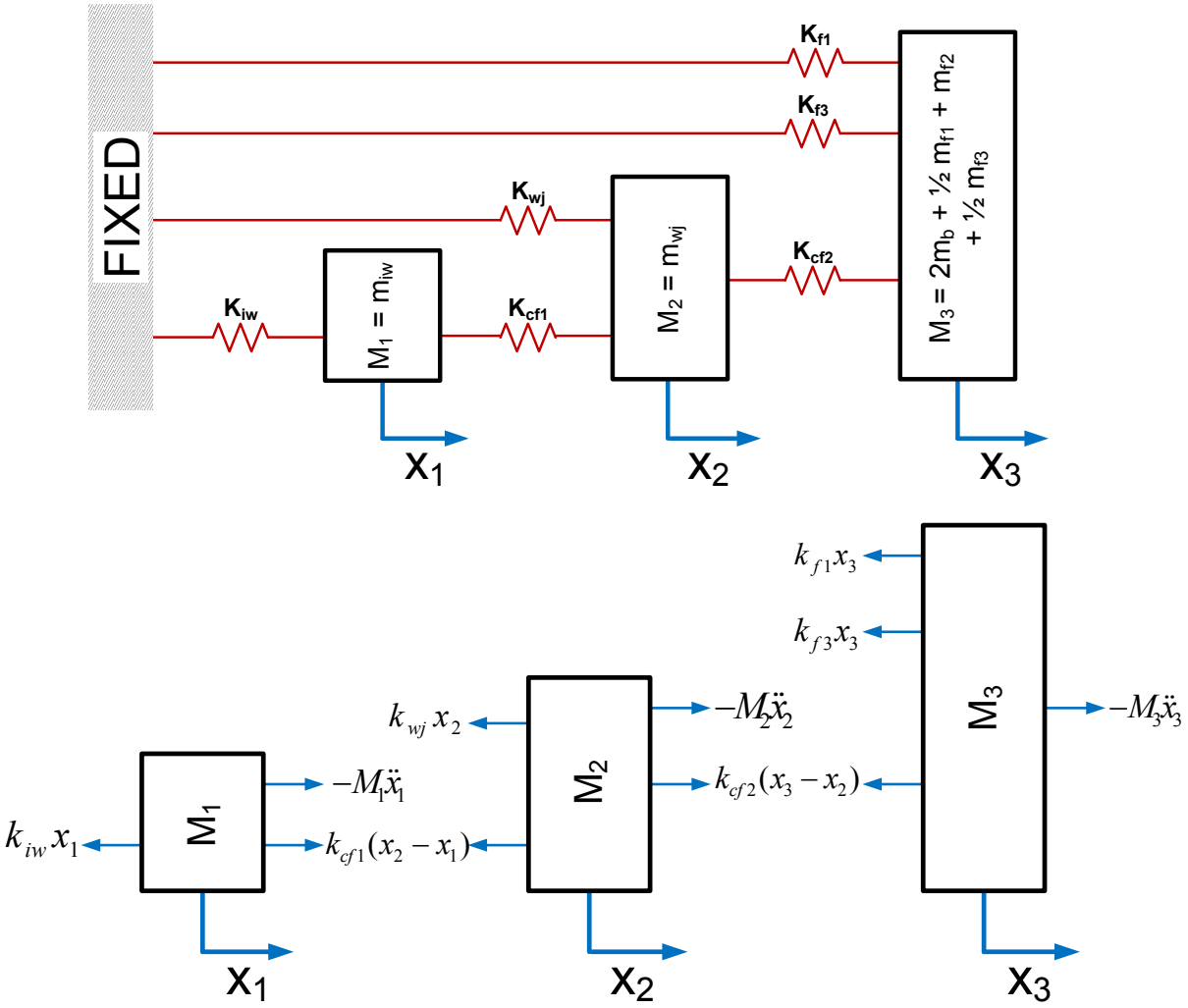


Figure 51 - Network diagram (top) and free-body diagrams (bottom) governing radial motion

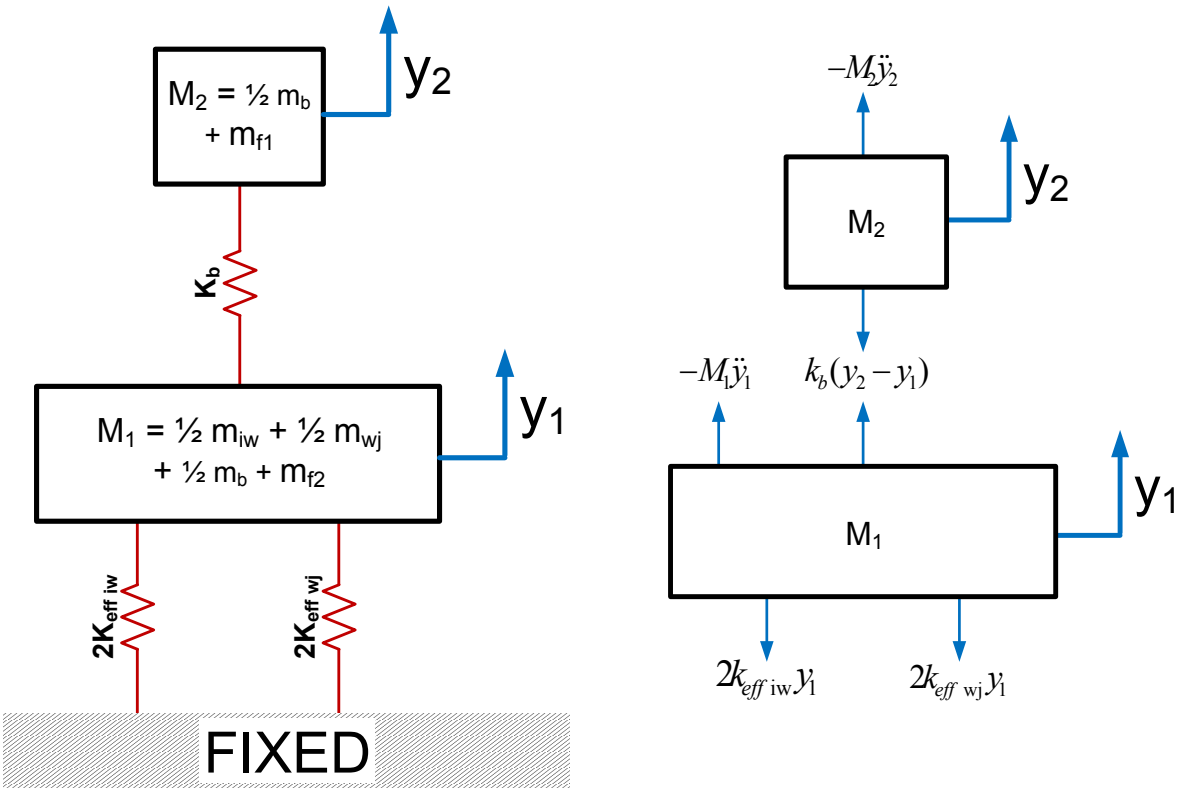


Figure 52 –Network diagram (left) and free-body diagrams (right) governing longitudinal motion

Owing to the relatively high stiffness of the test chamber, the calculated natural frequencies were several orders of magnitude higher than the expected acoustic excitation frequencies (i.e. MHz vs Hz) so it was concluded that resonance was unlikely to be a major concern during operation. Despite this conclusion, the chamber was closely observed for resonant behavior the first time it was subjected to a slow frequency sweep.

Appendix D Acoustic Pressure Amplitude

One of the analyses associated with generating the acoustic pressure was determining the constitutive relationship between volumetric compression and pressure generation in the test fluids. This understanding was essential in sizing the linear actuator piston stroke required for the acoustic excitation experiments.

It was known early on that this relationship would depend largely on the amount of residual gas in the chamber however rather than try to account for this in the analysis, design features were incorporated into the chamber to remove as much of it as possible. The constitutive analysis was thus assumed to be that for a compressed liquid, i.e.:

$$\beta_T = -\frac{1}{V}\left(\frac{\partial V}{\partial p}\right)_T$$

Equation 5 - Isothermal Compressibility Equation for a Compressed Liquid

Where β_T is the isothermal compressibility, V is the volume of fluid, and p is the pressure.

By modifying this partial differential equation to treat finite changes in volume and pressure, a relationship between piston displacement and pressure was developed. This is detailed below.

$$\beta_T = \frac{1}{V_{0_{fluid}}}\left(\frac{V_{0_{fluid}} - V_{fluid}}{p_0 - p}\right)_T$$

Where $V_{0_{fluid}}$ is the initial volume of fluid in the chamber,

V_{fluid} is the compressed fluid volume,

β_T is the isothermal compressibility of the fluid,

and $p_0 - p$ is the gauge pressure of the fluid.

Incorporating the chamber geometry:

$$V_{0_{fluid}} = Area_{chamber} \times Length_{chamber} = \frac{\pi {D_{chamber}}^2}{4} \times L_{chamber}$$

$$V_{fluid} = V_{0_{fluid}} - \ Piston\ Volume\ Displacement$$
$$= \frac{\pi {D_{chamber}}^2}{4} \times L_{chamber} - \frac{\pi {D_{piston}}^2}{4} \times L_{piston\ stroke}$$

Where $L_{chamber}$ and $D_{chamber}$ are the axial length and diameter of the inner wall of the chamber, and $L_{piston\ stroke}$ and D_{piston} are the piston stroke and piston diameter respectively

Therefore by combining and rearranging the above equations and assuming the fluid is initially at atmospheric pressure, the gauge pressure resulting from a given piston displacement is:

$$p = \left(\frac{1}{\beta_T}\right)\left(\frac{\frac{\pi {D_{piston}}^2}{4} \times L_{piston\ stroke}}{\frac{\pi {D_{chamber}}^2}{4} \times L_{chamber}}\right)$$

Equation 6 - Constitutive Relationship between Piston Stroke and Sample Pressure

When this analysis was initially performed it was decided to size the linear actuator by using the isothermal compressibility data for water since it was readily available. For the experimental pressures, this yielded a pairing of a 14mm diameter piston with a 37µm piston stroke. As stated in section 3.3.3.7 this pairing ultimately proved unsuccessful owing to difficulties in removing all of the residual gas in the test chamber. The subsequent use of a much larger piston with a larger stroke compensated for this shortcoming.

Appendix E Description of Damage to the Viscometer

During the chamber pressurization the viscometer began reporting erroneous viscosity data so the equipment was dismantled for investigation. Close inspection revealed that the viscometer shaft was bent during this initial pressurization.

Efforts were made to pack the granular oil sand test sample into the chamber as uniformly as possible before commencing the first of the oil sand acoustic excitation experiments. Despite these precautions however it is theorized that a slight inhomogeneous packing caused an internal lateral flow as the oil sand was redistributed to a homogeneous density under pressure. This internal flow could have applied the lateral force which bent the viscometer shaft.

The schematics in Figure 53 illustrate how inhomogeneous packing prior to pressurization could have resulted in the bending of the viscometer shaft. In the left schematic, the left side of the chamber is more densely packed with oil sand than the right side of the chamber. During pressurization, the internal flow of oil sand would be from left to right so as to have a homogeneous density distribution within the chamber. As the right schematic illustrates, this would cause a lateral bending force on the viscometer bulb (from left to right) resulting in the bending of the viscometer shaft.

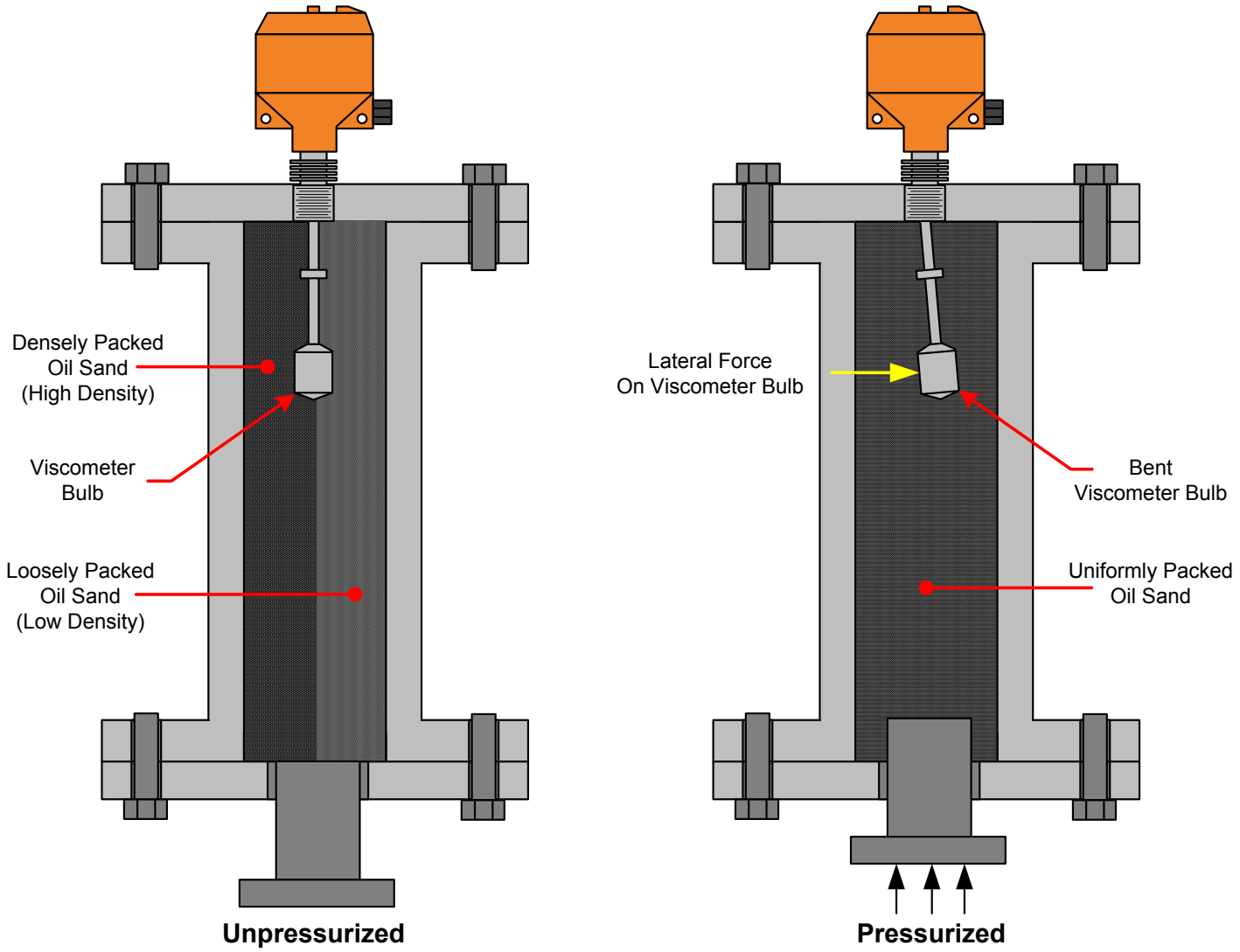
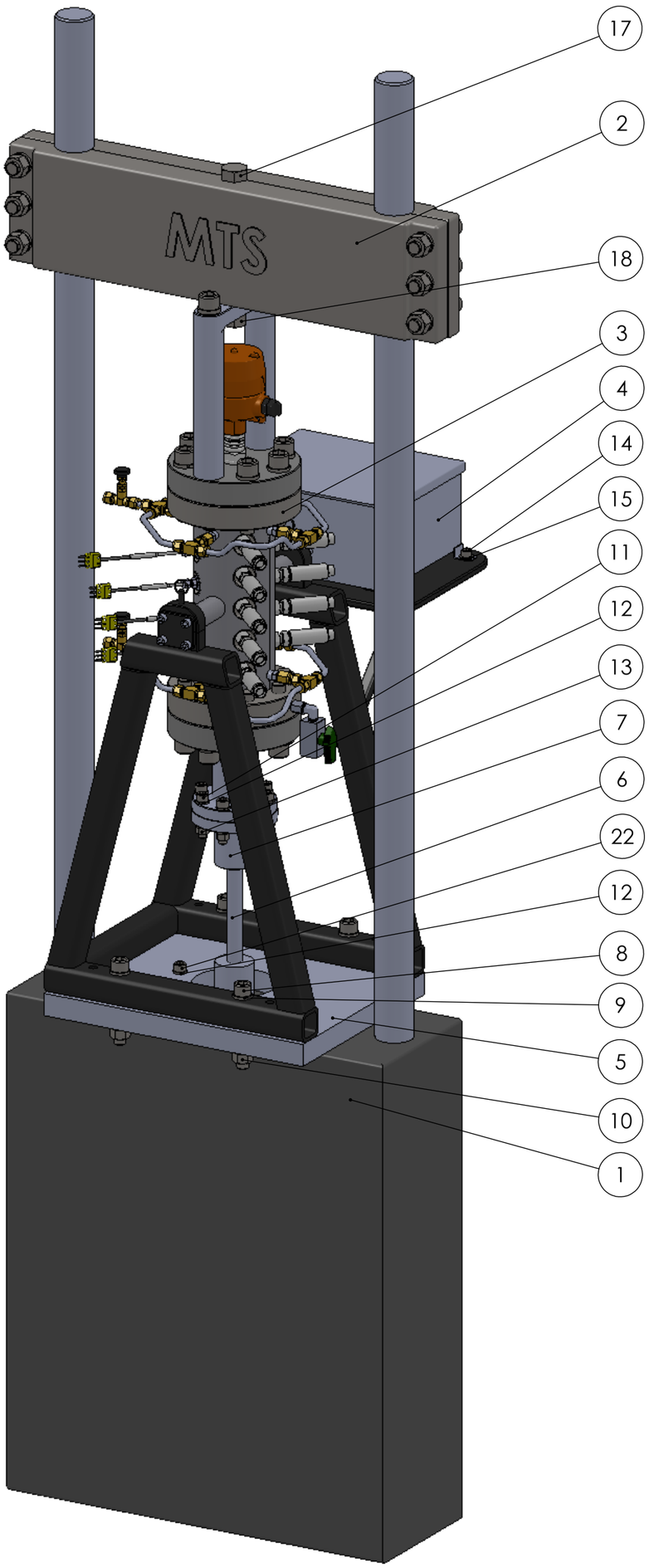


Figure 53 - Schematic diagrams showing inhomogeneous packing of oil sand in the test chamber (left) and material resettling which caused the viscometer to be bent (right)

Appendix F Solidworks Drawings



ITEM NO.	PART NUMBER	QTY.
1	Tensile Testing Machine	1
2	Tensile Testing Machine Cross Head	2
3	Test Chamber Assembly	1
4	Electronics Enclosure	1
5	Mounting Plate	1
6	Threaded Piston Rod	1
7	Hydraulic Piston Coupling	1
8	3/4-10 Bolt	4
9	3/4 Lock Washer	8
10	3/4-10 Nut	4
11	1/2-20 Bolt	8
12	1/2 Lock Washer	22
13	1/2-20 Nut	8
14	7/16-14 Bolt	4
15	7/16 Lock Washer	8
16	7/16-14 Nut (Not Shown)	4
17	1-14 Bolt	1
18	1-14 Nut	1
22	1/2-13 Bolt	6

DIMENSIONS ARE IN INCHES
TOLERANCES UNLESS SPECIFIED:
FRACTIONAL ± 1/32
ANGULAR: MACH ± 1° BEND ± 2°
TWO PLACE DECIMAL ± .010
THREE PLACE DECIMAL ± .005

MATERIAL: VARIOUS

FINISH UNLESS SPECIFIED: –

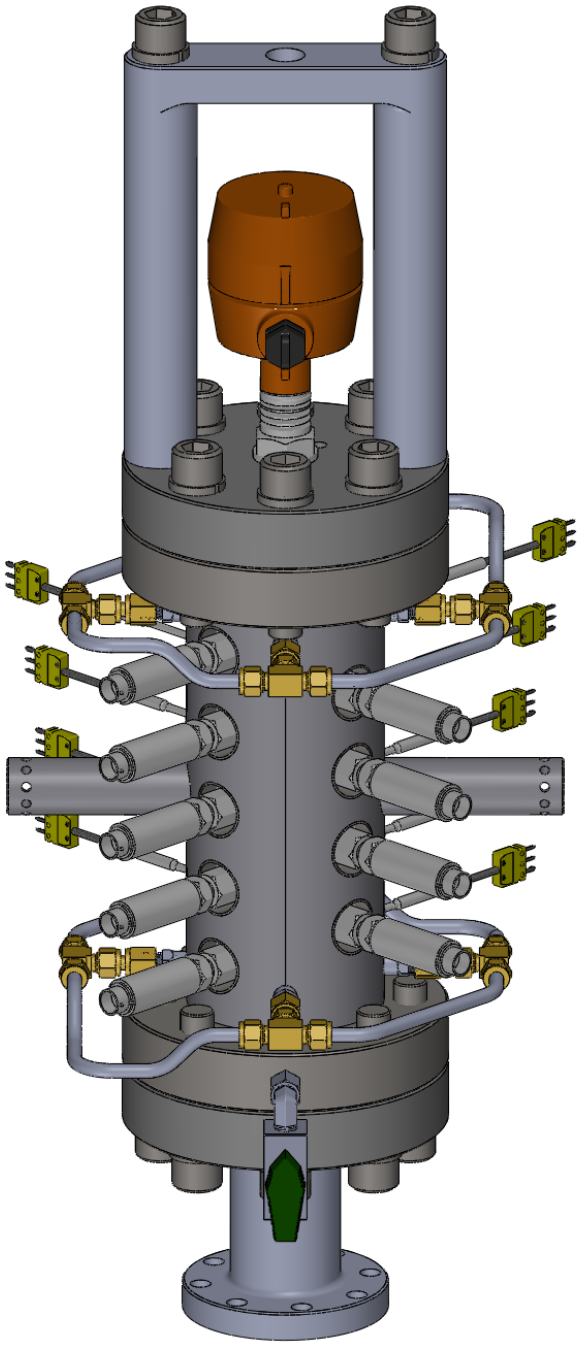
DO NOT SCALE DRAWING

	NAME	DATE	SIGNATURE
DRAWN	M. EVANS	13-JUL-10	
CHECKED			
ENG APPR.			
MFG APPR.			

NOTES:

UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

SIZE: B
DWG. NAME: TENSILE TESTING MACHINE INSTALLATION
SCALE: 1:8
WEIGHT:
SHEET 1 OF 1



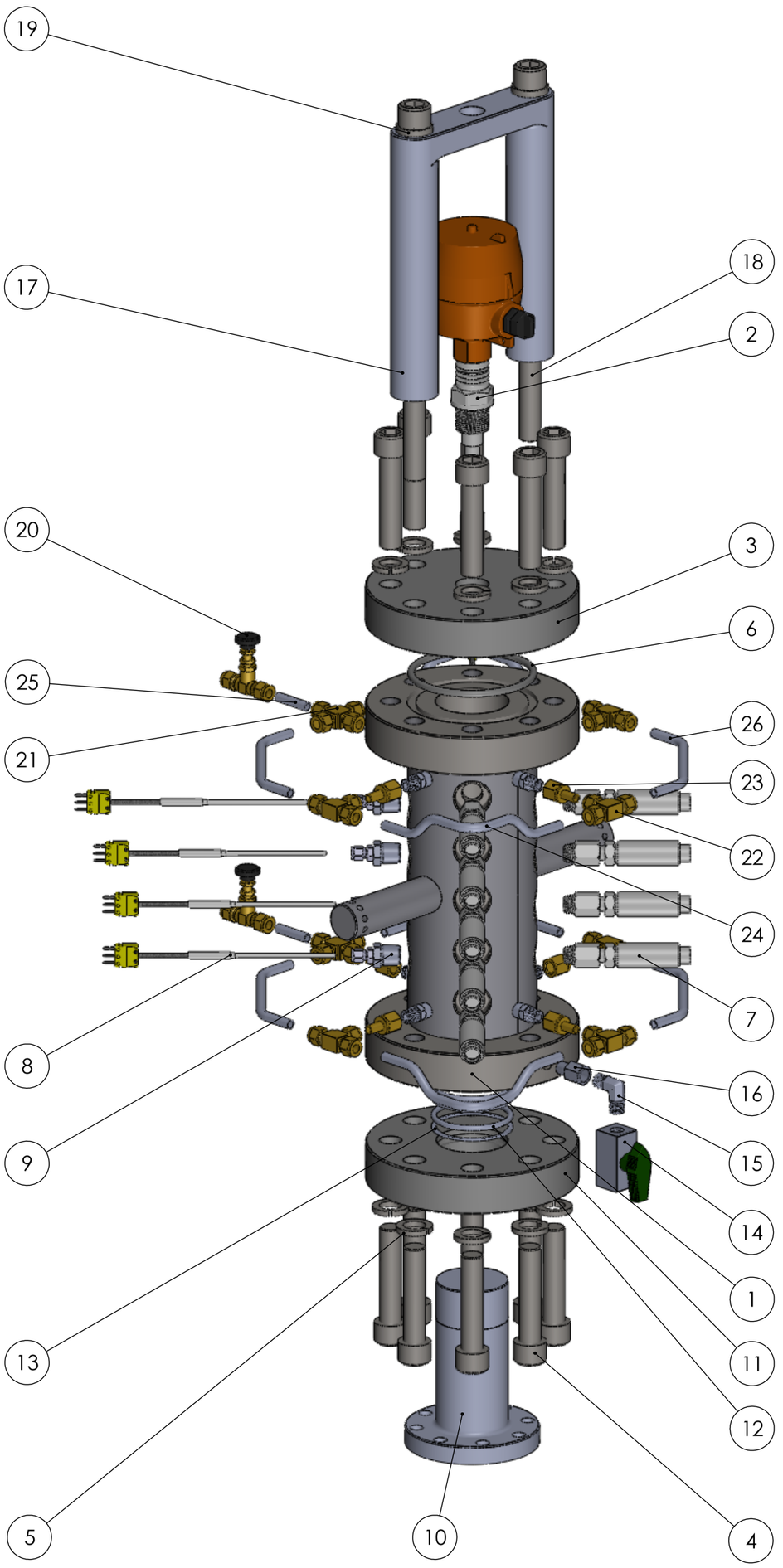
		NAME	DATE	SIGNATURE
DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005	DRAWN	M. EVANS	12-JUL-10	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

MATERIAL: VARIOUS
FINISH UNLESS SPECIFIED: -
DO NOT SCALE DRAWING

NOTES:
1. SHOWN WITH HYDRAULIC PISTON

SIZE	DWG. NO.	
A	ASSEMBLED TEST CHAMBER	
SCALE:1:5	WEIGHT:	SHEET 1 OF 2



ITEM NO.	COMPONENT	QTY.
1	Test Chamber	1
2	Viscometer	1
3	Viscometer Flange	1
4	7/8-14 Flange Bolt	14
5	7/8 Flange Washer	14
6	Flange O-Ring	2
7	Pressure Transducer	9
8	RTD Probe	9
9	Swagelok RTD Fitting	9
10	Hydraulic Piston	1
11	Hydraulic Piston Flange	1
12	Hydraulic Piston O-Ring Parker 2-336	2
13	Hydraulic Piston Backup Ring 2-336	2
14	Priming Valve	1
15	1/4 NPTM 90° Elbow	1
16	1/4 NPTM to 1/4 NPTF Adapter	1
17	Mounting Bracket	1
18	7/8-14 Mounting Bracket Bolt	2
19	7/8 Mounting Bracket Washer	2
20	Swagelok 3/8 Tube Needle Valve	2
21	Swagelok 3/8 Tube Cross Fitting	2
22	Swagelok 3/8 Tube T Fitting	6
23	1/4 NPTM to 3/8 Tube Adapter	8
24	Swagelok 3/8 Tubing	4
25	Swagelok 3/8 Tubing	2
26	Swagelok 3/8 Tubing	4

DIMENSIONS ARE IN INCHES
TOLERANCES UNLESS SPECIFIED:
FRACTIONAL ± 1/32
ANGULAR: MACH ± 1° BEND ± 2°
TWO PLACE DECIMAL ± .010
THREE PLACE DECIMAL ± .005

MATERIAL: VARIOUS

FINISH UNLESS SPECIFIED: –

DO NOT SCALE DRAWING

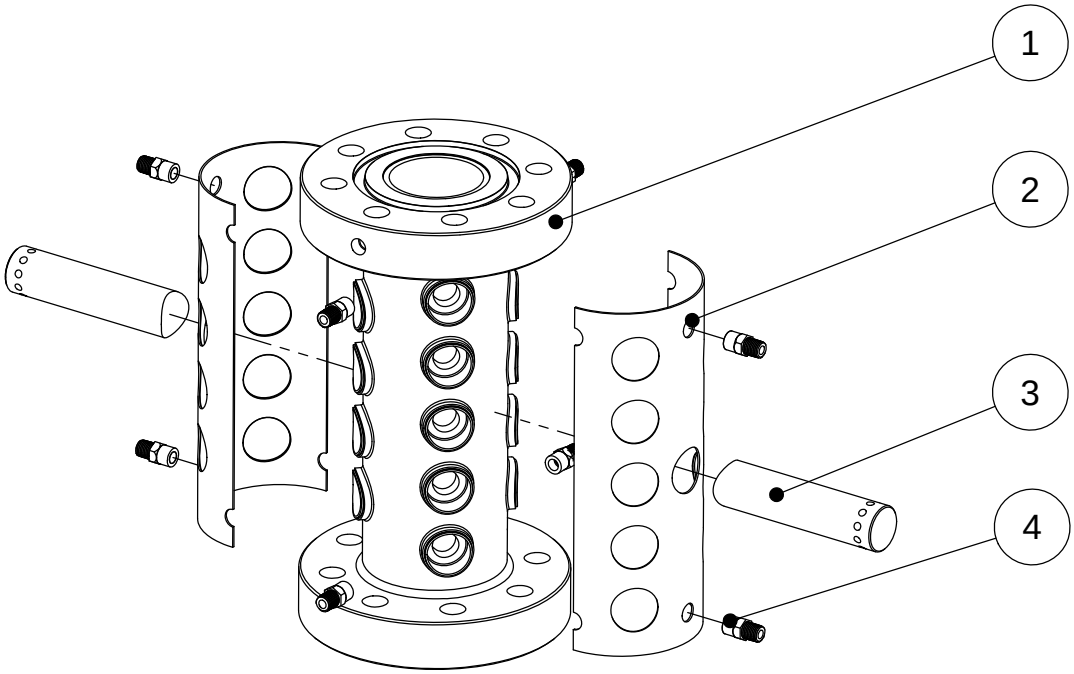
	NAME	DATE	SIGNATURE
DRAWN	M. EVANS	13-JUL-10	
CHECKED			
ENG APPR.			
MFG APPR.			

NOTES:
1. SHOWN HERE WITH HYDRAULIC PISTON

UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

SIZE: B
DWG. NAME: EXPLODED TEST CHAMBER
SCALE: 1:5
WEIGHT:
SHEET 2 OF 2

ITEM NO.	PART NAME	QTY.
1	Cylinder Body	1
2	Water Jacket	2
3	Support Pins	2
4	Water Jacket Welded Ports	8



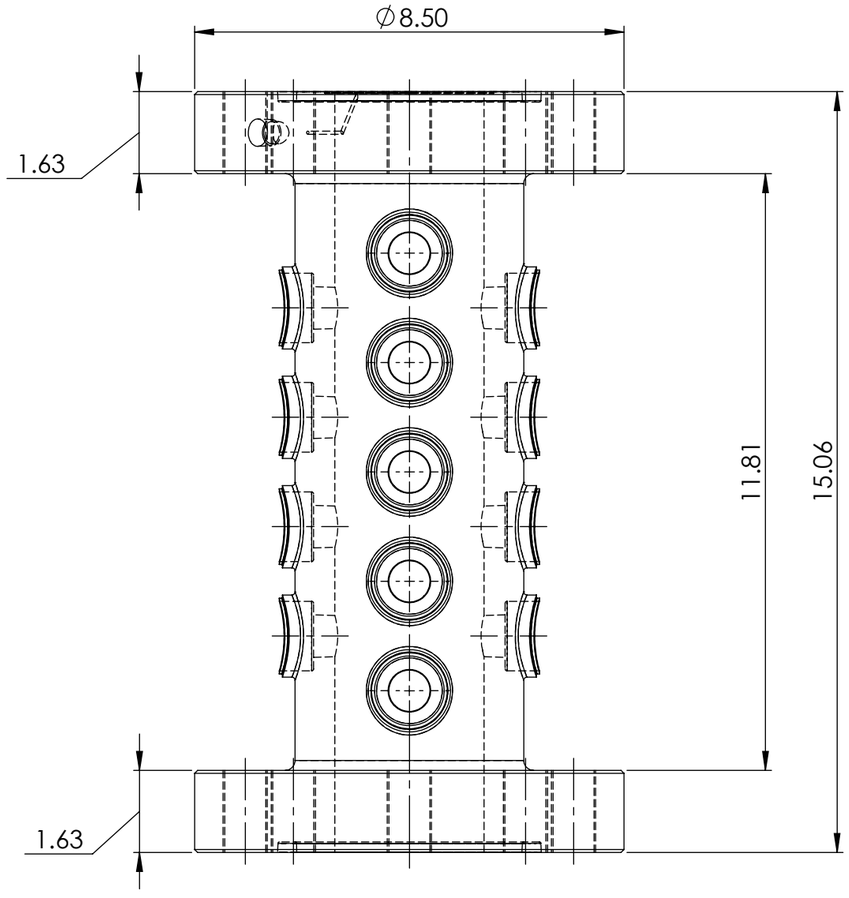
		NAME	DATE	SIGNATURE
DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005	DRAWN	M. EVANS	02-APR-08	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

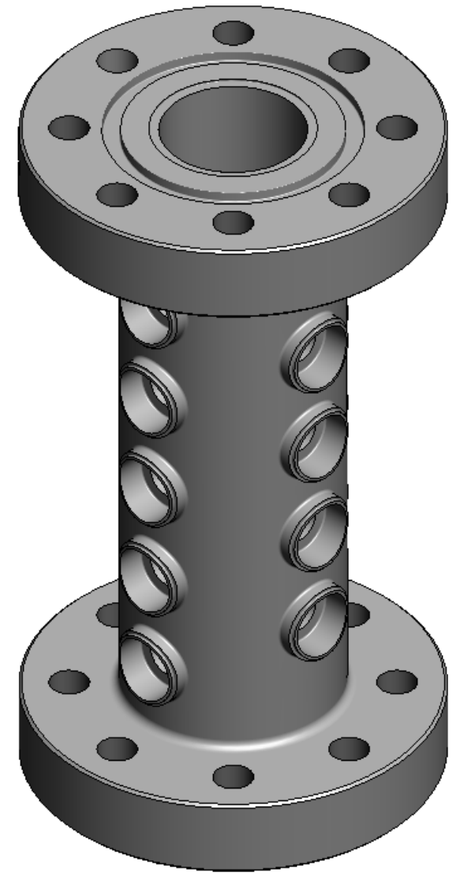
UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

MATERIAL: VARIOUS
FINISH UNLESS SPECIFIED: 63
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES
3. WELD ALL COMPONENTS TOGETHER
4. WELD SUPPORT PINS TO BOTH THE CYLINDER BODY AND WATER JACKET

SIZE	DWG. NAME	
A	TEST CHAMBER MANUFACTURING	
SCALE:1:6	WEIGHT:	SHEET 1 OF 1





PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF THE IMPERIAL OIL CENTRE FOR OIL SANDS INNOVATION (COSI). ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF COSI IS PROHIBITED.

DIMENSIONS ARE IN INCHES
TOLERANCES UNLESS SPECIFIED:
FRACTIONAL ± 1/32
ANGULAR: MACH ± 1° BEND ± 2°
TWO PLACE DECIMAL ± .010
THREE PLACE DECIMAL ± .005

MATERIAL
17-4 STAINLESS STEEL

FINISH UNLESS SPECIFIED
63

DO NOT SCALE DRAWING

	NAME	DATE	SIGNATURE
DRAWN	M. EVANS	22-SEP-08	
CHECKED			
ENG APPR.			
MFG APPR.			

NOTES:

1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

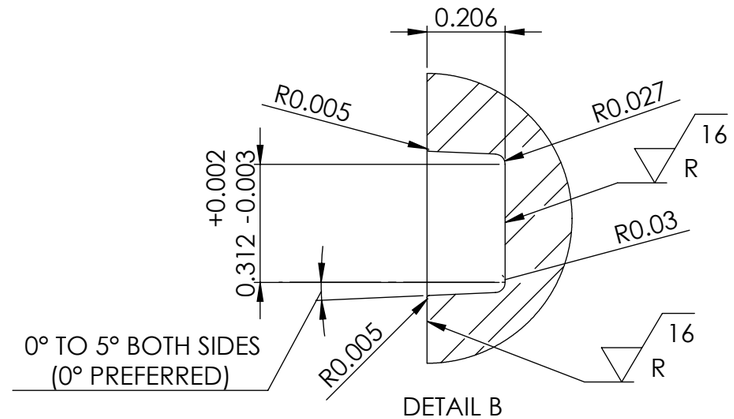
UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

SIZE	DWG. NAME
B	TEST CHAMBER BODY
SCALE:1:3	WEIGHT:

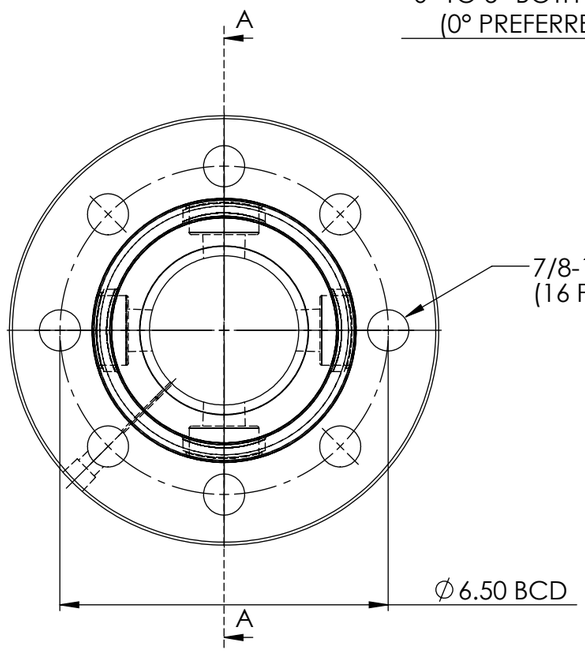
SHEET 1 OF 3

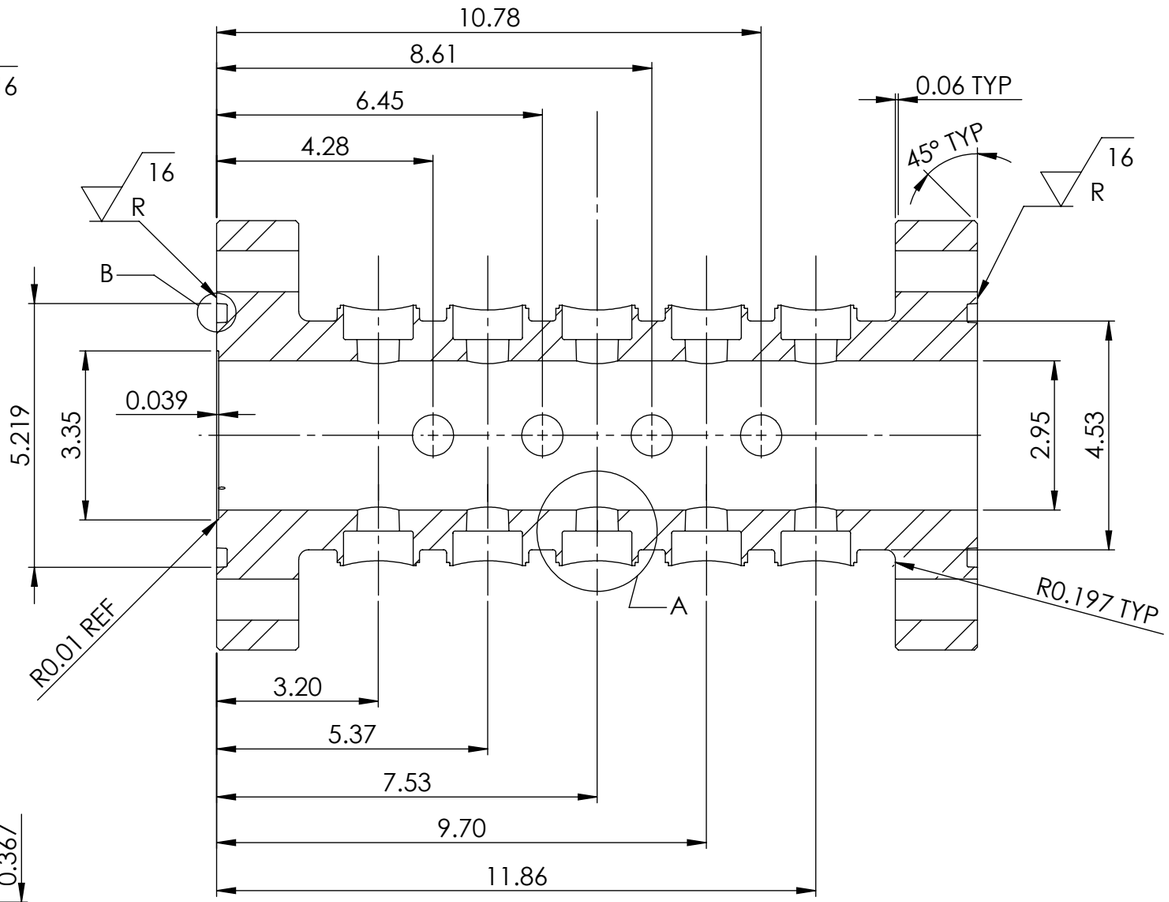
DRAWN	CHECKED	ENG APPR.	MFG APPR.	DATE
M.E.				22-SEP-08

O-RING GLAND (BOTH ENDS)



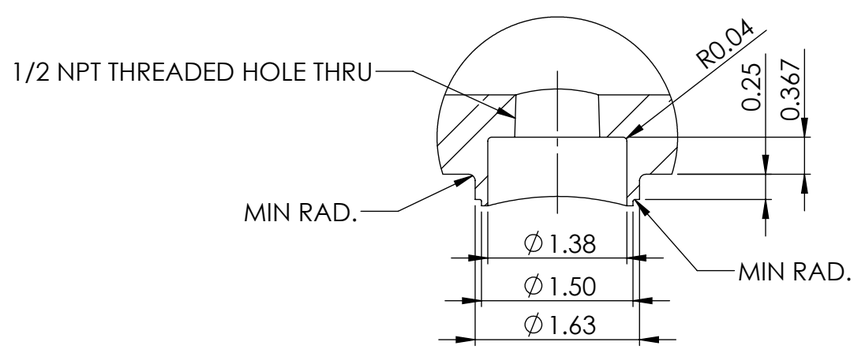
DETAIL B
SCALE 5 : 2





SECTION A-A

SENSOR PORT (18 PLACES)



DETAIL A
SCALE 2 : 3

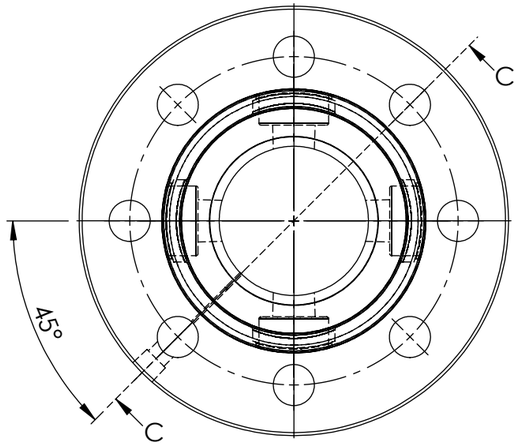
UNIVERSITY OF ALBERTA

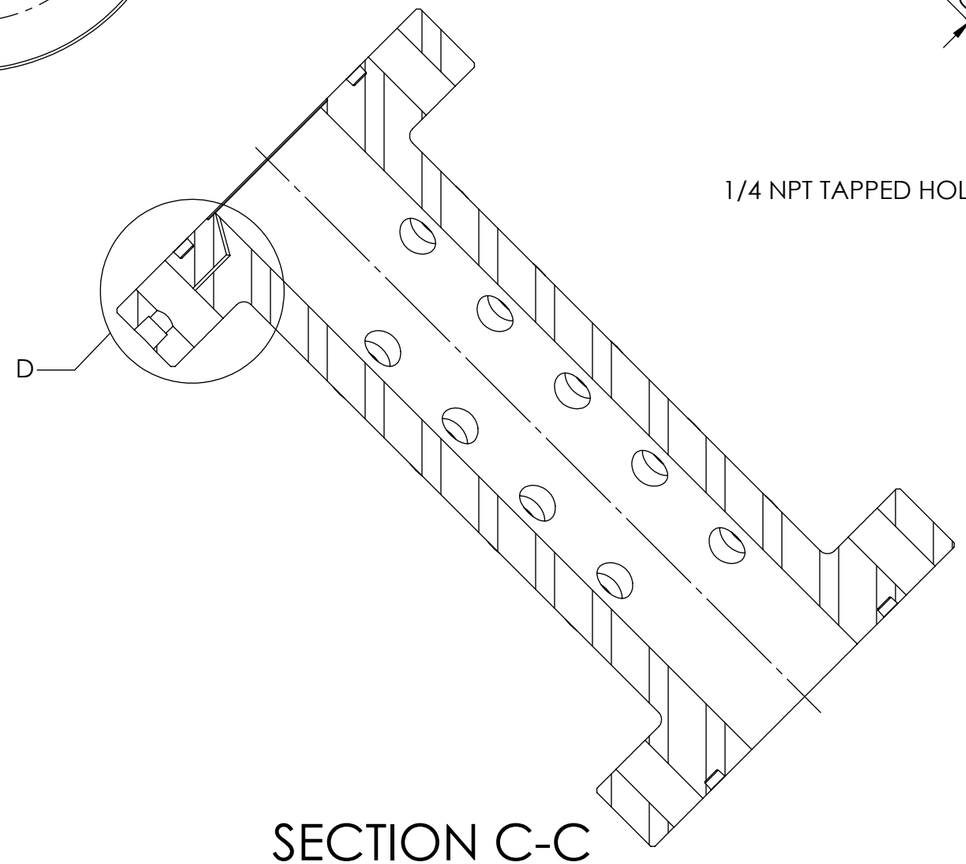
DEPARTMENT OF
MECHANICAL ENGINEERING

4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

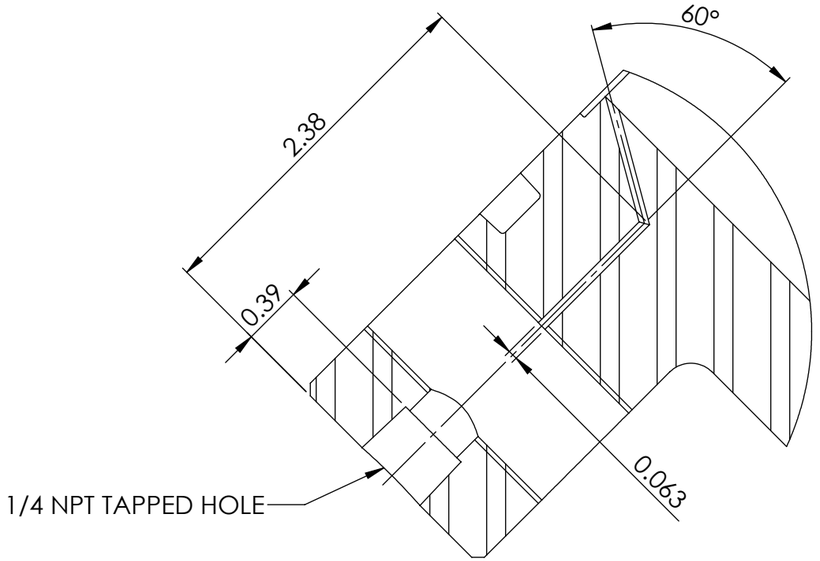
SIZE	DWG. NAME	
B	TEST CHAMBER BODY	
SCALE:1:3	WEIGHT:	SHEET 2 OF 3

DRAWN	CHECKED	ENG APPR.	MFG APPR.	DATE
M.E.				22-SEP-08





SECTION C-C



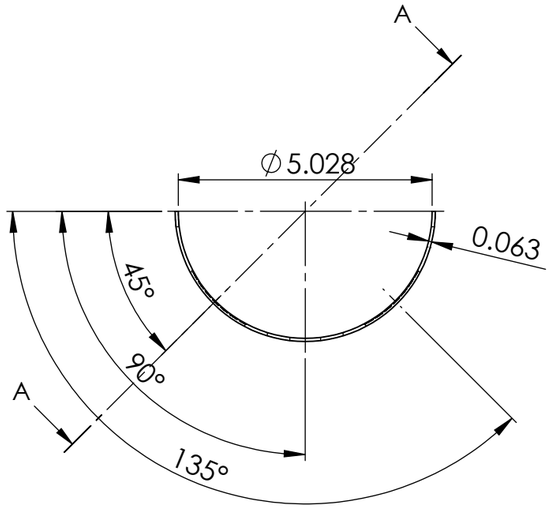
DETAIL D
SCALE 1 : 1

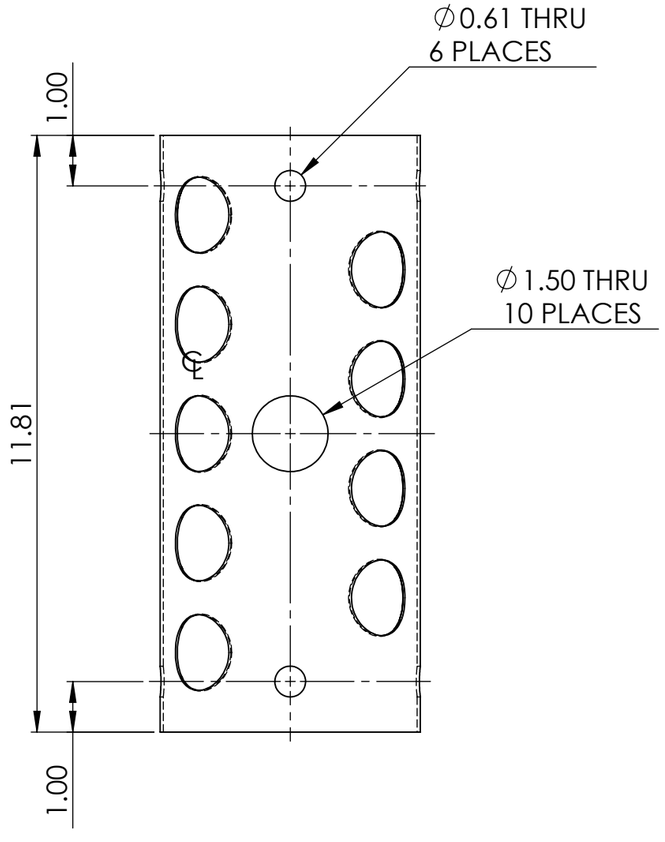
UNIVERSITY OF ALBERTA

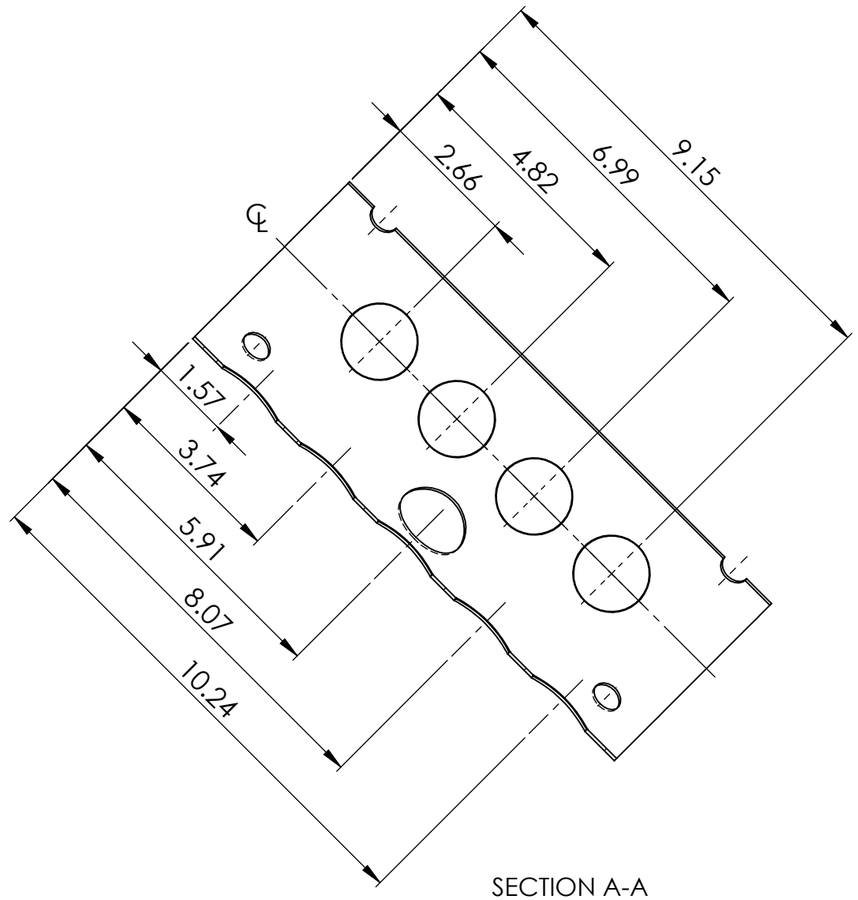
DEPARTMENT OF
MECHANICAL ENGINEERING

4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

SIZE	DWG. NAME	
B	TEST CHAMBER BODY	
SCALE:1:3	WEIGHT:	SHEET 3 OF 3







SECTION A-A

DIMENSIONS ARE IN INCHES
TOLERANCES UNLESS SPECIFIED:
FRACTIONAL ±1/32
ANGULAR: MACH ± 1° BEND ± 2°
TWO PLACE DECIMAL ± .010
THREE PLACE DECIMAL ± .005

MATERIAL: 316 STAINLESS

FINISH UNLESS SPECIFIED: 63

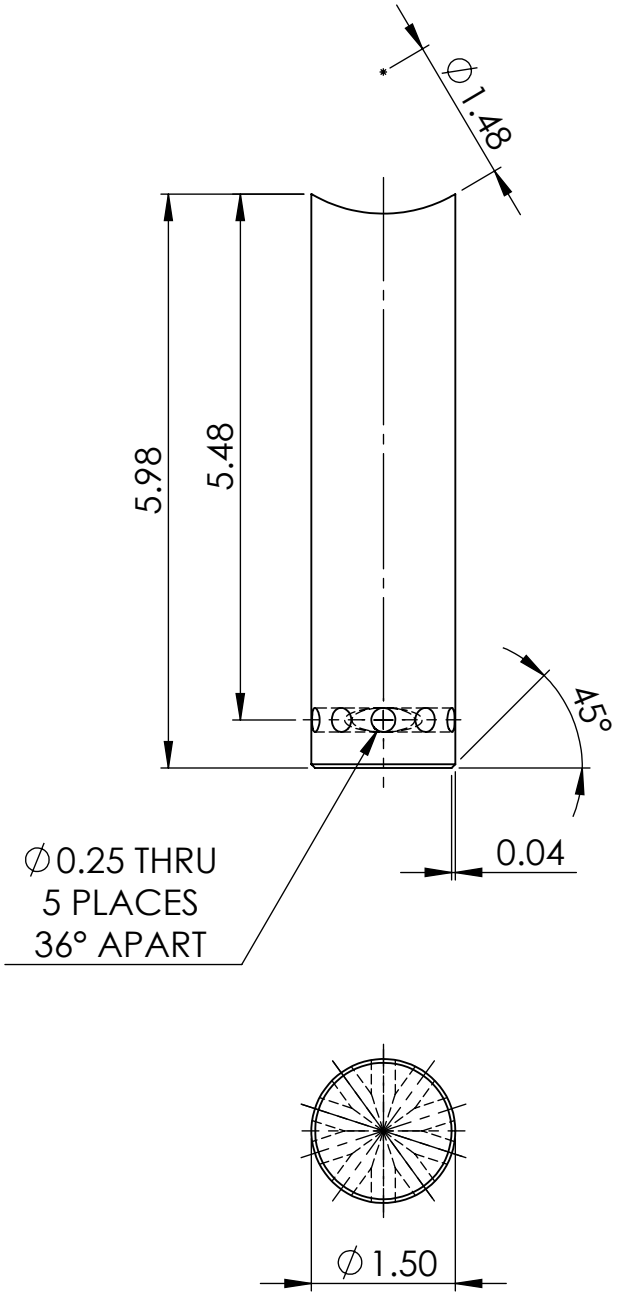
DO NOT SCALE DRAWING

	NAME	DATE	SIGNATURE
DRAWN	M. EVANS	22-JUN-08	
CHECKED			
ENG APPR.			
MFG APPR.			

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

SIZE	DWG. NO.	
B	TEST CHAMBER WATER JACKET	
SCALE:1:3	WEIGHT:	SHEET 1 OF 1



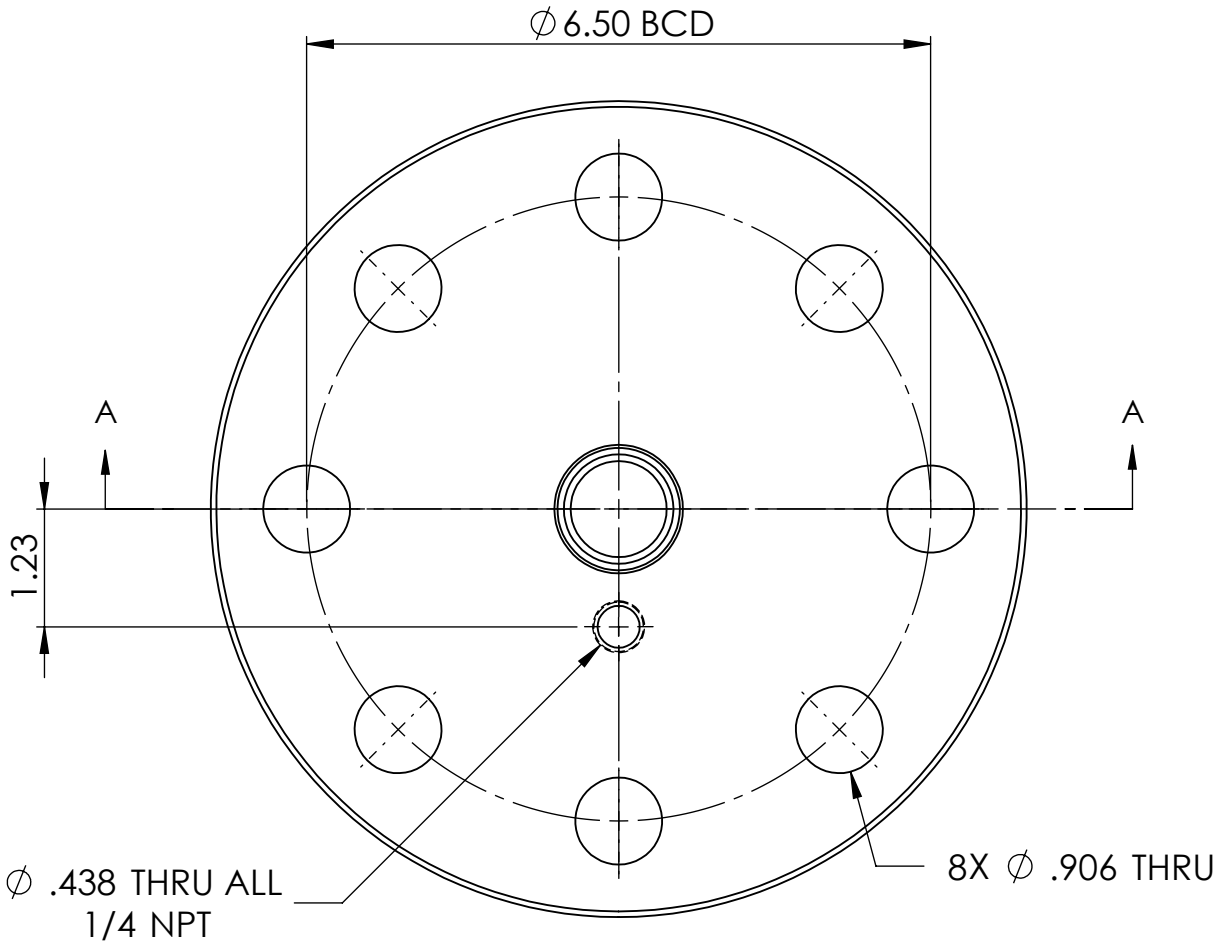
DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005		NAME	DATE	SIGNATURE
	DRAWN	M. EVANS	22-JUN-08	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

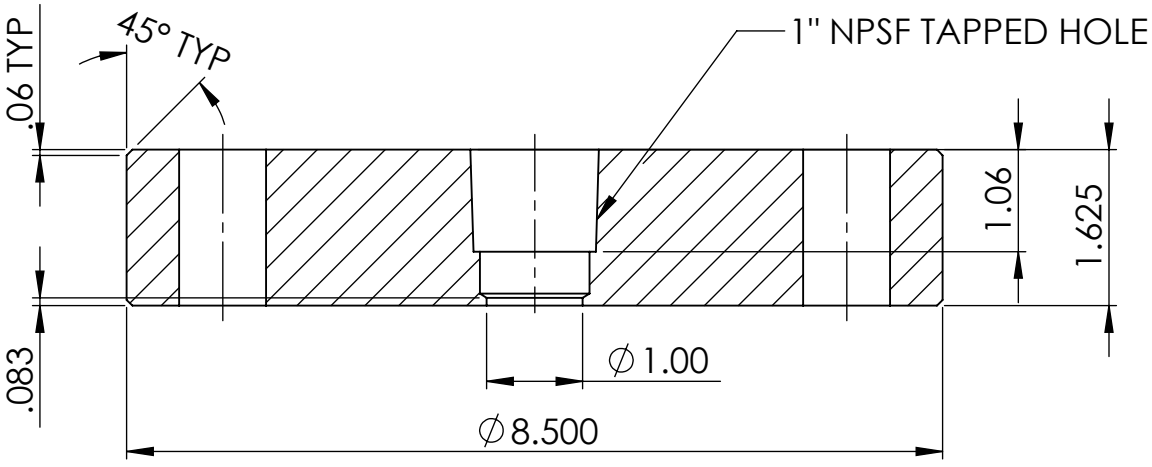
UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

MATERIAL: 316 STAINLESS STEEL
FINISH UNLESS SPECIFIED: 63
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

SIZE	DWG. NO.	
A	TEST CHAMBER TRUNNION	
SCALE:1:2	WEIGHT:	SHEET 1 OF 1





SECTION A-A

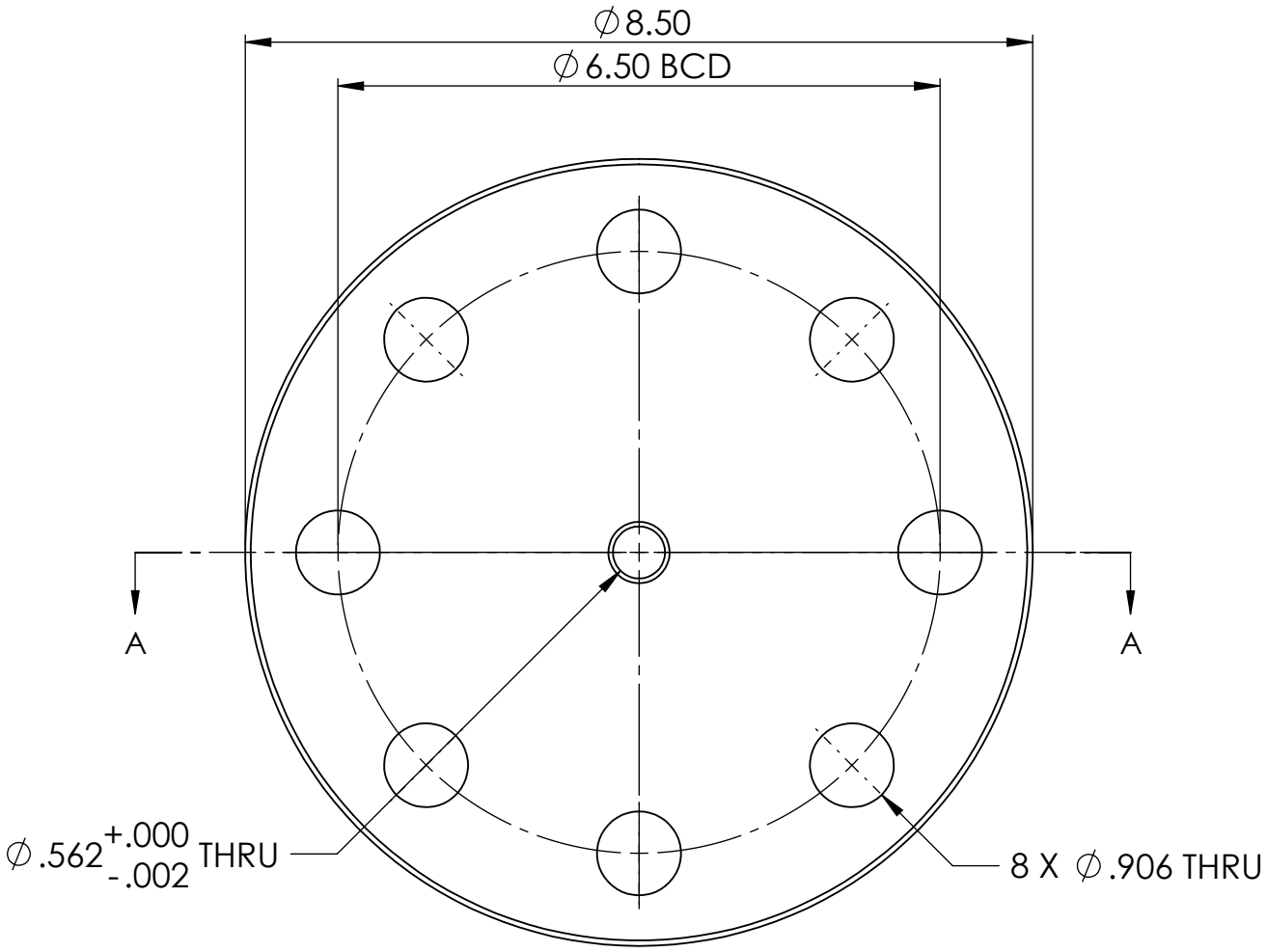
		NAME	DATE	SIGNATURE
DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005	DRAWN	M. EVANS	02-APR-08	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

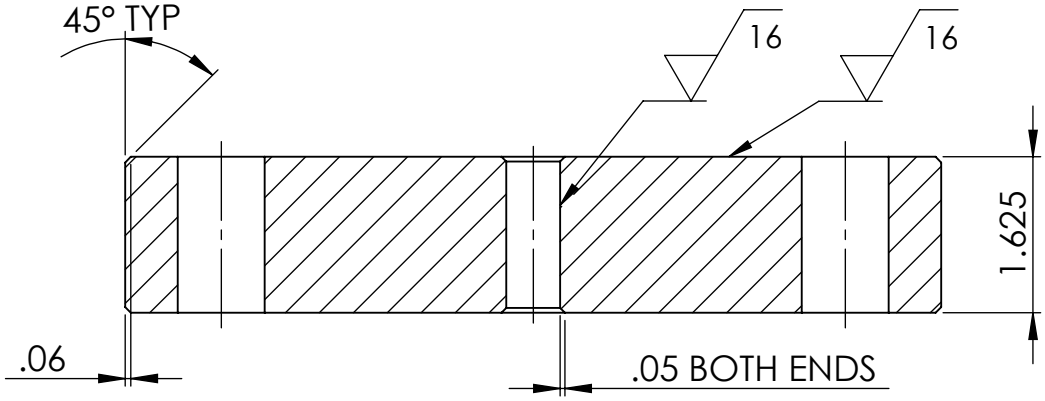
UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

MATERIAL: 17-4 STAINLESS STEEL
FINISH UNLESS SPECIFIED: 63
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

SIZE	DWG. NAME
A	VISCOMETER FLANGE
SCALE:1:2	WEIGHT:
	SHEET 1 OF 1





SECTION A-A

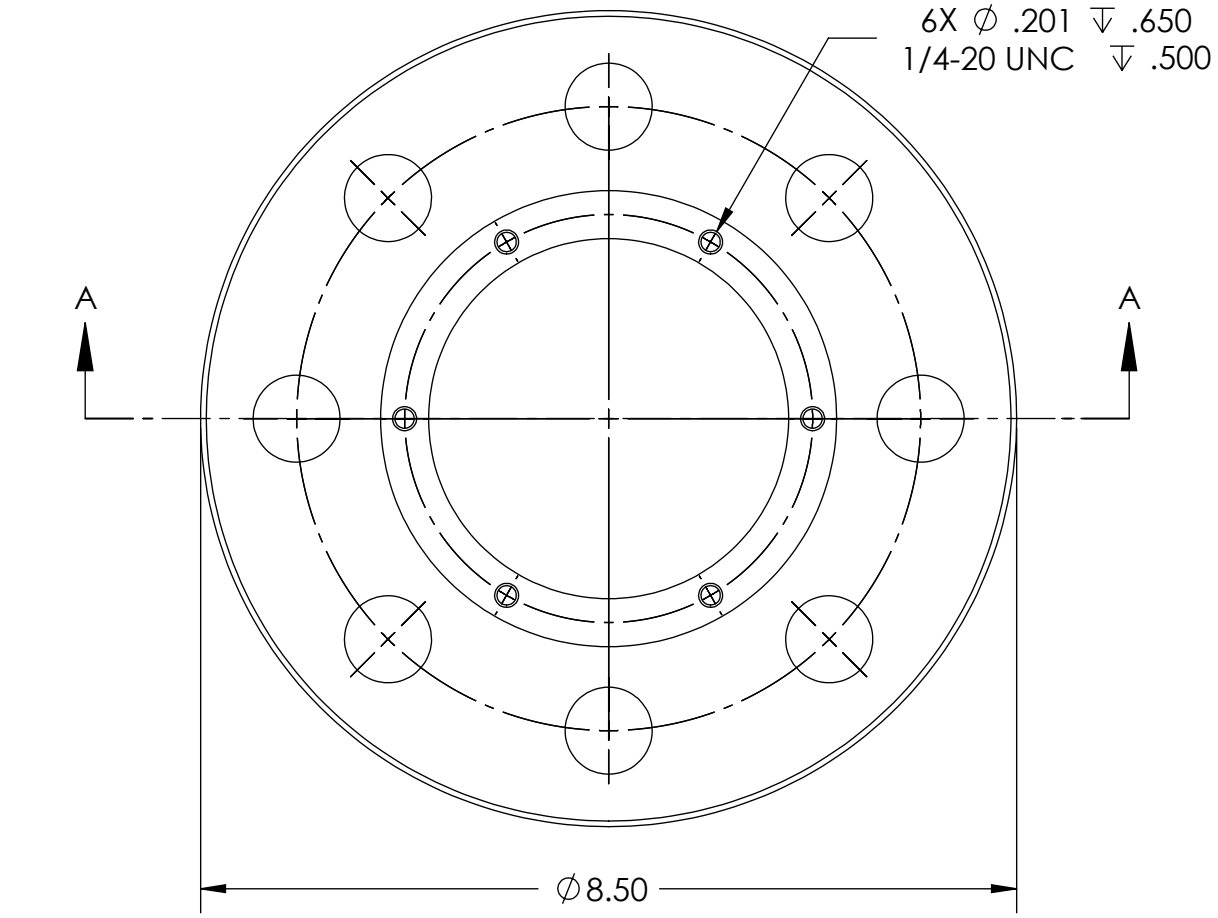
		NAME	DATE	SIGNATURE
DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005	DRAWN	M. EVANS	02-APR-08	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

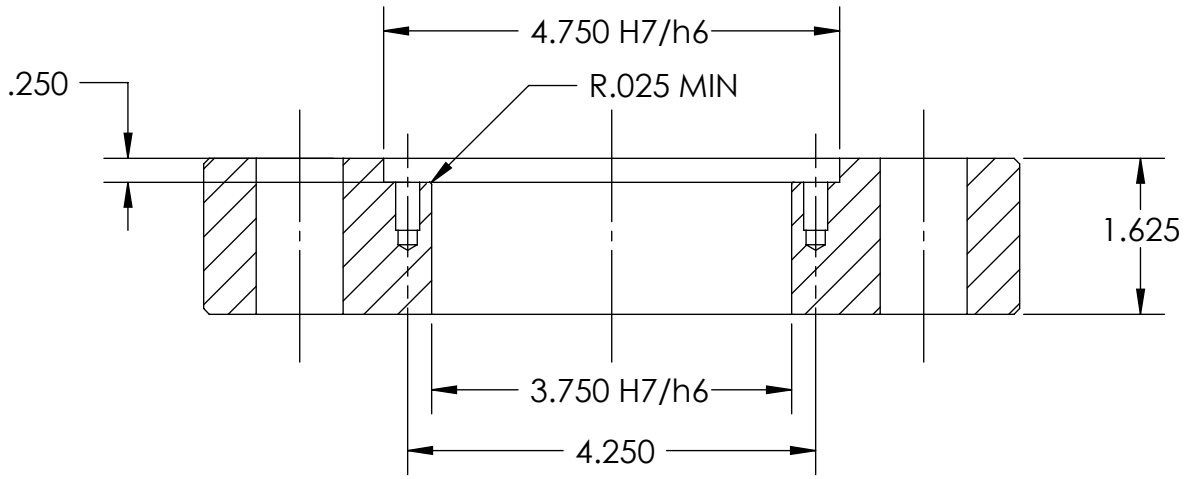
UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

MATERIAL: 17-4 STAINLESS STEEL
FINISH UNLESS SPECIFIED: 63
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

SIZE	DWG. NAME
A	HYDRAULIC PISTON FLANGE
SCALE:1:2	WEIGHT:
	SHEET 1 OF 1





SECTION A-A

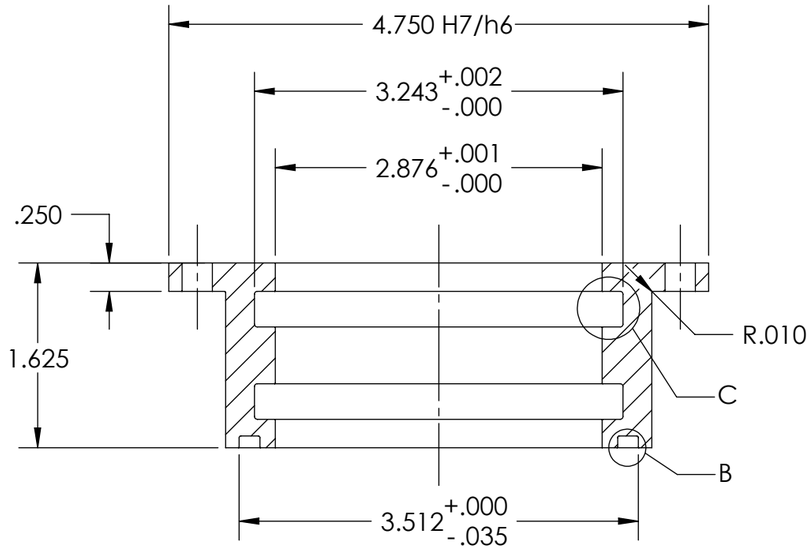
		NAME	DATE	SIGNATURE
DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005	DRAWN	M. EVANS	02-APR-08	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

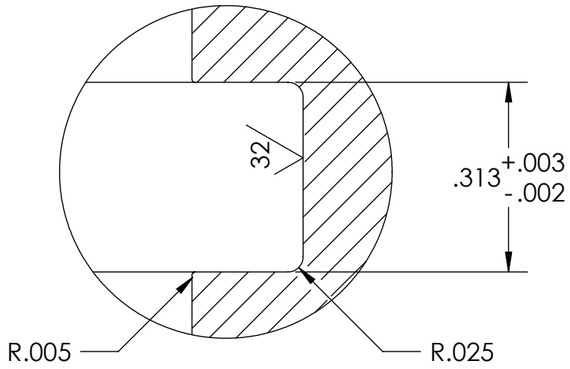
MATERIAL: MODIFY EXISTING
FINISH UNLESS SPECIFIED: 63
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

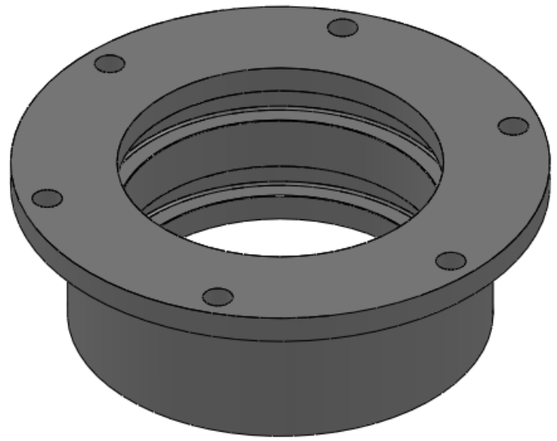
SIZE: A
DWG. NO.: PISTON FLANGE RETROFIT
SCALE:1:2
WEIGHT:
SHEET 1 OF 1

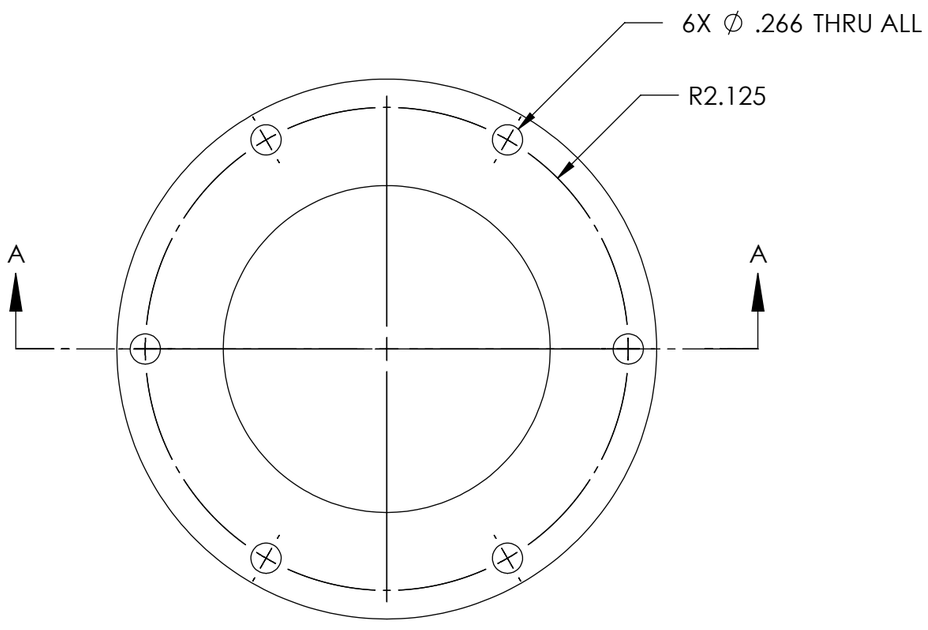


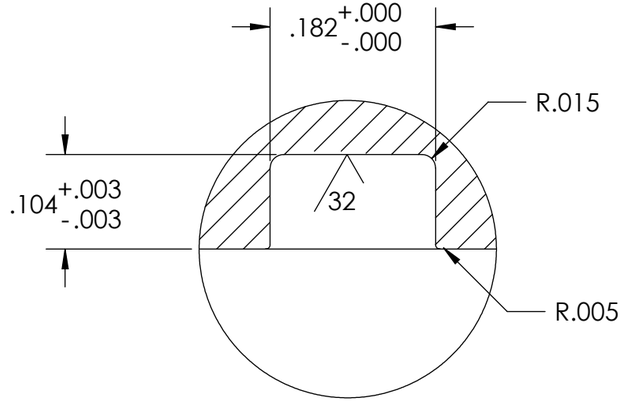
SECTION A-A



DETAIL C
SCALE 4 : 1
(2 PLACES)







DETAIL B
SCALE 6 : 1

DIMENSIONS ARE IN INCHES
TOLERANCES UNLESS SPECIFIED:
FRACTIONAL ± 1/32
ANGULAR: MACH ± 1° BEND ± 2°
TWO PLACE DECIMAL ± .010
THREE PLACE DECIMAL ± .005

MATERIAL: STAINLESS STEEL

FINISH UNLESS SPECIFIED: 63

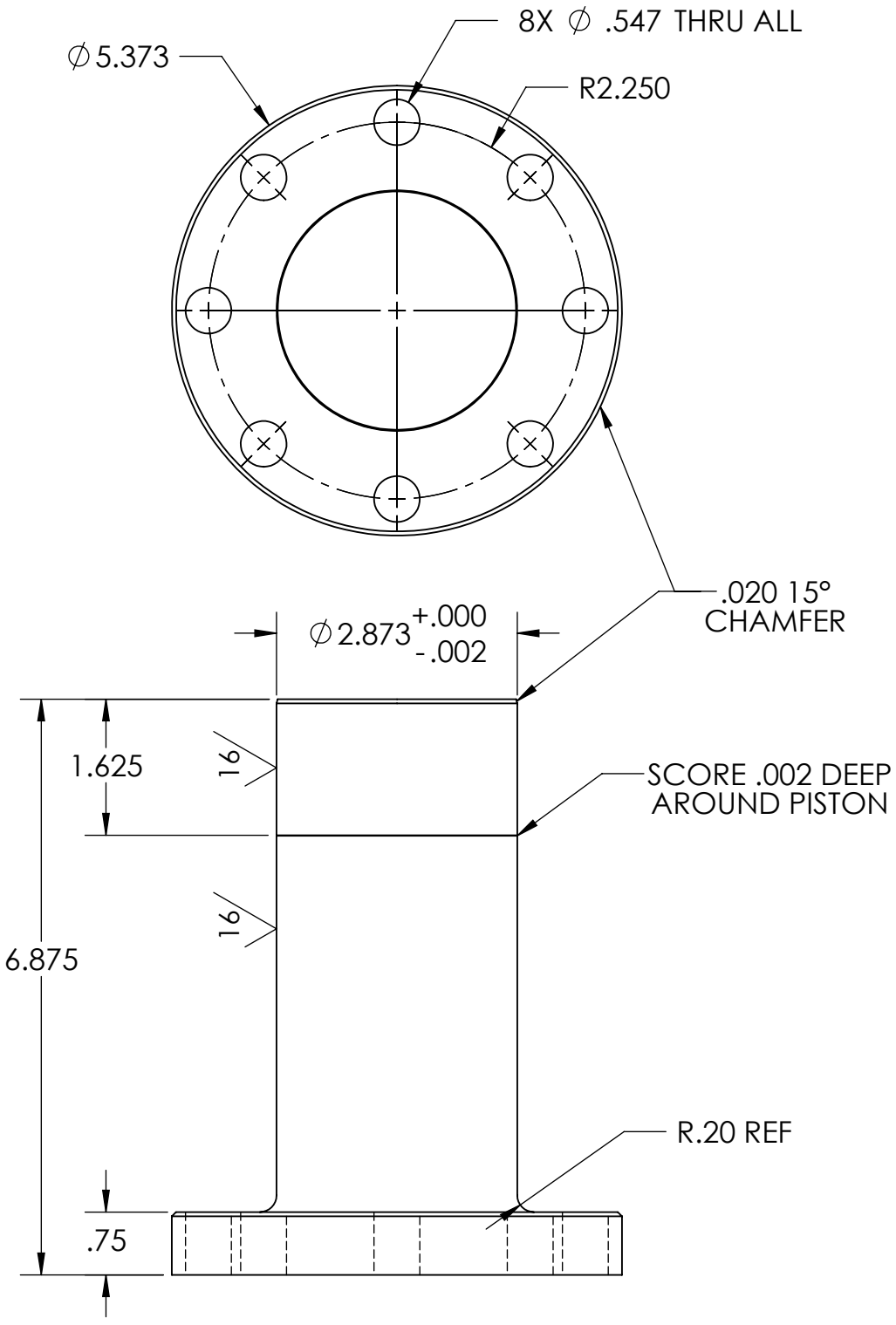
DO NOT SCALE DRAWING

	NAME	DATE	SIGNATURE
DRAWN	M. EVANS	21-OCT-09	
CHECKED			
ENG APPR.			
MFG APPR.			

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

SIZE: B
DWG. NAME: PISTON FLANGE INSERT
SCALE:3:4
WEIGHT:
SHEET 1 OF 1



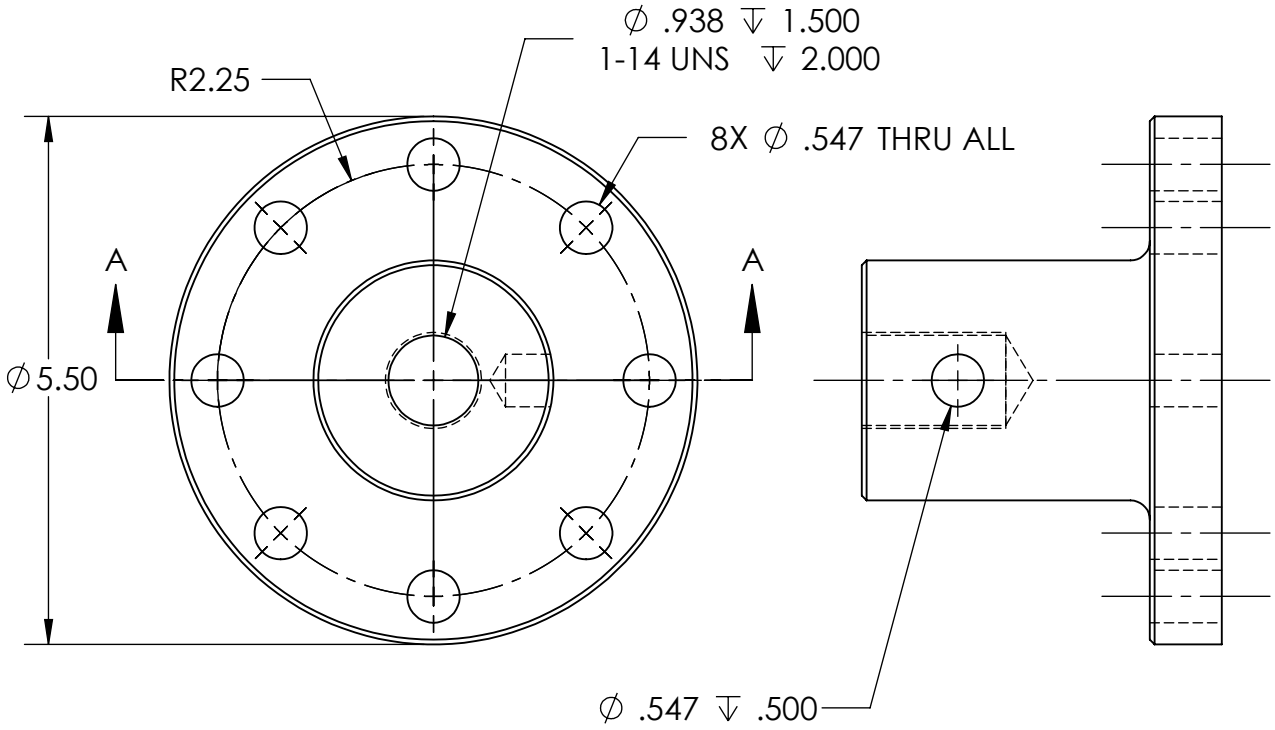
		NAME	DATE	SIGNATURE
DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005	DRAWN	M. EVANS	02-APR-08	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

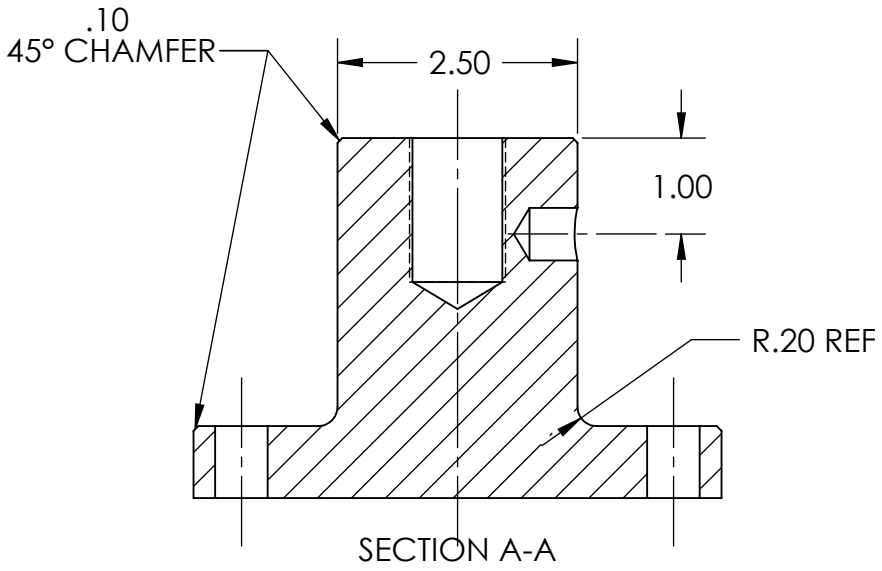
UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

MATERIAL: STAINLESS STEEL
FINISH UNLESS SPECIFIED: 63
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

SIZE	DWG. NO.	
A	HYDRAULIC PISTON	
SCALE:1:2	WEIGHT:	SHEET 1 OF 1





	NAME	DATE	SIGNATURE
DRAWN	M. EVANS	02-APR-08	
CHECKED			
ENG APPR.			
MFG APPR.			

DIMENSIONS ARE IN INCHES

TOLERANCES UNLESS SPECIFIED:
FRACTIONAL ±1/32
ANGULAR: MACH ± 1° BEND ± 2°
TWO PLACE DECIMAL ± .010
THREE PLACE DECIMAL ± .005

PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF THE IMPERIAL OIL CENTRE FOR OIL SANDS INNOVATION (COSI). ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF COSI IS PROHIBITED.

MATERIAL
STAINLESS STEEL

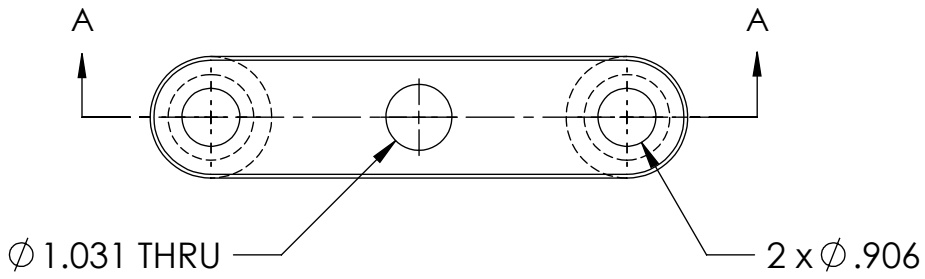
FINISH UNLESS SPECIFIED
63

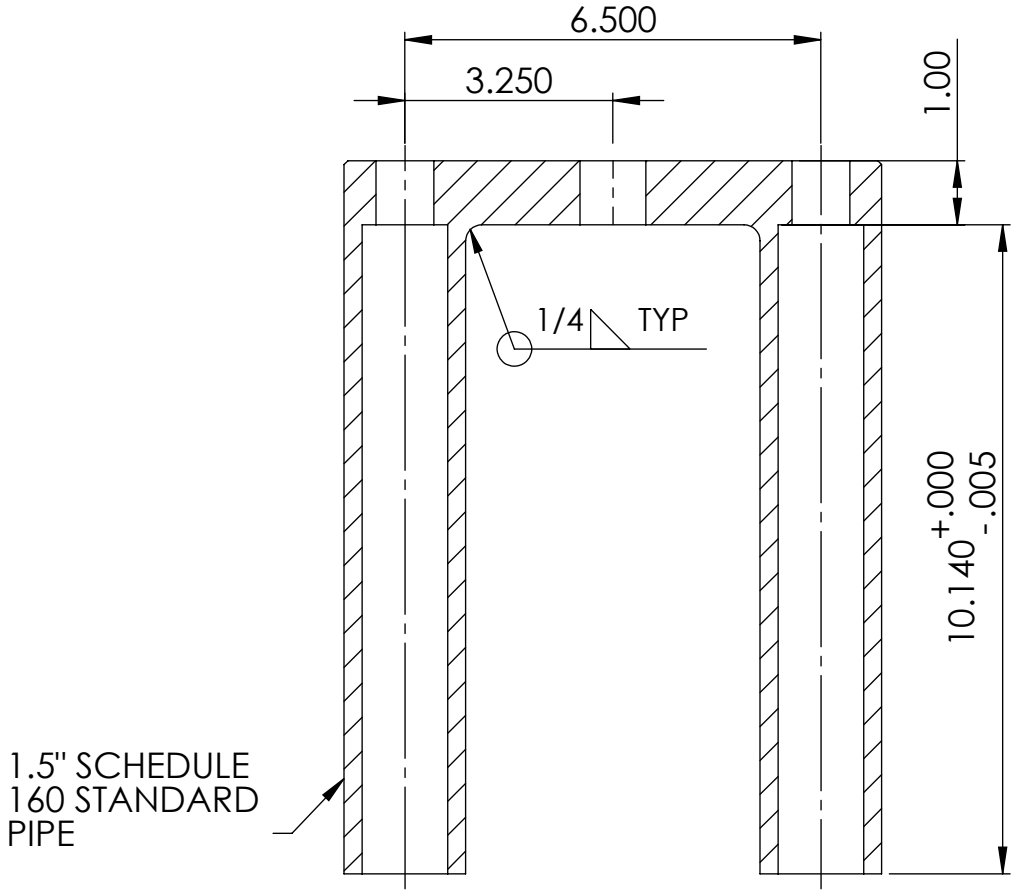
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

SIZE	DWG. NO.	
A	HYDRAULIC PISTON COUPLING	
SCALE:1:2	WEIGHT:	SHEET 1 OF 1





SECTION A-A

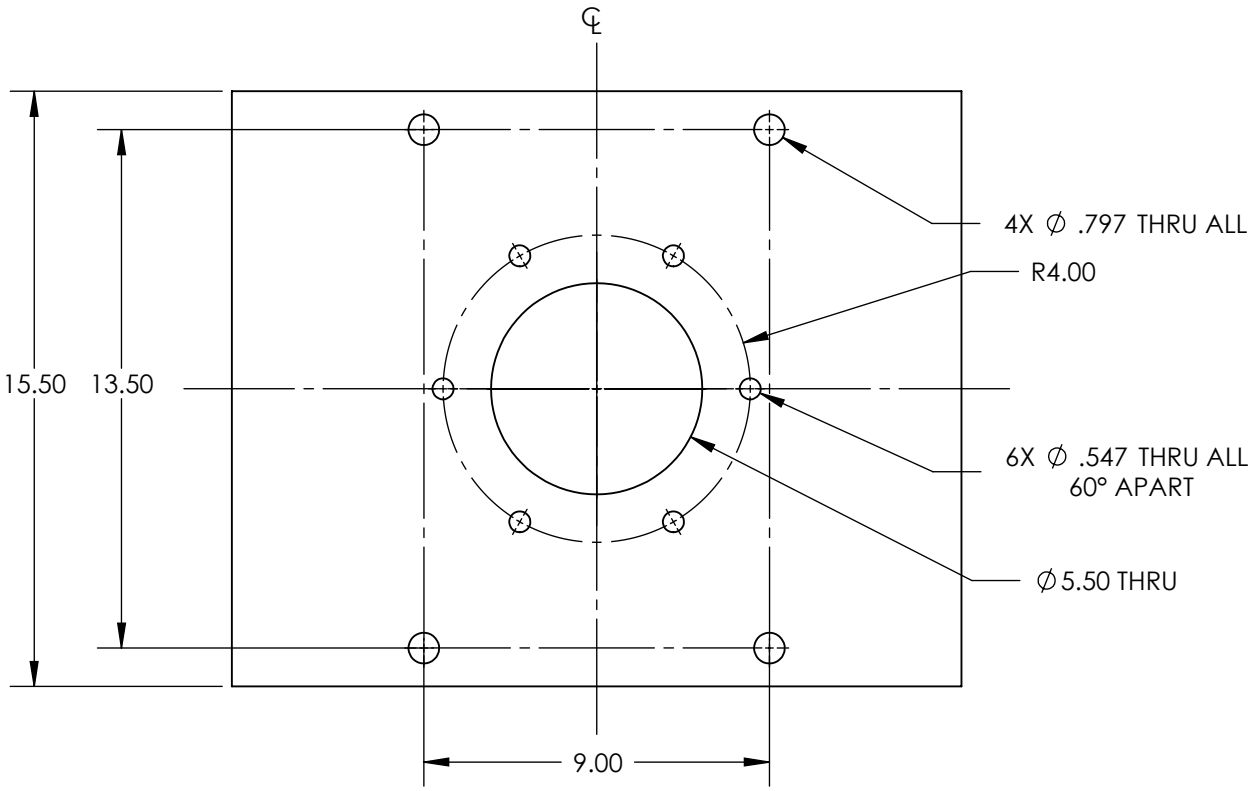
		NAME	DATE	SIGNATURE
DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005	DRAWN	M. EVANS	02-APR-08	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

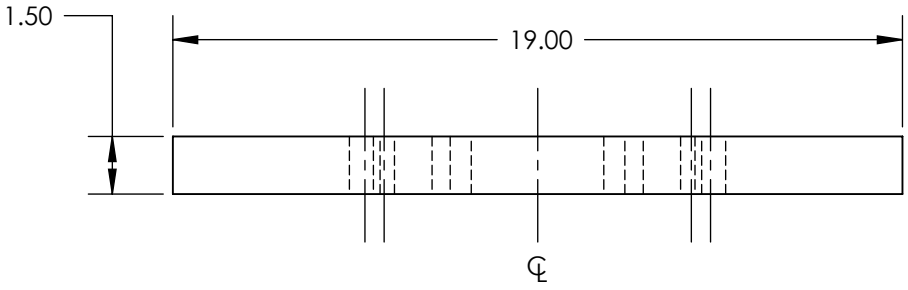
UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

MATERIAL: STEEL
FINISH UNLESS SPECIFIED: 63
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES
3. WELD PIPE TO CROSS BAR

SIZE	DWG. NO.	
A	CHAMBER ANCHOR	
SCALE:1:3	WEIGHT:	SHEET 1 OF 1





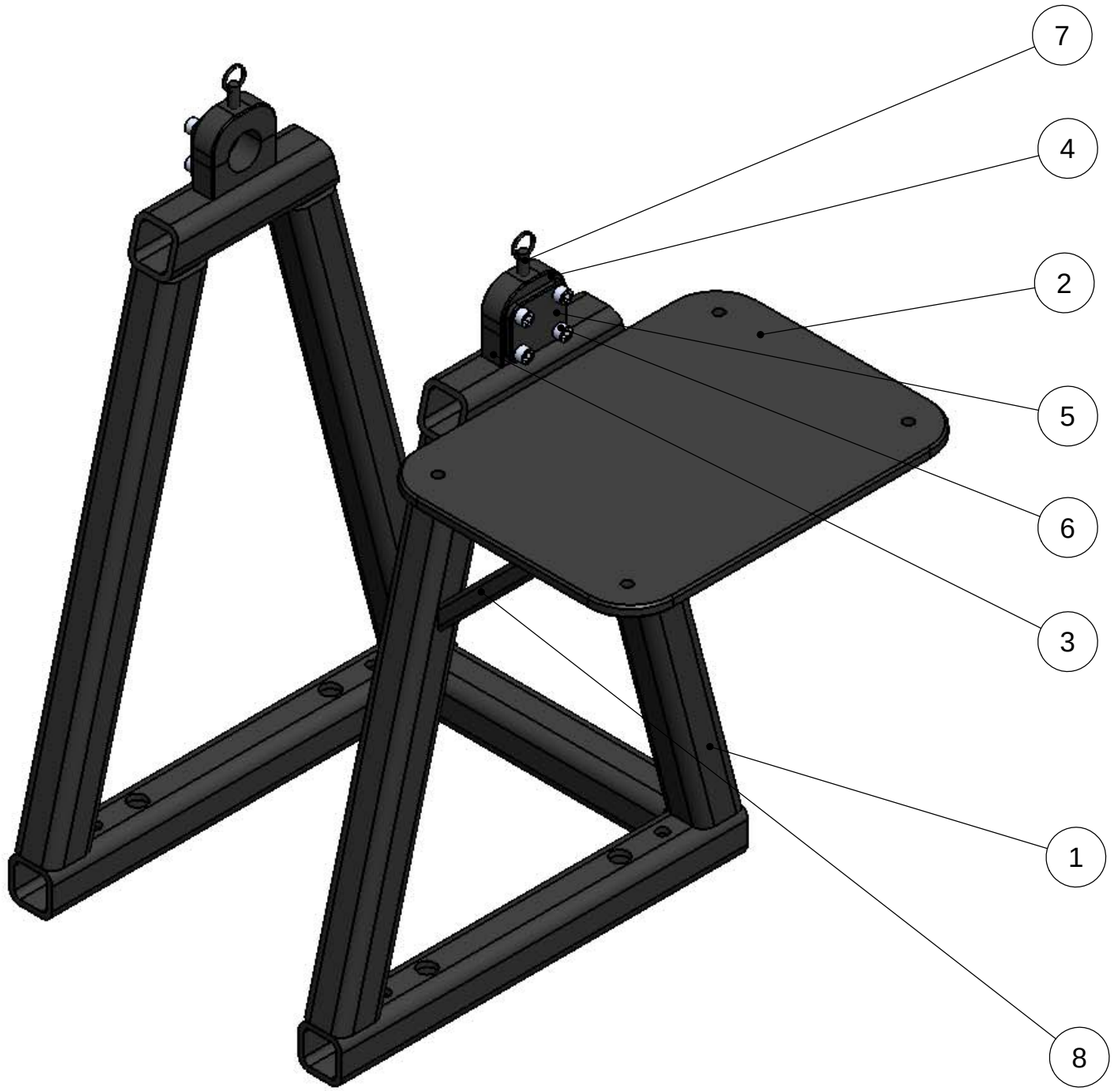
		NAME	DATE	SIGNATURE
DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005	DRAWN	M. EVANS	02-APR-08	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

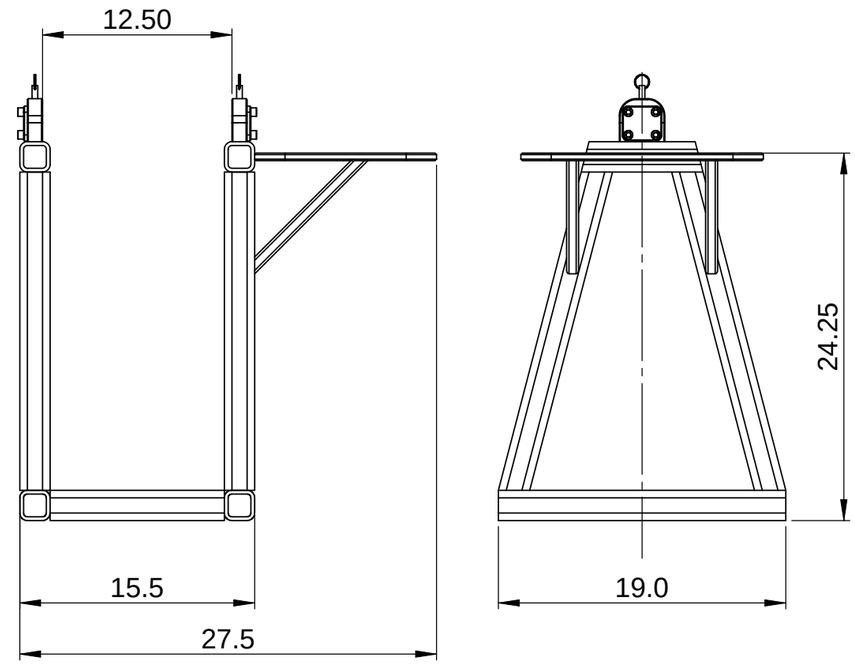
MATERIAL: VARIOUS
FINISH UNLESS SPECIFIED: 63
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

SIZE	DWG. NO.	
A	TENSILE TESTING MACHINE MOUNTING PLATE	
SCALE:1:5	WEIGHT:	SHEET 1 OF 1



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Frame	2" x 1/4" Standard Square Tube	1
2	Frame Table		1
3	Support Clamp Base		2
4	Support Clamp Top		2
5	Support Clamp End Plate		2
6	3-8_16 Socket Head Cap Screw	McMaster-Carr #92196A622	8
7	Spring Plunger	McMaster-Carr #94975A411	2
8	Table Support		2



SCALE 1:10

DIMENSIONS ARE IN INCHES
TOLERANCES UNLESS SPECIFIED:
FRACTIONAL ±1/32
ANGULAR: MACH ± 1° BEND ± 2°
TWO PLACE DECIMAL ± .010
THREE PLACE DECIMAL ± .005

MATERIAL: STEEL

FINISH UNLESS SPECIFIED: 63

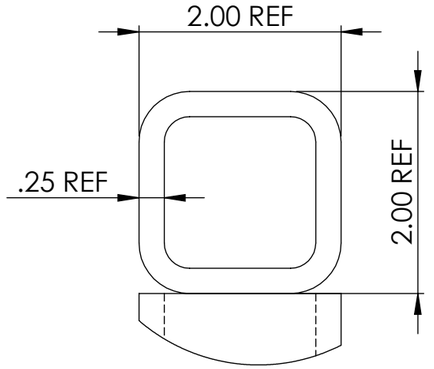
DO NOT SCALE DRAWING

	NAME	DATE	SIGNATURE
DRAWN	M. EVANS	17-JUN-08	
CHECKED			
ENG APPR.			
MFG APPR.			

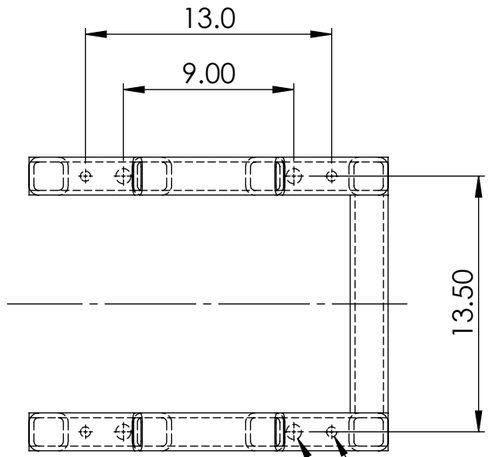
NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES
3. WELD LOWER SUPPORT CLAMPS TO FRAME
4. WELD TABLE TO FRAME
5. PAINT COMPLETED ASSEMBLY

UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

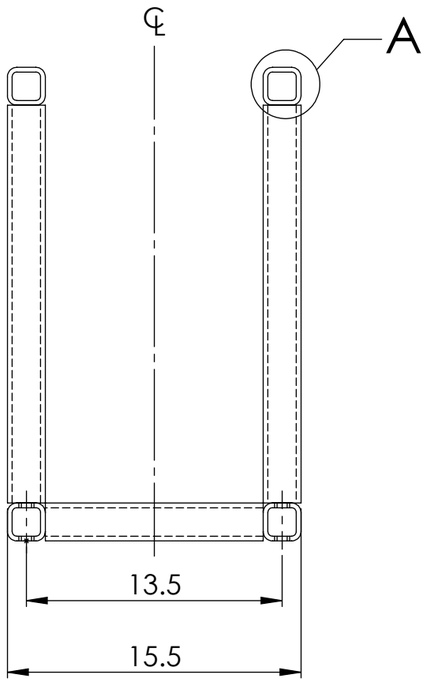
SIZE: B
DWG. NAME: FRAME ASSEMBLY
SCALE:1:5
WEIGHT:
SHEET 1 OF 1

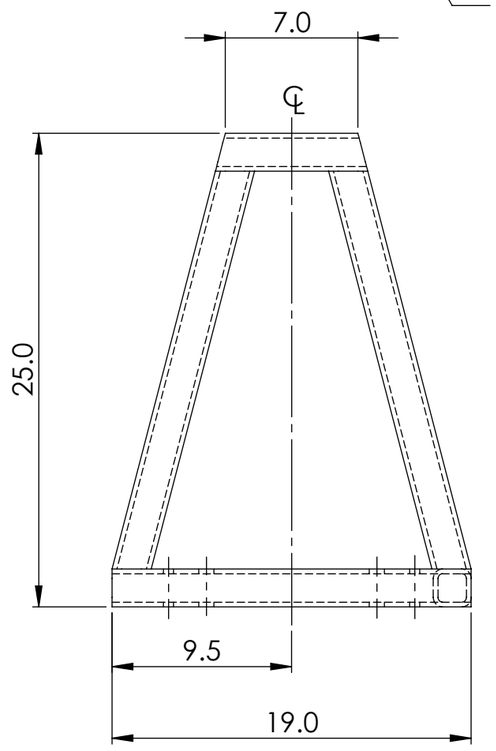


DETAIL A
SCALE 1 : 1.5



4X ∅ .531 ↧ 2.000
4X ∅ .797 THRU ALL





DIMENSIONS ARE IN INCHES
TOLERANCES UNLESS SPECIFIED:
FRACTIONAL ±1/32
ANGULAR: MACH ± 1° BEND ± 2°
TWO PLACE DECIMAL ± .010
THREE PLACE DECIMAL ± .005

MATERIAL
2" x 1/4" SQUARE STEEL TUBE

FINISH UNLESS SPECIFIED
-

DO NOT SCALE DRAWING

	NAME	DATE	SIGNATURE
DRAWN	M. EVANS	02-APR-08	
CHECKED			
ENG APPR.			
MFG APPR.			

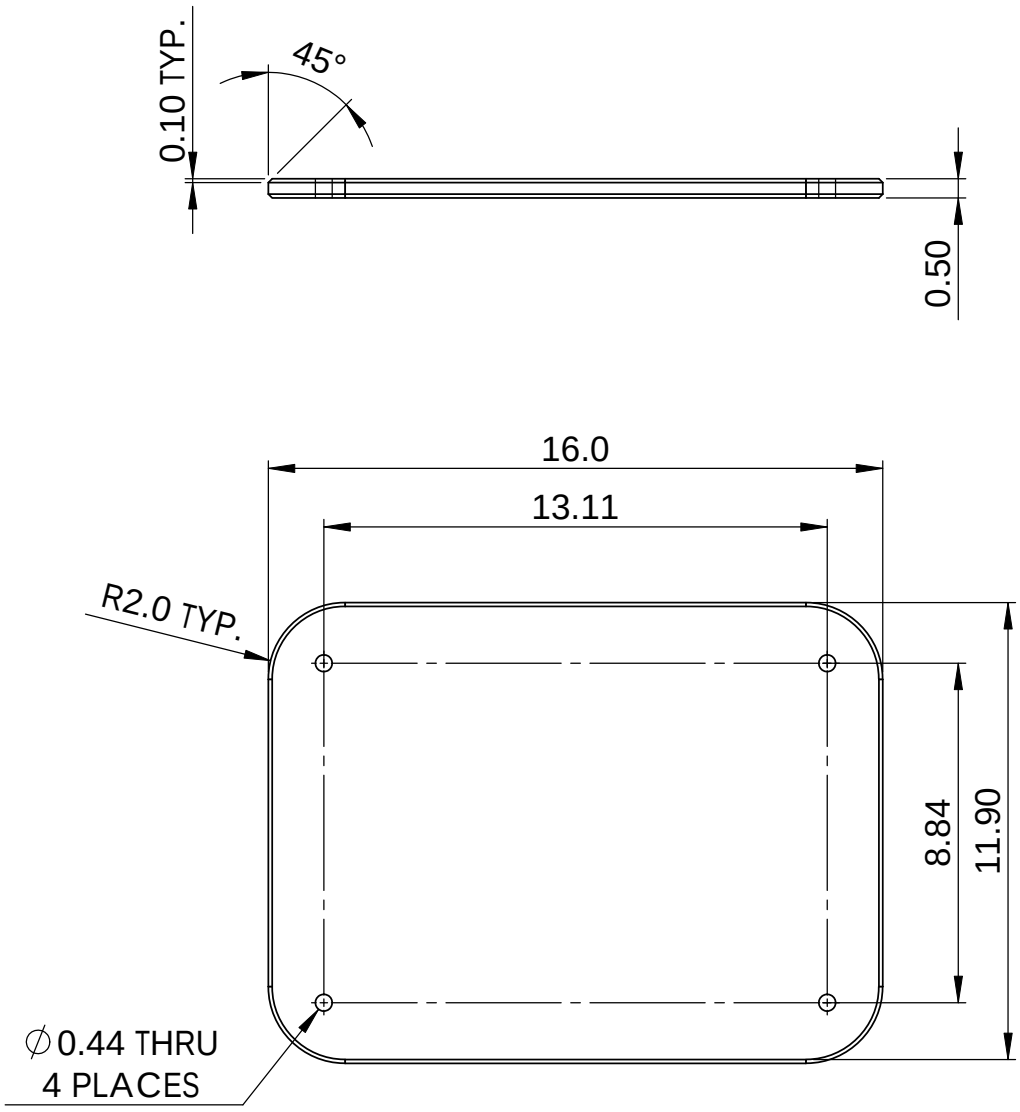
NOTES:

1. WELD TOGETHER
2. REMOVE ALL BURRS
3. CHAMFER SHARP EDGES
4. PAINT GREY

UNIVERSITY OF ALBERTA
DEPARTMENT OF
MECHANICAL ENGINEERING

4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

SIZE	DWG. NO.	
B	FRAME BODY	
SCALE:1:8	WEIGHT:	SHEET 1 OF 1



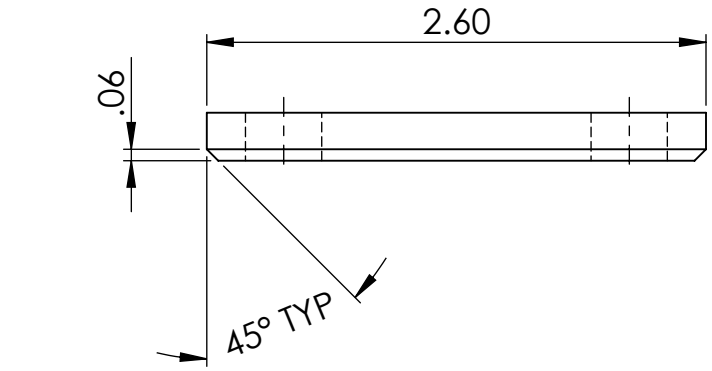
		NAME	DATE	SIGNATURE
DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005	DRAWN	M. EVANS	22-JUN-08	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

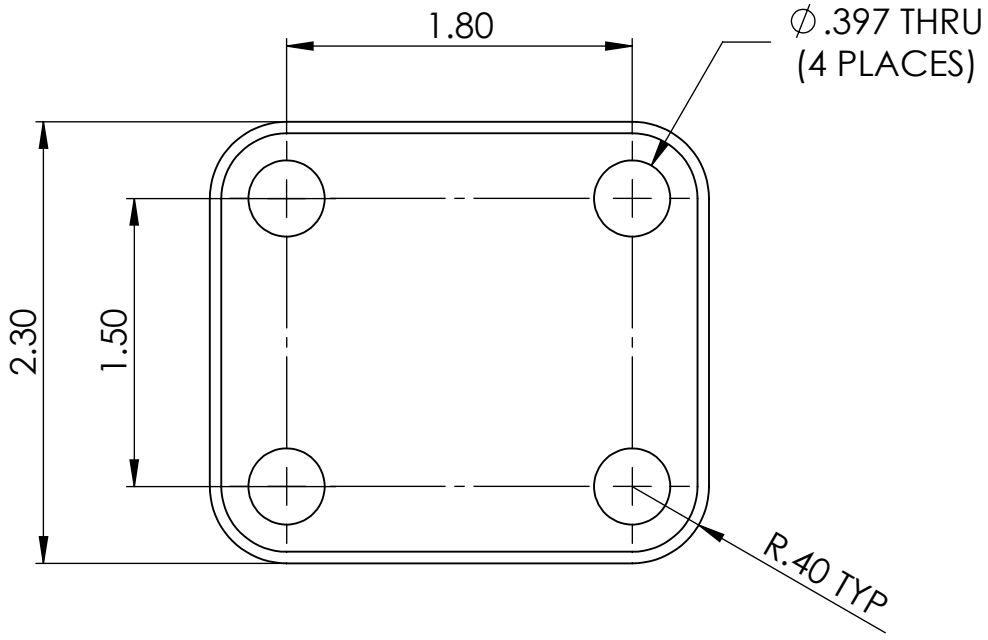
UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

MATERIAL: STEEL
FINISH UNLESS SPECIFIED: 63
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

SIZE	DWG. NAME	
A	FRAME TABLE	
SCALE:1:5	WEIGHT:	SHEET 1 OF 1





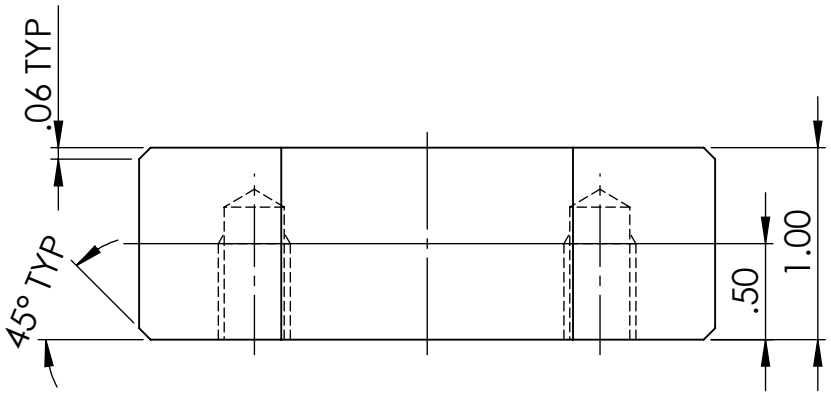
DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005		NAME	DATE	SIGNATURE
	DRAWN	M. EVANS	02-APR-08	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

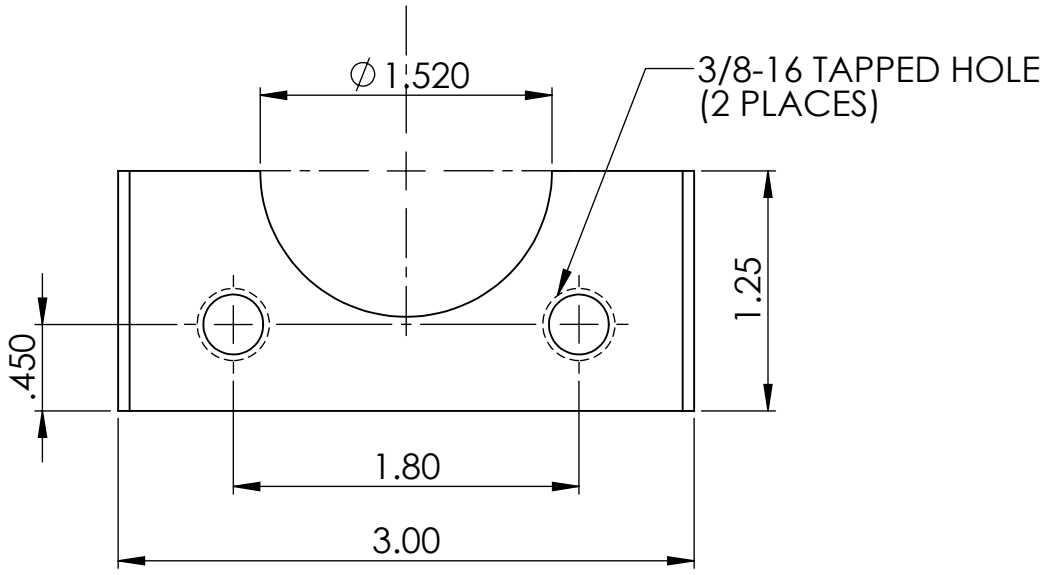
UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

MATERIAL: STEEL
FINISH UNLESS SPECIFIED: 63
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

SIZE	DWG. NO.	
A	TRUNNION END CAP	
SCALE:1:1	WEIGHT:	SHEET 1 OF 1





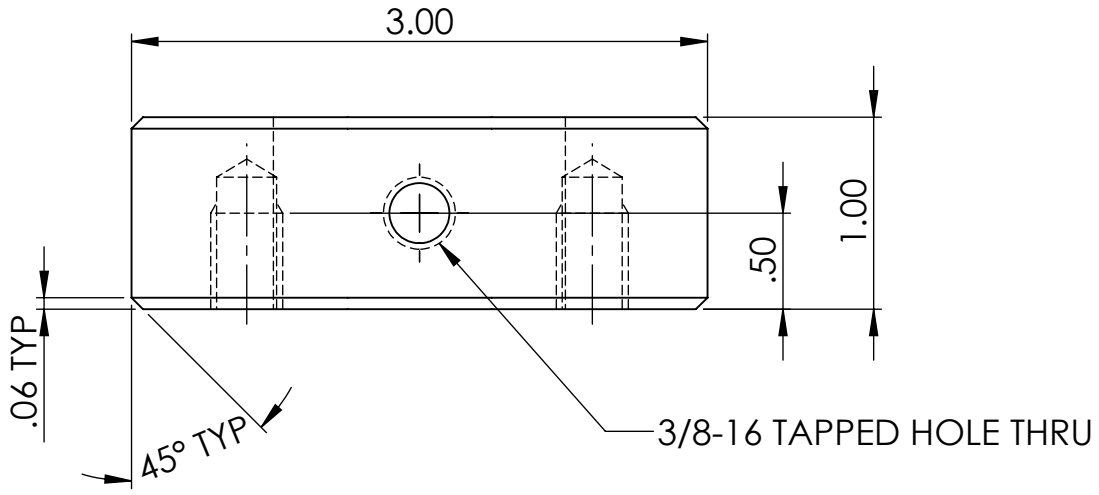
DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005		NAME	DATE	SIGNATURE
	DRAWN	M. EVANS	02-APR-08	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

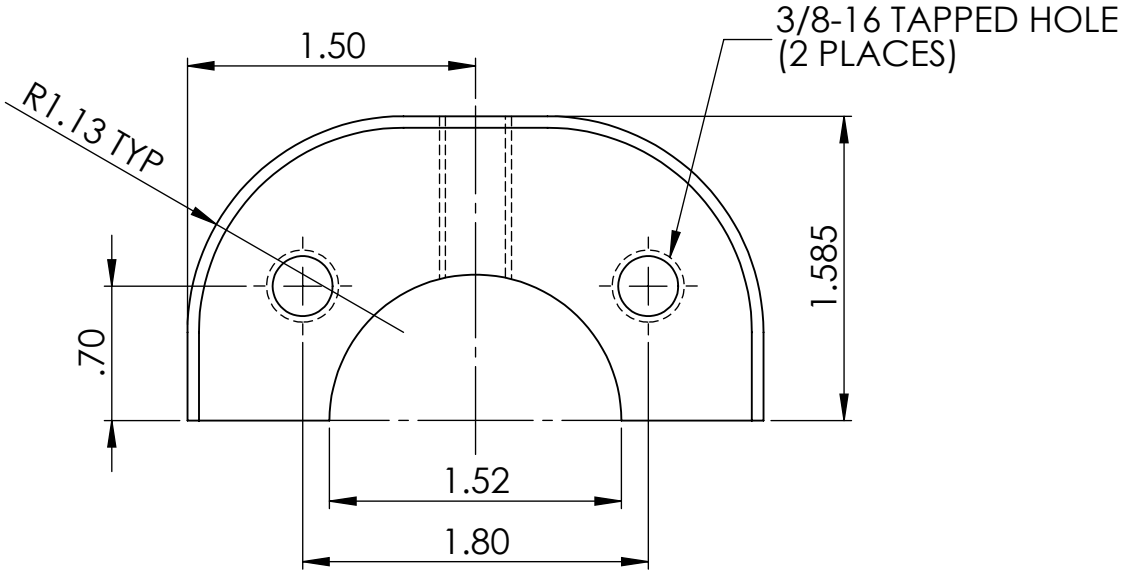
UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

MATERIAL: STEEL
FINISH UNLESS SPECIFIED: 63
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

SIZE	DWG. NO.	
A	TRUNNION BASE	
SCALE:1:1	WEIGHT:	SHEET 1 OF 1





DIMENSIONS ARE IN INCHES TOLERANCES UNLESS SPECIFIED: FRACTIONAL ±1/32 ANGULAR: MACH ± 1° BEND ± 2° TWO PLACE DECIMAL ± .010 THREE PLACE DECIMAL ± .005		NAME	DATE	SIGNATURE
	DRAWN	M. EVANS	02-APR-08	
	CHECKED			
	ENG APPR.			
	MFG APPR.			

UNIVERSITY OF ALBERTA
DEPARTMENT OF MECHANICAL ENGINEERING
4-9 MECH. ENG. BLDG.
EDMONTON, ALBERTA
T6G 2G8

MATERIAL: STEEL
FINISH UNLESS SPECIFIED: 63
DO NOT SCALE DRAWING

NOTES:
1. REMOVE ALL BURRS
2. CHAMFER SHARP EDGES

SIZE	DWG. NO.	
A	TRUNNION TOP CAP	
SCALE:1:1	WEIGHT:	SHEET 1 OF 1

Appendix G Monitoring and Control Software Source Code

The following code was written in the National Instruments LabWindows/CVI 9.0 programming environment. It includes the main function, equipment initialization functions, automated experiment procedure, student coded instrument functions, and all GUI setup. The text size has been reduced to conserve space.

```
//-----------------------------------------------------------
//
// A BASIC log and disply program to communicate with a
// NATIONAL INSTRUMENTS USB DAQ unit and up to 3 serial
// devices
//
// Written by: Marc D. Evans
// University of Alberta
// Last Updated: June 2010
//
// Based on a code framework written by Dr. David S. Nobes
//
//-----------------------------------------------------------

#include "Monitoring and Control Program.h"
#include <NIDAQmx.h>
#include <formatio.h>
#include <stdio.h>
#include <rs232.h>
#include <utility.h>
#include <ansi_c.h>
#include <analysis.h>
#include <cvirte.h>
#include <userint.h>
#include <ctype.h>
#include "Monitoring and Control Program_Declare.h"
#include "Monitoring and Control Program.h"
#include "NIDAQmx.h"
#include "DAQmxIOctrl.h"
#include "visatype.h"
#include "tkafg3k.h"

ViSession   tkafg3k;

#define DAQmxErrChk(functionCall) if( DAQmxFailed(DAQerror=(functionCall)) ) goto Error; else

// DAQ Globals
    int32       DAQerror=0;
    TaskHandle  taskHandle=0;
    char        chan[256];
    int32       DAQrate;
    uInt32      DAQsampsPerChan;
    int32       DAQnumRead;
    uInt32      DAQnumChannels;
    float64     *DAQdata=NULL;
    float64     *DAQArray_Out=NULL;
    int         DAQlog;
    char        DAQerrBuff[2048]={'\0'};
    double      DAQoutMean=0;
    int         DAQ_Task_Started=0;

/*****************************************************************************************************************************************/
//  888b     d888        d8888 8888888 888b    888         d88P  8888888b.     d8888 888b    888 8888888888 888      .d8888b.
//  8888b   d8888       d88888   888   8888b   888        d88P   888   Y88b   d88888 8888b   888 888        888     d88P  Y88b
//  88888b.d88888      d88P888   888   88888b  888       d88P    888    888  d88P888 88888b  888 888        888     Y88b.
//  888Y88888P888     d88P 888   888   888Y88b 888      d88P     888   d88P d88P 888 888Y88b 888 8888888    888      "Y888b.
//  888 Y888P 888    d88P  888   888   888 Y88b888     d88P      8888888P" d88P  888 888 Y88b888 888        888         "Y88b.
//  888  Y8P  888   d88P   888   888   888  Y88888    d88P       888      d88P   888 888  Y88888 888        888           "888
//  888   "   888  d8888888888   888   888   Y8888   d88P        888     d8888888888 888   Y8888 888        888     Y88b  d88P
//  888       888 d88P     888 8888888 888    Y888  d88P         888    d88P     888 888    Y888 8888888888 88888888 "Y8888P"
/*****************************************************************************************************************************************/

//*******************************************************
// main             : MCP main function
//*******************************************************
int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)
        return -1;  /* out of memory */
    if ((panelHandle    = LoadPanel (0, "Monitoring and Control Program.uir", PANEL)) < 0)
        return -1;
    // Set the panel variables
        g_Handle        = LoadPanel (0, "Monitoring and Control Program.uir", G_SETUP);
        com_Handle      = LoadPanel (0, "Monitoring and Control Program.uir", COM);
        a_Handle        = LoadPanel (0, "Monitoring and Control Program.uir", ABOUT);
        s_Handle        = LoadPanel (0, "Monitoring and Control Program.uir", SAVE_Con);

    DisplayPanel (panelHandle);
        DSN_Init();
        DSN_Init2();
    SetSleepPolicy (VAL_SLEEP_MORE);

    // Maximise the panel
    //SetPanelAttribute (panelHandle, ATTR_WINDOW_ZOOM, VAL_MAXIMIZE);
    RunUserInterface ();
    DSN_Update_Graphics();

    DiscardPanel (panelHandle);
    CloseCVIRTE ();
    return 0;
}

//*******************************************************
// ExitCallback     : Exit menu
//*******************************************************
void CVICALLBACK ExitCallback (int menuBar, int menuItem, void *callbackData, int panel)

{
            DisplayPanel (s_Handle);
}

//*******************************************************
// QuitCallback     : Main panel quit
//*******************************************************
int CVICALLBACK QuitCallback (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            DisplayPanel (s_Handle);
            break;
        case EVENT_RIGHT_CLICK:

            break;
```

```
        }
    return 0;
}
//*********************************************************
// DSN_MainPanelQuit: Main panel quit function
//*********************************************************
void DSN_MainPanelQuit(void)
{int i;

            QuitUserInterface (0);

return;
}
//*********************************************************
// Save_MCP_Config: Save the current configuration on exit
//*********************************************************
int CVICALLBACK Save_MCP_Config (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
        switch (control)
            {
            case SAVE_Con_SAVE_YES:
                DSN_Save_Vars();
                DSN_Save_Config(0);
                HidePanel (s_Handle);
                DSN_MainPanelQuit();
                break;
            case SAVE_Con_SAVE_NO:
                HidePanel (s_Handle);
                DSN_MainPanelQuit();
                break;
            case SAVE_Con_SAVE_CANCEL:
                HidePanel (s_Handle);
                break;
            }
            break;
        case EVENT_RIGHT_CLICK:

            break;
        }
    return 0;
}
//*********************************************************
// SHOW AND CLOSE CALLBACKS FOR OTHER PANELS
//*********************************************************

//*********************************************************
// GRAPH SETUP
//*********************************************************
int CVICALLBACK SHOW_Graphs (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:

            break;
        case EVENT_RIGHT_CLICK:
            DisplayPanel (g_Handle);
            break;
        }
    return 0;
}

int CVICALLBACK CLOSE_Graphs (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            HidePanel (g_Handle);
            break;
        case EVENT_RIGHT_CLICK:

            break;
        }
    return 0;
}

//*********************************************************
// ABOUT
//*********************************************************
void CVICALLBACK SHOW_About (int menuBar, int menuItem, void *callbackData,
        int panel)
{
  DisplayPanel (a_Handle);
}

int CVICALLBACK CLOSE_About (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            HidePanel (a_Handle);
            break;
        case EVENT_RIGHT_CLICK:

            break;
        }
    return 0;
}
//*********************************************************
// COMMUNICATIONS SETUP
//*********************************************************
void CVICALLBACK SHOW_Com (int menuBar, int menuItem, void *callbackData,
        int panel)
{
  DisplayPanel (com_Handle);
}

int CVICALLBACK CLOSE_Com (int panel, int control, int event,
```

```
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            HidePanel (com_Handle);
            break;
        case EVENT_RIGHT_CLICK:

            break;
        }
    return 0;
}

/*****************************************************************************************************************************/
//         d8888  .d8888b.   .d88888b.  888     888 8888888 .d8888b. 8888888 888888888 8888888 .d88888b.  888b    888
//        d88888 d88P  Y88b d88P" "Y88b 888     888   888  d88P  Y88b  888       888       888  d88P" "Y88b 8888b   888
//       d88P888 888    888 888     888 888     888   888  Y88b.       888       888       888  888     888 88888b  888
//      d88P 888 888        888     888 888     888   888   "Y888b.    888       888       888  888     888 888Y88b 888
//     d88P  888 888        888     888 888     888   888      "Y88b.  888       888       888  888     888 888 Y88b888
//    d88P   888 888    888 888 Y8b 888 888     888   888        "888  888       888       888  888     888 888  Y88888
//   d8888888888 Y88b  d88P Y88b.Y8b88P Y88b. .d88P   888  Y88b  d88P  888       888       888  Y88b. .d88P 888   Y8888
//  d88P     888  "Y8888P"   "Y888888"   "Y88888P"  8888888 "Y8888P" 8888888     888     8888888 "Y88888P"  888    Y888
/*****************************************************************************************************************************/


//*******************************************************
// DSN_SHIFT_DAQ    : Will step data along an array
//                  : For all data that can be plotted
//*******************************************************
void DSN_SHIFT_DAQ(void)
{
    Shift (step, NUM, 1, step);
    Shift (S1A, NUM, 1, S1A);
    Shift (S2A, NUM, 1, S2A);
    Shift (S3A, NUM, 1, S3A);
    Shift (S4A, NUM, 1, S4A);
    Shift (S5A, NUM, 1, S5A);
    Shift (S6A, NUM, 1, S6A);
    //Shift (S7A, NUM, 1, S7A);     //*** Omit while S7A is External Bath Probe
    Current_Radial_Temps[1] = S8A[0];
    Shift (S8A, NUM, 1, S8A);
    Shift (S9A, NUM, 1, S9A);
    Shift (S1B, NUM, 1, S1B);
    Shift (S2B, NUM, 1, S2B);
    Shift (S3B, NUM, 1, S3B);
    Shift (S4B, NUM, 1, S4B);
    Shift (S5B, NUM, 1, S5B);
    Shift (S6B, NUM, 1, S6B);
    Current_Radial_Temps[2] = S7B[0];
    Shift (S7B, NUM, 1, S7B);
    Current_Radial_Temps[0] = S8B[0];
    Shift (S8B, NUM, 1, S8B);
    Shift (S9B, NUM, 1, S9B);
    Shift (Piezo_Temp, NUM, 1, Piezo_Temp);
    Shift (Logging_Trig_Array, NUM, 1, Logging_Trig_Array);

return;
}

//*******************************************************
// DSN_SHIFT_Slow   : Will step data along an array
//                  : For all data that can be plotted
//*******************************************************
void DSN_SHIFT_Slow(int instrument)
{
    switch(instrument)
    {
        case 0:     // Bath Temperature Array
            Shift (Bath_Temp,       NUM, 1, Bath_Temp);
            break;
        case 1:     // Bath Setpoint Array
            Shift (Bath_Setpoint,   NUM, 1, Bath_Setpoint);
            break;
        case 2:     // Average Live Viscosity Array
            Shift (Ave_L_Visc,      NUM, 1, Ave_L_Visc);
            break;
        case 3:     // Average Temperature Corrected Viscosity Array
            Shift (Ave_TC_Visc,     NUM, 1, Ave_TC_Visc);
            break;
        case 4:     // Bulb Temperature Array
            Shift (Bulb_Temp,       NUM, 1, Bulb_Temp);
            break;
        case 5:     // Pressure Controller Setpoint
            Shift (Stat_P_Setpoint, NUM, 1, Stat_P_Setpoint);
            break;
        case 6:     // External Bath Probe
            Current_Radial_Temps[3] = S7A[0];
            Shift (S7A, NUM, 1, S7A);       // Only while S7A is External Bath Probe
            break;
        case 7:     // PACE5000 Pressure
            Shift (PACE_Pressure, NUM, 1, PACE_Pressure);
            break;
    }

return;
}

/*****************************************************************/
/* Slow_ThreadFunction ():  - Separate Thread for Slow Components   */
/*****************************************************************/
int CVICALLBACK Slow_ThreadFunction (void *functionData)
{   double T1,T2,Hz;

    int i, j, k = 0;
    int count = 0;

/*  Start a loop that will process events for this thread */
    while (Slow_quitflag == 1)
    {
            ProcessSystemEvents ();

            //---------------------------------------------

            if(Is_COM1) // Cole Parmer Fluid Bath
```

```
            {

                FlushInQ (SP_comport);
                FlushOutQ(SP_comport);

                DSN_Poll_Bath_Temp();
                DSN_Poll_Bath_Setpoint();
                DSN_Poll_Bath_Probe();

                GetCtrlVal(panelHandle, PANEL_S7A, &Probe_Temp_Trigger);
                Temperature_Error = Probe_Temp_Trigger - wb_setpoint;   // Error

            }
            //---------------------------------------------

            if(Is_COM2) // Viscojet Viscometer
            {

                FlushInQ (SP2_comport);
                FlushOutQ(SP2_comport);

                DSN_Poll_Viscosity();
                DSN_Poll_Bulb_Temperature();

            }
            //---------------------------------------------

            if(Is_COM3) // GE PACE5000 Pressure Controller
            {

                DSN_Poll_Pressure_Setpoint();
                DSN_Poll_Current_Pressure();

                Pressure_Error = (PACE_Pressure_Global - sp_setpoint)/sp_setpoint;  // Error

            }
            //---------------------------------------------

            ProcessSystemEvents ();

            //---------------------------------------------
    }
return 0;
}


/*************************************************************/
/* DAQThreadFunction ():    - Separate Thread for DAQ        */
/*                          - Logs to file/graphs            */
/*************************************************************/
int CVICALLBACK DAQThreadFunction (void *functionData)
{   double T1,T2,Hz;

    double TEMP[10];
    int i, j, k = 0;
    int count = 0;

/*  Start a loop that will process events for this thread */
    while (DAQquitflag == 1)
    {       ++SAMPLES;
            T1 = Timer ();              // Start TIME

            ProcessSystemEvents ();

            //---------------------------------------------

            step[0] = Timer ();         // Get Time

            if(Is_DAQ)
            {
                DSN_Run_DAQ();
                Logging_Trig_Array[0] = Logging_Trigger;
            }

            //---------------------------------------------

            ProcessSystemEvents ();
            SetCtrlVal(panelHandle, PANEL_SAMPLES, SAMPLES);

            //---------------------------------------------

            if(Is_Log)
                DSN_LogFile();          // Log to File
            ProcessSystemEvents ();

            CmtGetLock (DAQ_lockHandle);
            if(Is_Graph)
                DSN_Graph();            // Do Graphing
            CmtReleaseLock (DAQ_lockHandle);

            DSN_SHIFT_DAQ();            // SHIFT all DAQ arrays
            //---------------------------------------------

    }
return 0;
}

//*****************************************************************************
// LOG_ON_OFF       : Check what ports to run and whether to log to a file
//                  : Check current experiment configuration
//*****************************************************************************
int CVICALLBACK LOG_ON_OFF (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            GetCtrlVal(panelHandle, PANEL_LOG_COM3,         &Is_COM3);      // Check Devices to Log
            GetCtrlVal(panelHandle, PANEL_LOG_COM1,         &Is_COM1);
            GetCtrlVal(panelHandle, PANEL_LOG_COM2,         &Is_COM2);
            GetCtrlVal(panelHandle, PANEL_LOG_DAQ,          &Is_DAQ);
            GetCtrlVal(panelHandle, PANEL_LOG_FUNCTION_GEN, &Is_Function_Gen);
            GetCtrlVal(panelHandle, PANEL_LOG_Graph,        &Is_Graph);
            GetCtrlVal(panelHandle, PANEL_LOG_LOGtoFile,    &Is_Log);
```

```
            GetCtrlVal(panelHandle, PANEL_EXP_TYPE,         &Exp_Type);     // Check Experiment Type
            GetCtrlVal(panelHandle, PANEL_RUN_MODE,         &Run_Mode);     // Check Control Type

            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_1A, &Is_S1A);       // Check Sensors to Log
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_1B, &Is_S1B);       // "On" means Pressure Transducer
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_2A, &Is_S2A);       // "Off" means RTD
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_2B, &Is_S2B);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_3A, &Is_S3A);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_3B, &Is_S3B);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_4A, &Is_S4A);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_4B, &Is_S4B);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_5A, &Is_S5A);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_5B, &Is_S5B);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_6A, &Is_S6A);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_6B, &Is_S6B);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_7A, &Is_S7A);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_7B, &Is_S7B);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_8A, &Is_S8A);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_8B, &Is_S8B);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_9A, &Is_S9A);
            GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_9B, &Is_S9B);

            GetCtrlVal(panelHandle, PANEL_Amp_On_1, &Amp_On_1);
            GetCtrlVal(panelHandle, PANEL_Amp_On_2, &Amp_On_2);
            GetCtrlVal(panelHandle, PANEL_Amp_On_3, &Amp_On_3);
            GetCtrlVal(panelHandle, PANEL_Amp_On_4, &Amp_On_4);
            GetCtrlVal(panelHandle, PANEL_Amp_On_5, &Amp_On_5);

            DSN_Get_Sensor_Positions(); // Retrieve Sensor Radial Positions from Screen

            DSN_Update_Graphics();      // Update On-screen Graphics

            break;
        case EVENT_RIGHT_CLICK:

            break;
        }
    return 0;
}

//*********************************************************
// START_STOP       : Start/Stop control of Timer Loop
//                  : Use a separate thread
//*********************************************************
int CVICALLBACK START_STOP (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{   int test, test2;
    int i,j,delay;

    switch (event)
        {
        case EVENT_COMMIT:
            DSN_Save_Vars();
            GetCtrlVal(panelHandle, PANEL_START_STOP, &test2);
            SAMPLES = 0;
            SetCtrlVal(panelHandle, PANEL_SAMPLES, SAMPLES);

            if(!test2)  // Stop Instruments
            {

            DSN_Update_Graphics();

            //--------------
            // STOP DAQ
            //--------------
                if(Is_DAQ)
                {
                DAQquitflag = 0;
                Delay(1);

                // Turn off the Logging Trigger
                Logging_Trigger = 0;
                SetCtrlVal(panelHandle, PANEL_LOGGING_TRIGGER_LED,0);

                DAQmxStopTask(taskHandle);
                DAQmxClearTask(taskHandle);
                DAQ_Task_Started = 0;

            if( DAQdata )
                free(DAQdata);
            if( DAQArray_Out )
                free(DAQArray_Out);

                CmtReleaseThreadPoolFunctionID (DEFAULT_THREAD_POOL_HANDLE, DAQthreadID);
                CmtReleaseThreadPoolFunctionID (DEFAULT_THREAD_POOL_HANDLE, Slow_threadID);

                Delay(1);
                }

            //--------------------------------------
            // STOP COM 1 - Cole Parmer Fluid Bath
            //--------------------------------------
                if(Is_COM1)
                {
                    FlushInQ (SP_comport);
                    FlushOutQ(SP_comport);

                    DSN_Bath_Power_Off();
                }

            //--------------------------------------
            // STOP COM 3 - GE Pressure Controller
            //--------------------------------------
                if(Is_COM3)
                {
                    FlushInQ (SP2_comport);
                    FlushOutQ(SP2_comport);

                    DSN_Shut_Down_P_Controller();

                    Delay(1);

                }
                Delay(1);
                CmtDiscardLock (DAQ_lockHandle);
```

```
}

else    // Start Instruments
{

//----------------------------------
//  Thread for Slow Instruments
//----------------------------------

/*  Start a new thread function in the Default Thread Pool */
Slow_cmtStatus = CmtScheduleThreadPoolFunction (DEFAULT_THREAD_POOL_HANDLE,
                          Slow_ThreadFunction, NULL, &Slow_threadID);
Slow_quitflag = 1;

// Create a thread lock
CmtNewLock (NULL, 0, &Slow_lockHandle);


//--------------------------------------
// SETUP COM 1 - Cole Parmer Fluid Bath
//--------------------------------------

if(Is_COM1)
{
    FlushInQ (SP_comport);
    FlushOutQ(SP_comport);

    DSN_Bath_Power_On();
    bytes_read = ComRdTerm (SP_comport, read_check, 1, 13); // Read ! from COM Port

    if (bytes_read < 0)
    {
        MessagePopup ("Warning", "Fluid Bath did not Communicate");
        DAQquitflag = 0;
        break;
    }

    Delay(2);
    DSN_Poll_Bath_Setpoint();
    // Updates bath setpoint readout once
    SetCtrlVal(panelHandle, PANEL_WB_SETPOINT, wb_setpoint);
    DSN_Set_Bath_External();                                // Set to External Probe
}

//--------------------------------------
// SETUP COM 2 - Viscojet Viscometer
//--------------------------------------
if(Is_COM2)
{
    FlushInQ (SP2_comport);
    FlushOutQ(SP2_comport);

    bytes_read = -1;

    DSN_Poll_Viscosity();

    if (bytes_read < 0)
    {
        MessagePopup ("Warning", "Viscometer did not Communicate");
        DAQquitflag = 0;
        break;
    }
}

//--------------------------------------------
// SETUP COM 3 - GE Pressure Controller
//--------------------------------------------
if(Is_COM3)
{
    FlushInQ (SP3_comport);
    FlushOutQ(SP3_comport);

    DSN_Initialize_P_Controller();

}

//--------------------------------------
// SETUP Function Generator
//--------------------------------------
if(Is_Function_Gen)
{
    DSN_Init_FuncGen();
}

//--------------------------------------
// SETUP DAQ
//--------------------------------------
if(Is_DAQ)
{
    DSN_Setup_DAQ();
}

/*  Start a new thread function in the Default Thread Pool */
DAQcmtStatus = CmtScheduleThreadPoolFunction (DEFAULT_THREAD_POOL_HANDLE,
                          DAQThreadFunction, NULL,
                          &DAQthreadID);
DAQquitflag = 1;

// Create a thread lock
CmtNewLock (NULL, 0, &DAQ_lockHandle);

//--------------------------------------------------
// Undim Monitored Readouts and Adjust Control Types
//--------------------------------------------------
DSN_Update_Graphics();

//--------------------------------------------
// Jump Into Control Loop (if Auto Run Mode)
//--------------------------------------------
if(!Run_Mode)
{
    if(Exp_Type)    // Static Experiment
    {
        // Read in Test Parameters
        GetCtrlVal(panelHandle, PANEL_P_MAX_T, &P_Max_T);
```

```
GetCtrlVal(panelHandle, PANEL_P_MIN_T, &P_Min_T);
GetCtrlVal(panelHandle, PANEL_P_T_INCREMENTS, &P_T_Increments);
GetCtrlVal(panelHandle, PANEL_P_MAX_SP, &P_Max_SP);
GetCtrlVal(panelHandle, PANEL_P_MIN_SP, &P_Min_SP);
GetCtrlVal(panelHandle, PANEL_P_SP_INCREMENTS, &P_SP_Increments);

// Determine Total Number of Experiment Steps
Experiment_Steps = (P_T_Increments + 1)*(P_SP_Increments + 1);

// Repeat the Up-Down Step Program "Num_Cycles" Times
for(Num_Cycles=0 ; Num_Cycles < 1 ; Num_Cycles++)
{
    //--------------------------------------
    // Upward Static Experiment Loop:
    //--------------------------------------
    for(Current_Step=0 ; Current_Step <= P_T_Increments ; Current_Step++)
    {
        ProcessSystemEvents();

        //--------------------------------------
        // Step 1 - Set Next Temperature Point
        //--------------------------------------
                                                    // "+ Current..." for Upward Loop
        wb_setpoint = P_Min_T + Current_Step*((P_Max_T - P_Min_T)/P_T_Increments);
        DSN_Set_Bath_Setpoint();

        //--------------------------------------
        // Step 2 - Let Temperature Stabilize
        //--------------------------------------
        while(Temperature_Error > 1 || Temperature_Error < -1)
        {
            ProcessSystemEvents();

            if(wb_setpoint == 0)        // Error is undefined at 0
            {   // Defined as close enough to "zero"
                if(Probe_Temp_Trigger < 0.1 && Probe_Temp_Trigger > -0.1)
                {
                    Temperature_Error = 0;
                }
            }
        }

        // Underdamped system so exits the above loop at first overshoot

        // System is stable once a peak temperature is within the error bounds

        // Initialize Errors so that we enter the loop
        Upwards_Temp_Error = 99999;
        Downwards_Temp_Error = -99999;

        // Collect NUM PANEL_S7A readings
        for(meas_count = 0 ; meas_count < NUM ; meas_count++ )
        {
            GetCtrlVal(panelHandle, PANEL_S7A, &S7A_Damping_Array[meas_count]);
            Delay(2);
        }
        // Average the NUM readings
        Mean (S7A_Damping_Array, NUM, &Last_Temperature);
        Delay(2);

        while(Upwards_Temp_Error > 0.01 && Downwards_Temp_Error < -0.01)
        {
            /* Damp out high frequency responses */

            // Clear Damping Array
            Clear1D (S7A_Damping_Array, NUM);
            // Collect NUM PANEL_S7A readings
            for(meas_count = 0 ; meas_count < NUM ; meas_count++ )
            {
                GetCtrlVal(panelHandle, PANEL_S7A, &S7A_Damping_Array[meas_count]);
                Delay(2);
            }
            // Average the NUM readings
            Mean (S7A_Damping_Array, NUM, &Latest_Temperature);

            // Compare against the last set of NUM readings to determine increasing or decreasing

            // Temperature is Increasing
            if(Latest_Temperature > Last_Temperature)
            {
                Upwards_Temp_Error = (Latest_Temperature - wb_setpoint)/wb_setpoint;
            }
            // Temperature is Decreasing
            if(Latest_Temperature < Last_Temperature)
            {
                Downwards_Temp_Error = (Latest_Temperature - wb_setpoint)/wb_setpoint;
            }
            // Error is undefined at 0 setpoint
            if(wb_setpoint == 0)
            {
                // Temperature is Increasing
                if(Latest_Temperature > Last_Temperature)
                {   // Not a percent
                    Upwards_Temp_Error = Latest_Temperature - wb_setpoint;
                }
                // Temperature is Decreasing
                if(Latest_Temperature < Last_Temperature)
                {   // Not a percent
                    Downwards_Temp_Error = Latest_Temperature - wb_setpoint;
                }
                // Define as close enough to "zero"
                if(Upwards_Temp_Error < 0.1 && Downwards_Temp_Error > -0.1)
                {
                    Upwards_Temp_Error = 0;
                    Downwards_Temp_Error = 0;
                }
            }

            Last_Temperature = Latest_Temperature;
            ProcessSystemEvents();
        }

        //--------------------------------------
        // Step 3 - Vary Pressure over the Range
        //--------------------------------------
```

```
    if(Is_COM3)
    {
        for(Pressure_Step=0 ; Pressure_Step <= P_SP_Increments ; Pressure_Step++)
        {
            // Calculate Next Pressure Setpoint
            sp_setpoint = P_Min_SP + Pressure_Step*((P_Max_SP- P_Min_SP)/P_SP_Increments);

            // Send and Adjust to New Setpoint
            DSN_Change_P_Setpoint(sp_setpoint);

            // Initialize Errors so that we enter the loop
            Upwards_Press_Error = 99999;
            Downwards_Press_Error = -99999;
            DSN_Poll_Current_Pressure();
            GetCtrlVal(panelHandle, PANEL_REGULATOR_PRESSURE, &Last_Pressure);
            Delay(2);

            // Let Pressure Stabilize
            while(Upwards_Press_Error > 0.01 && Downwards_Press_Error < -0.01)
            {
                DSN_Poll_Current_Pressure();
                GetCtrlVal(panelHandle, PANEL_REGULATOR_PRESSURE, &Latest_Pressure);

                // Pressure is Increasing
                if(Latest_Pressure > Last_Pressure)
                {
                    Upwards_Press_Error = (Latest_Pressure - sp_setpoint)/sp_setpoint;
                }
                // Pressure is Decreasing
                if(Latest_Pressure < Last_Pressure)
                {
                    Downwards_Press_Error = (Latest_Pressure - sp_setpoint)/sp_setpoint;
                }
                // Error is undefined at 0 setpoint
                if(sp_setpoint == 0)
                {
                    // Pressure is Increasing
                    if(Latest_Pressure > Last_Pressure)
                    {   // Not a percent
                        Upwards_Press_Error = Latest_Pressure - sp_setpoint;
                    }
                    // Pressure is Decreasing
                    if(Latest_Pressure < Last_Pressure)
                    {   // Not a percent
                        Downwards_Press_Error = Latest_Pressure - sp_setpoint;
                    }
                    // Defined as close enough to "zero"
                    if(Upwards_Press_Error < 0.1 && Downwards_Press_Error > -0.1)
                    {
                        Upwards_Press_Error = 0;
                        Downwards_Press_Error = 0;
                    }
                }

                Last_Pressure = Latest_Pressure;
                ProcessSystemEvents();
            }

            // Turn on Logging Trigger
            Logging_Trigger = 1;
            SetCtrlVal(panelHandle, PANEL_LOGGING_TRIGGER_LED,1);

            // Delay for a time at Setpoint
            for(delay=1 ; delay <= 600 ; delay++)
            {
                Delay(1);
                ProcessSystemEvents ();
            }

            // Turn off the Logging Trigger
            Logging_Trigger = 0;
            SetCtrlVal(panelHandle, PANEL_LOGGING_TRIGGER_LED,0);
        }
    }

    // Vent the System Pressure
    if(Is_COM3)
    {
        DSN_Vent_Pressure();
    }

    //---------------------------------------
    // Step 4 - Update Progress Bar
    //---------------------------------------
    //Percent_Complete = 50*(Current_Step+1)/Experiment_Steps;
    //SetCtrlVal(panelHandle, PANEL_EXP_PROGRESS, Percent_Complete);
}


//---------------------------------------
// Downward Static Experiment Loop:
//---------------------------------------
for(Current_Step=0 ; Current_Step <= P_T_Increments ; Current_Step++)
{
    ProcessSystemEvents ();

    //---------------------------------------
    // Step 1 - Set Next Temperature Point
    //---------------------------------------
                                                // "- Current..." for Downward Loop
    wb_setpoint = P_Max_T - Current_Step*((P_Max_T - P_Min_T)/P_T_Increments);
    DSN_Set_Bath_Setpoint();

    //---------------------------------------
    // Step 2 - Let Temperature Stabilize
    //---------------------------------------
    while(Temperature_Error > 1 || Temperature_Error < -1)
    {
        ProcessSystemEvents();

        if(wb_setpoint == 0)        // Error is undefined at 0
        {   // Defined as close enough to "zero"
            if(Probe_Temp_Trigger < 1 && Probe_Temp_Trigger > -1)
            {
                Temperature_Error = 0;
```

```
        }
    }
}

// Underdamped system so exits the above loop at first overshoot
// System is stable once a peak temperature is within the error bounds

// Initialize Errors so that we enter the loop
Upwards_Temp_Error = 99999;
Downwards_Temp_Error = -99999;

// Collect NUM PANEL_S7A readings
for(meas_count = 0 ; meas_count < NUM ; meas_count++ )
{
    GetCtrlVal(panelHandle, PANEL_S7A, &S7A_Damping_Array[meas_count]);
    Delay(2);
}
// Average the NUM readings
Mean (S7A_Damping_Array, NUM, &Last_Temperature);

Delay(2);

while(Upwards_Temp_Error > 0.01 && Downwards_Temp_Error < -0.01)
{
    /* Damp out high frequency responses */

    // Clear Damping Array
    Clear1D (S7A_Damping_Array, NUM);
    // Collect NUM PANEL_S7A readings
    for(meas_count = 0 ; meas_count < NUM ; meas_count++ )
    {
        GetCtrlVal(panelHandle, PANEL_S7A, &S7A_Damping_Array[meas_count]);
        Delay(2);
    }
    // Average the NUM readings
    Mean (S7A_Damping_Array, NUM, &Latest_Temperature);

    // Compare against the last set of NUM readings to determine increasing or decreasing


    // Temperature is Increasing
    if(Latest_Temperature > Last_Temperature)
    {
        Upwards_Temp_Error = (Latest_Temperature - wb_setpoint)/wb_setpoint;
    }
    // Temperature is Decreasing
    if(Latest_Temperature < Last_Temperature)
    {
        Downwards_Temp_Error = (Latest_Temperature - wb_setpoint)/wb_setpoint;
    }
    // Error is undefined at 0 setpoint
    if(wb_setpoint == 0)
    {
        // Temperature is Increasing
        if(Latest_Temperature > Last_Temperature)
        {
            Upwards_Temp_Error = Latest_Temperature - wb_setpoint;  // Not a percent
        }
        // Temperature is Decreasing
        if(Latest_Temperature < Last_Temperature)
        {
            Downwards_Temp_Error = Latest_Temperature - wb_setpoint;// Not a percent
        }
        // Defined as close enough to "zero"
        if(Upwards_Temp_Error < 0.1 && Downwards_Temp_Error > -0.1)
        {
            Upwards_Temp_Error = 0;
            Downwards_Temp_Error = 0;
        }
    }

    Last_Temperature = Latest_Temperature;
    ProcessSystemEvents();
}

//--------------------------------------
// Step 3 - Vary Pressure over the Range
//--------------------------------------
if(Is_COM3)
{
    for(Pressure_Step=0 ; Pressure_Step <= P_SP_Increments ; Pressure_Step++)
    {
        // Calculate Next Pressure Setpoint
        sp_setpoint = P_Min_SP + Pressure_Step*((P_Max_SP- P_Min_SP)/P_SP_Increments);

        // Send and Adjust to New Setpoint
        DSN_Change_P_Setpoint(sp_setpoint);

        // Initialize Errors so that we enter the loop
        Upwards_Press_Error = 99999;
        Downwards_Press_Error = -99999;
        DSN_Poll_Current_Pressure();
        GetCtrlVal(panelHandle, PANEL_REGULATOR_PRESSURE, &Last_Pressure);
        Delay(2);

        // Let Pressure Stabilize
        while(Upwards_Press_Error > 0.01 && Downwards_Press_Error < -0.01)
        {
            DSN_Poll_Current_Pressure();
            GetCtrlVal(panelHandle, PANEL_REGULATOR_PRESSURE, &Latest_Pressure);

            // Pressure is Increasing
            if(Latest_Pressure > Last_Pressure)
            {
                Upwards_Press_Error = (Latest_Pressure - sp_setpoint)/sp_setpoint;
            }
            // Pressure is Decreasing
            if(Latest_Pressure < Last_Pressure)
            {
                Downwards_Press_Error = (Latest_Pressure - sp_setpoint)/sp_setpoint;
            }
            // Error is undefined at 0 setpoint
            if(sp_setpoint == 0)
            {
                // Pressure is Increasing
```

```
                            if(Latest_Pressure > Last_Pressure)
                            {
                                Upwards_Press_Error = Latest_Pressure - sp_setpoint;    // Not a percent
                            }
                            // Pressure is Decreasing
                            if(Latest_Pressure < Last_Pressure)
                            {
                                Downwards_Press_Error = Latest_Pressure - sp_setpoint;  // Not a percent
                            }
                            // Defined as close enough to "zero"
                            if(Upwards_Press_Error < 0.1 && Downwards_Press_Error > -0.1)
                            {
                                Upwards_Press_Error = 0;
                                Downwards_Press_Error = 0;
                            }
                        }

                        Last_Pressure = Latest_Pressure;
                        ProcessSystemEvents();
                    }

                    // Turn on Logging Trigger
                    Logging_Trigger = 1;
                    SetCtrlVal(panelHandle, PANEL_LOGGING_TRIGGER_LED,1);

                    // Delay for a time at Setpoint
                    for(delay=1 ; delay <= 600 ; delay++)
                    {
                        Delay(1);
                        ProcessSystemEvents ();
                    }

                    // Turn off the Logging Trigger
                    Logging_Trigger = 0;
                    SetCtrlVal(panelHandle, PANEL_LOGGING_TRIGGER_LED,0);
                }
            }

            // Vent the System Pressure
            if(Is_COM3)
            {
                DSN_Vent_Pressure();
            }

            //---------------------------------------
            // Step 4 - Update Progress Bar
            //---------------------------------------
            //Percent_Complete = 100*(Current_Step+1)/Experiment_Steps;
            //SetCtrlVal(panelHandle, PANEL_EXP_PROGRESS, Percent_Complete);
        }
    }

    wb_setpoint = 20;  // Setpoint to 20 once experiment is completed (in case it doesn't shut down)

    printf("Successfully Exited the Loop");

    DSN_Set_Bath_Setpoint();
    DAQquitflag = 0; // Shut down experiment
    DSN_Bath_Power_Off();
}

// ********************************* //

else    // Dynamic Experiment
{
    // Read in Test Parameters
    GetCtrlVal(panelHandle, PANEL_P_MAX_T, &P_Max_T);
    GetCtrlVal(panelHandle, PANEL_P_MIN_T, &P_Min_T);
    GetCtrlVal(panelHandle, PANEL_P_T_INCREMENTS, &P_T_Increments);
    GetCtrlVal(panelHandle, PANEL_DP_STIM_DURATION, &DP_Stim_Duration);
    GetCtrlVal(panelHandle, PANEL_MTS_SPAN, &MTS_Span);
    GetCtrlVal(panelHandle, PANEL_P_MAX_F, &P_Max_F);
    GetCtrlVal(panelHandle, PANEL_P_MIN_F, &P_Min_F);
    GetCtrlVal(panelHandle, PANEL_P_F_INCREMENTS, &P_F_Increments);

    GetCtrlVal(panelHandle, PANEL_DWELL_TIME, &StimDwellTime);
    GetCtrlVal(panelHandle, PANEL_Waveform_1, &FuncGen_Waveform);

    Num_Amp_Cycles = 0;     // Set default number of amplitude cycles to zero

    // Read in Dynamic Pressure Amplitude Parameters
    if (Amp_On_1)
    {
        GetCtrlVal(panelHandle, PANEL_P_DP_AMP_1, &DP_Amp_Array[Num_Amp_Cycles]);
        GetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_1, &FuncGenVolt_Array[Num_Amp_Cycles]);
        Num_Amp_Cycles++;
    }
    if (Amp_On_2)
    {
        GetCtrlVal(panelHandle, PANEL_P_DP_AMP_2, &DP_Amp_Array[Num_Amp_Cycles]);
        GetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_2, &FuncGenVolt_Array[Num_Amp_Cycles]);
        Num_Amp_Cycles++;
    }
    if (Amp_On_3)
    {
        GetCtrlVal(panelHandle, PANEL_P_DP_AMP_3, &DP_Amp_Array[Num_Amp_Cycles]);
        GetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_3, &FuncGenVolt_Array[Num_Amp_Cycles]);
        Num_Amp_Cycles++;
    }
    if (Amp_On_4)
    {
        GetCtrlVal(panelHandle, PANEL_P_DP_AMP_4, &DP_Amp_Array[Num_Amp_Cycles]);
        GetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_4, &FuncGenVolt_Array[Num_Amp_Cycles]);
        Num_Amp_Cycles++;
    }
    if (Amp_On_5)
    {
        GetCtrlVal(panelHandle, PANEL_P_DP_AMP_5, &DP_Amp_Array[Num_Amp_Cycles]);
        GetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_5, &FuncGenVolt_Array[Num_Amp_Cycles]);
        Num_Amp_Cycles++;
    }


    // Experiment Loop:
    // Notes:   For safety, loop only works for a single temperature setpoint
```

```
//          Reconfigure MTS Machine before changing temperature setpoints

    // 1. Set the Amplitude Setpoint
    // 2. Set the Temperature Setpoint
    // 3. Wait till setpoint is reached
    // 4. Run the stimulation cycle at one frequency setpoint
    // 5. Update Progress Bar
    // 6. Dwell Period
    // 7. Update Progress Bar
    // 8. Repeat for all frequencies
    // 9. Repeat for all amplitudes with a dwell between amplitudes

//-------------------------------------
// Dynamic Experiment Loop:
//-------------------------------------
for(Amplitude_Cycle = 0 ; Amplitude_Cycle <= Num_Amp_Cycles - 1 ; Amplitude_Cycle++)    // Amplitude Loop
    {
        // Set Percentage Complete to Zero on First Run
        if (Amplitude_Cycle == 0)
        {
            Percent_Complete = 0;
            SetCtrlVal(panelHandle, PANEL_EXP_PROGRESS, Percent_Complete);
        }
        else // Changing to next amplitude -> Dwell for 30 minutes to recover viscosity
        {
            Logging_Trigger=3;
            for(Current_Step=0 ; Current_Step<=1800 ; Current_Step++)
                {
                    Delay(1);
                }
        }

        Total_Exp_Time = Num_Amp_Cycles * (P_F_Increments+1) * (DP_Stim_Duration + StimDwellTime);

        for(Current_Step=0 ; Current_Step <= P_F_Increments ; Current_Step++)   // Frequency Loop
        {

            ProcessSystemEvents();

            // Switch the Logging Trigger
            Logging_Trigger = 0;

            //-------------------------------------
            // Step 1 - Set Temperature Setpoint
            //-------------------------------------
            if(Is_COM1)
            {
                wb_setpoint = P_Min_T;
                DSN_Set_Bath_Setpoint();
            }
            //-------------------------------------
            // Step 2 - Let Temperature Stabilize
            //-------------------------------------
            if(Is_COM1)
            {
                // Indicate setpoint not reached
                SetCtrlVal(panelHandle, PANEL_WAITING_TEMP_LED, 1);

                while(Temperature_Error > 0.1 || Temperature_Error < -0.1)
                {
                    ProcessSystemEvents();

                    if(wb_setpoint == 0)        // Error is undefined at 0
                    {   // Defined as close enough to "zero"
                        if(Probe_Temp_Trigger < 1 && Probe_Temp_Trigger > 1)
                        {
                            Temperature_Error = 0;
                        }
                    }
                }
                // Indicate setpoint reached
                SetCtrlVal(panelHandle, PANEL_WAITING_TEMP_LED, 0);
            }

            //-----------------------------------------------------------
            // Step 3 - Run Stimulation Cycle at One Frequency Setpoint
            //-----------------------------------------------------------

            // Determine Amplitude Setpoint and Voltage
            DP_Amplitude = DP_Amp_Array[Amplitude_Cycle];
            FuncGenVoltage = FuncGenVolt_Array[Amplitude_Cycle];
            SetCtrlVal(panelHandle, PANEL_CURR_AMP_STEP, DP_Amplitude);

            // Determine Frequency Setpoint
            freq_setpoint = P_Min_F + Current_Step*((P_Max_F - P_Min_F)/P_F_Increments);
            SetCtrlVal(panelHandle, PANEL_CURR_FREQ_STEP, freq_setpoint);

            // Decide Whether to Run the Function Generator Output
            if (freq_setpoint == 0 || DP_Amplitude == 0)
            {
                FuncGenVoltage = 0;
                No_FuncOutput = 1;
            }
            else
            {
                No_FuncOutput = 0;
            }

            // Update GUI with Function Generator Outputs
            SetCtrlVal (panelHandle, PANEL_VOLTAGE, FuncGenVoltage);
            SetCtrlVal (panelHandle, PANEL_FREQUENCY,freq_setpoint);

            // Switch the Logging Trigger
            Logging_Trigger = 1;
            SetCtrlVal(panelHandle, PANEL_LOGGING_TRIGGER_LED,1);

            if (No_FuncOutput == 1)
            {
                // Do not run function generator output
            }
            else
            {
                // Configure and Turn On Function Generator Channel 1
                DSN_Toggle_FuncGen_On();
            }
```

```
                                // Run Frequency Stimulation for Specified Duration
                                for(delay=1 ; delay <= DP_Stim_Duration ; delay++)
                                {
                                    Delay(1);
                                    ProcessSystemEvents ();
                                }

                                // Turn Off the Function Generator Output
                                DSN_Toggle_FuncGen_Off();

                                //--------------------------------------
                                // Step 4 - Update Progress Bar
                                //--------------------------------------
                                Percent_Complete = 100*(DP_Stim_Duration + (Current_Step)*(DP_Stim_Duration + StimDwellTime)
                                        + Amplitude_Cycle*(P_F_Increments+1)*(DP_Stim_Duration + StimDwellTime))/Total_Exp_Time;
                                SetCtrlVal(panelHandle, PANEL_EXP_PROGRESS, Percent_Complete);

                                //------------------------
                                // Step 5 - Dwell Period
                                //------------------------

                                // Switch the Logging Trigger
                                Logging_Trigger = 2;
                                SetCtrlVal(panelHandle, PANEL_LOGGING_TRIGGER_LED,0);
                                SetCtrlVal(panelHandle, PANEL_THIXOTROPY_LED, 1);

                                FuncGenVoltage = 0;
                                freq_setpoint = 0;
                                SetCtrlVal (panelHandle, PANEL_VOLTAGE, FuncGenVoltage);
                                SetCtrlVal (panelHandle, PANEL_FREQUENCY,freq_setpoint);

                                // Delay for Dwell Period
                                for(delay=1 ; delay <= StimDwellTime ; delay++)
                                {
                                    Delay(1);
                                    ProcessSystemEvents ();
                                }

                                SetCtrlVal(panelHandle, PANEL_THIXOTROPY_LED, 0);

                                //--------------------------------------
                                // Step 6 - Update Progress Bar
                                //--------------------------------------
                                Percent_Complete = 100*(((Current_Step+1)*(DP_Stim_Duration + StimDwellTime)
                                        + Amplitude_Cycle*(P_F_Increments+1)*(DP_Stim_Duration + StimDwellTime))/Total_Exp_Time);
                                SetCtrlVal(panelHandle, PANEL_EXP_PROGRESS, Percent_Complete);

                            }
                        }

                    DAQquitflag = 0; // Shut down experiment
                    DSN_Bath_Power_Off();
                    SetCtrlVal(panelHandle, PANEL_START_STOP, 0);
                    DSN_Update_Graphics();
                    //-------------------------------------
                    // Update On-screen Graphics
                    //-------------------------------------
                    DSN_Update_Graphics();
                    MessagePopup ("Monitoring and Control Program", "The experiment was completed successfully.");

                }
            }


        case EVENT_RIGHT_CLICK:

            break;
        }
        }
    return 0;
}

/*****************************************************************************************************************/
// 8888888 888b    888  .d8888b. 88888888888 8888888b.  888     888 888b     d888 8888888888 888b    888 88888888888
//   888   8888b   888 d88P  Y88b    888     888   Y88b 888     888 8888b   d8888 888        8888b   888     888
//   888   88888b  888 Y88b.         888     888    888 888     888 88888b.d88888 888        88888b  888     888
//   888   888Y88b 888  "Y888b.      888     888   d88P 888     888 888Y88888P888 8888888    888Y88b 888     888
//   888   888 Y88b888     "Y88b.    888     8888888P"  888     888 888 Y888P 888 888        888 Y88b888     888
//   888   888  Y88888       "888    888     888 T88b   888     888 888  Y8P  888 888        888  Y88888     888
//   888   888   Y8888 Y88b  d88P    888     888  T88b  Y88b. .d88P 888   "   888 888        888   Y8888     888
// 8888888 888    Y888  "Y8888P"     888     888   T88b  "Y88888P"  888       888 8888888888 888    Y888     888
//
//
//
// 8888888888 888     888 888b    888  .d8888b. 88888888888 8888888 .d88888b.  888b    888  .d8888b.
// 888        888     888 8888b   888 d88P  Y88b    888       888  d88P" "Y88b 8888b   888 d88P  Y88b
// 888        888     888 88888b  888 888    888    888       888  888     888 88888b  888 Y88b.
// 8888888    888     888 888Y88b 888 888           888       888  888     888 888Y88b 888  "Y888b.
// 888        888     888 888 Y88b888 888           888       888  888     888 888 Y88b888     "Y88b.
// 888        888     888 888  Y88888 888    888    888       888  888     888 888  Y88888       "888
// 888        Y88b. .d88P 888   Y8888 Y88b  d88P    888       888  Y88b. .d88P 888   Y8888 Y88b  d88P
// 888         "Y88888P"  888    Y888  "Y8888P"     888     8888888 "Y88888P"  888    Y888  "Y8888P"
/*****************************************************************************************************************/


//*************************************/
//  National Instruments Compact DAQ     /
//*************************************/

//-------------------------------------------------------
// DSN_Setup_DAQ    : Sets up the DAQ Task and RTD Channels
//-------------------------------------------------------

void DSN_Setup_DAQ(void)
{

    int i,count=0;

    // Retrieve Averaging Number and Sampling Rate from Panel

    GetCtrlVal(panelHandle,PANEL_DAQ_AVERAGING_NUMBER,&DAQsampsPerChan);
    GetCtrlVal(panelHandle,PANEL_DAQ_RATE,&DAQrate);

    //*************************************/
```

```
// Create the DAQ task                       /
//*****************************************/

DAQmxErrChk (DAQmxCreateTask("",&taskHandle));

//*****************************************/
// Create all the Necessary Channels        /
//*****************************************/

//----------------------------------------
if(Is_S1A)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai0",
        "Voltage_1A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod1/ai0",
        "Temperature_1A", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S1B)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai1",
        "Voltage_1B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod1/ai1",
        "Temperature_1B", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S2A)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai2",
        "Voltage_2A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod1/ai2",
        "Temperature_2A", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S2B)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai3",
        "Voltage_2B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod1/ai3",
        "Temperature_2B", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S3A)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai4",
        "Voltage_3A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod2/ai0",
        "Temperature_3A", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S3B)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai5",
        "Voltage_3B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod2/ai1",
        "Temperature_3B", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S4A)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai6",
        "Voltage_4A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod2/ai2",
        "Temperature_4A", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S4B)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai7",
        "Voltage_4B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
```

```
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod2/ai3",
        "Temperature_4B", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S5A)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai8",
        "Voltage_5A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod3/ai0",
        "Temperature_5A", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S5B)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai9",
        "Voltage_5B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod3/ai1",
        "Temperature_5B", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S6A)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai10",
        "Voltage_6A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod3/ai2",
        "Temperature_6A", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S6B)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai11",
        "Voltage_6B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod3/ai3",
        "Temperature_6B", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S7A)  // Create channel but do not use its data while external probe is in position S7A!!
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai12",
        "Voltage_7A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod4/ai0",
        "Temperature_7A", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S7B)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai13",
        "Voltage_7B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod4/ai1",
        "Temperature_7B", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S8A)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai14",
        "Voltage_8A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod4/ai2",
        "Temperature_8A", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
        DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
}
//----------------------------------------
if(Is_S8B)
{
    // Sensor is a Pressure Transducer
    DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai15",
        "Voltage_8B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
}
else
{
     // Sensor is an RTD
    DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod4/ai3",
```

```
            "Temperature_8B", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
            DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
    }
    //-----------------------------------------
    if(Is_S9A)
    {
        // Sensor is a Pressure Transducer
        DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai16",
            "Voltage_9A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
    }
    else
    {
         // Sensor is an RTD
        DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod5/ai0",
            "Temperature_9A", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
            DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
    }
    //-----------------------------------------
    if(Is_S9B)
    {
        // Sensor is a Pressure Transducer
        DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, "cDAQ1Mod6/ai17",
            "Voltage_9B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
    }
    else
    {
         // Sensor is an RTD
        DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod5/ai1",
            "Temperature_9B", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
            DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
    }
    //-----------------------------------------
    if(Exp_Type)
    {
        // Static Experiment
    }
    else
    {
        // Dynamic Experiment - Create Piezo RTD Channel
        DAQmxErrChk(DAQmxCreateAIRTDChan (taskHandle, "cDAQ1Mod5/ai3",
            "Piezo_Temp", 0, 150, DAQmx_Val_DegC, DAQmx_Val_Pt3851,
            DAQmx_Val_3Wire, DAQmx_Val_Internal, 0.001, 100));
    }

    //*********************************************/
    // Setup the DAQ Timing and Number of Channels  /
    //*********************************************/

    // Set the clock timing
    DAQmxErrChk (DAQmxCfgSampClkTiming (taskHandle,"", DAQrate,
                        DAQmx_Val_Rising, DAQmx_Val_ContSamps, DAQsampsPerChan));

    // Set the number of channels to scan
    DAQmxErrChk (DAQmxGetTaskAttribute(taskHandle,DAQmx_Task_NumChans,&DAQnumChannels));

    // Make space for the data
    if( (DAQdata=malloc(DAQsampsPerChan*DAQnumChannels*sizeof(float64)))==NULL ) {
        MessagePopup("Error","Not enough memory");
        goto Error;
        }
    // Make space for the averaging aray
    if( (DAQArray_Out=malloc(DAQsampsPerChan*sizeof(float64)))==NULL ) {
        MessagePopup("Error","Not enough memory");
        goto Error;
        }

    //Exit
    Error:
        if( DAQmxFailed(DAQerror) )
            {
                DAQmxGetExtendedErrorInfo(DAQerrBuff,2048);
                DAQquitflag =0;
            }

        if( DAQmxFailed(DAQerror) )
                MessagePopup("DAQmx Error",DAQerrBuff);
}

//-----------------------------------------------------------
// DSN_Run_DAQ  : Run the DAQ Task and RTD Channels
//-----------------------------------------------------------

void DSN_Run_DAQ(void)
{
    int i;
    double pressure_scaling_factor = 200;   // psi per Volt

    /*********************************************/
    // DAQmx Start DAQ Task                       /
    /*********************************************/

    if(DAQ_Task_Started)
    {
        // Do not attempt to start task
    }
    else
    {
        DAQmxErrChk (DAQmxStartTask(taskHandle));
    }

    DAQ_Task_Started = 1;
    ProcessDrawEvents();

    /*********************************************/
    // DAQmx Read Code                            /
    /*********************************************/

    Clear1D (DAQdata, DAQsampsPerChan*DAQnumChannels);
    DAQmxReadAnalogF64(taskHandle,DAQsampsPerChan,10.0,DAQmx_Val_GroupByChannel,
                        DAQdata,DAQsampsPerChan*DAQnumChannels,&DAQnumRead,NULL);

        if( DAQnumRead>0 )
            for(i=0;i<DAQnumChannels;i++)
            {
                // Extract Data for one Channel
```

```
Subset1D (DAQdata, DAQnumChannels*DAQnumRead, i*DAQnumRead, DAQnumRead, DAQArray_Out);

// Average the Data Points
Mean (DAQArray_Out,DAQnumRead, &DAQoutMean);

switch(i)
{
    case 0:
        if(Is_S1A)  // Pressure
        {
            DAQoutMean = DAQoutMean * pressure_scaling_factor;
        }
        SetCtrlVal(panelHandle,PANEL_S1A,DAQoutMean);
        S1A[0] = DAQoutMean;
        break;
    case 1:
        if(Is_S1B)  // Pressure
        {
            DAQoutMean = DAQoutMean * pressure_scaling_factor;
        }
        SetCtrlVal(panelHandle,PANEL_S1B,DAQoutMean);
        S1B[0] = DAQoutMean;
        break;
    case 2:
        if(Is_S2A)  // Pressure
        {
            DAQoutMean = DAQoutMean * pressure_scaling_factor;
        }
        SetCtrlVal(panelHandle,PANEL_S2A,DAQoutMean);
        S2A[0] = DAQoutMean;
        break;
    case 3:
        if(Is_S2B)  // Pressure
        {
            DAQoutMean = DAQoutMean * pressure_scaling_factor;
        }
        SetCtrlVal(panelHandle,PANEL_S2B,DAQoutMean);
        S2B[0] = DAQoutMean;
        break;
    case 4:
        if(Is_S3A)  // Pressure
        {
            DAQoutMean = DAQoutMean * pressure_scaling_factor;
        }
        SetCtrlVal(panelHandle,PANEL_S3A,DAQoutMean);
        S3A[0] = DAQoutMean;
        break;
    case 5:
        if(Is_S3B)  // Pressure
        {
            DAQoutMean = DAQoutMean * pressure_scaling_factor;
        }
        SetCtrlVal(panelHandle,PANEL_S3B,DAQoutMean);
        S3B[0] = DAQoutMean;
        break;
    case 6:
        if(Is_S4A)  // Pressure
        {
            DAQoutMean = DAQoutMean * pressure_scaling_factor;
        }
        SetCtrlVal(panelHandle,PANEL_S4A,DAQoutMean);
        S4A[0] = DAQoutMean;
        break;
    case 7:
        if(Is_S4B)  // Pressure
        {
            DAQoutMean = DAQoutMean * pressure_scaling_factor;
        }
        SetCtrlVal(panelHandle,PANEL_S4B,DAQoutMean);
        S4B[0] = DAQoutMean;
        break;
    case 8:
        if(Is_S5A)  // Pressure
        {
            DAQoutMean = DAQoutMean * pressure_scaling_factor;
        }
        SetCtrlVal(panelHandle,PANEL_S5A,DAQoutMean);
        S5A[0] = DAQoutMean;
        break;
    case 9:
        if(Is_S5B)  // Pressure
        {
            DAQoutMean = DAQoutMean * pressure_scaling_factor;
        }
        SetCtrlVal(panelHandle,PANEL_S5B,DAQoutMean);
        S5B[0] = DAQoutMean;
        break;
    case 10:
        if(Is_S6A)  // Pressure
        {
            DAQoutMean = DAQoutMean * pressure_scaling_factor;
        }
        SetCtrlVal(panelHandle,PANEL_S6A,DAQoutMean);
        S6A[0] = DAQoutMean;
        break;
    case 11:
        if(Is_S6B)  // Pressure
        {
            DAQoutMean = DAQoutMean * pressure_scaling_factor;
        }
        SetCtrlVal(panelHandle,PANEL_S6B,DAQoutMean);
        S6B[0] = DAQoutMean;
        break;
    case 12:
        /* Since external bath probe is in position S7A, we do not use
        the data from this channel.  The channel was created to make sure
        all other channels still fit this "switch" statement.  Temperature
        for S7A is obtained using the DSN_Poll_Bath_Probe function in the
        slow equipment loop.  Uncomment this section if the probe is moved
        to another position. */

//      if(Is_S7A)  // Pressure
//      {
//          DAQoutMean = DAQoutMean * pressure_scaling_factor;
//      }
```

```
//                      SetCtrlVal(panelHandle,PANEL_S7A,DAQoutMean);
//                      S7A[0] = DAQoutMean;
                        break;
                    case 13:
                        if(Is_S7B)  // Pressure
                        {
                            DAQoutMean = DAQoutMean * pressure_scaling_factor;
                        }
                        SetCtrlVal(panelHandle,PANEL_S7B,DAQoutMean);
                        S7B[0] = DAQoutMean;
                        break;
                    case 14:
                        if(Is_S8A)  // Pressure
                        {
                            DAQoutMean = DAQoutMean * pressure_scaling_factor;
                        }
                        SetCtrlVal(panelHandle,PANEL_S8A,DAQoutMean);
                        S8A[0] = DAQoutMean;
                        break;
                    case 15:
                        if(Is_S8B)  // Pressure
                        {
                            DAQoutMean = DAQoutMean * pressure_scaling_factor;
                        }
                        SetCtrlVal(panelHandle,PANEL_S8B,DAQoutMean);
                        S8B[0] = DAQoutMean;
                        break;
                    case 16:
                        if(Is_S9A)  // Pressure
                        {
                            DAQoutMean = DAQoutMean * pressure_scaling_factor;
                        }
                        SetCtrlVal(panelHandle,PANEL_S9A,DAQoutMean);
                        S9A[0] = DAQoutMean;
                        break;
                    case 17:
                        if(Is_S9B)  // Pressure
                        {
                            DAQoutMean = DAQoutMean * pressure_scaling_factor;
                        }
                        SetCtrlVal(panelHandle,PANEL_S9B,DAQoutMean);
                        S9B[0] = DAQoutMean;
                        break;
                    case 18:
                        SetCtrlVal(panelHandle,PANEL_PIEZO_TEMP,DAQoutMean);
                        Piezo_Temp[0] = DAQoutMean;
                        break;
                }
            }
//Exit
Error:
    if( DAQmxFailed(DAQerror) )
        {
            DAQmxGetExtendedErrorInfo(DAQerrBuff,2048);
            DAQquitflag =0;
        }
}

//**********************************
//  Cole Parmer Fluid Bath
//**********************************

//-------------------------------------------------------
// DSN_Bath_Power_On    : Powers ON the fluid bath
//-------------------------------------------------------

void DSN_Bath_Power_On(void)
{
    char* power_on = "SO1\r";   // Define Command to Power On

    FlushInQ(SP_comport);       // Flush the Input and Output Queue
    FlushOutQ(SP_comport);

    stringsize = StringLength (power_on);
    ComWrt (SP_comport, power_on, stringsize);  // Powers On

    Is_Bath = 1;    // Define Bath Power Global as "On"
}

//-------------------------------------------------------
// DSN_Bath_Power_Off   : Powers OFF the fluid bath
//-------------------------------------------------------

void DSN_Bath_Power_Off(void)
{
    char* power_off = "SO0\r";  // Define Command to Power On

    FlushInQ(SP_comport);           // Flush the Input and Output Queue
    FlushOutQ(SP_comport);

    stringsize = StringLength (power_off);
    ComWrt (SP_comport, power_off, stringsize); // Powers Off

    Is_Bath = 0;    // Define Bath Power Global as "Off"
}

//-------------------------------------------------------
// DSN_Poll_Bath_Temp   : Polls Internal Bath Temperature
//-------------------------------------------------------

void DSN_Poll_Bath_Temp(void)
{
    char*   ask_temp = "RT\r",      // Command to Request Internal Temperature
            ascii_temp[20];         // ASCII response to the temperature query

    double bath_temp;               // Decimal Value

    FlushInQ(SP_comport);           // Flush the Input and Output Queue
    FlushOutQ(SP_comport);

    stringsize = StringLength (ask_temp);
    ComWrt (SP_comport, ask_temp, stringsize);              // Send temperature request to COM Port

    ascii_temp[0] = '\0';
    bytes_read = ComRdTerm (SP_comport, ascii_temp, 9, 13); // Read incoming temperature from COM Port
```

```
    bath_temp = atof(ascii_temp);                           // Convert ASCII to Double Format

    SetCtrlVal(panelHandle, PANEL_WB_TEMP, bath_temp);      // Updates bath temperature on screen

    CmtGetLock (Slow_lockHandle);
    DSN_SHIFT_Slow(0);
    Bath_Temp[0] = bath_temp;                               // Log value in Bath Temperature Array
    CmtReleaseLock (Slow_lockHandle);
}

//---------------------------------------------------------
// DSN_Poll_Bath_Setpoint   : Polls Fluid Bath Setpoint
//---------------------------------------------------------

void DSN_Poll_Bath_Setpoint(void)
{
    char*   ask_setpoint = "RS\r",  // Command to Request Setpoint Temperature
            ascii_setpoint[20];     // ASCII response to the setpoint query

    FlushInQ(SP_comport);           // Flush the Input and Output Queue
    FlushOutQ(SP_comport);

    stringsize = StringLength (ask_setpoint);
    ComWrt (SP_comport, ask_setpoint, stringsize);              // Send setpoint request to COM Port

    ascii_setpoint[0] = '\0';
    bytes_read = ComRdTerm (SP_comport, ascii_setpoint, 9, 13); // Read incoming setpoint from COM Port

    wb_setpoint = atof(ascii_setpoint);                         // Convert ASCII to Double Format

    SetCtrlVal(panelHandle, PANEL_WB_SETPOINT, wb_setpoint);    // Updates bath setpoint on screen

    CmtGetLock (Slow_lockHandle);
    DSN_SHIFT_Slow(1);
    Bath_Setpoint[0] = wb_setpoint;                             // Log value in Setpoint Array
    CmtReleaseLock (Slow_lockHandle);
}

//---------------------------------------------------------------------
// (Callback) Get_New_Setpoint  : Gets Fluid Bath Setpoint from Screen
//---------------------------------------------------------------------

int CVICALLBACK Get_New_Setpoint (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            GetCtrlVal(panelHandle, PANEL_WB_SETPOINT, &wb_setpoint);
            DSN_Set_Bath_Setpoint();

            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}

//---------------------------------------------------------
// DSN_Set_Bath_Setpoint    : Sets Fluid Bath Setpoint
//---------------------------------------------------------

void DSN_Set_Bath_Setpoint(void)
{
    char  ascii_setpoint[15],
          wholeNum_string[4],
          decNum_string[4];

    char* setpointcode = "SS";
    char* decPoint_string = ".";

    int wholeNum,
        decNum;

    wholeNum = wb_setpoint;                     // Captures the whole number portion of the setpoint
    decNum = (wb_setpoint-wholeNum)*100;        // Captures the 2 decimal places of the setpoint

    sprintf(wholeNum_string, "%i", wholeNum);
    sprintf(decNum_string, "%i", decNum);
                                                // Assembles the command as shown below
    strcpy(ascii_setpoint, setpointcode);       // SS
    strcat(ascii_setpoint, wholeNum_string);    // SSXXX
    strcat(ascii_setpoint, decPoint_string);    // SSXXX.
    strcat(ascii_setpoint, decNum_string);      // SSXXX.XX
    strcat(ascii_setpoint, "\r");               // SSXXX.XX\r

    FlushInQ(SP_comport);                       // Flush the Input and Output Queue
    FlushOutQ(SP_comport);

    stringsize = StringLength (ascii_setpoint);
    ComWrt (SP_comport, ascii_setpoint, stringsize);    // Sends new setpoint to COM Port
}

//---------------------------------------------------------------------
// (Callback) Select_Bath_Fluid : Sets Fluid Bath Temperature Alarms
//---------------------------------------------------------------------

int CVICALLBACK Select_Bath_Fluid (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    char    ascii_highalarm[15],
            ascii_lowalarm[15],
            wholeNum_string[4],
            decNum_string[3];
    char*   highalarmcode = "SH";
    char*   lowalarmcode = "SL";
    char*   decPoint_string = ".";

    double  setpoint,
            highalarm_value,
            lowalarm_value,
            ethglyc_maxtemp = 105,  // Max safe operating temperature for ethylene glycol°C
```

```
            ethglyc_mintemp = -30,  // Min safe operating temperature for ethylene glycol°C
            distH2O_maxtemp = 100,  // Max safe operating temperature for distilled water °C
            distH2O_mintemp = 0;    // Min safe operating temperature for distilled water °C

    int     selected_fluid,
            wholeNum;

    switch (event)
    {
        case EVENT_COMMIT:

            GetCtrlVal(panelHandle,PANEL_BATHFLUID, &selected_fluid);   // Find which fluid is selected
            GetCtrlVal(panelHandle,PANEL_WB_SETPOINT, &setpoint);       // Find current setpoint

            //-----------------------------------------------------
            // 1st - Set the Setpoint Limits within the Program
            //-----------------------------------------------------

            switch (selected_fluid)
                {
                case 0:     // Ethylene Glycol

                    // Change Valid Setpoint Range
                    SetCtrlAttribute(panelHandle,PANEL_WB_SETPOINT,ATTR_MAX_VALUE,ethglyc_maxtemp);
                    SetCtrlAttribute(panelHandle,PANEL_WB_SETPOINT,ATTR_MIN_VALUE,ethglyc_mintemp);

                    highalarm_value = ethglyc_maxtemp;
                    lowalarm_value = ethglyc_mintemp;

                    break;

                case 1:     // Distilled Water

                    // Change Valid Setpoint Range
                    SetCtrlAttribute(panelHandle,PANEL_WB_SETPOINT,ATTR_MAX_VALUE,distH2O_maxtemp);
                    SetCtrlAttribute(panelHandle,PANEL_WB_SETPOINT,ATTR_MIN_VALUE,distH2O_mintemp);

                    highalarm_value = distH2O_maxtemp;
                    lowalarm_value = distH2O_mintemp;

                    break;

                }

            //-------------------------------------------------------
            // 2nd - Send the High and Low Alarm Values to the Bath
            //-------------------------------------------------------

            // High Alarm //

            wholeNum = highalarm_value;                     // Captures the whole number portion of the high alarm
            sprintf(wholeNum_string, "%i", wholeNum);
                                                            // Assembles the command as shown below
            strcpy(ascii_highalarm, highalarmcode);         // SH
            strcat(ascii_highalarm, wholeNum_string);       // SHXXX
            strcat(ascii_highalarm, "\r");                  // SHXXX\r

            FlushInQ(SP_comport);                           // Flush the Input and Output Queue
            FlushOutQ(SP_comport);

            stringsize = StringLength (ascii_highalarm);
            ComWrt (SP_comport, ascii_highalarm, stringsize);   // Sends new high alarm to COM Port


            // Low Alarm //

            wholeNum = lowalarm_value;                      // Captures the whole number portion of the low alarm
            sprintf(wholeNum_string, "%i", wholeNum);
                                                            // Assembles the command as shown below
            strcpy(ascii_lowalarm, lowalarmcode);           // SL
            strcat(ascii_lowalarm, wholeNum_string);        // SLXXX
            strcat(ascii_lowalarm, "\r");                   // SLXXX\r

            FlushInQ(SP_comport);                           // Flush the Input and Output Queue
            FlushOutQ(SP_comport);

            stringsize = StringLength (ascii_lowalarm);
            ComWrt (SP_comport, ascii_lowalarm, stringsize);    // Sends new high alarm to COM Port

            break;

        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}

//-----------------------------------------------------------
// DSN_Set_Bath_External    : Sets Bath to use Remote Probe
//-----------------------------------------------------------

void DSN_Set_Bath_External(void)
{
    char* power_on = "Sr1\r";   // Define Command to Power On

    FlushInQ(SP_comport);       // Flush the Input and Output Queue
    FlushOutQ(SP_comport);

    stringsize = StringLength (power_on);
    ComWrt (SP_comport, power_on, stringsize);  // Powers On
}

//-----------------------------------------------------------
// DSN_Poll_Bath_Probe  : Polls External Probe Temperature
//-----------------------------------------------------------

void DSN_Poll_Bath_Probe(void)
{
    char*   ask_temp = "RR\r",      // Command to Request External Temperature
            ascii_temp[20];         // ASCII response to the temperature query

    double probe_temp;              // Decimal Value

    FlushInQ(SP_comport);           // Flush the Input and Output Queue
```

```
    FlushOutQ(SP_comport);

    stringsize = StringLength (ask_temp);

    // Send temperature request to COM Port
    ComWrt (SP_comport, ask_temp, stringsize);

    ascii_temp[0] = '\0';

    // Read incoming temperature from COM Port
    bytes_read = ComRdTerm (SP_comport, ascii_temp, 9, 13);

    // Convert ASCII to Double Format
    probe_temp = atof(ascii_temp);

    // Updates probe temperature on screen (S7A)
    SetCtrlVal(panelHandle, PANEL_S7A, probe_temp);

    CmtGetLock (Slow_lockHandle);
    DSN_SHIFT_Slow(6);
    S7A[0] = probe_temp;            // Log value in S7A Array
    CmtReleaseLock (Slow_lockHandle);
}

//*********************************
//  Viscojet Viscometer
//*********************************

//-----------------------------------------------
// DSN_Poll_Viscosity       : Request Viscosity
//-----------------------------------------------

void DSN_Poll_Viscosity(void)
{
    double viscosity_response;

    char    response[200]="\0",
            hex_ave_live_visc[9]="\0",
            hex_ave_corr_visc[9]="\0",
            hex_live_visc[9]="\0";

    FlushInQ(SP2_comport);                          // Flush the Input and Output Queue
    FlushOutQ(SP2_comport);

    ComWrt (SP2_comport, ":010410000004E7\n", 17);  // Poll Viscosities
    Delay (1.5);                                    // Allow Time for Instrument to Respond
    inqlen = GetInQLen (SP2_comport);               // Check Response Length

    bytes_read = ComRdTerm (SP2_comport, response, inqlen, 10); // Read Hex Response

    //printf ("bytes_read = %i\n", bytes_read);
    //printf ("The complete hex response is %s\n", response);

    CopyBytes (hex_ave_live_visc, 0, response, 7, 8);           // Parse Response
    CopyBytes (hex_ave_corr_visc, 0, response, 15, 8);
    CopyBytes (hex_live_visc, 0, response, 23, 8);

    // Converts to Decimal and Updates Readouts
    viscosity_response = DSN_Hex_to_Float(hex_ave_live_visc, PANEL_AVE_LIVE_VISCOSITY);

    CmtGetLock (Slow_lockHandle);
    DSN_SHIFT_Slow(2);
    Ave_L_Visc[0] = viscosity_response;                         // Updates Array
    CmtReleaseLock (Slow_lockHandle);

    viscosity_response = DSN_Hex_to_Float(hex_ave_corr_visc, PANEL_AVE_CORR_VISC);

    CmtGetLock (Slow_lockHandle);
    DSN_SHIFT_Slow(3);
    Ave_TC_Visc[0] = viscosity_response;
    CmtReleaseLock (Slow_lockHandle);
}

//-------------------------------------------------------------------------
// DSN_Poll_Bulb_Temperature    : Requests viscometer bulb temperature
//-------------------------------------------------------------------------

void DSN_Poll_Bulb_Temperature(void)
{
    double bulb_temp_response;

    char response[200]="\0",
         hex_visc_bulb_temp[9]="\0";

    FlushInQ(SP2_comport);                              // Flush the Input and Output Queue
    FlushOutQ(SP2_comport);

    ComWrt (SP2_comport, ":010410100003D8\n", 17);      // Poll Bulb Temperature
    Delay (1.75);                                       // Allow Time for Instrument to Respond
    inqlen = GetInQLen (SP2_comport);                   // Check Response Length

    ComRdTerm (SP2_comport, response, inqlen, 10);      // Read Hex Response
    CopyBytes (hex_visc_bulb_temp, 0, response, 7, 8);  // Parse Response

    // Converts to Decimal and Updates Readouts
    bulb_temp_response = DSN_Hex_to_Float(hex_visc_bulb_temp, PANEL_BULB_TEMPERATURE);

    CmtGetLock (Slow_lockHandle);
    DSN_SHIFT_Slow(4);
    Bulb_Temp[0] = bulb_temp_response;                  // Updates Array
    CmtReleaseLock (Slow_lockHandle);
}

//-------------------------------------------------------------------------
// DSN_Hex_to_Float     : Converts 8 bit ASCII Hex Float to Decimal Number
//-------------------------------------------------------------------------

double DSN_Hex_to_Float(char* ascii_hex_in, int output_indicator)
{
    double float_out;   // Float Output from Hex-to-Float Function

    int     binary_out,
            i,
            integer_exponent,
            whole_num_array_size = 0,
```

```
        dec_num_array_size = 0;

char    //ascii_hex_in[9],                  // Step 1 Array
        binary_from_hex[36] = "\0",         // Step 2 Arrays
        ascii_hex_chunk[5],                 //
        sign_array[2] = "\0",               // Step 3 Arrays
        exponent_array[9] = "\0",           //
        mantissa_array[25] = "1",           // Leading 1 Normalizes Mantissa
        whole_num_array[200] = "\0",        // Step 5 Arrays
        dec_num_array[200] = "\0",          //
        complete_binary_num[200] = "\0";    // Step 6 Array

char    *end_pointer;
char    *decimal_point = ".";

float   float_whole,
        float_decimal = 0.0;

// Define binary digit arrays
char    *zero = "0000";
char    *one = "0001";
char    *two = "0010";
char    *three = "0011";
char    *four = "0100";
char    *five = "0101";
char    *six = "0110";
char    *seven = "0111";
char    *eight = "1000";
char    *nine = "1001";
char    *ten = "1010";
char    *eleven = "1011";
char    *twelve = "1100";
char    *thirteen = "1101";
char    *fourteen = "1110";
char    *fifteen = "1111";



        /*********************************************/
        /*  Step 1: Obtain Hex Input String          */
        /*********************************************/

        //printf ("The hex number is %s\n", ascii_hex_in);

        /*********************************************/
        /*  Step 2: Convert Hex String to Binary     */
        /*********************************************/

        for(i=0;ascii_hex_in[i]!=NULL;i++)
        {
            switch(ascii_hex_in[i])
            {
            case '0':
                strcat (binary_from_hex, zero);
                break;
            case '1':
                strcat (binary_from_hex, one);
                break;
            case '2':
                strcat (binary_from_hex, two);
                break;
            case '3':
                strcat (binary_from_hex, three);
                break;
            case '4':
                strcat (binary_from_hex, four);
                break;
            case '5':
                strcat (binary_from_hex, five);
                break;
            case '6':
                strcat (binary_from_hex, six);
                break;
            case '7':
                strcat (binary_from_hex, seven);
                break;
            case '8':
                strcat (binary_from_hex, eight);
                break;
            case '9':
                strcat (binary_from_hex, nine);
                break;
            case 'a':
            case 'A':
                strcat (binary_from_hex, ten);
                break;
            case 'b':
            case 'B':
                strcat (binary_from_hex, eleven);
                break;
            case 'c':
            case 'C':
                strcat (binary_from_hex, twelve);
                break;
            case 'd':
            case 'D':
                strcat (binary_from_hex, thirteen);
                break;
            case 'e':
            case 'E':
                strcat (binary_from_hex, fourteen);
                break;
            case 'f':
            case 'F':
                strcat (binary_from_hex, fifteen);
                break;
            default:
                //printf("Entered number is not Hexadecimal. Printed value is not correct.");
                break;
            }
        }
        //printf ("The hex number in binary is %s\n", binary_from_hex);

        /*********************************************************************************/
        /*  Step 3: Break Binary String into Sign, Exponent, and Mantissa Arrays         */
```

```
            /*****************************************************************************/

            sign_array[0] = binary_from_hex[0];             // Copy Sign (Bit 31)

            for (i=0; i<8; i++)                             // Copy Exponent (Bits 30-23)
            {
                exponent_array[i] = binary_from_hex[i+1];
            }

            for (i=0; i<23; i++)                            // Copy Mantissa (Bits 22-0)
            {
                mantissa_array[i+1] = binary_from_hex[i+9];
            }

            //printf ("The binary sign bit is %s\n", sign_array);
            //printf ("The binary exponent is %s\n", exponent_array);
            //printf ("The binary mantissa is %s\n", mantissa_array);

            /************************************************/
            /*  Step 4: Compute Integer Value of Exponent       */
            /************************************************/

                                                    // -127 is there because exponent is biased
            integer_exponent = strtol (exponent_array, &end_pointer, 2) - 127;

            //printf ("The integer exponent is %i\n", integer_exponent);

            /*****************************************************************************/
            /*  Step 5: Shift Decimal Position According to Exponent and Create New Binary  */
            /*          Number Arrays for the Whole and Decimal Portions of the Float       */
            /*****************************************************************************/

            if (integer_exponent > 0)   // Whole and Decimal Portions Both Exist
            {
                for (i=0; i < (integer_exponent + 1); i++)  // Fill Whole Number Binary Array
                {
                    whole_num_array[i] = mantissa_array[i];
                    //printf ("The binary whole number array is %s\n", whole_num_array);
                }

                for (i=0; i < (23 - integer_exponent); i++) // Fill Decimal Number Binary Array
                {
                    dec_num_array[i] = mantissa_array[(integer_exponent + 1 + i)];
                }
            }

            else    // Only Decimal Portion Exists
            {
                for (i=0; i < (-1 - integer_exponent); i++) // Add Leading Zeros
                {
                    dec_num_array[i] = '0';
                }
                // Fill Decimal Number Binary Array (Shifted by Leading Zeros)
                strcat(dec_num_array, mantissa_array);
            }

            //printf ("The binary decimal number array is %s\n", dec_num_array);

            /*********************************************************************/
            /*  Step 6: Compute Base 10 Values for Whole and Decimal Portions   */
            /*          and Assemble Complete Floating Point Number             */
            /*********************************************************************/

            // Assemble the Complete Binary String as a Check
            strcat(complete_binary_num, whole_num_array);
            strcat(complete_binary_num, decimal_point);
            strcat(complete_binary_num, dec_num_array);

            //printf("The complete binary number is %s\n", complete_binary_num);

            // Compute Whole Number Portion of Float
            float_whole = strtol(whole_num_array, &end_pointer, 2);

            // Compute Decimal Number Portion of Float
            for(i=0;dec_num_array[i]!=NULL;i++)
            {
                switch(dec_num_array[i])
                {
                case '0':
                    break;
                case '1':
                    float_decimal = float_decimal + pow(2, -i-1);
                    break;
                }
            }

            float_out = float_whole + float_decimal;

            switch(sign_array[0])       // Determine Sign of Floating Point Number
            {
            case '0':
                break;
            case '1':
                float_out = - float_out;
                break;
            }

            //printf("The floating point decimal is %f\n", float_decimal);
            //printf("The final floating point number is %f\n\n", float_out);

            // Sends Decimal Result out to the Panel
            SetCtrlVal(panelHandle, output_indicator, float_out);

            return float_out;
        }

//**********************************
//  GE Pressure Controller
//**********************************

//-----------------------------------------------------------
// DSN_Initialize_P_Controller  : Sets to Remote Operation
//                              : Sets Units to PSI
//                              : Set to Overdamped Control
//                              : Set Notation Code to N1
```

```
//                              : Set Interrupt to End of Conversion
//                              : Vents the System
//                              : Zero's the Instrument
//-------------------------------------------------------------

void DSN_Initialize_P_Controller(void)
{
    char* remote_mode = "R1\r";         /* Command to Set to Remote Mode */
    char* set_units = "S1\r";           /* Command to Set Units to psi */
    char* set_control_rate = "J1\r";    /* Command to Set Overdamped Control */
                                        /* Vents Instrument */
    char* zero = "O1\r";                /* Command to Zero Instrument */
    char* notation_code = "N1\r";       /* Command to Set Notation Code to N1 */
    char* interrupt_code = "I4\r";      /* Command to Set Interrupt on End of Conversion */

    FlushInQ(SP3_comport);
    FlushOutQ(SP3_comport);
    stringsize = StringLength (remote_mode);
    ComWrt (SP3_comport, remote_mode, stringsize);      // Sets control mode at COM Port

    FlushInQ(SP3_comport);
    FlushOutQ(SP3_comport);
    stringsize = StringLength (set_units);
    ComWrt (SP3_comport, set_units, stringsize);        // Send Units Command

    FlushInQ(SP3_comport);
    FlushOutQ(SP3_comport);
    stringsize = StringLength (set_control_rate);
    ComWrt (SP3_comport, set_control_rate, stringsize); // Send Control Rate Command

    DSN_Vent_Pressure();                                // Vents the System Pressure

    FlushInQ(SP3_comport);                              // Clears the In and Out Q
    FlushOutQ(SP3_comport);
    stringsize = StringLength (zero);
    ComWrt (SP3_comport, zero, stringsize);             // Send Zero Command

    FlushInQ(SP3_comport);
    FlushOutQ(SP3_comport);
    stringsize = StringLength (notation_code);
    ComWrt (SP3_comport, notation_code, stringsize);    // Send Notation Code Command

    FlushInQ(SP3_comport);
    FlushOutQ(SP3_comport);
    stringsize = StringLength (interrupt_code);
    ComWrt (SP3_comport, interrupt_code, stringsize);   // Send Interrupt Code Command

}

//-------------------------------------------------------------
// DSN_Change_P_Setpoint        : Changes the Setpoint
//                              : Controls to New Setpoint
//-------------------------------------------------------------

double DSN_Change_P_Setpoint(double new_p_setpoint)
{
    char    ascii_setpoint[20],
            wholeNum_string[5],
            decNum_string[4];

    char* setpointcode = "P=";
    char* decPoint_string = ".";
    char* start_control = "C1\r";

    double  max_pressure = 1450.38; // Maximum pressure allowed by controller

    int wholeNum,
        decNum;

    if(new_p_setpoint <= max_pressure)      // Safe Setpoint Pressure
    {
        wholeNum = new_p_setpoint;          /* Captures the whole number portion of the setpoint */
        decNum = (new_p_setpoint-wholeNum)*100; /* Captures the 2 decimal places of the setpoint */

        sprintf(wholeNum_string, "%i", wholeNum);
        sprintf(decNum_string, "%i", decNum);
                                            /* Assembles the Command */
        strcpy(ascii_setpoint, setpointcode);       /* P= */
        strcat(ascii_setpoint, wholeNum_string);    /* P=XXX */
        strcat(ascii_setpoint, decPoint_string);    /* P=XXX. */
        strcat(ascii_setpoint, decNum_string);      /* P=XXX.XX */
        strcat(ascii_setpoint, "\r");               /* P=XXX.XX\r */


        FlushInQ(SP3_comport);      /* Clears the In and Out Q */
        FlushOutQ(SP3_comport);

        stringsize = StringLength (ascii_setpoint);
        ComWrt (SP3_comport, ascii_setpoint, stringsize);   // Sends new setpoint to COM Port

        stringsize = StringLength (start_control);
        ComWrt (SP3_comport, start_control, stringsize);        // Controls to New Setpoint

        SetCtrlVal(panelHandle, PANEL_REGULATOR_SETPOINT, new_p_setpoint);
    }
    else
    {
    MessagePopup ("Warning", "Desired Setpoint is Outside Safe Working Pressure of the Controller");
    }
    return 0;
}


//-------------------------------------------------------------
// Change_P_Setpoint (Callback) : For manual mode
//                              : Calls DSN_Change_P_Setpoint
//-------------------------------------------------------------

int CVICALLBACK REGULATOR_SETPOINT_CALLBACK (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
```

```
            GetCtrlVal(panelHandle, PANEL_REGULATOR_SETPOINT,
                &p_setpoint_for_callback);                      // Retrieve New Value

            DSN_Change_P_Setpoint(p_setpoint_for_callback);     // Adjust to New Setpoint

            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}

//---------------------------------------------------------
// DSN_Vent_Pressure    : Slowly vent the system pressure
//
//---------------------------------------------------------

void DSN_Vent_Pressure(void)
{
    double step_down_pressure, current_pressure;

    char* stop_control = "C0\r";

    int delay,
        vent_steps,
        current_vent_step,
        pressure_per_vent_step = 5; // Vent ??psi per vent step

    DSN_Poll_Current_Pressure();
    current_pressure = PACE_Pressure_Global;

    vent_steps = current_pressure/pressure_per_vent_step;

    for(current_vent_step = 1; current_vent_step <= vent_steps; current_vent_step++)
    {
        step_down_pressure = current_pressure - current_vent_step*pressure_per_vent_step;

        if(step_down_pressure < 0)
        {
            step_down_pressure = 0;
        }

        // Vent pressure in small steps
        DSN_Change_P_Setpoint(step_down_pressure);

        // Change Setpoint Readout
        SetCtrlVal(panelHandle, PANEL_REGULATOR_SETPOINT, step_down_pressure);
        for(delay=1 ; delay <= 10 ; delay++)
        {
            Delay(1);
            DSN_Poll_Current_Pressure();
            ProcessSystemEvents ();
        }
    }

    step_down_pressure = 0;
    DSN_Change_P_Setpoint(step_down_pressure);      // Final vent to zero psi
    for(delay=1 ; delay <= 30 ; delay++)
        {
            Delay(1);    // Final Vent
            ProcessSystemEvents ();
        }

    // Change Setpoint Readout to Zero
    SetCtrlVal(panelHandle, PANEL_REGULATOR_SETPOINT, 0.00);
    DSN_Poll_Current_Pressure();

    stringsize = StringLength (stop_control);
    ComWrt (SP3_comport, stop_control, stringsize);  // Turns off Control Module
}

//---------------------------------------------------------
// DSN_Shut_Down_P_Controller   : Vent the System
//                              : Set PACE5000 to Local Mode
//---------------------------------------------------------

void DSN_Shut_Down_P_Controller(void)
{
    char* local_mode = "R0\r";          /* Command to Set to Local Mode */

    DSN_Vent_Pressure();

    FlushInQ(SP3_comport);                          // Clears the In and Out Q
    FlushOutQ(SP3_comport);
    stringsize = StringLength (local_mode);
    ComWrt (SP3_comport, local_mode, stringsize);   // Sets control mode at COM Port
}

//---------------------------------------------------------
// DSN_Poll_Pressure_Setpoint   : Polls Pressure Setpoint
//---------------------------------------------------------

void DSN_Poll_Pressure_Setpoint(void)
{
    char*   data_select = "D1\r";   // Command to set reading to Setpoint
    char*   req_reading = "\r",     // Command to Request a Reading
            ascii_response[20],     // ASCII response to query
            ascii_p_setpoint[20];   // ASCII setpoint

    double pressure_setpoint;       // Decimal Value

    FlushInQ(SP3_comport);                      // Flush the Input and Output Queue
    FlushOutQ(SP3_comport);

    stringsize = StringLength (data_select);
    ComWrt (SP3_comport, data_select, stringsize); // Set Reading to Setpoint

    Delay(0.5);                                 // Delay to allow for end of conversion

    FlushInQ(SP3_comport);                      // Flush the Input and Output Queue
    FlushOutQ(SP3_comport);

    stringsize = StringLength (req_reading);
    ComWrt (SP3_comport, req_reading, stringsize); // Send Read request to COM Port
```

```
    ascii_response[0] = '\0';
    bytes_read = ComRdTerm (SP3_comport,
        ascii_response, 7, 13);     // Read data from COM Port (7 digits including ".")

    pressure_setpoint = atof(ascii_response);   // Convert ASCII to Double Format

    SetCtrlVal(panelHandle, PANEL_REGULATOR_SETPOINT,
        pressure_setpoint);                     // Updates pressure setpoint on screen

    CmtGetLock (Slow_lockHandle);
    DSN_SHIFT_Slow(5);
    Stat_P_Setpoint[0] = pressure_setpoint;     // Log value in Setpoint Array
    CmtReleaseLock (Slow_lockHandle);
}

//-------------------------------------------------------
// DSN_Poll_Current_Pressure    : Polls Current Pressure
//-------------------------------------------------------

void DSN_Poll_Current_Pressure(void)
{
    char*   data_select = "D0\r";   // Command to set reading to Current Pressure
    char*   req_reading = "\r",     // Command to Request a Reading
            ascii_response[20],     // ASCII response to query
            ascii_current_p[20];    // ASCII setpoint

    double current_pressure;        // Decimal Value

    FlushInQ(SP3_comport);                      // Flush the Input and Output Queue
    FlushOutQ(SP3_comport);

    stringsize = StringLength (data_select);
    ComWrt (SP3_comport, data_select, stringsize); // Set Reading to Current Pressure

    Delay(0.5);                                 // Delay to allow for end of conversion

    FlushInQ(SP3_comport);                      // Flush the Input and Output Queue
    FlushOutQ(SP3_comport);

    stringsize = StringLength (req_reading);
    ComWrt (SP3_comport, req_reading, stringsize); // Send Read request to COM Port

    Delay(0.5);

    ascii_response[0] = '\0';
    bytes_read = ComRdTerm (SP3_comport,
        ascii_response, 7, 13);                 // Read data from COM Port (7 digits including ".")

    current_pressure = atof(ascii_response);    // Convert ASCII to Double Format
    PACE_Pressure_Global = current_pressure;    // Send current pressure to a global variable

    SetCtrlVal(panelHandle, PANEL_REGULATOR_PRESSURE,
        current_pressure);                      // Updates pressure setpoint on screen

    CmtGetLock (Slow_lockHandle);
    DSN_SHIFT_Slow(7);
    PACE_Pressure[0] = current_pressure;        // Log value in PACE_Pressure Array
    CmtReleaseLock (Slow_lockHandle);
}

//**********************************
//  Tektronics Function Generator
//**********************************

//-------------------------------------------------------
// DSN_Init_FuncGen     : Initializes Function Generator
//-------------------------------------------------------

void DSN_Init_FuncGen(void)
{
    tkafg3k_InitWithOptions ("USB0::0x0699::0x0346::C030622::INSTR", VI_TRUE, VI_TRUE,
                             "Simulate=0,RangeCheck=1,QueryInstrStatus=1,Cache=1",
                                      &tkafg3k);
}

//-------------------------------------------------------------
// DSN_Set_FuncGen_Out      : Sets Function Generator Output
//-------------------------------------------------------------

void DSN_Set_FuncGen_Out(void)

{
    ViReal64 Amplitude_1;
    ViReal64 Frequency_1;

    int wave1;          //wavetype
    ViInt32 WaveType1;

    int mode1;           //run mode
    ViInt32 ModeType1;

    double phase1;       //phase modulation frequency
    ViInt32 PhaseType1;

    ViReal64 phaseangle1;     //phase angle
    int mwave_1;

    ViBoolean enablephasemod_1;


    GetCtrlVal (panelHandle, PANEL_VOLTAGE, &Amplitude_1);  //get info from gui
    GetCtrlVal (panelHandle, PANEL_FREQUENCY,&Frequency_1);
    GetCtrlVal (panelHandle, PANEL_Waveform_1,&wave1);

    ModeType1 = TKAFG3K_VAL_OPERATE_CONTINUOUS;     //set to continuous operation
    phaseangle1 = 0;                                //set phase angle in degrees
    phase1 = 25000;                                 //set phase modulation frequency in Hz
    PhaseType1 = TKAFG3K_VAL_PM_INTERNAL_SINE;      //for the phase modulation waveform
    enablephasemod_1 = VI_FALSE;                    //disable phase modulation


    switch (wave1)          //select desired waveform for channel 1
        {
        case 0:
```

```
            WaveType1 = TKAFG3K_VAL_WFM_SINE;
            break;
        case 1:
            WaveType1 = TKAFG3K_VAL_WFM_SQUARE;
            break;
        case 2:
            WaveType1 = TKAFG3K_VAL_WFM_RAMP;
            break;
        case 3:
            WaveType1 = TKAFG3K_VAL_WFM_PULS;
            break;
        case 4:
            WaveType1 = TKAFG3K_VAL_WFM_PRN;
            break;
        case 5:
            WaveType1 = TKAFG3K_VAL_WFM_DC;
            break;
        case 6:
            WaveType1 = TKAFG3K_VAL_WFM_SINC;
            break;
        case 7:
            WaveType1 = TKAFG3K_VAL_WFM_GAUS;
            break;
        case 8:
            WaveType1 = TKAFG3K_VAL_WFM_LOR;
            break;
        case 9:
            WaveType1 = TKAFG3K_VAL_WFM_ERIS;
            break;
        case 10:
            WaveType1 = TKAFG3K_VAL_WFM_EDEC;
            break;
        case 11:
            WaveType1 = TKAFG3K_VAL_WFM_HAV;
            break;

        }


    //tkafg3k_ConfigureOutputMode (tkafg3k, TKAFG3K_VAL_OUTPUT_FUNC);
    tkafg3k_ConfigurePMEnabled (tkafg3k, "1", enablephasemod_1);
    tkafg3k_ConfigurePMSource (tkafg3k, "1", TKAFG3K_VAL_PM_INTERNAL);
    tkafg3k_ConfigurePMInternalByChannel (tkafg3k, "1", 90, PhaseType1, phase1);
    tkafg3k_ConfigureOperationMode (tkafg3k, "1", (ViInt32)ModeType1);
    tkafg3k_ConfigureStandardWaveform (tkafg3k, "1", WaveType1, Amplitude_1, 0.0, Frequency_1, phaseangle1);
}

//---------------------------------------------------------------------
// DSN_Toggle_FuncGen_On        : Configures the Output
//                              : Sets Function Generator Output On
//---------------------------------------------------------------------

void DSN_Toggle_FuncGen_On(void)

{
    DSN_Set_FuncGen_Out();
    tkafg3k_ConfigureOutputEnabled (tkafg3k, "1", VI_TRUE);
    SetCtrlVal(panelHandle, PANEL_FUNCGEN_ONOFF, 1);
}

//---------------------------------------------------------------------
// DSN_Toggle_FuncGen_Off       : Sets Function Generator Output Off
//---------------------------------------------------------------------

void DSN_Toggle_FuncGen_Off(void)

{
    tkafg3k_ConfigureOutputEnabled (tkafg3k, "1", VI_FALSE);
    SetCtrlVal(panelHandle, PANEL_FUNCGEN_ONOFF, 0);
}


//---------------------------------------------------------------------------------------
// DSN_Read_FuncGen_UI_Output (Callback)    : Manual Function Generator Output Toggle
//---------------------------------------------------------------------------------------

int CVICALLBACK Read_FuncGen_UI_Output (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    int OutputState;

    switch (event)
    {
        case EVENT_COMMIT:

            GetCtrlVal(panelHandle,PANEL_FUNCGEN_ONOFF, &OutputState);

            switch (OutputState)          // Set desired output state
                {
                case 0: // Channel 1 Off
                    tkafg3k_ConfigureOutputEnabled (tkafg3k, "1", VI_FALSE);
                    break;
                case 1: // Channel 1 On
                    tkafg3k_ConfigureOutputEnabled (tkafg3k, "1", VI_TRUE);
                    break;
                }

            break;
    }
    return 0;
}

//---------------------------------------------------------------------
// DSN_Read_FuncGen_UI      : Reads function generator setpoints on UI
//                          : Sends values to the function generator
//---------------------------------------------------------------------

int CVICALLBACK Read_FuncGen_UI (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            DSN_Set_FuncGen_Out();  //call the function to read the UI values and set the output
```

```
            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}


/*****************************************************************************************/
//   .d8888b.  8888888b.         d8888 8888888b.  888    888 8888888 888b    888  .d8888b.
//  d88P  Y88b 888   Y88b       d88888 888   Y88b 888    888   888   8888b   888 d88P  Y88b
//  888    888 888    888      d88P888 888    888 888    888   888   88888b  888 888    888
//  888        888   d88P     d88P 888 888   d88P 8888888888   888   888Y88b 888 888
//  888  88888 8888888P"     d88P  888 8888888P"  888    888   888   888 Y88b888 888  88888
//  888    888 888 T88b     d88P   888 888        888    888   888   888  Y88888 888    888
//  Y88b  d88P 888  T88b   d8888888888 888        888    888   888   888   Y8888 Y88b  d88P
//   "Y8888P88 888   T88b d88P     888 888        888    888 8888888 888    Y888  "Y8888P88
/*****************************************************************************************/

//*************************************************************
// OnOff_Graphs     : For Turning Graphs ON and OFF (Dimming)
//*************************************************************
int CVICALLBACK OnOff_Graphs (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{   int test;
    switch (event)
        {
        case EVENT_COMMIT:
            switch(control)
                {
                case G_SETUP_OnOff_G_1:
                    GetCtrlVal (g_Handle, G_SETUP_OnOff_G_1, &test);
                    if(test)
                      {
                      SetCtrlAttribute (panelHandle, PANEL_G_1, ATTR_DIMMED, 0);
                      OnOff_G1 = 1;
                      }
                    else
                      {
                      SetCtrlAttribute (panelHandle, PANEL_G_1, ATTR_DIMMED, 1);
                      OnOff_G1 = 0;
                      }
                    break;
                case G_SETUP_OnOff_G_2:
                    GetCtrlVal (g_Handle, G_SETUP_OnOff_G_2, &test);
                    if(test)
                      {
                      SetCtrlAttribute (panelHandle, PANEL_G_2, ATTR_DIMMED, 0);
                      OnOff_G2 = 1;
                      }
                    else
                      {
                      SetCtrlAttribute (panelHandle, PANEL_G_2, ATTR_DIMMED, 1);
                      OnOff_G2 = 0;
                      }
                    break;
                case G_SETUP_OnOff_G_3:
                    GetCtrlVal (g_Handle, G_SETUP_OnOff_G_3, &test);
                    if(test)
                      {
                      SetCtrlAttribute (panelHandle, PANEL_G_3, ATTR_DIMMED, 0);
                      OnOff_G3 = 1;
                      }
                    else
                      {
                      SetCtrlAttribute (panelHandle, PANEL_G_3, ATTR_DIMMED, 1);
                      OnOff_G3 = 0;
                      }
                    break;
                case G_SETUP_OnOff_G_4:
                    GetCtrlVal (g_Handle, G_SETUP_OnOff_G_4, &test);
                    if(test)
                      {
                      SetCtrlAttribute (panelHandle, PANEL_G_4, ATTR_DIMMED, 0);
                      OnOff_G4 = 1;
                      }
                    else
                      {
                      SetCtrlAttribute (panelHandle, PANEL_G_4, ATTR_DIMMED, 1);
                      OnOff_G4 = 0;
                      }
                    break;
                }
            break;
        case EVENT_RIGHT_CLICK:

            break;
        }
    return 0;
}

//*************************************************************
// DSN_Graph        : Calls Graphing Function and Scales X Axes
//*************************************************************
void DSN_Graph(void)
{
// GRAPH #1
if(OnOff_G1)
{
DSN_Graph_Select(plotVar_G_1, PANEL_G_1);
    if(plotVar_G_1>=0)
    {
    SetAxisScalingMode (panelHandle, PANEL_G_1, VAL_XAXIS,
                        VAL_MANUAL, step[0]-X_Range_G_1, step[0]);
    if(Y_Mode_G_1)
     SetAxisScalingMode (panelHandle, PANEL_G_1, VAL_LEFT_YAXIS,
                         VAL_AUTOSCALE,0 ,0 );
    else
      SetAxisScalingMode (panelHandle, PANEL_G_1, VAL_LEFT_YAXIS,
                          VAL_MANUAL, Y_Min_G_1, Y_Max_G_1);
    }
}
// GRAPH #2
```

```
if(OnOff_G2)
{
DSN_Graph_Select(plotVar_G_2, PANEL_G_2);
    if(plotVar_G_2>=0)
    {
    SetAxisScalingMode (panelHandle, PANEL_G_2, VAL_XAXIS, VAL_MANUAL,
                                step[0]-X_Range_G_2, step[0]);
    if(Y_Mode_G_2)
     SetAxisScalingMode (panelHandle, PANEL_G_2, VAL_LEFT_YAXIS,
                         VAL_AUTOSCALE,0 ,0 );
    else
      SetAxisScalingMode (panelHandle, PANEL_G_2, VAL_LEFT_YAXIS,
                          VAL_MANUAL, Y_Min_G_2, Y_Max_G_2);
    }
}
// GRAPH #3
if(OnOff_G3)
{
DSN_Graph_Select(plotVar_G_3, PANEL_G_3);
    if(plotVar_G_3>=0)
    {
    SetAxisScalingMode (panelHandle, PANEL_G_3, VAL_XAXIS, VAL_MANUAL,
                                step[0]-X_Range_G_3, step[0]);
    if(Y_Mode_G_3)
     SetAxisScalingMode (panelHandle, PANEL_G_3, VAL_LEFT_YAXIS,
                         VAL_AUTOSCALE,0 ,0 );
    else
      SetAxisScalingMode (panelHandle, PANEL_G_3, VAL_LEFT_YAXIS,
                          VAL_MANUAL, Y_Min_G_3, Y_Max_G_3);
    }
}
// GRAPH #4
if(OnOff_G4)
{
DSN_Graph_Select(plotVar_G_4, PANEL_G_4);

    if(plotVar_G_4>=0)
    {
    SetAxisScalingMode (panelHandle, PANEL_G_4, VAL_XAXIS,
            VAL_AUTOSCALE, step[0]-X_Range_G_4, step[0]);
    if(Y_Mode_G_4)
     SetAxisScalingMode (panelHandle, PANEL_G_4, VAL_LEFT_YAXIS,
                         VAL_AUTOSCALE,0 ,0 );
    else
      SetAxisScalingMode (panelHandle, PANEL_G_4, VAL_LEFT_YAXIS,
                          VAL_MANUAL, Y_Min_G_4, Y_Max_G_4);
    }
}

return;
}

//*************************************************************
// DSN_Graph_Select     : Plots the Selected Variable
//*************************************************************
void DSN_Graph_Select(int plotVar, int Panel_Graph)
{
switch(plotVar)
    {   case -1:
            DeleteGraphPlot (panelHandle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
            break;
        case 0: // Fluid Bath Temperature & Setpoint
            DeleteGraphPlot (panelHandle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
            PlotXY (panelHandle, Panel_Graph, step, Bath_Temp, NUM, VAL_DOUBLE,
                    VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1, VAL_RED);
            PlotXY (panelHandle, Panel_Graph, step, Bath_Setpoint, NUM, VAL_DOUBLE,
                    VAL_DOUBLE, VAL_FAT_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_WHITE);    // Not Working
            break;
        case 1: // Viscosity
            DeleteGraphPlot (panelHandle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
            PlotXY (panelHandle, Panel_Graph, step, Ave_L_Visc, NUM, VAL_DOUBLE, VAL_DOUBLE,
                VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1, VAL_RED);
            break;
        case 2: // Experiment Stage
            DeleteGraphPlot (panelHandle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
            PlotXY (panelHandle, Panel_Graph, step, Logging_Trig_Array, NUM, VAL_DOUBLE,
                VAL_DOUBLE, VAL_FAT_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);
            break;
        case 3: // Radial Temperature Profile
            DeleteGraphPlot (panelHandle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
            PlotXY (panelHandle, Panel_Graph, Key_Radial_Positions, Current_Radial_Temps, 4,
                VAL_DOUBLE, VAL_DOUBLE, VAL_CONNECTED_POINTS, VAL_SOLID_CIRCLE, VAL_DASH, 1, VAL_RED);
            break;
        case 4: //
            DeleteGraphPlot (panelHandle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);

            break;

    }
return;
}
//*****************************************************************************
// GRAPHS   : Call to Setup Graphs Anytime a Value on Graph Panel is Changed
//*****************************************************************************
int CVICALLBACK DSN_SETUP_G (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            DSN_GraphSetup();
            break;
        case EVENT_RIGHT_CLICK:

            break;
        }
    return 0;
}
//*****************************************************************************
// DSN_GraphSetup       : Setup Graph Scaling to Match the Graph Panel Values
//*****************************************************************************
void DSN_GraphSetup(void)
{
// Update the variables from the panels
DSN_Save_Vars();
```

```
//Read Graph #1
    if(Y_Mode_G_1)
       {
       SetCtrlAttribute (g_Handle, G_SETUP_Y_Mode_G_1,  ATTR_LABEL_TEXT, "Auto");
       SetCtrlAttribute (g_Handle, G_SETUP_Y_Min_G_1,   ATTR_DIMMED, 1);
       SetCtrlAttribute (g_Handle, G_SETUP_Y_Max_G_1,   ATTR_DIMMED, 1);
       }
    else
        {
        SetCtrlAttribute (g_Handle, G_SETUP_Y_Mode_G_1, ATTR_LABEL_TEXT, "Fixed");
        SetCtrlAttribute (g_Handle, G_SETUP_Y_Min_G_1,  ATTR_DIMMED, 0);
        SetCtrlAttribute (g_Handle, G_SETUP_Y_Max_G_1,  ATTR_DIMMED, 0);
        }
        DSN_GraphName(plotVar_G_1, PANEL_G_1,G_SETUP_G1_L_1,G_SETUP_G1_T_1,
                                             G_SETUP_G1_L_2,G_SETUP_G1_T_2,
                                             G_SETUP_G1_L_3,G_SETUP_G1_T_3,
                                             G_SETUP_G1_L_4,G_SETUP_G1_T_4);
//Read Graph #2
    if(Y_Mode_G_2)
       {
       SetCtrlAttribute (g_Handle, G_SETUP_Y_Mode_G_2,  ATTR_LABEL_TEXT, "Auto");
       SetCtrlAttribute (g_Handle, G_SETUP_Y_Min_G_2,   ATTR_DIMMED, 1);
       SetCtrlAttribute (g_Handle, G_SETUP_Y_Max_G_2,   ATTR_DIMMED, 1);
       }
    else
        {
        SetCtrlAttribute (g_Handle, G_SETUP_Y_Mode_G_2, ATTR_LABEL_TEXT, "Fixed");
        SetCtrlAttribute (g_Handle, G_SETUP_Y_Min_G_2,  ATTR_DIMMED, 0);
        SetCtrlAttribute (g_Handle, G_SETUP_Y_Max_G_2,  ATTR_DIMMED, 0);
        }
        DSN_GraphName(plotVar_G_2, PANEL_G_2,G_SETUP_G2_L_1,G_SETUP_G2_T_1,
                                             G_SETUP_G2_L_2,G_SETUP_G2_T_2,
                                             G_SETUP_G2_L_3,G_SETUP_G2_T_3,
                                             G_SETUP_G2_L_4,G_SETUP_G2_T_4);
//Read Graph #3
    if(Y_Mode_G_3)
       {
       SetCtrlAttribute (g_Handle, G_SETUP_Y_Mode_G_3,  ATTR_LABEL_TEXT, "Auto");
       SetCtrlAttribute (g_Handle, G_SETUP_Y_Min_G_3,   ATTR_DIMMED, 1);
       SetCtrlAttribute (g_Handle, G_SETUP_Y_Max_G_3,   ATTR_DIMMED, 1);
       }
    else
        {
        SetCtrlAttribute (g_Handle, G_SETUP_Y_Mode_G_3, ATTR_LABEL_TEXT, "Fixed");
        SetCtrlAttribute (g_Handle, G_SETUP_Y_Min_G_3,  ATTR_DIMMED, 0);
        SetCtrlAttribute (g_Handle, G_SETUP_Y_Max_G_3,  ATTR_DIMMED, 0);
        }
        DSN_GraphName(plotVar_G_3, PANEL_G_3,G_SETUP_G3_L_1,G_SETUP_G3_T_1,
                                             G_SETUP_G3_L_2,G_SETUP_G3_T_2,
                                             G_SETUP_G3_L_3,G_SETUP_G3_T_3,
                                             G_SETUP_G3_L_4,G_SETUP_G3_T_4);
//Read Graph #4
    if(Y_Mode_G_4)
       {
       SetCtrlAttribute (g_Handle, G_SETUP_Y_Mode_G_4,  ATTR_LABEL_TEXT, "Auto");
       SetCtrlAttribute (g_Handle, G_SETUP_Y_Min_G_4,   ATTR_DIMMED, 1);
       SetCtrlAttribute (g_Handle, G_SETUP_Y_Max_G_4,   ATTR_DIMMED, 1);
       }
    else
        {
        SetCtrlAttribute (g_Handle, G_SETUP_Y_Mode_G_4, ATTR_LABEL_TEXT, "Fixed");
        SetCtrlAttribute (g_Handle, G_SETUP_Y_Min_G_4,  ATTR_DIMMED, 0);
        SetCtrlAttribute (g_Handle, G_SETUP_Y_Max_G_4,  ATTR_DIMMED, 0);
        }
        DSN_GraphName(plotVar_G_4, PANEL_G_4,G_SETUP_G4_L_1,G_SETUP_G4_T_1,
                                             G_SETUP_G4_L_2,G_SETUP_G4_T_2,
                                             G_SETUP_G4_L_3,G_SETUP_G4_T_3,
                                             G_SETUP_G4_L_4,G_SETUP_G4_T_4);
return;
}

//*****************************************************
// DSN_GraphName        : Setup Graph Appearance
//*****************************************************
void DSN_GraphName(int Val, int GRAPH, int L1, int T1, int L2, int T2,
                                       int L3, int T3, int L4, int T4)
{
    switch(Val)
        {
        case -1:
             SetCtrlAttribute (panelHandle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
             SetCtrlAttribute (panelHandle, GRAPH, ATTR_YNAME, "Not Plotting");
             SetCtrlAttribute (panelHandle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_RIGHT_YAXIS);
             SetCtrlAttribute (panelHandle, GRAPH, ATTR_YNAME, "");
             SetCtrlAttribute (g_Handle, L1, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_Handle, T1, "");
             SetCtrlAttribute (g_Handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_Handle, T2, "");
             SetCtrlAttribute (g_Handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_Handle, T3, "");
             SetCtrlAttribute (g_Handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_Handle, T4, "");
             break;
        case 0: // Fluid Bath Temperature & Setpoint
             SetCtrlAttribute (panelHandle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
             SetCtrlAttribute (panelHandle, GRAPH, ATTR_YNAME, "Bath Temperature [°C]");
             SetCtrlAttribute (g_Handle, L1, ATTR_FRAME_COLOR, VAL_RED);
                   SetCtrlVal (g_Handle, T1, "Internal Bath Temp");
             SetCtrlAttribute (g_Handle, L2, ATTR_FRAME_COLOR, VAL_WHITE);
                   SetCtrlVal (g_Handle, T2, "Bath Setpoint");
             SetCtrlAttribute (g_Handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_Handle, T3, "");
             SetCtrlAttribute (g_Handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_Handle, T4, "");
             break;
        case 1: // Viscosity
             SetCtrlAttribute (panelHandle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
             SetCtrlAttribute (panelHandle, GRAPH, ATTR_YNAME, "Viscosity [cP]");
             SetCtrlAttribute (g_Handle, L1, ATTR_FRAME_COLOR, VAL_RED);
                   SetCtrlVal (g_Handle, T1, "Viscosity");
             SetCtrlAttribute (g_Handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_Handle, T2, "");
             SetCtrlAttribute (g_Handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_Handle, T3, "");
             SetCtrlAttribute (g_Handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
```

```
                SetCtrlVal (g_Handle, T4, "");
            break;
        case 2: // Experiment Stage
            SetCtrlAttribute (panelHandle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
            SetCtrlAttribute (panelHandle, GRAPH, ATTR_YNAME, "Experiment Stage");
            SetCtrlAttribute (g_Handle, L1, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                SetCtrlVal (g_Handle, T1, "0 (Temp), 1 (Active), 2 (Thixo.)");
                SetCtrlAttribute (panelHandle, GRAPH, ATTR_YDIVISIONS, 3);
            SetCtrlAttribute (g_Handle, L2, ATTR_FRAME_COLOR, VAL_RED);
                SetCtrlVal (g_Handle, T2, "");
            SetCtrlAttribute (g_Handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                SetCtrlVal (g_Handle, T3, "");
            SetCtrlAttribute (g_Handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                SetCtrlVal (g_Handle, T4, "");
            break;
        case 3: // Radial Temperature Profile
            SetCtrlAttribute (panelHandle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
            SetCtrlAttribute (panelHandle, GRAPH, ATTR_YNAME, "Temperature [°C]");
            SetCtrlAttribute (panelHandle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
            SetCtrlAttribute (panelHandle, GRAPH, ATTR_XNAME, "Radial Position [mm]");
            SetCtrlAttribute (panelHandle, GRAPH, ATTR_XDIVISIONS, 3);
            SetCtrlAttribute (g_Handle, L1, ATTR_FRAME_COLOR, VAL_RED);
                SetCtrlVal (g_Handle, T1, "Temperature");
            SetCtrlAttribute (g_Handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                SetCtrlVal (g_Handle, T2, "");
            SetCtrlAttribute (g_Handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                SetCtrlVal (g_Handle, T3, "");
            SetCtrlAttribute (g_Handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                SetCtrlVal (g_Handle, T4, "");
            break;
        case 4:
            SetCtrlAttribute (panelHandle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
            SetCtrlAttribute (panelHandle, GRAPH, ATTR_YNAME, "");
            SetCtrlAttribute (g_Handle, L1, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                SetCtrlVal (g_Handle, T1, "");
            SetCtrlAttribute (g_Handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                SetCtrlVal (g_Handle, T2, "");
            SetCtrlAttribute (g_Handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                SetCtrlVal (g_Handle, T3, "");
            SetCtrlAttribute (g_Handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                SetCtrlVal (g_Handle, T4, "");
            break;
        }
return;
}

/*****************************************************************************************************************/
//   .d8888b.   .d88888b.  888b     d888        .d8888b.  8888888888 88888888888 888     888 8888888b.
//  d88P  Y88b d88P" "Y88b 8888b   d8888       d88P  Y88b 888            888     888     888 888   Y88b
//  888    888 888     888 88888b.d88888       Y88b.      888            888     888     888 888    888
//  888        888     888 888Y88888P888        "Y888b.   8888888        888     888     888 888   d88P
//  888        888     888 888 Y888P 888           "Y88b. 888            888     888     888 8888888P"
//  888    888 888     888 888  Y8P  888             "888 888            888     888     888 888
//  Y88b  d88P Y88b. .d88P 888   "   888       Y88b  d88P 888            888     Y88b. .d88P 888
//   "Y8888P"   "Y88888P"  888       888        "Y8888P"  8888888888     888      "Y88888P"  888
/*****************************************************************************************************************/

//*******************************************************//
//          COM #1 - Cole Parmer Fluid Bath               //
//*******************************************************//
//-------------------------------------------------------
// SP_OPEN          : Call to Open the Com Port
//-------------------------------------------------------
int CVICALLBACK SP_OPEN (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            DSN_SP_OPEN();
            break;
        case EVENT_RIGHT_CLICK:
            break;
        }
    return 0;
}
//-------------------------------------------------------
// DSN_SP_OPEN      : Opens the com port
//-------------------------------------------------------
void DSN_SP_OPEN(void)
{   int test;
    SetCtrlVal (com_Handle, COM_SP_STRING_1, '\0');
    SetCtrlVal (com_Handle, COM_SP_STRING_2, '\0');
    SetCtrlVal (com_Handle, COM_SP_STRING_3, '\0');
    SetCtrlVal (com_Handle, COM_SP_STRING_4, '\0');
    rmd2[0]='\0';
    DSN_GetSPConfig();

    SP_port_open = 0;  /* initialize flag to 0 - unopened */
    DisableBreakOnLibraryErrors ();
    RS232Error = OpenComConfig (SP_comport, "", SP_baudrate, SP_parity,
                                SP_databits, SP_stopbits, SP_inputq, SP_outputq);
    SetCtrlVal  (com_Handle, COM_NUM,   RS232Error);
    EnableBreakOnLibraryErrors ();

    if (RS232Error)
        {
        DisplayRS232Error ();
        }
    else
        {
        SP_port_open = 1;
        SetCtrlVal  (com_Handle, COM_NUM,   SP_port_open);
        GetCtrlVal (com_Handle, COM_SP_XMODE, &SP_xmode);
        SetXMode (SP_comport, SP_xmode);
        GetCtrlVal (com_Handle, COM_SP_CTS, &SP_ctsmode);
        SetCTSMode (SP_comport, SP_ctsmode);
        GetCtrlVal (com_Handle, COM_SP_TIMEOUT, &SP_timeout);
        SetComTime (SP_comport, SP_timeout);
        }

    //SetCtrlVal (com_Handle, COM_SP_STRING_2, rmd2);

    if (SP_port_open)
        {
```

```
        SetCtrlAttribute (com_Handle,  COM_SP_LED,   ATTR_DIMMED, 0);
        SetCtrlAttribute (panelHandle, PANEL_SP_LED, ATTR_DIMMED, 0);


        test=1;
        if(test)
            {
            SetCtrlVal (com_Handle,  COM_SP_LED,   1);
            SetCtrlVal (panelHandle, PANEL_SP_LED, 1);
            }
        else
            {
            SetCtrlVal (com_Handle,  COM_SP_LED,   0);
            SetCtrlVal (panelHandle, PANEL_SP_LED, 0);
            }

        }

return;
}
//---------------------------------------------------------
// DSN_GetSPConfig  : Gets COM Configuration from Panel
//---------------------------------------------------------
void DSN_GetSPConfig (void)
{

    DSN_Save_Vars();

    #ifdef _NI_unix_
        SP_devicename[0]=0;
    #else
        GetLabelFromIndex (com_Handle, COM_SP_COM, SP_portindex,
                           SP_devicename);
    #endif
}

//*********************************************************//
//          COM #2 - Viscojet Viscometer                   //
//*********************************************************//
//---------------------------------------------------------
// SP2_OPEN         : Call to Open the Com Port
//---------------------------------------------------------
int CVICALLBACK SP2_OPEN (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            DSN_SP2_OPEN();
            break;
        case EVENT_RIGHT_CLICK:
            break;
        }
    return 0;
}
//---------------------------------------------------------
// DSN_SP2_OPEN     : Opens the com port
//---------------------------------------------------------
void DSN_SP2_OPEN(void)
{   int test;
    SetCtrlVal (com_Handle, COM_SP_STRING_1, '\0');
    SetCtrlVal (com_Handle, COM_SP_STRING_2, '\0');
    SetCtrlVal (com_Handle, COM_SP_STRING_3, '\0');
    SetCtrlVal (com_Handle, COM_SP_STRING_4, '\0');
    rmd2[0]='\0';
    DSN_GetSP2Config();

    SP2_port_open = 0;  /* initialize flag to 0 - unopened */
    DisableBreakOnLibraryErrors ();
    RS232Error = OpenComConfig (SP2_comport, "", SP2_baudrate, SP2_parity,
                                SP2_databits, SP2_stopbits, SP2_inputq, SP2_outputq);
    SetCtrlVal  (com_Handle, COM_NUM,   RS232Error);
    EnableBreakOnLibraryErrors ();

    if (RS232Error)
        {
        DisplayRS232Error ();
        }
    else
        {
        SP2_port_open = 1;
        SetCtrlVal  (com_Handle, COM_NUM,   SP2_port_open);
        GetCtrlVal (com_Handle, COM_SP_XMODE_2, &SP2_xmode);
        SetXMode (SP2_comport, SP2_xmode);
        GetCtrlVal (com_Handle, COM_SP_CTS_2, &SP2_ctsmode);
        SetCTSMode (SP2_comport, SP2_ctsmode);
        GetCtrlVal (com_Handle, COM_SP_TIMEOUT_2, &SP2_timeout);
        SetComTime (SP2_comport, SP2_timeout);
        }

    //SetCtrlVal (com_Handle, COM_SP2_STRING_2, rmd2);

    if (SP2_port_open)
        {
        SetCtrlAttribute (com_Handle,  COM_SP_LED_2,   ATTR_DIMMED, 0);
        SetCtrlAttribute (panelHandle, PANEL_SP_LED_2, ATTR_DIMMED, 0);


        test=1;
        if(test)
            {
            SetCtrlVal (com_Handle,  COM_SP_LED_2,   1);
            SetCtrlVal (panelHandle, PANEL_SP_LED_2, 1);
            }
        else
            {
            SetCtrlVal (com_Handle,  COM_SP_LED_2,   0);
            SetCtrlVal (panelHandle, PANEL_SP_LED_2, 0);
            }

        }

return;
}
//---------------------------------------------------------
```

```
// DSN_GetSP2Config : Gets COM Configuration from Panel
//-----------------------------------------------------
void DSN_GetSP2Config (void)
{

    DSN_Save_Vars();

    #ifdef _NI_unix_
        SP2_devicename[0]=0;
    #else
        GetLabelFromIndex (com_Handle, COM_SP_COM_2, SP2_portindex,
                           SP2_devicename);
    #endif
}

//*****************************************************//
//          COM #3 - GE Pressure Controller              //
//*****************************************************//
//-----------------------------------------------------
// SP3_OPEN         : Call to Open the Com Port
//-----------------------------------------------------
int CVICALLBACK SP3_OPEN (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            DSN_SP3_OPEN();
            break;
        case EVENT_RIGHT_CLICK:
            break;
        }
    return 0;
}
//-----------------------------------------------------
// DSN_SP3_OPEN     : Opens the com port
//-----------------------------------------------------
void DSN_SP3_OPEN(void)
{   int test;
    SetCtrlVal (com_Handle, COM_SP_STRING_1, '\0');
    SetCtrlVal (com_Handle, COM_SP_STRING_2, '\0');
    SetCtrlVal (com_Handle, COM_SP_STRING_3, '\0');
    SetCtrlVal (com_Handle, COM_SP_STRING_4, '\0');
    rmd2[0]='\0';
    DSN_GetSP3Config();

    SP3_port_open = 0;  /* initialize flag to 0 - unopened */
    DisableBreakOnLibraryErrors ();
    RS232Error = OpenComConfig (SP3_comport, "", SP3_baudrate, SP3_parity,
                                SP3_databits, SP3_stopbits, SP3_inputq, SP3_outputq);
    SetCtrlVal  (com_Handle, COM_NUM,   RS232Error);
    EnableBreakOnLibraryErrors ();

    if (RS232Error)
        {
        DisplayRS232Error ();
        }
    else
        {
        SP3_port_open = 1;
        SetCtrlVal  (com_Handle, COM_NUM,   SP3_port_open);
        GetCtrlVal (com_Handle, COM_SP_XMODE_3, &SP3_xmode);
        SetXMode (SP3_comport, SP3_xmode);
        GetCtrlVal (com_Handle, COM_SP_CTS_3, &SP3_ctsmode);
        SetCTSMode (SP3_comport, SP3_ctsmode);
        GetCtrlVal (com_Handle, COM_SP_TIMEOUT_3, &SP3_timeout);
        SetComTime (SP3_comport, SP3_timeout);
        }

    //SetCtrlVal (com_Handle, COM_SP3_STRING_3, rmd3);

    if (SP3_port_open)
        {
        SetCtrlAttribute (com_Handle,  COM_SP_LED_3,   ATTR_DIMMED, 0);
        SetCtrlAttribute (panelHandle, PANEL_SP_LED_3, ATTR_DIMMED, 0);


        test=1;
        if(test)
            {
            SetCtrlVal (com_Handle,  COM_SP_LED_3,   1);
            SetCtrlVal (panelHandle, PANEL_SP_LED_3, 1);
            }
        else
            {
            SetCtrlVal (com_Handle,  COM_SP_LED_3,   0);
            SetCtrlVal (panelHandle, PANEL_SP_LED_3, 0);
            }

        }

return;
}
//-----------------------------------------------------
// DSN_GetSP3Config : Gets COM Configuration from Panel
//-----------------------------------------------------
void DSN_GetSP3Config (void)
{

    DSN_Save_Vars();

    #ifdef _NI_unix_
        SP3_devicename[0]=0;
    #else
        GetLabelFromIndex (com_Handle, COM_SP_COM_3, SP3_portindex,
                           SP3_devicename);
    #endif
}

//*************************************************************//
// DisplayRS232Error    : Display error information to the user.  //
//*************************************************************//
void DisplayRS232Error (void)
{
    char ErrorMessage[200];
```

```
    switch (RS232Error)
        {
        default :
            if (RS232Error < 0)
                {
                Fmt (ErrorMessage, "%s<RS232 error number %i", RS232Error);
                MessagePopup ("RS232 Message", ErrorMessage);
                }
            break;
        case 0  :
            MessagePopup ("RS232 Message", "No errors.");
            break;
        case -2 :
            Fmt (ErrorMessage, "%s", "Connection failed. \n"
                 "Check port settings.");
            MessagePopup ("RS232 Message", ErrorMessage);
            break;
        case -3 :
            Fmt (ErrorMessage, "%s", "No port is open.\n"
                 "Check port settings.");
            MessagePopup ("RS232 Message", ErrorMessage);
            break;
        case -99 :
            Fmt (ErrorMessage, "%s", "Timeout error.\n"
                 "Check port settings.");
            MessagePopup ("RS232 Message", ErrorMessage);
            break;
        }
}

/*****************************************************************************************************************/
//   .d8888b.   .d88888b.  888b    888 8888888888 8888888 .d8888b.        8888888888 8888888 888      8888888888
//  d88P  Y88b d88P" "Y88b 8888b   888 888          888  d88P  Y88b       888          888   888      888
//  888    888 888     888 88888b  888 888          888  888    888       888          888   888      888
//  888        888     888 888Y88b 888 8888888      888  888              8888888      888   888      8888888
//  888        888     888 888 Y88b888 888          888  888  88888       888          888   888      888
//  888    888 888     888 888  Y88888 888          888  888    888       888          888   888      888
//  Y88b  d88P Y88b. .d88P 888   Y8888 888          888  Y88b  d88P       888          888   888      888
//   "Y8888P"   "Y88888P"  888    Y888 888        8888888 "Y8888P88       888        8888888 88888888 8888888888
/*****************************************************************************************************************/

//*******************************************************
// Load_Config      : Load variables from config file
//                  : Update screen from variables
//*******************************************************
void CVICALLBACK Load_Config (int menuBar, int menuItem, void *callbackData,
        int panel)
{
    DSN_Load_Config();
    DSN_Load_Vars();
    DSN_Update_Graphics();
}
//*******************************************************
// Save_Config      : Save screen to variables
//                  : Save variables to config file
//*******************************************************
void CVICALLBACK Save_Config (int menuBar, int menuItem, void *callbackData,
        int panel)
{
    DSN_Save_Vars();
    DSN_Save_Config(0);
}
//*******************************************************
// Save_Config_As   : Save screen to variables
//                  : Save variables to config file
//                  : Allow the config file to be selected
//*******************************************************
void CVICALLBACK Save_Config_As (int menuBar, int menuItem, void *callbackData,
        int panel)
{
    DSN_Save_Vars();
    DSN_Save_Config(1);
}
//*******************************************************
// DSN_Save_Vars    : Save screen to variables
//*******************************************************
void DSN_Save_Vars(void)
{  int i;
    double J_temp,ALPHA_temp;

// MAIN PANEL

    // COM Ports
    GetCtrlVal(panelHandle, PANEL_LOG_COM3,         &Is_COM3);
    GetCtrlVal(panelHandle, PANEL_LOG_COM2,         &Is_COM2);
    GetCtrlVal(panelHandle, PANEL_LOG_COM1,         &Is_COM1);
    GetCtrlVal(panelHandle, PANEL_LOG_DAQ,          &Is_DAQ);
    GetCtrlVal(panelHandle, PANEL_LOG_FUNCTION_GEN, &Is_Function_Gen);
    GetCtrlVal(panelHandle, PANEL_LOG_Graph,        &Is_Graph);
    GetCtrlVal(panelHandle, PANEL_LOG_LOGtoFile,    &Is_Log);
    // A Sensors
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_1A,     &Is_S1A);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_2A,     &Is_S2A);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_3A,     &Is_S3A);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_4A,     &Is_S4A);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_5A,     &Is_S5A);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_6A,     &Is_S6A);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_7A,     &Is_S7A);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_8A,     &Is_S8A);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_9A,     &Is_S9A);
    // B Sensors
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_1B,     &Is_S1B);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_2B,     &Is_S2B);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_3B,     &Is_S3B);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_4B,     &Is_S4B);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_5B,     &Is_S5B);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_6B,     &Is_S6B);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_7B,     &Is_S7B);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_8B,     &Is_S8B);
    GetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_9B,     &Is_S9B);
    // Sensor Positions
    DSN_Get_Sensor_Positions();
    // Equipment Setup
    GetCtrlVal(panelHandle, PANEL_EXP_TYPE,             &Exp_Type);
    GetCtrlVal(panelHandle, PANEL_RUN_MODE,             &Run_Mode);
```

```
// GRAPH PANEL
    //Read Graph #1
    GetCtrlVal( g_Handle,   G_SETUP_Var_G_1,        &plotVar_G_1);
    GetCtrlVal( g_Handle,   G_SETUP_X_Range_G_1,    &X_Range_G_1);
    GetCtrlVal( g_Handle,   G_SETUP_Y_Mode_G_1,     &Y_Mode_G_1);
    GetCtrlVal( g_Handle,   G_SETUP_Y_Min_G_1,      &Y_Min_G_1);
    GetCtrlVal( g_Handle,   G_SETUP_Y_Max_G_1,      &Y_Max_G_1);
    if (Y_Min_G_1 >= Y_Max_G_1)
          Y_Min_G_1 = Y_Max_G_1-0.5;
          SetCtrlVal( g_Handle,     G_SETUP_Y_Min_G_1,      Y_Min_G_1);
    //Read Graph #2
    GetCtrlVal( g_Handle,   G_SETUP_Var_G_2,        &plotVar_G_2);
    GetCtrlVal( g_Handle,   G_SETUP_X_Range_G_2,    &X_Range_G_2);
    GetCtrlVal( g_Handle,   G_SETUP_Y_Mode_G_2,     &Y_Mode_G_2);
    GetCtrlVal( g_Handle,   G_SETUP_Y_Min_G_2,      &Y_Min_G_2);
    GetCtrlVal( g_Handle,   G_SETUP_Y_Max_G_2,      &Y_Max_G_2);
    if (Y_Min_G_2 >= Y_Max_G_2)
          Y_Min_G_2 = Y_Max_G_2-0.5;
          SetCtrlVal( g_Handle,     G_SETUP_Y_Min_G_2,      Y_Min_G_2);
    //Read Graph #3
    GetCtrlVal( g_Handle,   G_SETUP_Var_G_3,        &plotVar_G_3);
    GetCtrlVal( g_Handle,   G_SETUP_X_Range_G_3,    &X_Range_G_3);
    GetCtrlVal( g_Handle,   G_SETUP_Y_Mode_G_3,     &Y_Mode_G_3);
    GetCtrlVal( g_Handle,   G_SETUP_Y_Min_G_3,      &Y_Min_G_3);
    GetCtrlVal( g_Handle,   G_SETUP_Y_Max_G_3,      &Y_Max_G_3);
    if (Y_Min_G_3 >= Y_Max_G_3)
          Y_Min_G_3 = Y_Max_G_3-0.5;
          SetCtrlVal( g_Handle,     G_SETUP_Y_Min_G_3,      Y_Min_G_3);
    //Read Graph #4
    GetCtrlVal( g_Handle,   G_SETUP_Var_G_4,        &plotVar_G_4);
    GetCtrlVal( g_Handle,   G_SETUP_X_Range_G_4,    &X_Range_G_4);
    GetCtrlVal( g_Handle,   G_SETUP_Y_Mode_G_4,     &Y_Mode_G_4);
    GetCtrlVal( g_Handle,   G_SETUP_Y_Min_G_4,      &Y_Min_G_4);
    GetCtrlVal( g_Handle,   G_SETUP_Y_Max_G_4,      &Y_Max_G_4);
    if (Y_Min_G_4 >= Y_Max_G_4)
          Y_Min_G_4 = Y_Max_G_4-0.5;
          SetCtrlVal( g_Handle,     G_SETUP_Y_Min_G_4,      Y_Min_G_4);

// COM PANEL
    GetCtrlVal( com_Handle, COM_SP_COM,             &SP_comport);
    GetCtrlVal( com_Handle, COM_SP_BR,              &SP_baudrate);
    GetCtrlVal( com_Handle, COM_SP_P,               &SP_parity);
    GetCtrlVal( com_Handle, COM_SP_DB,              &SP_databits);
    GetCtrlVal( com_Handle, COM_SP_SB,              &SP_stopbits);
    GetCtrlVal( com_Handle, COM_SP_INPUTQ,          &SP_inputq);
    GetCtrlVal( com_Handle, COM_SP_OUTPUTQ,         &SP_outputq);
    GetCtrlIndex(com_Handle, COM_SP_COM,            &SP_portindex);

    GetCtrlVal( com_Handle, COM_SP_COM_2,           &SP2_comport);
    GetCtrlVal( com_Handle, COM_SP_BR_2,            &SP2_baudrate);
    GetCtrlVal( com_Handle, COM_SP_P_2,             &SP2_parity);
    GetCtrlVal( com_Handle, COM_SP_DB_2,            &SP2_databits);
    GetCtrlVal( com_Handle, COM_SP_SB_2,            &SP2_stopbits);
    GetCtrlVal( com_Handle, COM_SP_INPUTQ_2,        &SP2_inputq);
    GetCtrlVal( com_Handle, COM_SP_OUTPUTQ_2,       &SP2_outputq);
    GetCtrlIndex(com_Handle, COM_SP_COM_2,          &SP2_portindex);

    GetCtrlVal( com_Handle, COM_SP_COM_3,           &SP3_comport);
    GetCtrlVal( com_Handle, COM_SP_BR_3,            &SP3_baudrate);
    GetCtrlVal( com_Handle, COM_SP_P_3,             &SP3_parity);
    GetCtrlVal( com_Handle, COM_SP_DB_3,            &SP3_databits);
    GetCtrlVal( com_Handle, COM_SP_SB_3,            &SP3_stopbits);
    GetCtrlVal( com_Handle, COM_SP_INPUTQ_3,        &SP3_inputq);
    GetCtrlVal( com_Handle, COM_SP_OUTPUTQ_3,       &SP3_outputq);
    GetCtrlIndex(com_Handle, COM_SP_COM_3,          &SP3_portindex);

// TEST PARAMETERS
    GetCtrlVal(panelHandle, PANEL_P_MIN_SP, &P_Min_SP);
    GetCtrlVal(panelHandle, PANEL_P_MAX_SP, &P_Max_SP);
    GetCtrlVal(panelHandle, PANEL_P_SP_INCREMENTS, &P_SP_Increments);

    GetCtrlVal(panelHandle, PANEL_P_MAX_T, &P_Max_T);
    GetCtrlVal(panelHandle, PANEL_P_MIN_T, &P_Min_T);
    GetCtrlVal(panelHandle, PANEL_P_T_INCREMENTS, &P_T_Increments);

    GetCtrlVal(panelHandle, PANEL_DP_STIM_DURATION, &DP_Stim_Duration);
    GetCtrlVal(panelHandle, PANEL_MTS_SPAN, &MTS_Span);

    GetCtrlVal(panelHandle, PANEL_P_F_INCREMENTS, &P_F_Increments);
    GetCtrlVal(panelHandle, PANEL_P_MAX_F, &P_Max_F);
    GetCtrlVal(panelHandle, PANEL_P_MIN_F, &P_Min_F);
    GetCtrlVal(panelHandle, PANEL_DWELL_TIME, &StimDwellTime);

    GetCtrlVal(panelHandle, PANEL_P_DP_AMP_1, &DP_Amp_Array_Save[0]);
    GetCtrlVal(panelHandle, PANEL_P_DP_AMP_2, &DP_Amp_Array_Save[1]);
    GetCtrlVal(panelHandle, PANEL_P_DP_AMP_3, &DP_Amp_Array_Save[2]);
    GetCtrlVal(panelHandle, PANEL_P_DP_AMP_4, &DP_Amp_Array_Save[3]);
    GetCtrlVal(panelHandle, PANEL_P_DP_AMP_5, &DP_Amp_Array_Save[4]);
    GetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_1, &FuncGenVolt_Array_Save[0]);
    GetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_2, &FuncGenVolt_Array_Save[1]);
    GetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_3, &FuncGenVolt_Array_Save[2]);
    GetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_4, &FuncGenVolt_Array_Save[3]);
    GetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_5, &FuncGenVolt_Array_Save[4]);

    GetCtrlVal(panelHandle, PANEL_Amp_On_1, &Amp_On_1);
    GetCtrlVal(panelHandle, PANEL_Amp_On_2, &Amp_On_2);
    GetCtrlVal(panelHandle, PANEL_Amp_On_3, &Amp_On_3);
    GetCtrlVal(panelHandle, PANEL_Amp_On_4, &Amp_On_4);
    GetCtrlVal(panelHandle, PANEL_Amp_On_5, &Amp_On_5);


return;
}
//******************************************************
// DSN_Load_Vars    : Load variables to the screen
//******************************************************
void DSN_Load_Vars(void)
{
// MAIN PANEL

    //COM Ports
    SetCtrlVal(panelHandle, PANEL_LOG_COM3,         Is_COM3);
    SetCtrlVal(panelHandle, PANEL_LOG_COM2,         Is_COM2);
    SetCtrlVal(panelHandle, PANEL_LOG_COM1,         Is_COM1);
```

```
    SetCtrlVal(panelHandle, PANEL_LOG_DAQ,          Is_DAQ);
    SetCtrlVal(panelHandle, PANEL_LOG_FUNCTION_GEN, Is_Function_Gen);
    SetCtrlVal(panelHandle, PANEL_LOG_Graph,        Is_Graph);
    SetCtrlVal(panelHandle, PANEL_LOG_LOGtoFile,    Is_Log);

// GRAPH PANEL

    //Read Graph #1
    SetCtrlVal( g_Handle,   G_SETUP_Var_G_1,        plotVar_G_1);
    SetCtrlVal( g_Handle,   G_SETUP_X_Range_G_1,    X_Range_G_1);
    SetCtrlVal( g_Handle,   G_SETUP_Y_Mode_G_1,     Y_Mode_G_1);
    SetCtrlVal( g_Handle,   G_SETUP_Y_Min_G_1,      Y_Min_G_1);
    SetCtrlVal( g_Handle,   G_SETUP_Y_Max_G_1,      Y_Max_G_1);
    //Read Graph #2
    SetCtrlVal( g_Handle,   G_SETUP_Var_G_2,        plotVar_G_2);
    SetCtrlVal( g_Handle,   G_SETUP_X_Range_G_2,    X_Range_G_2);
    SetCtrlVal( g_Handle,   G_SETUP_Y_Mode_G_2,     Y_Mode_G_2);
    SetCtrlVal( g_Handle,   G_SETUP_Y_Min_G_2,      Y_Min_G_2);
    SetCtrlVal( g_Handle,   G_SETUP_Y_Max_G_2,      Y_Max_G_2);
    //Read Graph #3
    SetCtrlVal( g_Handle,   G_SETUP_Var_G_3,        plotVar_G_3);
    SetCtrlVal( g_Handle,   G_SETUP_X_Range_G_3,    X_Range_G_3);
    SetCtrlVal( g_Handle,   G_SETUP_Y_Mode_G_3,     Y_Mode_G_3);
    SetCtrlVal( g_Handle,   G_SETUP_Y_Min_G_3,      Y_Min_G_3);
    SetCtrlVal( g_Handle,   G_SETUP_Y_Max_G_3,      Y_Max_G_3);
    //Read Graph #4
    SetCtrlVal( g_Handle,   G_SETUP_Var_G_4,        plotVar_G_4);
    SetCtrlVal( g_Handle,   G_SETUP_X_Range_G_4,    X_Range_G_4);
    SetCtrlVal( g_Handle,   G_SETUP_Y_Mode_G_4,     Y_Mode_G_4);
    SetCtrlVal( g_Handle,   G_SETUP_Y_Min_G_4,      Y_Min_G_4);
    SetCtrlVal( g_Handle,   G_SETUP_Y_Max_G_4,      Y_Max_G_4);

    // A Sensors
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_1A, Is_S1A);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_2A, Is_S2A);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_3A, Is_S3A);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_4A, Is_S4A);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_5A, Is_S5A);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_6A, Is_S6A);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_7A, Is_S7A);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_8A, Is_S8A);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_9A, Is_S9A);
    // B Sensors
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_1B, Is_S1B);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_2B, Is_S2B);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_3B, Is_S3B);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_4B, Is_S4B);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_5B, Is_S5B);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_6B, Is_S6B);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_7B, Is_S7B);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_8B, Is_S8B);
    SetCtrlVal(panelHandle, PANEL_SENSOR_SELECT_9B, Is_S9B);
    // Sensor Positions
    SetCtrlVal(panelHandle, PANEL_S1A_POSITION, S1A_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S2A_POSITION, S2A_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S3A_POSITION, S3A_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S4A_POSITION, S4A_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S5A_POSITION, S5A_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S6A_POSITION, S6A_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S7A_POSITION, S7A_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S8A_POSITION, S8A_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S9A_POSITION, S9A_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S1B_POSITION, S1B_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S2B_POSITION, S2B_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S3B_POSITION, S3B_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S4B_POSITION, S4B_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S5B_POSITION, S5B_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S6B_POSITION, S6B_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S7B_POSITION, S7B_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S8B_POSITION, S8B_Pos[0]);
    SetCtrlVal(panelHandle, PANEL_S9B_POSITION, S9B_Pos[0]);
    // Sensor Position Array
    Key_Radial_Positions[0] = S8B_Pos[0];
    Key_Radial_Positions[1] = S8A_Pos[0];
    Key_Radial_Positions[2] = S7B_Pos[0];
    Key_Radial_Positions[3] = S7A_Pos[0];
    // Equipment Setup
    SetCtrlVal(panelHandle, PANEL_EXP_TYPE,         Exp_Type);
    SetCtrlVal(panelHandle, PANEL_RUN_MODE,         Run_Mode);

// COM PANEL
    SetCtrlVal( com_Handle, COM_SP_COM,             SP_comport);
    SetCtrlVal( com_Handle, COM_SP_BR,              SP_baudrate);
    SetCtrlVal( com_Handle, COM_SP_P,               SP_parity);
    SetCtrlVal( com_Handle, COM_SP_DB,              SP_databits);
    SetCtrlVal( com_Handle, COM_SP_SB,              SP_stopbits);
    SetCtrlVal( com_Handle, COM_SP_INPUTQ,          SP_inputq);
    SetCtrlVal( com_Handle, COM_SP_OUTPUTQ,         SP_outputq);
    //SetCtrlIndex(com_Handle, COM_SP_COM,          SP_portindex);

    SetCtrlVal( com_Handle, COM_SP_COM_2,           SP2_comport);
    SetCtrlVal( com_Handle, COM_SP_BR_2,            SP2_baudrate);
    SetCtrlVal( com_Handle, COM_SP_P_2,             SP2_parity);
    SetCtrlVal( com_Handle, COM_SP_DB_2,            SP2_databits);
    SetCtrlVal( com_Handle, COM_SP_SB_2,            SP2_stopbits);
    SetCtrlVal( com_Handle, COM_SP_INPUTQ_2,        SP2_inputq);
    SetCtrlVal( com_Handle, COM_SP_OUTPUTQ_2,       SP2_outputq);

    SetCtrlVal( com_Handle, COM_SP_COM_3,           SP3_comport);
    SetCtrlVal( com_Handle, COM_SP_BR_3,            SP3_baudrate);
    SetCtrlVal( com_Handle, COM_SP_P_3,             SP3_parity);
    SetCtrlVal( com_Handle, COM_SP_DB_3,            SP3_databits);
    SetCtrlVal( com_Handle, COM_SP_SB_3,            SP3_stopbits);
    SetCtrlVal( com_Handle, COM_SP_INPUTQ_3,        SP3_inputq);
    SetCtrlVal( com_Handle, COM_SP_OUTPUTQ_3,       SP3_outputq);

// TEST PARAMETERS
    SetCtrlVal(panelHandle, PANEL_P_MIN_SP, P_Min_SP);
    SetCtrlVal(panelHandle, PANEL_P_MAX_SP, P_Max_SP);
    SetCtrlVal(panelHandle, PANEL_P_SP_INCREMENTS, P_SP_Increments);

    SetCtrlVal(panelHandle, PANEL_P_MAX_T, P_Max_T);
    SetCtrlVal(panelHandle, PANEL_P_MIN_T, P_Min_T);
    SetCtrlVal(panelHandle, PANEL_P_T_INCREMENTS, P_T_Increments);
```

```
    SetCtrlVal(panelHandle, PANEL_DP_STIM_DURATION, DP_Stim_Duration);
    SetCtrlVal(panelHandle, PANEL_MTS_SPAN, MTS_Span);

    SetCtrlVal(panelHandle, PANEL_P_F_INCREMENTS, P_F_Increments);
    SetCtrlVal(panelHandle, PANEL_P_MAX_F, P_Max_F);
    SetCtrlVal(panelHandle, PANEL_P_MIN_F, P_Min_F);
    SetCtrlVal(panelHandle, PANEL_DWELL_TIME, StimDwellTime);

    SetCtrlVal(panelHandle, PANEL_Amp_On_1, Amp_On_1);
    SetCtrlVal(panelHandle, PANEL_Amp_On_2, Amp_On_2);
    SetCtrlVal(panelHandle, PANEL_Amp_On_3, Amp_On_3);
    SetCtrlVal(panelHandle, PANEL_Amp_On_4, Amp_On_4);
    SetCtrlVal(panelHandle, PANEL_Amp_On_5, Amp_On_5);

    SetCtrlVal(panelHandle, PANEL_P_DP_AMP_1, DP_Amp_Array_Save[0]);
    SetCtrlVal(panelHandle, PANEL_P_DP_AMP_2, DP_Amp_Array_Save[1]);
    SetCtrlVal(panelHandle, PANEL_P_DP_AMP_3, DP_Amp_Array_Save[2]);
    SetCtrlVal(panelHandle, PANEL_P_DP_AMP_4, DP_Amp_Array_Save[3]);
    SetCtrlVal(panelHandle, PANEL_P_DP_AMP_5, DP_Amp_Array_Save[4]);

    SetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_1, FuncGenVolt_Array_Save[0]);
    SetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_2, FuncGenVolt_Array_Save[1]);
    SetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_3, FuncGenVolt_Array_Save[2]);
    SetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_4, FuncGenVolt_Array_Save[3]);
    SetCtrlVal(panelHandle, PANEL_FUNC_GEN_VOLT_5, FuncGenVolt_Array_Save[4]);


return;
}
//*******************************************************
// DSN_Init2        : Init some variable and do some setup
//*******************************************************
void DSN_Init2()
{ // Set the colours up

return;
}

//*******************************************************
//*******************************************************
//*******************************************************
```

Appendix H MATLAB Post Processing Software Source Code

```
%% MATLAB Plotting Script
% Written by: Marc Evans
% Last Updated: March 14, 2010
%
% This script generates a series of plots from a data file created by the
% Monitoring and Control Program that runs the COSI Acoustic Stimulation
% Chamber. The user is prompted to select the data file he/she wishes to
% plot and then the script generates all the relevant figures.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%------------------------------ START ---------------------------------%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Prompt the user for a log file and calibration file and imports all data
%------------------%
% Import Data File
%------------------%
% Opens a file selection box for *.log files
[fileToRead1,pathname] = uigetfile( ...
       {'*.log', 'Data Files (*.log)'; ...
        '*.*', 'All Files (*.*)'}, ...
        'Select Log File', ...
        'MultiSelect', 'off');

% If file selection is cancelled, pathname should be zero
% and nothing should happen
if pathname == 0
    return
end

% Import the file
newData1 = importdata(fileToRead1);

% Break the data up into a new structure with one field per column.
colheaders = genvarname(newData1.colheaders);
for i = 1:length(colheaders)
    dataByColumn1.(colheaders{i}) = newData1.data(:, i);
end

% Create new variables in the base workspace from those fields.
vars = fieldnames(dataByColumn1);
for i = 1:length(vars)
    assignin('base', vars{i}, dataByColumn1.(vars{i}));
end

% Prompt user to enter the name of the fluid
Fluid_Name = ...
    input(' Name of Fluid: \n Format (including brackets and dash): (Name - xx% mass) \n',↙
's');
if isempty(Fluid_Name)
    Fluid_Name = '**No fluid given**';
end

%-------------------------%
% Import Calibration File
%-------------------------%
% Opens a file selection box for *.txt files
[fileToRead1,pathname] = uigetfile( ...
       {'*.txt', 'Data Files (*.txt)'; ...
        '*.*', 'All Files (*.*)'}, ...
        'Select Calibration File', ...
```

```
        'MultiSelect', 'off');

% If file selection is cancelled, pathname should be zero
% and nothing should happen
if pathname == 0
    return
end

% Import the file
newData1 = importdata(fileToRead1);

% Break the data up into a new structure with one field per column.
colheaders = genvarname(newData1.colheaders);
for i = 1:length(colheaders)
    dataByColumn1.(colheaders{i}) = newData1.data(:, i);
end

% Create new variables in the base workspace from those fields.
vars = fieldnames(dataByColumn1);
for i = 1:length(vars)
    assignin('base', vars{i}, dataByColumn1.(vars{i}));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Prepare Variables
%--------------------------%
% Setup General Variables
%--------------------------%
% Scale Select Variables
Time_Hours = TIME./3600;
Exp_Stage = Logging_Trigger.*2;

% Apply Calibration Factors
S1A = S1A*mS1A + bS1A;
S2A = S2A*mS2A + bS2A;
S3A = S3A*mS3A + bS3A;
S4A = S4A*mS4A + bS4A;
S5A = S5A*mS5A + bS5A;
S6A = S6A*mS6A + bS6A;
%S7A = S7A*mS7A + bS7A;     % Bath Probe RTD
S8A = S8A*mS8A + bS8A;
S9A = S9A*mS9A + bS9A;
S1B = S1B*mS1B + bS1B;
S2B = S2B*mS2B + bS2B;
%S3B = S3B*mS3B + bS3B;     % Pressure Relief Valve
S4B = S4B*mS4B + bS4B;
S5B = S5B*mS5B + bS5B;
S6B = S6B*mS6B + bS6B;
S7B = S7B*mS7B + bS7B;
S8B = S8B*mS8B + bS8B;
S9B = S9B*mS9B + bS9B;

% Determine Experiment Start
Start_Index = find(Logging_Trigger==1);
Start_Index = Start_Index(1);

% Determine Viscosity at Experiment Start
Baseline_Visc = Ave_L_Visc(Start_Index);

% Determine Temperature Limits
Low_Temp_Limit = Bath_Setpoint(Start_Index)-1;
```

```
High_Temp_Limit = Bath_Setpoint(Start_Index)+1;

%------------------------%
% Density Correction
%------------------------%
Fluid_Density = ...
    input(' Please Input Fluid_Density [g/mL]: \n');
if isempty(Fluid_Density)
    Fluid_Density = 1;
end
Ave_L_Visc = Ave_L_Visc./Fluid_Density;

% %----------------------------------------%
% % Correct Temperature/Pressure Instability
% %----------------------------------------%
% % Store Uncorrected Viscosities
% ui = Ave_L_Visc;
%
% % Interpolate Expected Viscosity at Exact Setpoint Conditions
% Ts = Bath_Setpoint(1);
% Ps = 500;
% us = interp2(pressures,temperatures,Static_PT_Viscosities,Ps,Ts);
% us = us.*ones(length(ui),1);
%
% % Interpolate Expected Viscosity at Current Conditions
% Ti = S7A;
% Pi = S2B;
% ut = zeros(length(ui),1);
%
% % Repeat this loop once for each reading
% % Variables to keep track of progress
% counter = 0;
% processed_values = 0;
% display('Interpolating Viscosity Correction Factors')
% percent_done = 0
% total_values = length(ui);
% for i=1:length(ui)
%     ut(i) = interp2(pressures,temperatures,Static_PT_Viscosities,Pi(i),Ti(i));
%     counter=counter+1;
%         if counter>=1000
%             eval(['clc'])
%             processed_values=processed_values+counter;
%             display('Interpolating Viscosity Correction Factors')
%             percent_done = 100*(processed_values/total_values)
%             counter=0;
%         end
% end
%
% % Apply Correction Factor to each reading
% PT_Corr_Viscosities = ui+(us-ut);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Parse Data
%-----------------------------%
% Identify Dynamic Amplitudes
%-----------------------------%
% Store first dynamic pressure amplitude
Num_Dyn_Amps = 1;
Dyn_Amp_Array(1) = Dyn_Press_Amp(1);

% Count # of Distinct Dynamic Pressure Amplitude Steps in "Dyn_Press_Amp"
```

```
for i = 1:length(Dyn_Press_Amp)
    if Dyn_Press_Amp(i) ~= Dyn_Amp_Array(Num_Dyn_Amps)

        % Store # in "Num_Dyn_Amps"
        Num_Dyn_Amps = Num_Dyn_Amps+1;

        % Store Amplitudes in "Dyn_Amp_Array"
        Dyn_Amp_Array(Num_Dyn_Amps) = Dyn_Press_Amp(i);
    end
end

%---------------------------------%
% Parse Data by Dynamic Amplitude
%---------------------------------%
% Each column of a 'Parsed_XXX' variable holds data for one dynamic
% pressure amplitude series

% Variables to keep track of progress
counter = 0;
processed_values = 0;
percent_done = 0
total_values = length(vars)*length(TIME);

% Repeat this loop once for each dynamic amplitude series
for i=1:Num_Dyn_Amps
    % Find indices for one dynamic amplitude series
    Start_Indices = find(Dyn_Press_Amp==Dyn_Amp_Array(i));
    % Using index, assign values to parsed variable column 'i'
    for j=1:length(vars)
        % Repeat once for each index location
        for k=1:length(Start_Indices)
            eval(['Parsed_' vars{j} '(k,i)= dataByColumn1.(vars{j})(Start_Indices(k));']);

            counter=counter+1;
            if counter>=1000
                eval(['clc'])
                processed_values=processed_values+counter;
                display('Parsing Dynamic Amplitude Series')
                percent_done = 100*(processed_values/total_values)
                counter=0;
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Count Frequency Series and Parse Time
%-----------------------------%
% Count Frequency Series
%-----------------------------%
% Store first frequency
Num_Freq = 1;
Freq_Array(1) = Parsed_FuncGen_Freq(10,2);

% Count # of Distinct Frequency Steps in "Parsed_FuncGen_Freq"
for i = 1:length(Parsed_FuncGen_Freq)
    if Parsed_FuncGen_Freq(i,2) ~= Freq_Array(Num_Freq)
        if Parsed_FuncGen_Freq(i,2)~= 0
            % Store # in "Num_Freq"
            Num_Freq = Num_Freq+1;
```

```
            % Store Frequency in "Freq_Array"
            Freq_Array(Num_Freq) = Parsed_FuncGen_Freq(i,2);
        end
    end
end

% Zero each column of the Parsed_TIME matrix
Column_Subtraction = Parsed_TIME(1,:);
for i=1:size(Parsed_TIME,2)
    % Subtract the first time value in each column from the rows
    for j=1:length(Parsed_TIME)
        Parsed_TIME(j,i) = Parsed_TIME(j,i)-Column_Subtraction(i);
        % Zero any resulting negative numbers
        if Parsed_TIME(j,i) <0
            Parsed_TIME(j,i)=0;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Parse Frequency Series and Analyze Data
%---------------------------%
% Identify Frequency Series
%---------------------------%
% Repeat this loop once for each dynamic amplitude series
% (Once per column of Parsed_Ave_L_Visc)
Critical_Data=0;
for a=1:Num_Dyn_Amps
    % Find indices where stimulation is on (logging trigger = 1)
    Current_Column = Parsed_Logging_Trigger(:,a);
    Stimulation_On = find(Current_Column==1);

    % Check each index for new frequency series
    % Store viscosity in appropriate amplitude/frequency array location
    v=0;
    f=1;
    for z=1:length(Stimulation_On)-1
        if (Stimulation_On(z+1)-Stimulation_On(z)==1) % Same freq series
            i = i+1;
            Critical_Data(a,f,i) = Parsed_Ave_L_Visc(Stimulation_On(z),a);

        else % Reached next freq series
            f = f+1;
            i = 1;
            Critical_Data(a,f,i) = Parsed_Ave_L_Visc(Stimulation_On(z),a);
        end
    end
end

%---------------------------------%
% Calculate Viscosity Statistics
%---------------------------------%
% Repeat this loop once for each amplitude series
for a=1:size(Critical_Data,1)
    % Repeat this loop once for each frequency series
    for f=1:size(Critical_Data,2)
        % Find non-zero entries
        viscosities = zeros(1);
        h=1;
        for v=1:size(Critical_Data,3)
            if Critical_Data(a,f,v)~=0
```

```
                viscosities(h) = Critical_Data(a,f,v);
                h=h+1;
            end
        end

        % Only interested in the latter half of the stimulation data
        key_viscosities=zeros(1);
        j=1;
        for i=ceil(length(viscosities)/2):length(viscosities)
            key_viscosities(j)=viscosities(i);
            j=j+1;
        end

        % Calculate statistics
        Viscosity_Stats(a,f,1) = mean(key_viscosities);
        Viscosity_Stats(a,f,2) = max(key_viscosities);
        Viscosity_Stats(a,f,3) = min(key_viscosities);
        Viscosity_Stats(a,f,4) = std(key_viscosities);

    end
end

%-------------------------------------------------%
% Collect Viscosity Values in Interpolation Array
%-------------------------------------------------%
Acoustic_AF_Viscosities = Viscosity_Stats(:,:,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Calculate Parameters: Plotting Parameters
%---------------------------------%
% Calculate Plotting Parameters
%---------------------------------%
% Interpolate Viscosity Limits
temperatures = (20:20:80);
pressures = (0:200:1000);

Mid = interp2(pressures,temperatures,...
    Static_PT_Viscosities,500,Bath_Setpoint(1));

Upper = interp2(pressures,temperatures,...
    Static_PT_Viscosities,500,Bath_Setpoint(1)-0.25);
Upper_Baseline_Visc = Baseline_Visc+(Upper-Mid);

Lower = interp2(pressures,temperatures,...
    Static_PT_Viscosities,500,Bath_Setpoint(1)+0.25);
Lower_Baseline_Visc = Baseline_Visc-(Mid-Lower);

% Deal with extrapolation limit
if(isnan(Upper_Baseline_Visc) == 1)
    Upper_Baseline_Visc = Baseline_Visc+(Mid-Lower);
end
if(isnan(Lower_Baseline_Visc) == 1)
    Lower_Baseline_Visc = Baseline_Visc-(Upper-Mid);
end

% Calculate Max Viscosity for Axis Scaling
Max_Viscosity = max(max(Viscosity_Stats(:,:,1)));
if (Baseline_Visc > Max_Viscosity)
    Max_Viscosity = Baseline_Visc;
end
```

```
% Compute Viscosity Axis Ticks
viscosity_ticks = zeros(1);
tick_increment = 50;
current_tick = 0;
i = 1;
while current_tick < Max_Viscosity+tick_increment,
    viscosity_ticks(i) = current_tick;
    current_tick = current_tick+tick_increment;
    i = i+1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Generate Figure: Viscosity vs Frequency for Different Dynamic Pressures
%----------------------%
% Plot Figure
%----------------------%
% Create figure
figure1 = figure;
set(figure1,'Position',[80,80,1000,800]);

axes1 = axes('Parent',figure1,...
    'XLim',[min(Freq_Array) max(Freq_Array)],...
    'XTickMode','manual',...
    'XTick',[Freq_Array],...
    'YGrid','on',...
    'YLim',[min(viscosity_ticks) max(viscosity_ticks)],...
    'YTickMode','manual',...
    'YTick',[viscosity_ticks]);

box('on');
hold('on');

if(Dyn_Amp_Array(1)~=0)
h1 = errorbar(Freq_Array, Viscosity_Stats(1,:,1),Viscosity_Stats(1,:,4));
end
if (Num_Dyn_Amps>1)
    h2 = errorbar(Freq_Array, Viscosity_Stats(2,:,1),Viscosity_Stats(1,:,4));
end
if (Num_Dyn_Amps>2)
    h3 = errorbar(Freq_Array, Viscosity_Stats(3,:,1),Viscosity_Stats(1,:,4));
end
if (Num_Dyn_Amps>3)
    h4 = errorbar(Freq_Array, Viscosity_Stats(4,:,1),Viscosity_Stats(1,:,4));
end
if (Num_Dyn_Amps>4)
    h5 = errorbar(Freq_Array, Viscosity_Stats(5,:,1),Viscosity_Stats(1,:,4));
end

% Plot Baseline Viscosity
Base = line([0 ; Freq_Array(length(Freq_Array))],...
    [Baseline_Visc ; Baseline_Visc],'Color', 'k','LineWidth',2);
Upper_Lim = line([0 ; length(TIME)],[Lower_Baseline_Visc ; Lower_Baseline_Visc],...
    'LineStyle','--','Color', 'k','LineWidth',2);
Lower_Lim = line([0 ; length(TIME)],[Upper_Baseline_Visc ; Upper_Baseline_Visc],...
    'LineStyle','--','Color', 'k','LineWidth',2);

%----------------------%
% Customize Appearance
%----------------------%
% Title
title({'Viscosity vs Acoustic Stimulation Frequency and Amplitude',...
```

```
    Fluid_Name},...
    'FontWeight','bold');

% Axes
xlabel('Frequency [Hz]')
ylabel('Viscosity [cP]')

% Series
markersize = 7;

if(Dyn_Amp_Array(1)~=0)

    set(h1,'Marker','o','MarkerSize',markersize,...
        'Color', 'b', 'MarkerFaceColor','b',...
        'LineStyle','--') % freq_1
end
if (Num_Dyn_Amps>1)
    set(h2,'Marker','x','MarkerSize',markersize,...
        'Color', 'r', 'MarkerFaceColor','r',...
        'LineStyle','--') % freq_2
end
if (Num_Dyn_Amps>2)
    set(h3,'Marker','s','MarkerSize',markersize,...
        'Color', 'g', 'MarkerFaceColor','g',...
        'LineStyle','--') % freq_3
end
if (Num_Dyn_Amps>3)
    set(h4,'Marker','d','MarkerSize',markersize,...
        'Color', 'm', 'MarkerFaceColor','m',...
        'LineStyle','--') % freq_4
end
if (Num_Dyn_Amps>4)
    set(h5,'Marker','+','MarkerSize',markersize,...
        'Color', 'c', 'MarkerFaceColor','c',...
        'LineStyle','--') % freq_4
end

% Legend
% temp_legend = num2str(Dyn_Amp_Array');
% temp_legend(:,4) = ' ';
% temp_legend(:,5) = 'p';
% temp_legend(:,6) = 's';
% temp_legend(:,7) = 'i';
% temp_legend(size(temp_legend)+1,

legend('100psi','200psi','400psi','0psi at Setpoint T',...
    '0psi at Setpoint ± 0.25°C',...
    'Location','SouthEast');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Generate Figure: Viscosity, Temperature, Experiment Stage vs Time
%----------------------%
% Plot Figure
%----------------------%
% Create figure
figure1 = figure;
set(figure1,'Position',[0,0,1000,800]);

% Create axes
axes1 = axes('Parent',figure1,...
    'YColor',[0 0 1],...
```

```
    'Position',[0.13 0.11 0.775 0.815]);

box('on');
hold('all');

% Plot multiple lines using plotyy
[AX,H1,H2] = plotyy(Time_Hours,[Ave_L_Visc],...
    Time_Hours,[S7A,Exp_Stage],...
    'Parent',axes1);

% Plot constants
axes(AX(1));
H3 = line([0 ; length(TIME)],[Baseline_Visc ; Baseline_Visc],...
    'LineStyle','--');
axes(AX(2));
H4 = line([0 ; length(TIME)],[Low_Temp_Limit ; Low_Temp_Limit],...
    'LineStyle','--');
H5 = line([0 ; length(TIME)],[High_Temp_Limit ; High_Temp_Limit],...
    'LineStyle','--');

%----------------------%
% Customize Appearance
%----------------------%
% Title
title({'Viscosity and Temperature vs Time',...
    '(Also Showing Experiment Progression)'},...
    'FontWeight','bold');

% Axes
xlabel('Time [hrs]')
set(get(AX(1), 'Ylabel'),'String','Viscosity [cP]')
set(AX(1),'ycolor','b',...
    'YTick', [0:100:1000])

set(get(AX(2), 'Ylabel'),'String','Temperature [°C]')
ylim('manual');
ylim([0 100]);
set(AX(2),'ycolor',[0 0.498 0],...
    'YTick', [0:10:100])

% Series
set(H1(1),'Color','b','Marker','.','MarkerSize',1) %Viscosity
set(H2(1),'Color',[0 0.498 0],...   % Temperature
    'LineStyle','-',...
    'Marker','.',...
    'MarkerSize',1)
set(H2(2),'Color','r',...   % Experiment Stage
    'Marker','.',...
    'MarkerSize',1)

% Legend
legend([H1;H3;H2(1);H4;H2(2)],'Viscosity','Initial Viscosity',...
    'Center Temperature','±1°C from Setpoint','Experiment Stage',...
    'Location','SouthEast');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Generate Figure: Viscosity vs Time for Different Dynamic Pressures
%----------------------%
% Plot Figure
%----------------------%
% Create figure
```

```
figure1 = figure;
set(figure1,'Position',[0,0,1000,800]);

% Create axes
axes1 = axes('Parent',figure1,...
    'YColor','k',...
    'Position',[0.13 0.11 0.775 0.815]);

box('on');
hold('all');

AX = plot(Parsed_TIME, Parsed_Ave_L_Visc,'LineStyle','-','Marker','.',...
    'MarkerSize',1);

% Plot constants
Max_Time = max(max(Parsed_TIME));
H = line([0 ; Max_Time],[Baseline_Visc ; Baseline_Visc],...
    'LineStyle','--',...
    'Color','k');

%----------------------%
% Customize Appearance
%----------------------%
% Title
title({'Viscosity vs Time for Different Dynamic Pressure Amplitudes'},...
    'FontWeight','bold');

% Axes
xlabel('Time [sec]');
ylabel('Viscosity [cP]');

% Legend
HL = legend(num2str(transpose(Dyn_Amp_Array)),'Location','SouthEast');
v = get(HL,'title');
set(v,'string','Amplitude [psi]');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Generate Video: Radial Temperatures vs Time
%----------------------%
% Initialize variables
%----------------------%
radial_temps=[S8B(1) S8A(1) S7B(1) S7A(1)];
radial_positions=[S8B_Pos(1) S8A_Pos(1) S7B_Pos(1) S7A_Pos(1)];

%----------------------%
% Plot Figure
%----------------------%
% Create figure
figure1 = figure;
set(figure1,'Position',[50,50,640,480]);

% Create axes
axes1 = axes('Parent',figure1,...
    'YColor','k',...
    'Position',[0.13 0.11 0.775 0.815]);
axis([0 37.5 0 30])

box('on');
hold('all');

AX=plot(radial_positions, radial_temps,'o',...
```

```
    'MarkerSize',7,...
    'MarkerEdgeColor','b',...
    'MarkerFaceColor','r',...
    'LineStyle','-');
H4 = line([0 ; 37.5],[Low_Temp_Limit ; Low_Temp_Limit],...
    'LineStyle','--');
H5 = line([0 ; 37.5],[High_Temp_Limit ; High_Temp_Limit],...
    'LineStyle','--');

hold on

%----------------------%
% Customize Appearance
%----------------------%
% Title
title({'Radial Temperatures vs Time'},...
    'FontWeight','bold');

% Axes
xlabel('Radial Distance from Centre [mm]')
ylabel('Temperature [°C]')

% Legend
legend('Radial Temperature','±1°C from Setpoint');

%-------------------%
% Generate Movie
%-------------------%

% Define Movie
aviobj = avifile('RadialTemperatureMovie.avi')

counter=0
for i=1:length(TIME)-1
    Movie_Array = getframe(figure1);
    aviobj = addframe(aviobj,Movie_Array);
    radial_temps=[S8B(i+1) S8A(i+1) S7B(i+1) S7A(i+1)];
    set(AX,'XData',radial_positions,'YData',radial_temps);
    drawnow;

    counter=counter+1;
    if counter>=1000
        counter=0;
        check=i
    end
end

close(figure1)
aviobj = close(aviobj);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%------------------------------ END ----------------------------------%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```