DOE/PC/90512--2942

DE91 002041

Combustion Front Propagation in Underground Coal Gasification

Final Report

**Richard L. Dobbs II
William B. Krantz**

Work Performed Under Grant No.: DE-FG22-86PC90512

**For
U.S. Department of Energy
Office of Fossil Energy
Pittsburgh Energy Technology Center
P.O. Box 10940
Pittsburgh, Pennsylvania 15236**

**By
University of Colorado, Boulder
Department of Chemical Engineering
Boulder, Colorado 80309-0424**

October 1990

PREFACE

This is the final technical report for DOE grant DE-FG22-86PC90512. This grant continued the research on reverse combustion linking at the University of Colorado. One Ph.D. Thesis was completed under the sponsorship of this grant.

Thesis completed:

1. Richard L. Dobbs II, "Reverse Combustion Channel Propagation for Underground Coal Gasification," University of Colorado, Boulder, CO, 1990.

In addition to the work by the students cited above, an undergraduate student, Michael S. Jackson, participated in this research effort through the Undergraduate Research Opportunities Program.

Technical papers completed:

1. Dobbs II, R.L., and W.B. Krantz, "Optimizing Reverse Combustion with Promoters and Inhibitors", Proc. 13th Annual Underground Coal Gasification Symposium, U.S. DOE Report DOE/METC-88/6095 (DE88010255), pp. 202-210, (1987).
2. Dobbs II, R.L., and W.B. Krantz, "A Transient Model of Reverse Combustion Channel Propagation", Proc. 14th Annual Underground Coal Gasification Symposium, U.S. DOE Report DOE/METC-88/6097 (DE8801093), pp. 102-106, (1988).

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the following individuals and organizations: Financial support for this project was provided by the United States Department of Energy through DOE Grant No. DE-FG22-86PC90512. We thank Joseph W. Martin of DOE-METC for his interest in and encouragement of this work. We are quite grateful to the Washington Irrigation and Development Company (WIDCO), which generously provided access to the Centralia mine to obtain coal samples. Special thanks are extended to Roger Paul for arranging this access and Will Greenough for acting as our on-site liason. We thank Dennis McBreen, Rod Gilseth, Mark Hallsey, and Gordy Clapp of Soil Sampling Services, Inc. for their diligence and competence in executing the coal coring. Earl Jones and Kevin Leedy of the University of Colorado Facilities Management deserve special thanks for fabricating the reverse combustion pressure vessel. We thank Mike Jackson whose Undergraduate Research Opportunity Project and other work helped get the high pressure combustion laboratory in shape. Willy Grothe and Norm Taylor provided invaluable assistance in building the apparatus. We are very thankful to Kathy Dobbs who fabricated thermocouples, scrapped silicon sealant, and helped with hundreds of tasks associated with the experiments. Doug Wray provided able assistance in photography.

CONTENTS

CHAPTER

I. INTRODUCTION	1
1.1 Underground Coal Gasification	1
1.2 Linking	4
1.2.1 Need for Linking	4
1.2.2 Linking Techniques	4
1.3 Reverse Combustion	6
1.3.1 Nature of RC in Coal	6
1.3.2 Comparison with Diffusion-Limited Growth Mechanisms	8
1.3.3 Problems with RC	9
1.4 Objectives of this Study	9
Laboratory Studies:	9
Modeling Studies:	10
Application to UCG:	10
1.5 Scope	11
II. LITERATURE REVIEW	12
2.1 Scope	12
2.2 Reverse Combustion Front Propagation Modeling	12
2.3 Linear Stability Analyses	13
2.4 Two-dimensional Channel Evolution Modeling	15
2.5 Reverse Combustion Field Work	18
2.6 Experimental Studies	19
2.7 Critique of the Literature	21
III. OPTIMIZING REVERSE COMBUSTION LINKING WITH PRO- MOTERS AND INHIBITORS	23
3.1 Scope	23
3.2 Control of Reverse Combustion	23
3.3 Generalized Solution to RC Model	25
3.4 Discussion of Results	30
3.5 Summary	34

IV. A ONE-DIMENSIONAL MODEL FOR THE PROPAGATION OF A CURVED REVERSE COMBUSTION FRONT	35
4.1 Scope	35
4.2 Motivation	35
4.3 Nature of the Flow Field to a Reverse Combustion Finger	36
4.4 Formulation	39
4.5 Solution	47
4.5.1 Preliminaries	48
4.5.2 Variable and Parameter Expansions	48
4.5.3 Outer Zone Solutions	49
4.5.4 Inner Zone Solution	50
4.6 Nature of the Solution	53
4.6.1 Comments and Precautions	56
4.7 Application to Experimental Data Analysis	64
4.8 Application to Channel Growth Simulations	65
4.9 Summary	65
V. REVERSE COMBUSTION EXPERIMENTS IN SOLID COAL	66
5.1 Scope	66
5.2 Objectives of the Experimental Program	66
5.3 Design Considerations	67
5.3.1 Controlled Parameters	67
5.3.2 Observation of Channeling	67
5.3.3 Coal Samples	68
5.3.4 Safety	68
5.3.5 Reliability	69
5.4 Design Overview and Discussion	69
5.4.1 Overview of Apparatus and Experiments	69
5.4.2 Discussion of Key Design Features	70
5.5 Description of Apparatus	72
5.6 Operating Procedures	79
5.6.1 Sample Acquisition	79
5.6.2 Sample Preparation Procedure	80
5.6.3 Experiment Procedure	82
5.6.4 Post-burn Evaluation	84
5.7 Summary	85

VI. EXPERIMENT DESIGN AND RESULTS	86
6.1 Scope	86
6.2 Experiment Design	86
6.3 Observations	88
6.3.1 Phase One - Technique	88
6.3.2 Phase Two - First Block of Experiments	90
6.3.3 Phase Three - Second Block of Experiments.	90
6.3.4 Summary of Observations	92
6.4 Results	93
6.5 Comparison with Theory	99
6.6 Critique of Experiment Design	100
6.7 Discussion	102
6.8 Summary	103
VII. FORMULATION OF A TRANSIENT, NUMERICAL SIMULATION FOR REVERSE COMBUSTION CHANNEL EVOLUTION	104
7.1 Scope	104
7.2 Motivation	104
7.3 Strategy	104
7.4 Assumptions	106
7.5 Implementation	107
7.5.1 Reverse Combustion Model.	107
7.5.2 Simulation Procedure	109
7.5.3 Flow Calculations	109
7.5.4 Front Propagation	115
7.5.5 Remeshing	116
7.5.6 Input and Output	117
7.6 Appraisal of the Simulation	117
7.7 Summary	118
VIII. A CASE STUDY IN REVERSE COMBUSTION CHANNEL EVO- LUTION USING BOUNDARY ELEMENT FLOW CALCULATIONS	119
8.1 Scope	119
8.2 Model Formulation	119
8.3 Cases	123
8.3.1 Rectangular Slab Flow Configuration	124
8.3.2 Circular Flow Configuration	140

8.3.3 Two-well Flow Configuration	140
8.4 Discussion	150
8.5 Summary	155
IX. CONCLUSIONS AND RECOMMENDATIONS	156
9.1 Conclusions	156
9.1.1 Laboratory Studies	156
9.1.2 Modeling	157
9.2 Recommendations	158
9.2.1 Laboratory Studies	158
9.2.2 Modeling	159
9.2.3 Implications for Underground Coal Gasification	159
REFERENCES	161
NOMENCLATURE	167
A. COMPUTER PROGRAMS FROM CHAPTER 4	174
APPENDIX	
B. DATA ACQUISITION AND DISPLAY PROGRAMS	185
C. DATA ANALYSIS PROGRAMS	266
D. COMPUTER PROGRAM FROM CHAPTER 7	350
E. COMPUTER PROGRAM FROM CHAPTER 8	384

TABLES

Table

5.1 Ultimate analysis of coal from Upper Thompson Seam, Centralia, Washington, near site of coring	79
5.2 Ultimate analysis of coal from Upper Thompson Seam, Centralia, Washington, near site of coring	80
6.1 Summary of reverse combustion experimental data	94
6.2 Product gas composition at end of experiment run	97

FIGURES

Figure

1.1	Underground coal gasification in a horizontal coal seam.	2
1.2	Underground Coal Gasification in a steeply dipping coal seam.	2
1.3	The zones around a reverse combustion front.	7
3.1	Front velocity as a function of activation energy for constant values of pre-exponential factor.	31
3.2	Front velocity as a function of pre-exponential factor for constant values of activation energy.	32
3.3	Trial function parameter, β as function of the independent parameters describing reverse combustion.	33
4.1	Permeability zones for a reverse combustion finger.	37
4.2	Flow field around a reverse combustion finger.	38
4.3	Local frame of reference and coordinate system at a point on a reverse combustion channel	40
4.4	Local approximation for the gas flow field along a line passing through the combustion front.	41
4.5	Schematic of temperature and limiting reactant profiles.	45
4.6	Dimensionless Arrhenius Plot for Reverse Combustion Front Propagation . . .	54
4.7	Dimensionless plot of front temperature versus gas flux for a dimensionless activation energy of 10 and various heats of reaction.	55
4.8	Plot of a dimensionless group that is proportional to the dimensional velocity versus dimensionless gas flux for a dimensionless activation energy of 10 and various dimensionless heats of reaction.	57
4.9	Plot of dimensionless front temperature versus dimensionless gas flux for a dimensionless activation energy of 20 and various dimensionless heats of reaction.	58
4.10	Plot of a dimensionless group that is proportional to the dimensional velocity versus dimensionless gas flux for a dimensionless activation energy of 20 and various dimensionless heats of reaction.	59

4.11	Plot of dimensionless front temperature versus dimensionless gas flux for a dimensionless activation energy of 20 and various dimensionless versus dimensionless gas flux for a dimensionless activation energy of 10, a dimensionless heat of combustion of 5, and various values of dimensionless tangential heat transport.	60
4.12	Plot of a dimensionless group that is proportional to the dimensional velocity versus dimensionless gas flux for a dimensionless activation energy of 10, a dimensionless heat of combustion of 5 and various values of dimensionless tangential heat transport.	61
4.13	Front temperature versus gas flux for various levels of tangential conduction.	62
4.14	Front propagation velocity versus gas flux for various levels of tangential conduction.	63
5.1	Gas supply system and reverse combustion pressure vessel.	73
5.2	Gas clean-up system.	74
5.3	Reverse combustion sample container.	75
6.1	Experiment design.	89
6.2	Photograph of a cross-section of coal core for Experiment 5 near the channel tip.	91
6.3	Response of propagation velocity.	98
6.4	Arrhenius plot based on front speed.	101
7.1	Block flow diagram for SMOOTH, a reverse combustion channel evolution simulation.	108
8.1	Combustion front propagation relationship compared to the data of Visser <i>et al</i> (1985).	122
8.2	The rectangular slab configuration	125
8.3	A graphical display of the parameter values for the cases presented.	126
8.4	Record of channel evolution for Case 1 (rectangular slab configuration, superficial gas flux is 25 kg/hr/m ² , $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m ² , and $\mathcal{H}$ is 0.005).	127
8.5	Record of channel evolution for Case 2 (rectangular slab configuration, superficial gas flux is 50 kg/hr/m ² , $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m ² , and $\mathcal{H}$ is 0.005).	128
8.6	Record of channel evolution for Case 3 (rectangular slab configuration, superficial gas flux is 100 kg/hr/m ² , $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m ² , and $\mathcal{H}$ is 0.005).	129

8.7	Record of channel evolution for Case 4 (rectangular slab configuration, superficial gas flux is 200 kg/hr/m ² , $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m ² , and $\mathcal{H}$ is 0.005).	130
8.8	Record of channel evolution for Case 5 (rectangular slab configuration, $\mathcal{H}$ is 0.0, superficial gas flux is 50 kg/hr/m ² , and $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m ²).	132
8.9	Record of channel evolution for Case 6 (rectangular slab configuration, $\mathcal{H}$ is 0.005, superficial gas flux is 50 kg/hr/m ² , and $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m ²).	133
8.10	Record of channel evolution for Case 7 (rectangular slab configuration, $\mathcal{H}$ is 0.010, superficial gas flux is 50 kg/hr/m ² , and $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m ²).	134
8.11	Record of channel evolution for Case 8 (rectangular slab configuration, $\mathcal{H}$ is 0.015, superficial gas flux is 50 kg/hr/m ² , and $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m ²).	135
8.12	Record of channel evolution for Case 9 (rectangular slab configuration, $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m ² , the superficial gas flux is 50 kg/hr/m ² , and $\mathcal{H}$ is 0.005).	136
8.13	Record of channel evolution for Case 10 (rectangular slab configuration, $\mathcal{N}_{min}$ is equivalent to 20 kg/hr/m ² , the superficial gas flux is 50 kg/hr/m ² , and $\mathcal{H}$ is 0.005).	137
8.14	Record of channel evolution for Case 11 (rectangular slab configuration, $\mathcal{N}_{min}$ is equivalent to 30 kg/hr/m ² , the superficial gas flux is 50 kg/hr/m ² , and $\mathcal{H}$ is 0.005).	138
8.15	Record of channel evolution for Case 12 (rectangular slab configuration, $\mathcal{N}_{min}$ is equivalent to 40 kg/hr/m ² , the superficial gas flux is 50 kg/hr/m ² , and $\mathcal{H}$ is 0.005).	139
8.16	Record of channel evolution for Case 13 (circular configuration, the superficial gas flux to the initial channel shape is 50 kg/hr/m ² , $\mathcal{N}_{min}$ is equivalent to 30 kg/hr/m ² , and $\mathcal{H}$ is 0.0075).	141
8.17	Record of channel evolution for Case 14 (circular configuration, the superficial gas flux to the initial channel shape is 100 kg/hr/m ² , $\mathcal{N}_{min}$ is equivalent to 30 kg/hr/m ² , and $\mathcal{H}$ is 0.0075).	142
8.18	Record of channel evolution for Case 15 (circular configuration, the superficial gas flux to the initial channel shape is 150 kg/hr/m ² , $\mathcal{N}_{min}$ is equivalent to 30 kg/hr/m ² , and $\mathcal{H}$ is 0.0075).	143

8.19	Record of channel evolution for Case 16 (circular configuration, the superficial gas flux to the initial channel shape is 200 kg/hr/m^2 , $\mathcal{N}_{min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).	144
8.20	The two-well flow configuration.	145
8.21	Record of channel evolution for Case 17 (two-well configuration, the superficial gas flux to the initial channel shape is 50 kg/hr/m^2 , $\mathcal{N}_{min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).	146
8.22	Record of channel evolution for Case 18 (two-well configuration, the superficial gas flux to the initial channel shape is 100 kg/hr/m^2 , $\mathcal{N}_{min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).	147
8.23	Record of channel evolution for Case 19 (two-well configuration, the superficial gas flux to the initial channel shape is 150 kg/hr/m^2 , $\mathcal{N}_{min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).	148
8.24	Record of channel evolution for Case 20 (two-well configuration, the superficial gas flux to the initial channel shape is 200 kg/hr/m^2 , $\mathcal{N}_{min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).	149
8.25	Representative channel shape from Visser <i>et al.</i> (1985) for a rectangular slab configuration.	152
8.26	Representative channel shape from Visser <i>et al.</i> (1985) for a circular flow configuration.	153

SUMMARY

Reverse Combustion (RC) enhances coal seam permeability prior to Underground Coal Gasification. Understanding RC is necessary to improve its reliability and economics.

A curved RC front propagation model is developed, then solved by high activation energy asymptotics. It explicitly incorporates extinction (stoichiometric and thermal) and tangential heat transport (THT) (convection and conduction). THT arises from variation in combustion front temperature caused by tangential variation in the oxidant gas flux to the channel surface. Front temperature depends only weakly on THT; front velocity is strongly affected, with heat loss slowing propagation. The front propagation speed displays a maximum with respect to gas flux. Combustion promoters speed front propagation; inhibitors slow front propagation.

The propagation model is incorporated into 2-D simulations of RC channel evolution utilizing the boundary element method with cubic hermetian elements to solve the flow from gas injection wells through the coal to the convoluted, temporally evolving, channel surface, and through the channel to a gas production well. The simulated evolution of the combustion front closely resembles the fingers observed by Visser *et al.*, (1985). THT, combustion inhibitors, and increased gas flow are found to smooth the combustion front. Extinction causes combustion fingers to separate.

RC channel propagation is studied using 17 cm diameter subbituminous horizontally drilled coal cores. Sixteen experiments at pressures between 2000 and 3600 kPa, injected gas oxygen contents between 21% and 75%, and flows between 1 and 4 standard liters per minute are described. Similarity analysis led to scaling-down of large RC (≈ 1 m) to laboratory scale (≈ 5 cm). The utility of this scale-down is limited by heterogeneities with respect to permeability, volatile content, and reactivity, both within and between samples. Propagation velocity shows a strong synergistic increase at high levels of oxygen, pressure, and gas flow. Char combustion is observed, leaving ash-filled, irregularly shaped channels. Cracks are observed to penetrate the char zone surrounding the channel cores. Under these conditions, RC is not dominated by permeation processes; thus char-burning, diffusion-flame models of RC are needed.

CHAPTER I

INTRODUCTION

1.1 Underground Coal Gasification

Underground Coal Gasification (UCG) is an in-situ technique to recover the fuel or feedstock value of coal that is not economically available through conventional recovery technologies. Although coal is the most plentiful fossil fuel, only a small fraction of the potential coal reserves can be economically accessed through shaft or strip mining. The remaining coal has no value – unless alternative recovery technologies, such as UCG, are developed.

The basic process of UCG is shown in Figure 1.1. Consider a roughly horizontal coal seam sandwiched between underlying and overlying layers of rock. Two vertical wells are completed nearly to the bottom of the coal seam. A highly gas-permeable channel (or “link”) is created between these two wells, through a process which will be described in Section 1.2. A mixture of steam, air, and oxygen is forced into the gas-injection well. The partial combustion of coal in the gasifier cavity around the injection well creates heat that drives the endothermic gasification reactions between coal, oxygen, and water. The product gas is a mixture principally of carbon dioxide, carbon monoxide, hydrogen, methane, steam, and nitrogen, if air is used. It also contains in smaller amounts ethane, propane, and higher hydrocarbons, along with coal tars. This gas flows through a highly gas-permeable channel to the production well. At the surface, the gas is transported through pipelines to local users. For long distance transport, the gas must be upgraded to synthetic natural gas.

Over a period of weeks, the gasifier cavity grows outward and upward into the coal seam. When the cavity reaches the top of the coal seam, it continues to grow upward into the overlying rock through the process of rock spalling. After this point, the heating value of the gas, which is indicative of its value as either fuel or feedstock, declines with time. Heat is absorbed by rock and water brought into the gasifier cavity through the process of spalling-enhanced drying (Tantekin, 1986). This heat is no longer available to the endothermic gasification reactions. At some point, further operation of a gasifier cavity is not economical, and it must be shut down and a new cavity brought on stream. For a horizontal coal seam, the optimum cavity diameter is on the order of 20 meters (Cena, *et al.*, 1988).

An important alternative arrangement occurs for gasification of a steeply dipping bed, as shown in Figure 1.2. Although the underlying phenomena are the same, the orientation of the coal at a large angle to the horizontal affects the subsequent growth of the gasifier cavity. The lateral growth will be confined by rock along one axis, but the active chamber will translate upward along the coal seam, as rock, ash, and unconsumed char raise the floor

GASIFICATION OF A HORIZONTAL COAL SEAM

Figure 1.1: Underground coal gasification in a horizontal coal seam.

GASIFICATION OF A STEEPLY DIPPING COAL SEAM

Figure 1.2: Underground Coal Gasification in a steeply dipping coal seam.

of the chamber. The amount of heat lost to rock and water influx is more nearly constant with time; consequently more coal can be consumed in a single cavity. The cavities are approximately 20 meters wide and extend approximately 80 meters upward (Avasthi and Lee, 1984)

UCG is both a coal recovery and a coal conversion technology. For production of chemical feedstocks, it is an alternative to conventional mining (strip or shaft), followed by above-ground coal gasification. For production of fuel, it is an alternative to conventional mining, followed by direct utilization of coal or above-ground coal gasification. An extensive discussion of the advantages and disadvantages of UCG in comparison with these alternative technologies is outside of the scope of this work. The interested reader is referred to the *Proceedings of the Second through Fourteenth Annual Underground Coal Gasification Symposia*. Instead, the proposed advantages and disadvantages simply will be listed.

The principal advantage claimed for UCG over mining is that it will be able to use coal that is not economically available through shaft or strip mining because the coal is too deep, is of too low rank, contains too much ash, or a variety of other reasons. In addition, UCG would be less environmentally disruptive than strip mining, and would be safer and less labor intensive than shaft mining.

As a coal conversion technology, an advantage of UCG is that the magnitude of the solid-waste-disposal problem is greatly diminished, since most of the mineral content of the coal remains underground in the form of ash. For thick coal seams, UCG has a higher thermodynamic efficiency than above-ground coal gasification since heat initially lost to the surrounding coal is later regained as the gasifier cavity grows outward. Consequently, only heat absorbed by rock, by excessive water influx, and by coal which is not subsequently gasified, is ultimately lost to the process. By using the earth as a natural pressure vessel, capital costs for UCG are decreased. The virtual elimination of pressure vessels also decreases the lead time required to bring a large scale plant into operation. The combination of these two effects would make UCG more responsive to changing economic conditions than large-scale above-ground coal gasification plants. In common with above-ground gasification, UCG has an advantage over direct coal combustion in that the sulfur content of the coal is produced as H_2S rather than SO_2 , and thus can be recovered and converted into sulfur, rather than requiring scrubbing which creates a solid-waste-disposal problem.

The principal disadvantages of UCG are connected with it being an in-situ process. Geological features, such as fractures or partings, that are easily handled in mining can cause UCG to fail completely. There is relatively little direct control of the process. The materials of construction and design of the gasifier pressure vessel are mostly in the hands of nature. UCG is inherently a semi-batch, transient process, rather than being a continuous process. Post-burn ground subsidence must be considered, and may prevent utilization of UCG for low quality but shallow coal seams not suitable for mining. Ground water contamination from the cavity is a matter of important concern since most coal seams are aquifers. Other aquifers may be affected through rock spalling. Resource utilization is uncertain, but is lower than strip mining of the same seam. When UCG is used to produce fuel, which is subsequently burned, carbon dioxide is released into the atmosphere. This may have long term negative consequences through the Greenhouse Effect. As a final point, the economic

success of UCG is dependent on the prior creation of a highly permeable channel to allow the products of combustion to flow to the production well. Creation of this channel is referred to as "linking." UCG can be no more reliable or economic than linking. Linking is considered in more detail in Section 1.2.

1.2 Linking

1.2.1 Need for Linking

In UCG, the resource recovery rate (mass of coal gasified per unit time) for each gasifier cavity must be high. At low rates, excessive heat loss to the surroundings leads to poor thermal efficiency as manifested by poor product gas quality. Excessive heat loss to the surroundings also leads to poor coal resource utilization since the reaction temperature might not be sufficient to promote char combustion and gasification. The required injection gas flow rate and resultant product gas flow rate for efficient operation are large compared to the capacity of a coal seam to permeate gas at a reasonable pressure differential between the injection and production wells. This differential is limited both by technical and monetary constraints. The maximum injection pressure is limited by the compression costs, concern about fracturing the overburden, and environmental concerns about escape of combustion gases into surrounding formations. The minimum production back pressure is limited by the intrinsic value of producing gases at higher pressure, and also by the necessity to limit water influx into the cavity. Consequently, the resistance of the coal seam to flow between the injection and production wells must be decreased prior to UCG. Thus, a permeability enhancing step, i.e., linking, must precede UCG.

For optimal resource recovery (mass of coal gasified) per gasifier cavity, it is desirable that the link be created at the bottom of the coal seam to be gasified. Both modeling (Britten and Thorsness, 1987) and field excavation (Oliver, 1987; Cena, *et al.*, 1987) indicate that cavity growth downward from a link is minimal. The floor of the cavity is isolated from the hot, reactive environment of the rest of the gasifier by a pile of partially fused rock, ash and char. Upward growth is limited by the thickness of the coal seam. Lateral growth is eventually economically limited by exposure of increasing roof area, with its associated heat loss owing to the processes of spalling-enhanced drying. Thus, the resource recovery possible per well drilled, and hence the commercial success of UCG, are largely determined by the placement of the initial link.

1.2.2 Linking Techniques

A number of different linking techniques exist. Let us consider the comparative advantages and disadvantages of various linking methods.

Soviet experiences with hydraulic fracturing have found it to be effective only when the coal seam was already broken up by previous shaft mining or naturally occurring fractures. The first case is not of interest since UCG is proposed as an alternative to shaft mining. The

second case is not promising, since the fracture structure will tend to disrupt subsequent gasification because of excessive rock spalling and excessive loss of gas from the gasifier cavity.

Explosive fracturing has not met with success. Although it was hoped that it would create a fracture structure through the coal, thereby increasing its permeability, this fracture structure does not penetrate sufficiently far into the coal seam to link wells at any distance. If more powerful explosives are used to attempt greater penetration, the overlying rock would be disrupted, to the detriment of subsequent gasification. Furthermore, the cracks that form tend to close up again, since no material has been removed, and coal is structurally weak.

A method that has met with some success is electrolinking (EL), which has been studied recently by Dindi (1988). In this process, electrodes are placed into the bottom of the wells to be linked and a high electric potential is imposed on the intervening coal. Through a process of dielectric breakdown, small highly electrically conductive channels are created which, with luck and sufficient voltage, will link the two wells. These small links are then enlarged through electrocarbonization, which is a low voltage, high current process. An advantage of EL is that it is a homing process. A channel growing outward from one electrode tends to be attracted toward the other electrode. The disadvantage of this process is that channels tend to bifurcate excessively until so much current is passed that linking costs become prohibitive. If the voltage is allowed to decrease too much, the links will cease to propagate.

An obvious method, which is fairly successful, is to drill a well through the coal seam to directly connect vertical production and injection wells in the extended linked-well configuration (See Figure 1, p. 201 of Cena, *et al.*; 1988). Since an open hole is much more permeable than coal, the permeability will be markedly enhanced. With current drilling technologies, it is possible to pass within a few meters of the vertically drilled wells. This method must be coupled with a link-completion technique. This method suffers two disadvantages: First, the initial link is a single, small channel which is potentially susceptible to blockage in the early stages of cavity growth; second, directional drilling is expensive. Consequently, one horizontal well must be used by many gasifier cavities for these methods to be economical. A promising technique developed specifically for UCG is the Cornering-Water-Jet-Drill concept. This method is nearly right-angle drilling (Shirey and Engler, 1984).

A more sophisticated version of directional drilling is the Controlled Retraction Injection Point (CRIP) method. As implemented in the recent RM-I field test, this method uses a horizontal injection well equipped with a stainless steel liner and a retractable ignitor, an unlined horizontal production well, and vertical production well that is used in the link completion (Cena, *et al.*, 1988). After exhaustion of the preliminary gasifier cavity, a new injection point is started by burning through stainless steel liner with the retractable ignitor. Since this injection point is surrounded by virgin coal, this starts a new gasifier cavity which is operated until exhaustion. This process is repeated along the whole length of lined injection well. This process operated quite successfully in the RM-I field tests. Although the technical feasibility of this method has been established, it remains uncertain whether it is economically worthwhile. In addition to the drilling costs are the added liner costs. Although the liner is thin, stainless steel is far more costly than coal. As a method based

on directional drilling, it requires a link-completion method.

The method used most commonly as a final linking method is Reverse Combustion (RC), which is the focus of this work. It can also be used as a primary linking method. In RC, the small natural permeability of the coal seam is used to support a special mode of combustion which creates a char-filled, gas-permeable channel. This is an old method, having been regularly practiced on "commercial scale" in the Soviet Union beginning in the 1930's. It has the advantages of being a low cost (Gregg, 1980), simple method that has a homing tendency. Its disadvantage is that it is unreliable. Being a combustion process, buoyancy effects tend to gravitate it upwards. This is undesirable for the reasons discussed in the beginning of this section. If an open fracture exists, the channel will proceed to propagate up the fracture and then propagate along the top of the coal seam. It also has other problems which will be discussed in detail in Section 1.3.

Reverse Combustion is not a universally applicable linking technique. None exist. Instead, it has a limited but generally recognized role in UCG. It is the method of choice for use in gasification of steeply dipping beds. Here, the effect of a reverse combustion link deviating upwards even 5 meters has a minimal effect on commercial success. In horizontal coal seams, it is recognized as the method to achieve final linking of directionally drilled wells. These short distance links are often subsequently blasted clear of char with high flow rates of gas. If the coal seam has favorable shale partings which confine the RC link near the bottom of seam, RC could be used as a primary linking method. These partings would fail during the gasification phase, allowing recovery of the coal in the upper part of the seam. Through further development, the role of RC can be expanded to replace more expensive techniques, increasing the commercial viability of UCG.

1.3 Reverse Combustion

1.3.1 Nature of RC in Coal

In RC the combustion front propagates through a solid, countercurrently to the gas flow. The fuel for combustion is generally released from the solid, which undergoes a change in porosity. Figure 1.3 illustrates the zones around a RC front. A key point needed in understanding RC is the recognition that the combustion reaction is limited to a very thin zone. In the vast bulk of the coal, there is little chemical reaction taking place because the temperature is not significantly above the ambient temperature of the coal seam. The convection of heat due to cool injected gas flowing toward the combustion front prevents the thermal front from penetrating deep into the coal owing to solid-phase conduction of heat. As the front is approached, the coal undergoes devolatilization because of the heat conducting from the combustion front. Considering the coal to be a uniform homogeneous medium leads to the following description of reverse combustion: although the volatiles mix with the oxidant, no combustion takes place in this preheat zone; the reaction kinetics at these temperatures are too slow for significant reaction; at higher temperatures the preheat zone merges into the combustion zone; in this thin zone, the temperature is sufficient for

Figure 1.3: The zones around a reverse combustion front.

the combustion reactions to take place; without a large temperature increase, the reactions quickly progress until limited by exhaustion of the limiting reactant; thus the combustion zone merges into the char channel, where again no reactions take place, now because of lack of the limiting reactant. Certainly this simple continuum picture of reverse combustion only approximately applies to RC in coal, given the existence of cracks on the same scale as the thickness of the combustion zone.

A greater amount of heat will be released in the combustion reaction if the oxidant gas flux is larger. This in turn will cause the combustion front to propagate faster, since a portion of the heat released goes into heating and devolatilizing the coal in the preheat zone. Thus, the RC channel will grow toward any source of increased flow and hence toward any path of enhanced permeability. The devolatilization of the coal in the preheat zone increases its permeability so long as the coal does not swell excessively under heat treatment. The char channel thus serves as a path of least resistance for the gas flow. Consequently, the gas flow will further concentrate at the evolving tip of the channel. Thus, a hypothetical broad RC combustion front will evolve into one or more narrow, possibly branching, channels.

This process does not continue without limit. A point is reached where the additional heat from greater oxidant flow at the tip is completely absorbed by stabilizing mechanisms of heat transfer, and thus is unavailable for propagating the front. Thus, the front will not evolve into a cusp with a point of infinite curvature. The chief stabilizing mechanisms of heat transfer are downstream convection of heat and lateral conduction and convection of heat.

Consider now the interplay between devolatilization and combustion. At the tip of the channel, the combustion temperature is sufficient so that complete devolatilization ahead of the combustion front can be assumed. This in turn usually implies that at the tip, oxygen is the limiting reactant. Moving away from the tip, the combustion temperature decreases. At a trailing edge of the combustion front, the temperature is insufficient for complete devolatilization. Independent of any additional extinction mechanism, the minimum temperature required to begin devolatilization sets a minimum combustion front temperature at which the channel can propagate, which provides a lower limit for the oxidant gas flux for a propagating combustion front. The interaction between these phenomena and the gas flow to the channel surface determines the evolution of the shape of a RC channel.

In considering long term channel evolution, the simplest realistic configuration is flow from a single injection well to the channel surface. Important features of the channel evolution problem are not contained in geometrically simpler configurations such as an axially symmetric or slab configuration. Since the channel surface is highly convoluted and evolving in time, the determination of the flow to the channel surface is particularly challenging. It is not amenable to simple analytical solution, despite the deceptive simplicity of the underlying governing equations, and it is challenging for numerical methods.

1.3.2 Comparison with Diffusion-Limited Growth Mechanisms

If the flow through the coal is not dominated by channel flow in existing fractures, the equations describing the gas flow through the coal seam and char can be simplified through reasonable engineering assumptions to Laplace's equation for a zoned media. The gas flow to the combustion front determines the growth of the channel. Consequently, RC is partially analogous to other growth processes where the bulk is governed by Laplace's equation or more generally, the transient diffusion equation, and the flux to the surface drives the growth. If the surface acts as an attractor for the flux and the growth increases with greater flux, these processes are unstable and lead to complex growth structure. Since this partial analogy has been used in studying RC channel evolution, let us consider the similarities and differences between RC and several such processes.

In crystallization from a stationary solution, growth is conservative, that is, there is a proportionality between the total flux to the surface and the total growth. An additional destabilizing effect is that growth has an orientation effect because of crystal-packing considerations. Stabilization occurs because rearrangement of the crystal to lower energy configurations.

In displacement of a higher viscosity fluid by an immiscible lower viscosity fluid, growth is also conservative. In a Hele-Shaw cell, growth is stabilized by surface tension in the plane of the cell. For porous media, this long length scale stabilization effect is disrupted by the presence of the porous solid.

In RC, growth is nonconservative. Even for an active front, the propagation velocity is not linear with, let alone proportional to the gas flux. The possibility that part of the front is extinguished causes further deviation from conservative growth, and provides an additional destabilizing influence. Growth is not strictly a local effect. Only through modeling sim-

plifications is the channel growth separated from the complete temperature distribution in the channel core. The front is stabilized by heat transport. Tangential heat transport arises because of temperature variation on the channel surface. This in turn arises because of variation in the tangential and normal components of the flux to the surface. Thus, growth is a function of the tangential and normal components of the gas flux and also higher derivatives of the normal component of the gas flux.

1.3.3 Problems with RC

Reverse combustion is a difficult process to control. As a combustion process, RC tends to deviate upwards in a coal seam because of buoyancy effects. Geological fractures could cause immediate override of the coal seam. Alternatively, they could present an impenetrable barrier to channel progress. Swelling or plastic deformation of coal can limit the permeability enhancement arising from RC. If spontaneous ignition of the coal occurs, the subsequent release and condensation of coal tars could plug the coal seam, which would prevent successful linking.

Few techniques exist to cope with and mitigate these problems. There is no direct, independent control over the number, size, or permeability of the reverse combustion channels, or their paths. Consequently, there is a need to understand how process parameters indirectly affect channel characteristics. As an in-situ process, there are a limited number of control parameters; moreover the range of conventional control parameters is limited by technical and monetary constraints. Thus, there is a need to identify unconventional methods to improve RC linking. Field tests yield only limited information about RC linking at great expense. Consequently, better understanding of RC must come from laboratory studies and mathematical modeling.

1.4 Objectives of this Study

The goal of this study is to gain greater understanding of the evolution of reverse combustion channels, and from it to extract knowledge useful for the practitioner of RC linking for underground coal gasification. In pursuit of this goal, the following objectives are established:

Laboratory Studies:

- Construct an apparatus for studying combustion of porous solids at high pressure.
- Propagate RC in solid, subbituminous coal samples.
- Test scale-down laws for RC channel propagation by altering process parameters based on existing models.
- Study the response of channel characteristics to changes in process parameters.

- Explore using combustion tube data to extract global kinetic parameters for modeling RC in solid coal.
- Use flux determinations for assessing channel dynamics.

Modeling Studies:

- Develop an improved front-propagation model.
- Explore the use of the boundary-element method for flow calculations to study RC channel propagation.

Application to UCG:

- Identify novel methods that could lead to more reliable RC linking.
- Reassess existing theories and models for RC channel propagation in light of this study.
- Chart the direction for further research in RC.

Few experimental studies of RC in solid coal have been undertaken. Under moderate operating conditions, RC in coal is thought to propagate with char-filled channels too large to be observed in a laboratory-sized apparatus. Existing theory suggests that RC channels can be scaled-down to allow study in a laboratory apparatus by operating at high pressure and oxygen content. Small channels have been observed in previous experiments, but these channels were different in character from those envisioned by existing theory for operation at low pressure. This difference in character might result from operation with very low volatile matter coal or at extreme overburden pressure, and hence low coal permeability, or it might be an indication of deficiencies of existing theory. The feasibility and limitations of scale-down of RC channels by altering process parameters are worth further exploration.

In analyzing data from previous channeling RC experiments, no systematic attempts have been made to calculate the gas flux distribution to the channel surface. Since existing models for front propagation are based on gas flux, rather than gas flow, these analyses have been limited by the methods used to estimate the gas flux at the channel tip.

Existing models of reverse combustion channel propagation consist of a combination of a hydrodynamic analysis of the flow to a RC channel and a free-boundary condition. The free-boundary condition describes the propagation of the combustion front. Previous formulations of the propagation relationship display a basic inconsistency with the physics of RC in that they ignore lateral heat transport. Incorporation of this effect should lead to sounder basis for channel-evolution models.

Various methods have been used in RC channel-propagation models to calculate flow to the channel surface. The utility of the boundary-element method for this purpose is worth

investigation. The boundary-element method is a flexible, numerical technique that appears well suited to this task.

Given the constraints placed upon conventional control parameters for RC, it is beneficial to identify novel methods to improve the reliability and control of RC linking.

Existing RC theories and models are based on limited field test and experimental observations. It is important to reassess these theories in light of new observations. This assessment should consider both the basis for these models and also the utility of these theories for the practitioner of RC linking for UCG. The natural outcome of such a reassessment is an evaluation of the most beneficial direction for further research in RC.

1.5 Scope

This report is organized in the following manner. In Chapter 2, appropriate literature from previous work on one- and two-dimensional models for RC in coal, RC linking field tests, and RC laboratory experiments in coal are reviewed. In Chapter 3, a novel method to control RC linking is presented and evaluated in respect to existing theories for RC. As a side effect of this evaluation, the range of validity of the high-activation-energy asymptotic solution for planar RC of Britten (1984) is explored numerically. In Chapter 4, a model for the propagation of a curved RC front through homogeneous porous media is formulated. It considers lateral heat transport and some aspects of extinction. This model is solved through the use of high-activation-energy asymptotics. This solution is used both in the experimental data analysis and the subsequent two-dimensional channel propagation model. In Chapter 5, the design of the experimental apparatus is described, as well as operating and data evaluation procedures. In Chapter 6, the experiment design is presented along with results of the experiments and their significance. In Chapter 7, a transient, numerical 2-d model using the boundary-element method is developed. Key aspects of the numerical implementation are discussed. In Chapter 8, a case study of channel evolution using this model is presented. In Chapter 9, the conclusions and recommendations arising from this work are summarized. In Appendix A, the computer code arising from Chapter 4 is listed. In Appendix B, the software developed for the experimental data acquisition is listed. In Appendix C, the software developed for the data analysis is listed. In Appendix D, the computer code arising from Chapter 7 is listed. In Appendix E, the computer code from Chapter 8 is listed.

CHAPTER II

LITERATURE REVIEW

2.1 Scope

In this chapter, a selected portion of the literature highly relevant to channel propagation in RC linking is reviewed. Treatment of UCG in general is outside of the scope of this chapter; the interested reader is referred to the vast Soviet literature on UCG, much of which has been translated into English and is available through Lawrence Livermore Laboratories, the *Proceedings of Annual Underground Coal Gasification Symposia* (2nd through 14th), and the monograph *Underground Coal Gasification: The State of the Art* (Krantz and Gunn, editors; 1983). The organization of this chapter is as follows. First, models of RC front propagation are discussed. Linear stability analyses (LSA) of RC front propagation models are reviewed next. Channel growth models then are reviewed. Next, field work and experimental work that have been conducted in RC are summarized. A brief comparison of model predictions with experiments is made. A critique of the literature then follows.

2.2 Reverse Combustion Front Propagation Modeling

A number of models for the propagation of a planar reverse combustion front in a porous solid continuum have been developed. These models incorporate a heat balance, a momentum balance, and one or more species balances. For wholly one-dimensional modeling, the only aspects of the momentum balance that are of interest are the gas flux and pressure in the combustion zone. In preparation for subsequent LSA, the simplest possible description of flow in porous media, Darcy's law, is usually included. Normally the model PDE's are converted into an eigenvalue problem with ODE's by transforming the equations to a coordinate system moving with the front and then invoking a quasi-steady-state approximation as was done in the pioneering work of Kotowski and Gunn (1976). These models give the combustion front temperature and propagation velocity as a function of the gas flux and composition, the physical properties of the coal, and global kinetic parameters. They show that for typical conditions, the combustion zone (approximately 1 mm thick) is much thinner than the preheat zone (approximately 1 cm thick) which in turn is much less than the channel diameter (estimated to be 1 meter for operation at low pressure) or the well spacing (about 18.3 m). Kotowski and Gunn's (1976) numerical solution considered temperature-dependent properties. This model was solved by a shooting technique.

Many extensions and adaptations of Kotowski and Gunn's model have been made. For example, Su and Engelmann (1979) extended this model to higher pressures by considering pressure-dependent devolatilization, while Lovell (1982) considered wet reverse combustion

to the extreme of liquid water passing through the front. Corlett and Brandenburg (1977) examined Kotowski and Gunn's model and used phenomenological reasoning to obtain analytical relationships between the key parameters. Britten (1984) applied the technique of high-activation-energy asymptotic analysis to derive the same result as Corlett and Brandenburg. In addition, Britten considered heat losses to the ambient from a lumped parameter perspective. He found that the combustion front temperature varied little from that for adiabatic RC until heat losses were sufficient to cause significant incomplete combustion, but that front propagation rates decreased with increasing heat loss. Monteillier (1986) also applied high-activation-energy asymptotics, obtained the same result, but was more consistent mathematically in scaling the activation energy. Britten's reduced activation energy could be large, yet the high activation energy approximation could be invalid if the pre-exponential factor was sufficiently large or the heat of reaction sufficiently small. Monteillier's scaling corrects this problem.

Corlett and Carson (1977) suggested that this picture of reverse combustion is not representative of the process as it occurs in coal. For strict validity of the continuum approach, the coal would have to be porous at a length scale smaller than the thickness of the combustion zone. This would justify the assumption of premixed combustion. Although coal has a small-scale microstructure, most of the permeability resides in the cleat and fissure structure. For this reason, Corlett and Carson felt that explicit consideration of the fissured nature of coal was needed. This crack-burning model would be a diffusion flame. This is in contrast with the continuum approach, which presumes that the effect of flow in cleats can be reasonably approximated by retaining the premixed flame model, with diffusional effects absorbed into effective global kinetic parameters. These kinetic parameters might not correctly predict the effect of process parameters on propagation velocity if the combustion is dominated by crack-burning. Corlett and Carson did not present a crack-burning RC model to use as an alternative to the continuum models. Further development of continuum models has continued.

2.3 Linear Stability Analyses

Linear stability analyses (LSA) of continuum models show that a planar RC front is unstable to infinitesimal perturbations under most circumstances and will spontaneously evolve into fingers and channels even in a uniform porous media (Krantz and Gunn, 1977; Gunn and Krantz, 1980; Okutan, 1981; Britten, 1984; Britten and Krantz, 1985; Monteillier 1986). The reason for this instability was identified to be the increase in permeability that occurs upon passage of the combustion front, combined with a tendency of the front to propagate faster when subjected to increased gas flux. Examination of the linearized perturbation equations allows the stabilizing mechanism to be identified. The principal stabilizing mechanisms present for a planar reverse combustion front were identified to be heat convection through the front and conduction perpendicular to the front. By considering the length scale over which stabilizing and destabilizing influences manifest themselves, LSA produces a dispersion relation which identifies the growth rate of different wavelength infinitesimal disturbances. This relation allows identification of a most highly amplified wavelength, or

conversely, identification of circumstances that would lead to stability against infinitesimal perturbations. It suggests that a high gas flux rate would lead to a stable front.

RC is applied to coal seams that are not sufficiently permeable to allow the large gas flows required to stabilize a broad RC front. This suggests, however, that the high gas fluxes immediately at the production well during the process of ignition could lead to a stable configuration, that would only become unstable after a certain radius was reached. This motivated the linear stability analyses of Puri (1979) and Bumb (1981) which considered cylindrical and spherical geometries. Given the large diameter of the initial combustion face compared to the thickness of the combustion zone, they justifiably neglected any curvature effects, and considered only the effects of decreasing gas flux as the front expanded and the restriction on possible wavelengths imposed by periodicity. They considered the injection well to be at infinite distance, and thus assumed the basic-state growth to be symmetric.

Provided that the basic-state model is an adequate representation of RC and that perturbations to the system are random, the most highly amplified wavelength for a radially expanding front given by LSA could be characteristic of the initial physically observable manifestation of the instability. These analyses indicate that the system is not highly selective to a specific wavelength. This would suggest that the actual number of lobes originating from the instability would be highly dependent on the distribution of the actual disturbances, whether natural or imposed.

The wavelength of the initial disturbance need not correlate with the diameter of a fully developed finger, since the finger is not subject to the same periodicity constraints. It has been argued that since the wavelengths of a planar front are not subject to a periodicity constraint, the most highly amplified wavelength might correlate with the diameter of fully developed fingers. This hypothesis has led to a methodology for adapting the results of a linear stability analyses to create a lumped-parameter phenomenological model for channel propagation. Let us call this methodology "Applied Linear Stability Analysis" (ALSA) to distinguish it from the underlying linear stability analysis. In ALSA, one must postulate a relationship between the wavelength of the most highly amplified mode and the channel diameter. It has been assumed that the channel diameter is one-half the wavelength of the most highly amplified mode from the linear stability analysis. This linear stability analysis is conducted for a planar gas flux equal to the flow rate of gas to a channel divided by the projected cross-sectional area of the channel. Once converged values of channel diameter and the gas flux are found, the gas flux is inserted into the planar basic-state model for RC to evaluate the front speed. By using this methodology, it has been possible to fit reasonably the field data available for the Hanna, Wyoming field tests (Krantz and Gunn, 1977; Gunn and Krantz, 1980; Okutan, 1981; Krantz and Gunn, 1981; Britten *et al.*, 1982; Britten 1984). As pointed out by Britten (1984), this might be fortuitous.

Another lumped-parameter approach based on stability considerations is that of Mui and Miller (1983). They considered a combustion finger consisting of an elliptical tip followed by a straight walled-channel. By making several approximations, they generated a leading-order solution for the flow field in this specific configuration by using separation of variables. These approximations appear quite plausible for pointed channels, but not for blunt channels. The flux is presumed to monotonically decrease from its value at the channel

tip to a negligible value at the joint with the straight-walled channel. A blunt channel would have a maximum flux at the "corners" of the ellipse, not at the tip. Furthermore, the flux at the joint is not negligible for a blunt channel. They based their analysis on a particular, atypical case of Kotowski and Gunn's (Krantz and Gunn, 1981) where the relationship between propagation velocity and the gas flux reached a maximum at an unusually small gas flux, not found in most of Kotowski and Gunn's work, or in other planar RC models (see Britten, 1984, p. 112 for more typical results). However, given the lack of definitive kinetic parameters, it is possible to fit these parameters to achieve the desired maximum front speed at a sufficiently low gas flux. They produced reasonable agreement with the same field test data analyzed by the previous model.

In analyzing the conditions at the very tip of the channel, they argue:

"If we assume, as before, that the fastest-growing finger is favored, we expect to find a finger with a cylindrical channel but with a tip that is nearly flat [to minimize lateral heat transfer] and that moves with a velocity V^* [the maximum possible velocity for a planar front]. [The] Channel width will be that which makes [the] air flux approaching the flat tip equal to v_t^* , the value which produces a front velocity of V^* ."

As the gas flow is reduced, the tip would have to become more sharply pointed in order to attract sufficient flux. This would lead to lateral heat transport which would slow the propagating front compared to that predicted by the planar 1-d propagation relationship. Thus, this model would be inaccurate at lower gas flow rates. The same observations can also be made about ALSA.

2.4 Two-dimensional Channel Evolution Modeling

The solution to the continuum model equations is much more difficult for channel propagation than for planar front propagation because of the multiple spatial dimensions involved, the convoluted shape of the front, and the thinness of the combustion zone at the front, compared to the size of the channel. These difficulties have discouraged direct solution of the governing equations for channeling. Instead, attention has been limited to coupling an end result of planar models, the relationship between gas flux and propagation speed, with a hydrodynamic description of the flow to the channel. Preliminary nonlinear analyses of planar RC fronts show that there is some justification for this simplification (Monteillier, 1986; Clavin, *et al.*, 1986). To completely specify the channel propagation problem, one must specify the geometry of the coal seam and the initial shape of the combustion front. In general, this leads to a transient problem.

The transient numerical simulation of Thorsness *et al.* (1977) considered the two-well configuration. This model produced a smooth, broad, non-fingering reverse combustion front. Although spatially distributed heat and material balances were conducted, the principal feature of the model was a linear relationship between the combustion front velocity and the local gas flux. Given a temperature-dependent permeability, this problem is analogous to viscous displacement which is an unstable process. Consequently, small scale structures

should have arisen, but they did not. This failure has been variously interpreted. The authors considered it evidence that channeling must occur because of an initial preferred permeability path. Although they did not publish sufficient details on their numerical algorithm for this preliminary work to allow conclusive identification of the source of this behavior, they gave enough to raise the suspicion that it was entirely numerical. A grid of 340 points was used to model an entire two-dimensional two-well flow field.

Britten (1984) used conformal-mapping techniques along with complex-variable theory in studying the steady-state shape of a single combustion finger propagating along an infinitely long, comparatively thin, laterally bounded medium (the Hele-Shaw cell configuration). This technique was not particularly robust, but was sensitive to formulation of the extinction process. Consequently, the various propagation relationships that he used were only roughly based on his nonadiabatic model for planar RC with heat losses to the ambient that was discussed in Section 2.2. In particular, they did not show a maximum in the relationship between the propagation speed and the gas flux. Given the lack of any lateral stabilizing mechanism in the propagation rule, any suppression of small length scale instabilities was due to the restriction of considering only steady-state self-preserving smooth concave shapes. Britten envisioned his model as applying to the particle-board experiments conducted at the University of Colorado (Visser *et al.*, 1985). It does not consider lateral heat transport in the plane of the cell, but only losses to the surroundings. No stability analyses of the basic-state planar model have been undertaken, so the relative importance of these heat losses to stabilizing the front are not known. Britten recognized that heat transport in the channel core might have an important effect on the extinction phenomena, but he apparently did not consider the significance of lateral heat transport in the preheat zone.

Britten's model also assumes an infinite permeability change upon devolatilization. This assumption leads to reasonable estimates of the normal component of the flux through the channel surface for large but finite permeability changes. The same is not true for the tangential components. When a streamline crosses from one permeability zone to another, it diffracts. A physical demonstration of this phenomenon is given by Hele-Shaw and Hay (1901). This photograph shows flow through a Hele-Shaw cell, where the streamlines are visualized by dye injection. Through the use of a cut-out, a region of reduced resistance to flow is created in the shape of an ellipse placed obliquely to the flow. As the surface of the ellipse is approached from the outer, higher resistance zone, the streamlines smoothly bend toward the ellipse. The streamlines are nearly perpendicular to the surface where they contact the surface. The streamlines inside the lower permeability zone are essentially parallel. Consequently they are not uniformly perpendicular to the surface where they contact the surface, but instead meet at a highly variable angle. The reason why the streamlines diffract is clearly explained by Hubbert (1969). The normal component of the flux is continuous through the surface as required by conservation of mass. The pressure along the surface is the same, whether considered from the outside or the inside, hence the tangential component of the gradient in pressure is also the same. The tangential component of the flux is proportional to the negative of tangential component of the gradient in pressure, however, the proportionality constant (permeability) is not the same in the two different regions. Thus,

the tangential component of the flux is discontinuous across the surface and the streamlines diffract. If the permeability is infinite inside the channel, the channel surface is at a constant pressure, so the tangential component of the gradient is zero. However the magnitude of the tangential component of the flux is equal to the product of the gradient times the permeability, and thus is indeterminate, not zero as Britten assumed. The actual magnitude of tangential component of the flux in an infinite permeability zone can only be determined through a limiting process. For a finite but large permeability, lateral convection may have an important influence on channel geometry, since the tangential component of the gas flux could be large.

As in most previous studies, the underlying one-dimensional model assumes that oxygen is always the limiting reactant. However, this assumption must break down at some point on the channel surface, because if the channel does not advance, there is no fuel. Even if combustion is not extinguished, the channel cannot propagate if the temperature falls below that required for devolatilization.

This study did not lead to a predictive model for RC, even for the artificial Hele-Shaw cell configuration. The model does not capture enough of the physics of RC.

For the same geometry, Monteillier (1986) assumed a linear relationship between the propagation velocity and the gas flux, once a minimum gas flux necessary for sustaining combustion was reached. The results have cusps that would be suppressed by stabilizing effects in a reasonable model of RC. Since this formulation has no stabilizing effects, unique solutions were not found. This is largely equivalent to the results of the classic Saffman and Taylor formulation (1958) for viscous displacement. In considering the propagation of a finger of low viscosity liquid through a Hele-Shaw cell filled with high viscosity liquid, they found a family of solutions parameterized by the ratio of the width of the finger to the width of the cell.

Fuhrmann and coworkers (Fuhrmann, 1987; Fuhrmann, *et al.*, 1988) created a transient, two-dimensional model of channel propagation for the basic two-well configuration, by coupling electro-conductive analog flow calculations with a manually implemented propagation relationship. A piece of electrically conductive carbon paper serves as an analog to the coal seam. Electrodes painted with silver paint on the surface of the paper serve as analogs to the injection well and the channel surface. A modest electrical potential is imposed between these two electrodes. The electrical field thus created is an analog to the pressure field in the coal seam. By measuring the gradient in this analog field at the surface of the electrode that represents the RC channel, the gas flux distribution to the channel surface is calculated. This gas flux distribution is then input into the propagation relationship to determine the distance the front should propagate. At each time step, the combustion front was advanced by enlarging the representative electrode with silver paint on the conductive carbon paper. The accuracy of these calculations is questionable, since relatively large size time steps were taken. The channels were sometimes extremely thin, and had a strong tendency to bifurcate. Given the lack of any lateral stabilizing mechanism in the propagation rule, any suppression of these bifurcations was due to physical limitations in propagating the front, not mathematical effects.

2.5 Reverse Combustion Field Work

Our observations on the character of RC as practiced for RC linking come from two sources. The first is the long-term Soviet experience in UCG. The Soviet experience with RC comes from several decades of "commercial-scale" operation of UCG. This work produced an empirical understanding of RC in the particular coal basins in which it was practiced, but it did not lead to RC being a highly reliable process. As reviewed by Gregg *et al.* (1976), the Soviets observed that the superficial propagation velocity exhibited a weak maximum with respect to the rate of gas injection. They also observed that the propagation velocity increases with greater oxygen content in the injected gas. The resistance to flow of the resulting channel(s) increased when the oxygen content was greater.

The second source is the relatively small number of UCG field tests that have been carried out in North America and an even smaller number in Western Europe, and elsewhere. Although RC was used in some of them, none were RC field tests. None were intentionally terminated before link completion, and then excavated to observe the character of the reverse combustion channel and especially the growing front. The data from these field tests are insufficient to formulate an unambiguous understanding of RC linking.

A set of UCG field tests were carried out in the 1970's in Hanna, Wyoming that used RC linking in a thick seam of coal. In four successful tests, the linking time was approximately constant, despite completely different gas injection policies (Bartke and Gunn, 1983). The superficial propagation velocity was about 8 cm/hr with the combustion back pressure of about 300 kPa. The failure of two other tests was attributed to the presence of a fault between the wells. Thermocouples placed in instrumentation wells suggested the growth of two reverse combustion channels in the Hanna IV, Phase I field test.

A UCG field test in 1976 near Forestburg, Alberta, was excavated. The RC channel was char-filled, and hence suggestive that RC was occurring by the process envisioned by the continuum models, but since gasification took place prior to excavation, the native condition of RC channel can not unambiguously be identified. The link was approximately 0.3 m wide by 1 m high after gasification. The channel advanced at 6 cm/hr on the average, requiring 3700 m³ of air per meter of link with a combustion back pressure of near atmospheric. (Roehl *et al.*, 1978).

A field test was carried out in Pricetown, West Virginia, to explore the feasibility of UCG in deeper, swelling bituminous coals. The superficial propagation velocity was in the range of 6 to 9 cm/hr for different links with an injection gas rate of 0.28 m³/min and a link back pressure of 1200 kPa. In the initial reverse combustion linkage, insufficient permeability enhancement was achieved; hence an extensive procedure was pursued (Martin and Liberatore, 1980) to enhance it further.

Reverse Combustion was unsuccessful at creating a link at the Thulin, Belgium Test site (Ledent *et al.*, 1984). This field test was conducted with coal at 850 m with a back pressure of up 9500 kPa. The ALSA models predicted extremely small link diameters for these conditions and hence that RC would not succeed under the high combustion back pressure used. However, the plastic behavior of the coal must also be considered at this

depth.

This short overview does not cover all of the field test data that can be gleaned from the records, yet it does show that the field work provides no undisputable guide for model formulation, and relatively little against which to test models, given that results from one coal seam are difficult to generalize to other coal seams. All models are dependent on kinetic parameters, which might vary from coal to coal.

2.6 Experimental Studies

Reed *et al.* (1960) conducted some of the earliest laboratory studies of RC in their investigation for tar sands. They studied propagation of RC in small diameter combustion tubes filled with tar sands. In this and later work in tar sand (Cupps *et al.*, 1976), the peak front temperature displayed a weak dependence on the gas flux in the range investigated. The propagation velocity increased significantly when the gas flux was increased. In studying the effect of pressure, Wilson *et al.* (1963) found that increasing the pressure resulted in a decrease in the front temperature, while the propagation velocity increased for the same gas flux.

Arguing that reverse combustion in coal was not predominantly a porous media permeation process, but that combustion occurred in cracks, Pitin and coworkers (Pitin, 1959) conducted early inconclusive experimental studies of reverse combustion in slots created in wood and coal. Corlett and coworkers conducted more extensive experiments. (Corlett and Shearer, 1978; Carson, 1977). The work in wood showed the combustion front velocity to be insensitive to the gas flux, but to increase with the width of the slit in the range considered (1 to 5 mm). The work with coal was less extensive and less reproducible, but also showed an insensitivity of propagation velocity to gas flux, irrespective of whether char combustion occurred leaving an open channel, or whether combustion was entirely gas phase, leaving a core of char around the initial slot. The variation of the front temperature with gas flux was not reported. Since the cracks were placed there, these experiments do not demonstrate that RC linking actually occurs by this mode of operation.

Kotowski and Gunn's modeling studies were based on kinetic parameters derived from Reed and coworkers' tar-sand experiments. Lu (1980) studied propagation of a reverse combustion front in powdered coal to obtain kinetic parameters for coal to use with Kotowski and Gunn's model. Extensive efforts were undertaken to minimize channeling. Through the use of heaters to minimize lateral heat transfer, the process was made nearly one-dimensional. These experiments yielded no information on the validity of the continuum models for actual coal seams. The results of six experiments were reported, five of which were at constant pressure. For the constant pressure experiments, the experimental design consisted of two replicates of both a high and low gas flux case, along with an intermediate flux case. The constant pressure experiments (800 kPa) indicate an increase in front temperature with gas flux. The magnitude of this increase is uncertain, since the temperature variation between the replicates for the high flux case was 75 percent of the total observed temperature variation for all the experiments. Consequently Lu's data do not provide useful kinetic parameters, even if we assume that the kinetic parameters for powdered coal are similar to those for

solid coal. This assumption is questionable, given all of the interactions that contribute to the global kinetic parameters. The propagation velocity increased from about 8 to 21 cm/hr when the gas flux was increased from 33 kg/hr/m² to 134 kg/hr/m².

Particle board and balsa wood have been used in studies of channeling in RC (Cooney, 1979; Britten *et al.*, 1984; Visser *et al.*, 1985). Balsa wood has a very high directional permeability, such that it is not useful as an analog to coal. Hence, we focus our attention on the particle-board experiments. First, essentially one-dimensional experiments were undertaken in which RC was propagated down a long (40 cm) piece of particle board having a small square cross-section (1.9 cm X 1.9 cm). These experiments were well insulated laterally, but active measures to prevent lateral heat loss such as those used by Lu were not employed. No quantitative measurements or calculations were undertaken to estimate the magnitude of the lateral heat loss. If several outlying points are ignored, the temperature increase is observed to be basically linear with flux over a wide range of flow rates for 13 such experiments, although the minimum gas flux reported was over 100 kg/hr/m². In this range, the front velocity was insensitive to flow rates in view of the scatter in the data. The front velocity was in the range of 20-30 cm/hr.

In addition, experiments were undertaken with thin pieces of particle board, in both circular and slab geometry. Front propagation was monitored without altering the flow profiles via infrared thermography. This did not give quantitative information on the variation of the temperature along the front, due to thermal lag and smoothing that occurred as a result of transient conduction through the insulating layer of furnace cement. For the slab experiments, the propagation velocity of the leading channel tip ranged from 15 cm/hr at lower total flow rates to a maximum of 25 cm/hr at higher flow rates. After the burn, the slabs were sliced open, revealing the terminal shape of the channel. The total channel width was found to increase with total gas flux. The combustion front was lobed with each lobe having a width of about 4 cm. These lobes developed into fingers which grew away from each other. Trailing fingers progressed at slower rates, and eventually ceased to propagate. From a review of Visser *et al.* (1985) and the portion of the samples that have been retained, the terminal shape of the channels that ceased to propagate appears to be more sharply pointed than those on the active front. No accurate point-wise calculations of gas flux to the channel surface were made. The slab experiments were analyzed using ALSA. The channel diameter was found to agree closely with that predicted by the model. No comparison of the predicted versus observed propagation velocity was reported. No analysis of the circular geometry experiments has been published.

Preliminary experiments in RC in solid coal blocks of high rank European coal commenced at Aachen University of Technology in the early eighties (Breidung, 1982; Kurth *et al.* 1981). Significantly more refined and extensive experiments were reported recently (Schloemer and Wagner, 1988), during the course of our study. These experiments used a combustion apparatus capable of imposing mechanical stress on the coal equivalent to overburden pressures generated at 1500 m on a bituminous coal sample 17 cm diameter and a length of 40 cm. Over one hundred experiments were undertaken; 112 were identified as successful. The number of unsuccessful experiments was not reported. All experiments used an oxygen concentration appropriate to air injection. The same combustion back pressure

was used (2 MPa) in all experiments, with the simulated overburden pressure set at either 23 MPa or 6 MPa. The oxidant-gas flow rate was varied from 0.4 to over 8 kg/hr. For the results reported (approximately 26 data points), the front velocity increased with mass flow rate linearly from zero up to about 25 cm/hr at 2.4 kg/hr. It is not reported whether these data are only for those experiments that did not bifurcate or whether they include the propagation of multiple channels. The data scatter about 3 to 4 cm/hr, thus at low flow rates have an uncertainty of one-hundred-percent. Two experiments above this mass flux suggest a maximum propagation rate of 40 cm/hr around 6.5 kg/hr. No successful experiments occurred above 8 kg/hr. No direct temperature measurements were made. The channels created consisted of a small open core surrounded by a layer of coke. This structure closely resembles that observed by Corlett and coworkers for reverse combustion at ambient pressure in a manufactured slot. The channels were conclusively identified as originating in fissures and cleats in 40 percent of the experiments and in lenticular fusain spots in 16 percent of the experiments. This is good evidence that at high overburden pressure, small channels will track heterogeneities in the coal structure.

2.7 Critique of the Literature

The treatment of coal as a homogeneous media is probably the most fundamental limitation of current theory, but creation of a crack-burning model is beyond the scope of this work. Models of planar reverse combustion in porous media are mature. However, due to the channelling nature of RC, these models are directly useful only for analysis of carefully controlled laboratory experiments and as a basis for further mathematical development.

The basis of much of the current theory of RC comes from linear stability analyses (LSA) of planar, cylindrical, and spherical RC fronts in homogeneous media. These have been adapted to produce a lumped-parameter model for RC channel propagation. Given the complex nature of actual RC channels, there is a need to establish a more robust basis for RC theory. There have been limited efforts already to develop multidimensional RC models; more work is needed.

Existing one-dimensional models of RC are not an immediately suitable foundation for a numerical analysis of fully developed RC channels. LSA for some of these models suggests that tangential heat transport has a stabilizing effect, and that it arises from tangential variation in the gas flux. This effect has been ignored completely in adapting the planar results to curved fronts, although the corresponding LSA included it. An additional limitation of previous analytical models of reverse combustion is that they have presumed a superabundance of fuel. Although this is a reasonable assumption to make at the tip of a channel, it could break down at some point on the trailing edges of a combustion finger. This limitation should be removed if these models are to be more generally applicable.

Some of the existing two-dimensional work on combustion channel propagation has considered the steady-state propagation of a single combustion finger down an infinitely long lateral bounded domain (the Hele-Shaw cell configuration). This configuration has little resemblance to the actual field situation, making it difficult to adapt results to the true situation. Modeling attention should consider also the two-well configuration, since it

includes features that are not captured by either a slab or pure circular geometry. Previous numerical work in this area did not have the spatial resolution necessary to provide a reasonable model of RC channel propagation. An electroconductive model has been recently reported, but its accuracy is questionable. There is still a need for a transient numerical model, that directly simulates a reasonable two-dimensional analog of the full problem and is capable of resolving the channel shape.

Despite sustained interest in RC in the last fifteen years, there are relatively limited laboratory experimental data for RC. RC laboratory experiments with coal are expensive and time-consuming. The channelling characteristic of RC, so necessary for it to be useful for linking, serves as a barrier to easy interpretation of experimental results. The largest body of experimental work has been done with a single type of coal at a fixed oxygen content of the injected gas, and a fixed combustion pressure. This work was done to directly simulate RC at elevated pressures and high overburden pressures that would be typical of RC linking in deep European coals. In contrast, it has been proposed that RC at shallow depths typical of North American operation can be simulated in "miniature" by operating at elevated oxygen content. This idea is worth exploration.

CHAPTER III

OPTIMIZING REVERSE COMBUSTION LINKING WITH PROMOTERS AND INHIBITORS

3.1 Scope

In this chapter, a novel means of influencing reverse combustion linking is presented and evaluated in respect to existing theories for RC. This control parameter should allow reverse combustion to be better optimized for the specific field conditions. It should make reverse combustion either less expensive by allowing faster propagation of the channel at a given oxidant injection rate, or more effective by controlling the degree of permeability change. This chapter is condensed from a paper with the same title presented at the *Proceedings of the 13th Annual Underground Coal Gasification Symposium* (Dobbs and Krantz, 1987). The control of reverse combustion, by both conventional methods, and the unconventional method of using combustion inhibitors or promoters is discussed. Next, a more general solution to the reverse combustion model is developed. This solution is needed to test the effect of combustion promoters. A discussion of results follows. The chapter ends with a summary.

3.2 Control of Reverse Combustion

In reverse combustion linking, one wishes to achieve the desired permeability enhancement at the least cost. To do so, it is desirable to control the number of reverse combustion channels, their diameters, their permeabilities, and their propagation velocities. Unfortunately it is not possible to adjust a control parameter to change one of these characteristics without affecting all of them. It is necessary to choose the optimum set of control parameters.

Since reverse combustion linking occurs underground, it is controlled predominantly by what is injected into one of the wells and by the back pressure on the other well. Britten *et al.* (1982) have shown that increasing back pressure results in smaller channels that propagate faster. They also showed that increasing the oxygen content of the injected gas has the same result. In simulating the Hanna field tests, they showed that the propagation velocity should be almost independent of the total gas flow rate, while the channel diameter increased monotonically with the total gas flow rate; this result might be specific to the Hanna test site. Field tests in Belgium, France, USA and the USSR have in general supported these conclusions. These three parameters were considered to be the only means of influencing reverse combustion in the work of Britten *et al.* However, injecting water has been shown to decrease the propagation velocity in reverse combustion linking (Lovell, 1982). The effect of

these control parameters on the number of channels formed is not known with any certainty.

The problem with these methods is that they are expensive. Although decreasing the gas flow rate costs nothing, increasing the gas flow rate increases the compression cost in two ways since both the capacity and the discharge pressure must be increased. Increasing the back pressure on the production well requires greater discharge pressure. The production well back pressure can be reduced only so far, while still controlling water influx. Increasing the oxygen content requires supplementing cheap compressed air with expensive liquid oxygen or building an air separation plant. Decreasing the oxygen content increases compression costs by the amount of the diluent; in addition, the diluent must be supplied at some price. What is needed is a way to finesse reverse combustion into operating the desired way at low cost, instead of using these crude methods. A way to do this is implicit in previous work, but has not been developed explicitly before.

Britten (1984) observed that the behavior of his reverse combustion model was highly dependent on the kinetic parameters that he chose. Increasing the pre-exponential factor in the overall Arrhenius kinetics or decreasing the activation energy has the same effect as increasing the back pressure or the oxygen content; it decreases the channel diameter and increases the propagation velocity. He did not consider these to be control parameters. Rather, along with previous workers, he considered the kinetic parameters to be inherent properties of the coal, despite recognizing that reverse combustion is normally a gas-phase combustion process. As such, the overall kinetics should be functions of the composition at the combustion front, which is dependent on the devolatilization and pyrolysis products, and the composition of the gas injected as oxidant.

Addition of trace components acting as combustion inhibitors and promoters might greatly affect the overall combustion kinetics, by altering the effective kinetic parameters. This, in turn, would alter the RC propagation velocity and channel diameter. Controlling RC by addition of promoters or inhibitors might be more economic than using more gross measures such as altering the back pressure on the production well, or adjusting the oxygen content of the injected gas.

It is reasonable to expect that trace components can alter combustion kinetics. Combustion is a complex, multi-step chain reaction. Ignition is dependent on interplay between chain initiation, branching, and termination steps. This balance can be affected by trace additions. The effect is then propagated by the chain reaction. Examples of trace components affecting combustion can be easily found. Perhaps the most common one is the long-standing practice of using gasoline antiknocking agents such as tetraethyl lead. Addition of a small amount of butane to pure methane greatly decreases the spontaneous ignition temperature, presumably through increased amounts of free radicals acting as chain initiators. (The butane molecule is less thermally stable than the methane molecule.) Halon fire suppressants work on the principle of increasing the rate of the chain-termination reaction. Finally and more to the point, propane has been added to RC to maintain ignition (Gadelle *et al.*, 1985).

Britten and Krantz's (1985) high-activation-energy asymptotic solution provides an initial prediction of the effect of promoters and inhibitors. Unfortunately this solution can be rigorously defended only for high activation energies. In simulating the Hanna field tests, the activation energy was close to the strict limit of applicability. Addition of promoters

will result in activation energies outside this range. It is impossible to assess beforehand the accuracy of an asymptotic solution outside its range of strict applicability. It is important to determine if the asymptotic predictions carry over to lower activation energies, or whether they are simply artifacts of the asymptotic limit. Numerical solutions such as Kowtowski and Gunn's (1976) have the drawback they never can give a complete and final answer, but merely a series of test cases. Instead, in the following section, a more general solution to Britten and Krantz's (1985) planar model will be developed that is not explicitly limited to high activation energies. There is still an implicit limitation to this solution, since the model itself does not allow the possibility of devolatilization and pyrolysis occurring in the combustion zone, which would occur for sufficiently small activation energies.

3.3 Generalized Solution to RC Model

In this section, a generalized solution to Britten and Krantz's (1985) planar reverse combustion model will be developed. The model postulates that devolatilization occurs ahead of the combustion zone, and does not affect the characteristics of RC. Gas-phase combustion is assumed. The molecular weights of all the species are assumed to be equal; consequently, the mass fraction of a component is numerically equal to its mole fraction. The change in physical properties upon devolatilization is ignored. The heat of reaction is assumed to be constant and to combine the effects of the combustion and devolatilization reactions. The solution developed here is semi-analytic, in that the temperature and oxygen profiles are analytic functions of parameters numerically determined. The solution does not directly invoke a high-activation-energy assumption.

The one-dimensional species conservation equation for the oxidant in a stationary coordinate system is given by the following equation which is posed on a mass basis:

$$\varphi \rho_g \frac{\partial Y^*}{\partial t^*} + N_g \frac{\partial Y^*}{\partial x^*} = D \rho_g \frac{\partial^2 Y^*}{\partial x^{*2}} - \varphi r^* \quad (3.1)$$

The superscript * indicates dimensional variables. In this equation, φ is the porosity, ρ_g is the density of the gas phase, Y^* is the mass fraction of the oxidant, t^* is the time, N_g is the gas flux, x^* is the spatial coordinate, D is the effective molar diffusivity of the oxidant, and r^* is the volumetric reaction rate. The thermal energy conservation equation in a stationary coordinate system is given by the following equation:

$$C_s \rho_s \frac{\partial T^*}{\partial t^*} + C_g N_g \frac{\partial T^*}{\partial x^*} = \lambda \frac{\partial^2 T^*}{\partial x^{*2}} + q \varphi r^* \quad (3.2)$$

where C_s is the heat capacity of the solid phase, ρ_s is the density of solid phase, T^* is the absolute temperature, C_g is the heat capacity of the gaseous phase, λ is the effective thermal diffusivity, and q is the molar heat of reaction. The reaction rate is given by the equation:

$$r^* = A^* e^{-E/RT^*} P^2 Y^* \quad (3.3)$$

where A^* is the effective frequency factor, E is the effective activation energy, P is the absolute pressure and R is the universal gas constant. It assumes that the overall kinetics

are first-order in both the oxygen and the fuel, and that the fuel is present in large excess. This is slightly different than the reaction rate term used by Britten and Krantz, in that it considers the reaction rate to be a function of the partial pressures of the components instead of mass densities of the components, thus maintaining the link between thermodynamics and kinetics. These equations are converted to a moving coordinate system by using the following transformation:

$$T^*(x^*, t^*) \longrightarrow T^*(\eta^*, \tau^*) \quad (3.4)$$

$$Y^*(x^*, t^*) \longrightarrow Y^*(\eta^*, \tau^*) \quad (3.5)$$

$$\eta^* = x^* + Vt^* \quad (3.6)$$

$$\tau^* = t^* \quad (3.7)$$

where η^* is a spatial coordinate centered on the combustion front moving at a velocity of V . With these transformations, eqns. (3.1) and (3.2) become:

$$\varphi \rho_g \frac{\partial Y^*}{\partial \tau^*} + (\varphi \rho_g V + N_g) \frac{\partial Y^*}{\partial \eta^*} = D \frac{\partial^2 Y^*}{\partial \eta^{*2}} - \varphi r^* \quad (3.8)$$

$$C_s \rho_s \frac{\partial T^*}{\partial \tau^*} + (C_s \rho_s V + C_g N_g) \frac{\partial T^*}{\partial \eta^*} = \lambda \frac{\partial^2 T^*}{\partial \eta^{*2}} + q \varphi r^* \quad (3.9)$$

Let us now introduce the following dimensionless variables:

$$T = \frac{T^*}{E/R} \quad (3.10)$$

$$Y = \frac{Y^*}{Y_{-\infty}^*} \quad (3.11)$$

$$\eta = \frac{\eta^*}{\lambda/\Sigma} \quad (3.12)$$

$$\tau = \frac{\tau^*}{C_s \rho_s \lambda / \Sigma^2} \quad (3.13)$$

with $\Sigma = C_s \rho_s V + C_g N_g$. This results in the following two equations:

$$N_1 \frac{\partial Y}{\partial \tau} + (1 + N_2) \frac{\partial Y}{\partial \eta} = \frac{1}{Le} \frac{\partial^2 Y}{\partial \eta^2} - Ae^{-1/T} Y \quad (3.14)$$

$$\frac{\partial T}{\partial \tau} + \frac{\partial T}{\partial \eta} = \frac{\partial^2 T}{\partial \eta^2} + Q Ae^{-1/T} Y \quad (3.15)$$

The definition of the dimensionless parameters in the previous equations are as follows:

$$A = \frac{\lambda \varphi A^* P^2}{\Sigma N_g} \quad (3.16)$$

$$Q = \frac{Y_{-\infty} N_g q}{\Sigma E/R} \quad (3.17)$$

$$T_{-\infty} = \frac{T_{-\infty}^*}{E/R} \quad (3.18)$$

$$N_1 = \frac{\varphi \rho_g \Sigma}{C_s \rho_s N_g} \quad (3.19)$$

$$N_2 = \frac{\varphi \rho_g V}{N_g} \quad (3.20)$$

$$Le = \frac{\lambda N_g}{D \rho_g \Sigma} \quad (3.21)$$

Under typical reverse combustion conditions, $N_1 \ll 1$, $N_2 \ll 1$, and $1/Le \ll 1$. In addition, for the basic-state, the solution is at steady-state in the moving coordinate system. Hence, the terms containing partial derivatives with respect to τ drop out. This leaves the following simple set of equations:

$$\frac{dY}{d\eta} = -Ae^{-1/T}Y \quad (3.22)$$

$$\frac{dT}{d\eta} = \frac{d^2T}{d\eta^2} + QAe^{-1/T}Y \quad (3.23)$$

The appropriate boundary conditions are:

$$T = T_{-\infty} \quad @ \quad \eta = -\infty \quad (3.24)$$

$$Y = 0 \quad @ \quad \eta = \infty \quad (3.25)$$

$$Y = 1 \quad @ \quad \eta = -\infty \quad (3.26)$$

There is one more possible step that can be taken to reduce the order of the thermal energy equation. If the species equation is substituted into thermal energy equation, it is possible to analytically integrate once. After applying the boundary conditions, the following first-order differential equation in T arises:

$$T - \frac{dT}{d\eta} = -QY + T_{-\infty} + Q \quad (3.27)$$

There are three dimensionless parameters remaining in the previous equations: $T_{-\infty}$, Q , and A . Both Q and A contain the unknown front velocity through Σ . Thus, there are really only two independent dimensionless parameters that define the problem. These parameters can be chosen arbitrarily, with the exception that Σ must not be contained in them. It is useful to consider the independent parameters to be:

$$\frac{1}{T_{-\infty}} \quad \text{and} \quad \frac{AT_{-\infty}}{Q}$$

The virtue of this choice is that $1/T_{-\infty}$ is proportional to E and independent of A^* , whereas $(AT_{-\infty})/Q$ is proportional to A^* and independent of E . The third parameter, which depends on the other two parameters, is an eigenvalue or characteristic value. The eigenvalue is an

integral part of the *solution* to the model. It can be chosen arbitrarily, with the exception that it must contain the unknown front velocity, V (or equivalently Σ). In effect, the eigenvalue nature of the transformed equations is the price that is paid for changing the partial differential equations into ordinary differential equations by invoking the quasi-steady-state assumption.

Now, we will solve this problem approximately, using a trial function with adjustable parameters. If the nonlinear reaction rate term

$$r = Ae^{-1/T}Y \quad (3.28)$$

were known as a function of η , the two nonlinear coupled equations (Equations 3.22 and 3.23) could be easily solved. The general form for the spatial dependence of the reaction rate can be inferred. Ahead of the combustion zone, the reaction rate will decay toward zero because of lack of thermal energy. Behind the combustion front, the reaction rate will decay toward zero because of exhaustion of the oxidant. At the combustion front, the reaction rate will peak. In addition, to satisfy the boundary conditions on the species equation, the integral of the reaction rate from $\eta = -\infty$ to $\eta = +\infty$ must be 1. The simplest smooth trial function, r_t , for the reaction rate that exhibits this behavior is the normal probability distribution, which can be written as follows:

$$r_t = \frac{\beta}{\sqrt{\pi}} e^{-\beta^2 \eta^2} \quad (3.29)$$

where β is an adjustable parameter.

Using this trial function, it is possible to integrate Equation 3.22 to get the trial oxygen profile, Y_t , as a function of β :

$$Y_t = \frac{1}{2} - \frac{1}{2} \text{erf}(\beta\eta) \quad (3.30)$$

If this result is substituted into Equation 3.27, the following ODE is obtained:

$$T_t - \frac{dT_t}{d\eta} = -Q \left[\frac{1}{2} - \frac{1}{2} \text{erf}(\beta\eta) \right] + T_{-\infty} + Q \quad (3.31)$$

Let us define a new temperature variable:

$$\Theta = \frac{2}{Q}(T_t - T_{-\infty}) - 1 \quad (3.32)$$

With this definition, Equation 3.31 is transformed into:

$$\Theta - \frac{d\Theta}{d\eta} = \text{erf}(\beta\eta) \quad (3.33)$$

The solution is:

$$e^{-\eta}\Theta = - \int e^{-\eta} \text{erf}(\beta\eta) d\eta + C \quad (3.34)$$

After evaluating the integral (Abramowitz and Stegun, 1964) and applying the appropriate boundary condition, the solution can be written as:

$$\Theta = \text{erf}\beta\eta - e^{\eta-1/(4\beta^2)} [\text{erf}(\beta\eta + 1/(2\beta)) - 1] \quad (3.35)$$

Transforming back to T from Θ yields:

$$T_t = T_{-\infty} + \frac{Q}{2} \left[1 + \operatorname{erf} \beta \eta - e^{\eta - 1/(4\beta^2)} [\operatorname{erf}(\beta \eta + 1/(2\beta)) - 1] \right] \quad (3.36)$$

The oxygen profile, Y_t , and the temperature profile, T_t , can be substituted into Equation 3.28 to give:

$$r_r = e^{1/T_t} Y_t \quad (3.37)$$

where r_r is the reaction rate that results from assuming the trial reaction rate, r_t .

Now we must select appropriate values for β and Q as functions of $1/T_{-\infty}$ and $(T_{-\infty}A)/Q$ that make the resulting reaction rate approximately equal to the trial reaction rate. There are, however, a multitude of approximate criteria that can be used to define "approximately equal." Furthermore, we need an error estimate to evaluate whether this simple one-term trial function can adequately represent the true solution to the problem, preferably without recourse to resolving the problem by a different method. What we have done is to choose two different criteria that introduce different errors. This should give us an estimate of the error in the eigenvalue, which can in turn be used to evaluate the accuracy of the solution, without recourse to a higher-order-trial function expansion.

Two equations are needed to determine β and the eigenvalue, Q . The first criterion constitutes demanding that the resulting reaction rate reaches a maximum at $\eta = 0$. This implies that both the trial and resulting reaction rates are centered about the same point. This can be stated mathematically as:

$$\left. \frac{dr_r}{d\eta} \right|_{\eta=0} = 0 \quad (3.38)$$

In the first case, the criterion constitutes demanding that the reaction rate at $\eta = 0$ be equal for both the trial function and resulting reaction rate. This can be stated as

$$r_r|_{\eta=0} = r_t|_{\eta=0} \quad (3.39)$$

In the second case, the second criterion constitutes demanding that globally the conservation of species equation satisfy the boundary conditions. This can be stated as:

$$\int_{-\infty}^{\infty} r_r d\eta = 1 \quad (3.40)$$

The integral must be evaluated numerically. This was done using scaled hermetian quadrature with twenty points. The quadrature points were scaled so that they were concentrated in the reaction zone. The resulting integral was then scaled to account for the scaling of the quadrature points. The resulting two nonlinear algebraic equations were solved using an IMSL MATH/LIBRARY routine ZSPOW (IMSL, 1982) for the eigenvalue, Q , and the adjustable parameter, β , as functions of $1/T_{-\infty}$ and $(T_{-\infty}A)/Q$.

3.4 Discussion of Results

The resulting eigenvalue and adjustable parameter were insensitive to the choice of criteria in matching the assumed reaction-rate profile to the resulting reaction-rate profile. For the values of the independent parameters chosen, the eigenvalue determined for each case agreed within 2 percent (usually within 0.5 percent). The parameter β also agreed within 6.5 percent.

The average of the eigenvalues from the trial function solution agreed with the results predicted from Britten and Krantz's high-activation-energy asymptotic solution (after accounting for the slightly different reaction rate term) to within 34 percent. This is within the expected uncertainty of the asymptotic analysis.

As expected, the speed of propagation as a function of the activation energy shows the same trends for this solution as for Britten and Krantz's (1985) solution. Figure 3.1 shows the group $1/A$ as a function of $1/T_{-\infty}$, for constant values of the group $(AT_{-\infty})/Q$. For constant physical properties, $1/A$ is an increasing linear function of the propagation velocity when the pre-exponential factor is constant; $1/T_{-\infty}$ is proportional to the activation energy; and $(AT_{-\infty})/Q$ is proportional to the pre-exponential factor. This figure shows that for the planar RC model, the propagation velocity decreases monotonically with increasing activation energy.

The solution also shows the expected behavior with respect to varying the pre-exponential factor for the planar RC model: as the pre-exponential factor is increased, the propagation velocity also increases. Figure 3.2 shows $1/Q$ as a function of $(AT_{-\infty})/Q$ for constant values of $1/T_{-\infty}$. For constant values of the activation energy, $1/Q$ is an increasing linear function of the propagation velocity.

Figure 3.3 shows the variation of β with respect to the same parameters. The parameter β is proportional to the dimensionless peak reaction rate. It can also be considered to be inversely proportional to the thickness of the reaction zone in relationship to the thickness of the preheat zone.

In order to investigate the propensity of RC to form channels, it is necessary to conduct a stability analysis. To identify the final form of the channels requires a three-dimensional nonlinear stability analysis. This task would surely require a large commitment in time, and would push the capabilities of modern supercomputers. Instead of pursuing this task, we have adopted the expedient of looking only at the linear stability of a solution, on the assumption that this will give good correlating parameters for the overall stability of the model. Furthermore, due to time constraints, we will assume that since the trial function solution exhibits the same behavior as the planar high-activation-energy asymptotic solution, it is reasonable to assume that it will exhibit the same stability behavior. Thus, increasing the activation energy should produce larger channels that propagate more slowly, whereas increasing the pre-exponential factor should yield smaller channels that propagate faster.

Thus, the effect of using combustion promoters or inhibitors has been determined. The final tasks are to identify practical materials that might be effective promoters or inhibitors, and last of all, when they should be used. It is desirable that these materials be transported

Figure 3.1: Front velocity as a function of activation energy for constant values of pre-exponential factor.

Figure 3.2: Front velocity as a function of pre-exponential factor for constant values of activation energy.

Figure 3.3: Trial function parameter, β as function of the independent parameters describing reverse combustion.

to the combustion front in gaseous form in the injected gas. This lessens the likelihood that they will separate out and never reach the front.

Combustion promoters work by increasing the production of free radicals. They increase the effective pre-exponential factor and/or decrease the effective activation energy. Thus, they would result in smaller channels that propagate faster. They would be useful especially at shallow depths, where sufficient permeability enhancement occurs, for which it is desirable to make the channels propagate faster to reduce the costs of RC. The straight-chain saturated hydrocarbons, propane through normal hexane, are a potential family of promoters, since they are relatively inexpensive. In general, as the number of carbon atoms increases in this series, the material becomes less thermally stable, thus becoming more effective as a combustion promoter. However, as the chain length increases, the vapor pressure also decreases. This in turn, decreases the amount of promoter that can be carried by the injected gas at a given temperature and pressure. It is not necessary that the promoter be a pure material; in fact, a mixture of butane and pentane might be most suitable.

Combustion inhibitors generally work by chain termination through eliminating free radicals. They increase the effective activation energy and/or decrease the effective pre-exponential factor. Thus, their use would result in larger channels that propagate more slowly. Combustion inhibitors would be useful when insufficient permeability enhancement occurs owing to smaller channel size. A likely candidate inhibitor is the use of suitable amounts of halon fire suppressants.

3.5 Summary

The generalized solution developed in this chapter supports the hypothesis that combustion inhibitors or promoters could be effective tools in optimizing reverse combustion. Inhibitors should result in larger channels that propagate slower, whereas promoters should result in smaller channels that propagate faster. Suitable mixtures of normal pentane and butane should make effective, inexpensive combustion promoters. Halon gases should make effective combustion inhibitors. These ideas should be initially tested in laboratory scale reverse combustion tests.

CHAPTER IV

A ONE-DIMENSIONAL MODEL FOR THE PROPAGATION OF A CURVED REVERSE COMBUSTION FRONT

4.1 Scope

In this chapter, a one-dimensional continuum model for a weakly curved reverse combustion front is formulated then solved using high-activation-energy asymptotics. Here, a front is classified as being weakly curved if the explicit effects of the curvature of the combustion front can be ignored, but not the nonuniformity of the flow of gas through the combustion front. The nature of the solution is illustrated and discussed. Its application to data analysis of RC experiments is outlined. Its application as a front-propagation relationship for use in a RC channel growth is outlined. The chapter ends with a summary.

4.2 Motivation

This model is intended to serve two complementary purposes. First, it will aid in the analysis of RC channeling experiments in porous media, when coupled with calculations for the gas-flux to the experimentally observed combustion front. This would allow a spatially distributed analysis of the combustion front, which could lead to better understanding of RC channeling. Second, it will provide a combustion model for a transient numerical simulation of RC channel propagation. Since it considers the effect of tangential heat transport along the combustion front on the propagation velocity, this model provides better insight into RC than those used in previous studies. Both tangential heat conduction and convection are incorporated into the framework of existing models by a boundary-layer approach. Unlike previous planar models, the superabundance of fuel is not assumed, since this is not assured at all points on a fully developed RC finger. Along with heat loss, this provides a mechanism for extinction of the combustion front. This model is a natural extension of Britten's analysis of planar RC fronts, however, the nomenclature is different. This model differs from Britten's nonadiabatic model in that it considers heat transport along the combustion front, whereas Britten's model considers heat loss to the surroundings which are outside the porous media. Consequently, this analysis must consider heat gain as well as heat loss, whereas Britten's model could presume heat loss.

4.3 Nature of the Flow Field to a Reverse Combustion Finger

Previous RC front propagation models considered an idealized gas flow field for the oxidant gas to, and the product gas from the combustion front. The streamlines were parallel everywhere, and perpendicular to the combustion front. Even when considering a radially expanding front, the flow field was reduced to a time-dependent parallel flow through reasonable approximations. Here we wish to consider the effect of a more complex flow field on RC front propagation, without resorting to a multi-dimensional model. The basic effect of a more complex flow field is to introduce additional modes of heat transport that do not arise with planar or radial flow configurations. Our goal is include a reasonable approximation to these heat transfer modes in the framework of existing, purely one-dimensional models.

Before we start the model formulation, let us consider the characteristics of the actual flow field around a combustion finger. First, consider the permeability characteristics of a RC finger. As shown in Figure 4.1, inside the combustion front is the char zone which has a high permeability. Outside and away from the combustion front is the unaltered coal which has a low permeability. Separating the combustion front from the unaltered coal is a devolatilization zone in which the permeability smoothly changes and the gas flux is augmented by the release of combustible volatiles. The thickness of the devolatilization zone varies along the combustion front. It is thinnest at the tip of the finger.

Now consider the gas flow to this finger as illustrated in Figure 4.2. This figure qualitatively shows isobars and streamlines for the flow field. Far outside the front, the paths of the streamlines are governed by the overall flow configuration. Inside the channel, the streamlines are basically aligned with the axis of the finger. Near the tip, the streamlines pass through the devolatilization zone nearly perpendicular to the combustion front, without significantly changing direction. The tangential component (with respect to the combustion front) of the gas flux at the front is small near the tip whereas the normal component of the gas flux is large. Moving away from the tip, the streamlines enter the devolatilization zone with a small but increasing deviation from perpendicularity with the combustion front. In passing through the devolatilization zone, the streamlines exhibit a large change in direction. Far from the tip of the finger, the tangential component of the gas flux at the front is large, while the normal component of the gas flux is small.

To illustrate several points let us consider the behavior of the flow field in the limit of the devolatilization zone thickness becoming zero. In this limit, the release of volatile matter must cause a jump change in the normal component of the gas flux equal to the flux of volatile matter released, since no mass can be transported tangentially through a zone of zero thickness. In addition, the pressure gradient along the combustion front must be equal to the pressure gradient at the coal surface, since a zone of zero thickness would not present any resistance to flow. The pressure gradient at any point is governed by the total flow field; it is not directly a function of the local devolatilization. The tangential component of the gas flux is related to the pressure gradient by Darcy's law. Hence, the tangential component of the gas flux undergoes a jump change across the devolatilization zone proportional to the change in permeability, but not directly related to the flux of volatile matter released. For a

Figure 4.1: Permeability zones for a reverse combustion finger.

Figure 4.2: Flow field around a reverse combustion finger.

devolatilization zone of small but finite thickness, with a continuous variation in permeability and in the extent of devolatilization, the release of volatile matter will predominantly affect the normal component of the gas flux, whereas the change in permeability will predominantly affect the tangential components of the the gas flux.

4.4 Formulation

Attention is focused on a line passing perpendicular to the combustion front. This establishes a local, temporary frame of reference oriented with the combustion front, as illustrated in Figure 4.3. Only one tangential coordinate is shown in this figure. This frame of reference is fixed with respect to the solid phase. On this frame of reference, a local coordinate system is established. The temperature varies along the front, due to the nonuniformity of the gas flux to the combustion front. This tangential temperature variation introduces tangential modes of heat transport which we desire to incorporate into a simple model of RC front propagation. Rather than considering the full three-dimensional gas flow field, the tangential and normal components of the gas flux all along the line are approximated by their values at the combustion front, as illustrated in Figure 4.4. Only one tangential component is shown in this figure. This approximation is quite good inside of the RC channel, since the streamlines are nearly parallel. It degrades in passing outward through the devolatilization zone to the unaltered coal. However, the temperature decreases towards the ambient value on approximately this length scale. Away from the combustion front, where the gas flux assumption is weakest, the amount of tangential heat transport is negligible. Thus, this assumption leads to a reasonable model of RC.

Any necessary physical properties are taken to be given, suitably averaged, constant values. We ignore simultaneous combustion and devolatilization, but instead treat them as separate steps. The devolatilization is not explicitly considered; its only effect is to determine the composition of the gas at the start of combustion. The gas is treated as three components: oxygen, fuel, and inerts. The component "inerts" includes both noncombustible molecular species in the injected gas (typically nitrogen), as well as products of combustion such as carbon dioxide and water vapor. The combustion reaction has the following stoichiometry:

where the stoichiometric coefficient for fuel, β , is approximately one-half (Kotowski and Gunn, 1976). No serious error is introduced by assuming the mass fraction of a component is numerically equal to its mole fraction, since the three components have approximately the same average molecular weight.

We start with the model equations as given in Britten (1984). The energy and limiting reactant species balances on a mass basis are, respectively,

$$C_s \rho_s \frac{\partial T^*}{\partial t^*} + C_g N_g \cdot \nabla^* T^* = \lambda_s \nabla^{*2} T^* + q \varphi r^* \quad (4.2)$$

$$\varphi \rho_g \frac{\partial Y_l^*}{\partial t^*} + N_g \cdot \nabla^* Y_l^* = D_{eff} \rho_g \nabla^{*2} Y_l^* - \nu_l \varphi r^* \quad (4.3)$$

Figure 4.3: Local frame of reference and coordinate system at a point on a reverse combustion channel

←---
ACTUAL FLUX
VECTOR

←
APPROXIMATION
FOR FLUX VECTOR

Figure 4.4: Local approximation for the gas flow field along a line passing through the combustion front.

The reaction rate we shall use is

$$r^* = AP^2 e^{-E/(RT^*)} Y_l^* Y_{nl}^* \quad (4.4)$$

All properties are on a mass basis. The notation used is the universal gas constant R , the mass fraction of the nonlimiting reactant Y_{nl}^* , the heat capacity of the solid phase C_s , the heat capacity of the gas phase C_g , the density of the solid phase ρ_s , the density of the gas phase ρ_g , the absolute temperature T^* , the time t^* , the mass flux of gas N_g , the thermal conductivity of the solid phase λ_s , the heat of combustion q , the porosity φ , the reaction rate r^* , the mass fraction of the limiting reactant Y_l^* , the effective mass diffusivity D_{eff} , the stoichiometric coefficient of the limiting reactant ν_l , the Arrhenius pre-exponential factor A , the Arrhenius activation energy E , the absolute pressure P , the dimensional gradient ∇^* , and dimensional laplacian ∇^{*2} , in terms of the principal spatial coordinate x_1 , and the tangential coordinates x_2, x_3 . Thus, it is assumed that the overall kinetics are first-order in both the oxygen and the fuel. The reaction is modeled as a homogeneous gas-phase reaction. The overall kinetics are a function of the composition of the gas in the combustion zone. Thus, the kinetic parameters may be influenced by addition of combustion inhibitors or promoters into the injected gas. This could provide an additional controllable parameter for RC, as discussed in Chapter 3. This reaction rate is linearized with respect to the extent of reaction by approximating the mass fraction of the nonlimiting reactant with its value at the onset of combustion, $Y_{nl,f}^*$. The subscript f denotes the conditions at the front prior to combustion. This linearization simplifies the solution through matched asymptotic expansions, since only one species balance must be solved.

In one major aspect, the reaction rate differs from that used by Britten. In that model, the nonlimiting reactant (which was always assumed to be fuel) was presumed to be present in sufficiently large excess that its mass fraction was always unity. This assumption is not uniformly valid with respect to variations in the oxygen content of the injected gas. In contrast, this model determines the mass fractions of both fuel and oxygen at the onset of combustion. These mass fractions are affected by the extent of devolatilization as well as the composition of the injected gas. The nonlimiting reactant is determined considering the stoichiometric coefficients of the reactants. Even though the dependence of the mass fraction of the nonlimiting reactant with the extent of reaction is ignored, its overall dependence on the composition of the injected gas is considered. A less important difference is that the reaction rate is not expressed in terms of partial densities, which is equivalent to using the partial pressures and assuming that the pre-exponential factor has reciprocal temperature-squared dependence. The quality of the data from which reverse combustion kinetic parameters are extracted is not adequate to allow identification of the algebraic temperature-dependence of the pre-exponential factor.

The gas flux at the combustion front is given by

$$N_{g,f} = N_{g,a} + V_f \rho_{s,a} v_m f_{vmr}(T_f^*, P) \quad (4.5)$$

where $N_{g,f}$ is the mass gas flux at the combustion front, $N_{g,a}$ is the mass gas flux prior to devolatilization, V_f is the propagation speed of the front, $\rho_{s,a}$ is the coal density prior to devolatilization, v_m is the weight percent volatile matter in the coal, f_{vmr} is the fraction

of volatile matter released as a function of temperature, and T_f^* is the combustion front temperature. The composition at the front is given by

$$Y_{O_x,f}^* = \frac{N_{g,a}}{N_{g,f}} Y_{O_x,a}^* \quad (4.6)$$

$$Y_{I_n,f}^* = \frac{N_{g,a}}{N_{g,f}} Y_{I_n,a}^* \quad (4.7)$$

$$Y_{F_u,f}^* = 1 - Y_{O_x,f}^* - Y_{I_n,f}^* \quad (4.8)$$

where $Y_{O_x}^*$ is the oxygen mass fraction, $Y_{I_n}^*$ is the inerts mass fraction, and $Y_{F_u}^*$ is the fuel mass fraction, with the subscript f referring to conditions at the front before combustion, and the subscript a referring to conditions prior to devolatilization.

Scaling these equations allows selection of the following dimensionless variables.

$$T = \frac{T^*}{T_{amb}^*} \quad (4.9)$$

$$Y = \frac{Y_l^*}{Y_{l,f}^*} \quad (4.10)$$

$$x = \frac{x_1^*}{\lambda_s / C_g N_{g,x_1,f}} \quad (4.11)$$

$$t = \frac{t^*}{C_s \rho_s \lambda_s / C_g^2 N_{g,x_1,f}^2} \quad (4.12)$$

The notation is the dimensionless temperature T , the dimensionless concentration of the limiting reactant Y , the dimensionless spatial coordinate in the principal direction x , the dimensionless time t , the absolute ambient temperature far ahead of the front T_{amb}^* , the mass fraction of the limiting reactant at the front before combustion $Y_{l,f}^*$, the normal component of the gas flux at the front $N_{g,x_1,f}$. For later use, the tangential components of the gas flux are then $N_{g,x_2,f}$, and $N_{g,x_3,f}$.

The normal component of gas flux before devolatilization is assumed to be an externally specified parameter, as are the tangential components of the gas flux at the front. The normal component is modified via Equation 4.5 to account for the effect of devolatilization. This secondary effect should not be ignored when considering a range of oxygen contents for the injected gas. To do so would be inconsistent with the typical (although not universal) excess of fuel.

The characteristic dimensionless parameters for RC are

$$\mathcal{E} \equiv \frac{E}{RT_{amb}^*} \quad (4.13)$$

$$\mathcal{A} \equiv \frac{\nu_l \lambda_s \varphi A P^2 Y_{nl}}{C_g N_{g,x_1,f}^2} \quad (4.14)$$

$$\mathcal{Q} \equiv \frac{q Y_{l,f}^*}{C_g T_{amb}^* \nu_l} \quad (4.15)$$

$$\mathcal{B} \equiv \frac{\varphi \rho_s C_g}{\rho_s C_s} \quad (4.16)$$

$$\mathcal{L} \equiv \frac{\lambda_s}{D_{eff}\rho_g C_g} \quad (4.17)$$

$$(4.18)$$

The dimensionless parameters are $\mathcal{E}$, a dimensionless activation energy; $\mathcal{A}$, a dimensionless pre-exponential factor; $\mathcal{Q}$, a dimensionless heat of reaction; $\mathcal{B}$, a ratio of the specific heat of the gas phase to that of the solid; and $\mathcal{L}$, a Lewis number, which is a reciprocal dimensionless diffusion coefficient.

In terms of these dimensionless variables and dimensionless groups, the energy and material balances simplify to

$$\frac{\partial T}{\partial t} + \frac{\partial T}{\partial x} = \frac{\partial^2 T}{\partial x^2} + \mathcal{S} + \mathcal{Q}\mathcal{A}e^{-\mathcal{E}/T}Y \quad (4.19)$$

$$\mathcal{L}\frac{\partial Y}{\partial t} + \frac{\partial Y}{\partial x} = \mathcal{L}^{-1}\frac{\partial^2 Y}{\partial x^2} - \mathcal{A}e^{-\mathcal{E}/T}Y \quad (4.20)$$

The tangential heat transport by conduction and convection has been collected into the term $\mathcal{S}$, where

$$\begin{aligned} \mathcal{S} = & \left(\left(\frac{\lambda_s}{C_g N_{g,x1,f}} \right)^2 \frac{1}{\lambda_s T_{amb}^*} \right) \\ & * \left[\lambda_s \left(\frac{\partial^2 T^*}{\partial x_2^{*2}} + \frac{\partial^2 T^*}{\partial x_3^{*2}} \right) - C_g \left(N_{g,x2} \frac{\partial T^*}{\partial x_2^*} + N_{g,x3} \frac{\partial T^*}{\partial x_3^*} \right) \right] \end{aligned} \quad (4.21)$$

which from a one-dimensional perspective is a source or sink term arising from tangential heat transport. This term will not be treated exactly. To do so for arbitrary geometry would require direct numerical simulation of the governing equations. Instead, this term will be treated in a phenomenological manner, as will be detailed later.

The presentation of the appropriate boundary conditions for these equations will be postponed pending transformation of these equations into a moving coordinate system and invoking a quasi-steady-state-approximation.

The mapping to a new coordinate system is

$$(x, t) \longmapsto (\eta, \tau) \quad (4.22)$$

where

$$\eta \equiv x + Vt, \quad \tau \equiv t \quad (4.23)$$

Here η is a dimensionless spatial variable centered about the combustion front which moves at a quasi-steady, unknown velocity V , and τ is a dimensionless temporal variable introduced to clearly differentiate between partial derivatives while holding η fixed ($\partial/\partial\tau$), from partial derivatives while holding x fixed ($\partial/\partial t$). This coordinate system is shown in Figure 4.5, along with the temperature and limiting reactant profiles. The resulting energy and mass

Figure 4.5: Schematic of temperature and limiting reactant profiles.

balances in the moving coordinate system are then

$$\frac{\partial T}{\partial \tau} + (1 + V) \frac{\partial T}{\partial \eta} = \frac{\partial^2 T}{\partial \eta^2} + S + Q \mathcal{A} e^{-\mathcal{E}/T} Y \quad (4.24)$$

$$\mathcal{B} \frac{\partial Y}{\partial \tau} + (1 + \mathcal{B}V) \frac{\partial Y}{\partial \eta} = \mathcal{L}^{-1} \frac{\partial^2 Y}{\partial x^2} - \mathcal{A} e^{-\mathcal{E}/T} Y \quad (4.25)$$

In this new coordinate system we seek solutions for which the dependent variable profiles in the moving coordinate system are unchanging in time. Furthermore, we will invoke a quasi-steady-state-approximation (QSSA): That at any instant in time, the profiles and the resulting front velocity are reasonably approximated by the steady-state profiles and front velocity determined using the values of the dimensionless parameters for RC $\{\mathcal{E}, \mathcal{A}, Q, \mathcal{B}, \text{ and } \mathcal{L}\}$ evaluated at that time. This approximation restricts the applicability of this model to problems where the operating parameters vary slowly or not at all with time.

With the QSSA, and from order of magnitude scaling, which indicates that terms multiplied by either $\mathcal{B}$ or $\mathcal{L}^{-1}$ should be negligible and can be dropped, the heat and material balances are

$$(1 + V) \frac{\partial T}{\partial \eta} = \frac{\partial^2 T}{\partial \eta^2} + S + Q \mathcal{A} e^{-\mathcal{E}/T} Y \quad (4.26)$$

$$\frac{\partial Y}{\partial \eta} = -\mathcal{A} e^{-\mathcal{E}/T} Y \quad (4.27)$$

The appropriate boundary conditions are

$$\begin{aligned} T &\asymp 1 \quad \text{as } \eta \rightarrow -\infty \\ \frac{\partial T}{\partial \eta} &\asymp 0 \quad \text{as } \eta \rightarrow \infty \\ Y &\asymp 1 \quad \text{as } \eta \rightarrow -\infty \\ Y &\asymp 0 \quad \text{as } \eta \rightarrow \infty \end{aligned} \quad (4.28)$$

where the relational symbol $\asymp$ means "is asymptotic to." The first boundary condition is a statement that the front the temperature approaches ambient ahead of the front. The second boundary condition is a statement that the temperature approaches a constant value behind the front. The third boundary condition is a modeling approximation. Far ahead of the combustion front, the limiting reactant does approach a fixed value, either the injected oxygen or fuel concentration (usually zero). This concentration does not directly influence the combustion process. In the preheat zone, the concentration changes because of the release of volatiles. However, we model the process as if the volatiles are completely released ahead of the combustion front and ignore the variation in the preheat zone. In fact, the combustion zone is not in general distinct from the devolatilization zone. Still the details of devolatilization do not exert a major influence on reverse combustion and this level of approximation is consistent with that of the whole model. The fourth boundary condition is also a modeling approximation; complete combustion of the limiting reactant is assumed. Thus, this model is not appropriate for studying the details of extinction.

Tangential heat transport occurs because of the variability of temperature along the combustion front. This nonuniformity of temperature is caused by a variability of the gas flux along the combustion front. From previous one-dimensional models, the front velocity is known to vary markedly with the normal component of the gas flux, whereas the front temperature is a smooth monotonic function of the normal gas flux. Furthermore, Britten's (1984) nonadiabatic model suggests that the front temperature is not strongly affected by heat losses. Hence, for determining the magnitude of tangential heat transport, the variation of the front temperature will be considered a linear function of the normal gas flux, and the variation of the front velocity will be calculated.

Tangential conduction, a smoothing process, causes tangential heat transport to die off away from the front. The decline of tangential heat transport is approximated by an exponential decay, both ahead and behind the front. The length scale in which tangential heat transport is important ahead of the combustion front is nearly the same as the length scale of the preheat zone. The length scale in which tangential heat transport is important behind the combustion front is determined by the macroscopic distribution of heat in the reverse combustion channel. This length scale is not easily estimated. The tangential heat transport source or sink term is thus approximated by

$$S = S_f f = [S_d + S_v]f \quad (4.29)$$

where S_d is a strength of tangential conduction and S_v is a strength of tangential convection, as given by

$$S_d = \frac{\partial T_f}{\partial N_{g,x_1,f}} \left(\frac{\lambda_s}{C_g N_{g,x_1,f}} \right)^2 \left(\frac{\partial^2 N_{g,x_1,f}}{\partial x_2^{*2}} + \frac{\partial^2 N_{g,x_1,f}}{\partial x_3^{*2}} \right) \quad (4.30)$$

$$S_v = \frac{\partial T_f}{\partial N_{g,x_1,f}} \left(\frac{\lambda_s}{C_g N_{g,x_1,f}} \right)^2 \left(N_{g,x_2,f} \frac{\partial N_{g,x_1,f}}{\partial x_2^*} + N_{g,x_3,f} \frac{\partial N_{g,x_1,f}}{\partial x_3^*} \right) \frac{C_g}{\lambda} \quad (4.31)$$

The term S_f is a strength of tangential heat transport at the front. It varies continuously along the combustion surface, and thus accounts for the leading order tangential variation in the tangential heat transport. The η variation of the tangential heat transport is given by

$$f = \begin{cases} e^{G_\Theta \eta}, & \eta \leq 0 \\ e^{-G_\Theta \eta}, & \eta \geq 0 \end{cases} \quad (4.32)$$

$$(4.33)$$

4.5 Solution

In this section, an approximate solution for the curved front model formulated in the previous section is obtained by using the method of High-Activation-Energy Asymptotics, as employed by Britten (1984) to solve his nonadiabatic model. The most important difference

between the two solutions is that both heat loss and gain must be considered for the curved front model, whereas Britten considered only heat loss.

4.5.1 Preliminaries

The following variable transformations are employed to aid in the solution by high activation energy asymptotics:

$$T = T\alpha \quad (4.34)$$

$$\zeta = (1 + V)\eta \quad (4.35)$$

$$\delta = \frac{1}{\mathcal{E}} \quad (4.36)$$

The scale factor α is defined as the ratio of the ambient temperature to the combustion front temperature. The combustion front temperature is that at which the reaction rate reaches a maximum with respect to the spatial coordinate η . The reader should note that in Britten's work α is defined as the ratio of the ambient temperature to the maximum temperature. Although Britten refers to the maximum temperature as the combustion front temperature, the combustion front temperature is clearly not the maximum temperature if an arbitrary heat source term is considered. The rescaled problem is thus:

$$\frac{\partial T}{\partial \zeta} = \frac{\partial^2 T}{\partial \zeta^2} + \frac{\alpha S_f}{(1 + V)^2} f + \frac{\alpha A Q}{(1 + V)^2} e^{-\alpha/\delta T} Y \quad (4.37)$$

$$\frac{\partial Y}{\partial \zeta} = -\frac{A}{1 + V} e^{-\alpha/\delta T} Y \quad (4.38)$$

$$T \asymp \alpha, \quad \zeta \rightarrow -\infty \quad (4.39)$$

$$\frac{\partial T}{\partial \zeta} \asymp 0, \quad \zeta \rightarrow \infty \quad (4.40)$$

$$Y \asymp 1, \quad \zeta \rightarrow -\infty \quad (4.41)$$

$$Y \asymp 0, \quad \zeta \rightarrow \infty \quad (4.42)$$

$$f = \begin{cases} e^{G_{\Theta}\zeta/(1+V)}, & \zeta \leq 0 \\ e^{-G_{\Theta}\zeta/(1+V)}, & \zeta \geq 0 \end{cases} \quad (4.43)$$

4.5.2 Variable and Parameter Expansions

The expansions presented here are the same as those used by Britten (1984) for his adiabatic model. By considering α as $O(\delta^{1/2})$, the solution for the front temperature is simplified, although the accuracy of the front speed relationship suffers.

The outer zone expansions of the dependent variables are:

$$T = T_0 + \delta^{1/2} T_1 + O(\delta) \quad (4.44)$$

$$Y = Y_0 + \delta^{1/2}Y_1 + O(\delta). \quad (4.45)$$

In the inner zone, the independent variable is stretched:

$$\xi = \frac{\zeta}{\delta^{1/2}} \quad (4.46)$$

The inner zone expansions of the dependent variables are:

$$\mathcal{T} = 1 + \delta^{1/2}\theta_1(\xi) + O(\delta) \quad (4.47)$$

$$Y = y_0 + \delta^{1/2}y_1(\xi) + O(\delta) \quad (4.48)$$

We take α as the eigenvalue and expand it also:

$$\alpha = \delta^{1/2}\alpha_1 + O(\delta) \quad (4.49)$$

In addition, the unknown constant V (which is dependent on α), is expanded in a general asymptotic series:

$$V = V_0(\delta^{1/2}) + V_1(\delta^{1/2}) + \dots \quad \text{with} \quad \lim_{\delta^{1/2} \rightarrow 0} \frac{V_{i+1}}{V_i} = 0 \quad (4.50)$$

4.5.3 Outer Zone Solutions

The expansions for the outer zones are substituted into Equations 4.37 and 4.38. The leading order equations for the outer zone are:

$$\frac{\partial \mathcal{T}_0}{\partial \zeta} = \frac{\partial^2 \mathcal{T}_0}{\partial \zeta^2} + \mathcal{S}'_{f0} f \quad (4.51)$$

$$\frac{\partial Y_0}{\partial \zeta} = 0 \quad (4.52)$$

where

$$\mathcal{S}'_{f0} = \frac{\alpha_1 \delta^{1/2} \mathcal{S}_f}{(1 + V_0)^2}, \quad (4.53)$$

assuming $\mathcal{S}'_{f0} = O(1)$. The solution for the outer dependent variable profiles are obtained by integrating these equations, applying the appropriate asymptotic forms of the boundary conditions, and using the zeroth-order matching relationship: $\mathcal{T}_0^\pm = 1$. In expressing the matching conditions, the following nomenclature is used: $A_+ = \lim_{t \rightarrow 0} A(\eta)$; $\eta = |t|$. Similarly, $A_- = \lim_{t \rightarrow 0} A(\eta)$; $\eta = -|t|$. They are

$$\mathcal{T}_{\Theta 0} = e^\zeta + \frac{\mathcal{S}'_{f0}}{H_\Theta(H_\Theta - 1)} [e^\zeta - e^{H_\Theta \zeta}], \quad \zeta \leq 0 \quad (4.54)$$

$$\mathcal{T}_{\Theta 0} = 1 + \frac{\mathcal{S}'_{f0}}{H_\Theta(H_\Theta + 1)} [1 - e^{-H_\Theta \zeta}], \quad \zeta \geq 0 \quad (4.55)$$

$$Y_{\ominus 0} = 1, \quad \zeta \leq 0 \quad (4.56)$$

$$Y_{\oplus 0} = 0, \quad \zeta \geq 0 \quad (4.57)$$

where

$$H_{\ominus} = \frac{G_{\ominus}}{1 + V_0} \quad (4.58)$$

$$H_{\oplus} = \frac{G_{\oplus}}{1 + V_0} \quad (4.59)$$

4.5.4 Inner Zone Solution

The essential aspect of the solution of the inner reactive-zone is identification of a leading-order eigenvalue relationship between the combustion front temperature and the dimensionless parameters of the problem. The closed-form solution of the reduced reactive equations, which is not possible for arbitrary heat transport, is not necessary for this purpose. It is sufficient that the equations be integrated to a form that allows asymptotic matching, provided that an auxilliary equation can be generated from the known characteristics of the actual solution of the reduced equations. This allows the eigenvalue relationship to be uniquely determined.

First, let us substitute the inner zone expansions in the exponent of the Arrhenius kinetic expression, then expand in a Taylor's series about $\mathcal{T} = 1$. To leading order, this gives:

$$-\alpha/\delta\mathcal{T} = \frac{-\delta^{1/2}\alpha_1 + \dots}{\delta(1 + \delta^{1/2}\theta_1 + \dots)} \approx -\frac{\alpha_1}{\delta^{1/2}} + \alpha_1\theta_1 \quad (4.60)$$

The inner zone expansions and this linearization are substituted into the full set of equations. The leading-order inner zone equations are:

$$\frac{\partial y_0}{\partial \xi} = -\mathcal{D}_0 e^{\alpha_1 \theta_1} y_0 \quad (4.61)$$

$$\frac{\partial^2 \theta_1}{\partial \xi^2} = -\mathcal{Q}'_0 \mathcal{D}_0 e^{\alpha_1 \theta_1} y_0 \quad (4.62)$$

where $\mathcal{D}_0$ is a Damkohler number, and $\mathcal{Q}'_0$ is a dimensionless heat of reaction given by:

$$\mathcal{D}_0 = \frac{\mathcal{A}\delta^{1/2}e^{-\alpha_1/\delta^{1/2}}}{1 + V_0} \quad (4.63)$$

$$\mathcal{Q}'_0 = \frac{\delta^{1/2}\alpha_1 \mathcal{Q}}{1 + V_0} \quad (4.64)$$

Equation 4.61 and 4.62 are combined to eliminate the nonlinear Arrhenius temperature dependence, generating the following linear equation:

$$\frac{\partial^2 \theta_1}{\partial \xi^2} = \mathcal{Q}'_0 \frac{\partial y_0}{\partial \xi} \quad (4.65)$$

Equation 4.65 can be integrated once. Evaluating the resulting expression at $\xi = +\infty$ and using the matching relationship $\lim_{\xi \rightarrow +\infty} y_0 = Y_0^+$ yields:

$$\frac{\partial \theta_1}{\partial \xi} - \gamma_{\oplus} = Q'_0 y_0 \quad (4.66)$$

where $\gamma_{\oplus} = \frac{\partial \theta_1}{\partial \xi} \Big|_{\xi \rightarrow +\infty}$. Evaluating this expression at $\xi = -\infty$ and using the matching relationship $\lim_{\xi \rightarrow -\infty} y_0 = Y_0^-$ yields:

$$\gamma_{\ominus} - \gamma_{\oplus} = Q'_0 \quad (4.67)$$

where $\gamma_{\ominus} = \frac{\partial \theta_1}{\partial \xi} \Big|_{\xi \rightarrow -\infty}$. It is possible to obtain an equation completely in terms of the inner zone temperature θ_1 by substituting Equation 4.66 into Equation 4.61:

$$\frac{\partial^2 \theta_1}{\partial \xi^2} = -D_0 \left(\frac{\partial \theta_1}{\partial \xi} - \gamma_{\oplus} \right) e^{\alpha_1 \theta_1} \quad (4.68)$$

In the exceptional case of $\gamma_{\oplus} = 0$, Equation 4.68 can be easily integrate twice to yield an explicit, nonlinear solution for θ_1 , which can be matched in the conventional manner with the outer solution to generate the desired eigenvalue relationship. For $\gamma_{\oplus} \neq 0$, it can only be integrated once, yielding an intractable equation in terms of θ_1 and $\frac{\partial \theta_1}{\partial \xi}$. Matching the first-order differential equation with the outer solution does not completely determine the eigenvalue problem. Consequently, an auxilliary condition must be generated that provides additional information about the solution for Equation 4.68. To do so, we recall that the combustion front temperature is that at which the reaction rate reaches a maximum. Equation 4.62 shows that the reaction rate is equal to $-\frac{\partial^2 \theta_1}{\partial \xi^2}$. In addition, examination of the inner zone expansion Equation 4.47 shows that at the combustion front, $\theta_1 = 0$. Consequently,

$$-\frac{\partial^3 \theta_1}{\partial \xi^3} \Big|_{\theta_1=0} = 0 = D_0 \left[\frac{\partial^2 \theta_1}{\partial \xi^2} e^{\alpha_1 \theta_1} + \alpha_1 \left(\frac{\partial \theta_1}{\partial \xi} - \gamma_{\oplus} \right) \frac{\partial \theta_1}{\partial \xi} e^{\alpha_1 \theta_1} \right]_{\theta_1=0} \quad (4.69)$$

where Equation 4.68 has been used to evaluate the third derivative. Dividing by D_0 and using Equation 4.68 to eliminate the second derivative yields:

$$0 = \left[-D_0 \left(\frac{\partial \theta_1}{\partial \xi} - \gamma_{\oplus} \right) e^{2\alpha_1 \theta_1} + \alpha_1 \left(\frac{\partial \theta_1}{\partial \xi} - \gamma_{\oplus} \right) \frac{\partial \theta_1}{\partial \xi} e^{\alpha_1 \theta_1} \right]_{\theta_1=0} \quad (4.70)$$

which evaluates to

$$\frac{\partial \theta_1}{\partial \xi} \Big|_{\theta_1=0} = \frac{D_0}{\alpha_1} \quad (4.71)$$

Next, Equation 4.68 can be integrated once by using the substitution $z = \frac{\partial \theta_1}{\partial \xi} - \gamma_{\oplus}$. Application of the inverse substitution produces:

$$\frac{\partial \theta_1}{\partial \xi} - \gamma_{\oplus} + \gamma_{\oplus} \ln \left(\frac{\partial \theta_1}{\partial \xi} - \gamma_{\oplus} \right) = -\frac{D_0}{\alpha_1} e^{\alpha_1 \theta_1} + c_1 \quad (4.72)$$

The integration constant is obtained by evaluating this expression for $\theta_1 = 0$, yielding

$$\left. \frac{\partial \theta_1}{\partial \xi} \right|_{\theta_1=0} - \gamma_\Theta + \gamma_\Theta \ln \left(\left. \frac{\partial \theta_1}{\partial \xi} \right|_{\theta_1=0} - \gamma_\Theta \right) + \frac{\mathcal{D}_0}{\alpha_1} = c_1, \quad (4.73)$$

then using Equation 4.71 to eliminate the derivative:

$$2 \frac{\mathcal{D}_0}{\alpha_1} - \gamma_\Theta + \gamma_\Theta \ln \left(\frac{\mathcal{D}_0}{\alpha_1} - \gamma_\Theta \right) = c_1. \quad (4.74)$$

This result is inserted into Equation 4.72:

$$\frac{\partial \theta_1}{\partial \xi} + \gamma_\Theta \ln \left(\frac{\partial \theta_1}{\partial \xi} - \gamma_\Theta \right) = -\frac{\mathcal{D}_0}{\alpha_1} e^{\alpha_1 \theta_1} + 2 \frac{\mathcal{D}_0}{\alpha_1} + \gamma_\Theta \ln \left(\frac{\mathcal{D}_0}{\alpha_1} - \gamma_\Theta \right). \quad (4.75)$$

By evaluating Equation 4.75 at $\xi \rightarrow -\infty$, the eigenvalue relationship can be compactly expressed:

$$\gamma_\Theta + \gamma_\Theta \ln (\gamma_\Theta - \gamma_\Theta) = 2 \frac{\mathcal{D}_0}{\alpha_1} + \gamma_\Theta \ln \left(\frac{\mathcal{D}_0}{\alpha_1} - \gamma_\Theta \right) \quad (4.76)$$

with γ_Θ and γ_Θ determined by applying the degenerate matching conditions $\lim_{\xi \rightarrow \pm\infty} \frac{\partial \theta_1}{\partial \xi} = \frac{\partial T_0}{\partial \xi}^\pm$. They are:

$$\gamma_\Theta = 1 - \frac{S'_{f0}}{H_\Theta} \quad (4.77)$$

$$\gamma_\Theta = \frac{S'_{f0}}{1 + H_\Theta} \quad (4.78)$$

Substituting for γ_Θ and γ_Θ in Equation 4.76 yields:

$$1 - \frac{S'_{f0}}{H_\Theta} + \frac{S'_{f0}}{1 + H_\Theta} \ln \left(1 - \frac{S'_{f0}}{H_\Theta} - \frac{S'_{f0}}{1 + H_\Theta} \right) = 2 \frac{\mathcal{D}_0}{\alpha_1} + \frac{S'_{f0}}{1 + H_\Theta} \ln \left(\frac{\mathcal{D}_0}{\alpha_1} - \frac{S'_{f0}}{1 + H_\Theta} \right) \quad (4.79)$$

This equation contains both unknown constants, α_1 and V_0 . The second equation needed to determine V_0 is obtained by substituting γ_Θ and γ_Θ into Equation 4.67, yielding:

$$1 - \frac{S'_{f0}}{H_\Theta} - \frac{S'_{f0}}{1 + H_\Theta} = \mathcal{Q}'_0 = \frac{\delta^{1/2} \alpha_1 \mathcal{Q}}{1 + V_0} \quad (4.80)$$

Using this result, let us express the term containing the Damkohler number using the front temperature scaled with the ambient temperature, $T_f = 1/(\alpha_1 \delta^{1/2})$:

$$\frac{\mathcal{D}_0}{\alpha_1} = \frac{\mathcal{A} T_f^2 e^{-\mathcal{E}/T_f}}{\mathcal{Q} \mathcal{E}} \left[1 - S'_{f0} \left(\frac{1}{H_\Theta} + \frac{1}{1 + H_\Theta} \right) \right] \quad (4.81)$$

The adiabatic eigenvalue relationship is obtained by taking $S'_{f0} = 0$ in Equations 4.79 and 4.81.

$$\frac{1}{2} = \frac{\mathcal{A} T_f^2 e^{-\mathcal{E}/T_f}}{\mathcal{Q} \mathcal{E}} \quad (4.82)$$

It is the same as obtained by Britten for the adiabatic case, after correcting for the differences in the definitions for the front temperature.

To complete the solution for the curved front model, two assumptions about the boundary layer thickness for tangential heat transport are made. First, the thickness of the boundary layer in the preheat zone is assumed to be proportional to the characteristic length of the preheat zone. Second, the thickness of the boundary layer in the char zone is assumed to be much greater than that of the preheat zone. Given these assumptions, Equation 4.80 can be reduced to a quadratic equation in $1 + V_0$.

$$T_f(1 + V_0)^2 - Q(1 + V_0) - S_f/g = 0 \quad (4.83)$$

where

$$g = \frac{g'}{1 + g'} \quad (4.84)$$

$$g' = \frac{G_\Theta}{1 + V_0} \quad (4.85)$$

Equation 4.83 is solved to give an explicit relationship for the front speed. The positive (faster growing) root is assumed to be the one that is physically observable, following Britten's reasoning.

$$V_0 = \frac{Q + \sqrt{Q^2 + 4T_f S_f/g}}{2T_f} - 1 \quad (4.86)$$

This equation shows that the front speed is strongly influenced by tangential heat transport. The front temperature is less strongly influenced, due to the exponential dependence of the Arrhenius kinetic term. Comparison with the adiabatic case, in which higher order corrections to the eigenvalue can be obtained, suggests that for the front temperatures actually observed T_f should be replaced by $T_f - 1$ in Equation 4.86 to correct for the nonzero value of the ambient temperature.

$$V = \frac{Q + \sqrt{Q^2 + 4(T_f - 1)S_f/g}}{2(T_f - 1)} - 1 \quad (4.87)$$

For moderate amounts of lateral heat transport, it is reasonable to use the adiabatic eigenvalue relationship along with Equation 4.87.

4.6 Nature of the Solution

One-dimensional models of RC front propagation have two aspects. The first is a kinetic relationship that gives the front temperature as a function of the gas flux. For the adiabatic case, this relationship is Equation 4.82. This relationship can be displayed as an Arrhenius plot, as shown by Figure 4.6. The front temperature increases monotonically with gas flux. This behavior is illustrated in terms of dimensionless quantities by Figure 4.7, for various dimensionless heats of combustion and a dimensionless activation energy of 10. The square root of the reciprocal of $\mathcal{A}$ is proportional to the dimensional gas flux.

Figure 4.6: Dimensionless Arrhenius Plot for Reverse Combustion Front Propagation

Figure 4.7: Dimensionless plot of front temperature versus gas flux for a dimensionless activation energy of 10 and various heats of reaction.

The second aspect of a RC front propagation model is an overall energy balance, Equation 4.87. Examination of this equation shows that the dimensionless front speed decreases with increasing front temperature and hence with increasing gas flux. This is because as the front temperature is increased, more heat is convected by the gas passing through the front, which necessarily decreases the amount available to solid phase convection, thus the front slows down. The dimensionless speed is scaled with the gas flux. The dimensional front speed displays a maximum with respect to gas flux. To generate dimensionless results that are more immediately meaningful, the dimensionless front speed can be multiplied by the square root of the reciprocal of A . This compound parameter responds in the same fashion as the dimensional front speed. Typical response of the front speed is given by Figure 4.8 for a dimensionless activation energy of 10, and no tangential heat transport. Similar plots for a dimensionless activation energy of 20 are given by Figures 4.9 and 4.10.

Tangential heat transport has an effect similar to changing the dimensionless heat of reaction, except that the thickness of the tangential heat transport boundary layer scales with the front speed, so its effect is less at low gas flux. Typical behavior is illustrated in Figures 4.11 and 4.12. The computer program to generate the data for this plot, FEWCHT, is listed in Appendix A.

To achieve certain values of dimensionless parameters, supplemental fuel would have to be added to the oxidant gas, since insufficient fuel would be released by devolatilization. If we impose the additional normal restriction that no supplemental fuel is added, stoichiometric extinction can occur. This is illustrated by Figures 4.13 and 4.14 which show the dimensional front temperature and propagation speed versus the gas flux for various levels of tangential conduction. Note that at sufficiently low gas fluxes and sufficiently high gain of heat from tangential heat transport, the front speed decreases with gas flux. The kinetic parameters for this plot have been adjusted to yield a maximum propagation velocity of approximately 0.08 m/hr at 250 kg/hr/m² for the line representing no tangential conduction. This propagation speed is approximately that for the Hanna field tests.

4.6.1 Comments and Precautions

Global kinetic parameters provide a means of interpolating between data taken under various conditions of combustion back pressure, and oxygen content of the injected gas. Given the truly complex interactions that these kinetic parameters represent, the temptation to extrapolate should be resisted. Little confidence should be placed in results of such extrapolations. Of all parameters, perhaps the most dangerous to extrapolate is pressure, since the apparent order of the reaction has not been experimentally verified. Additional complications arise because of pressure dependent devolatilization (Su and Engelman, 1979). Without experimental verification that is currently lacking, global kinetic parameters can not be used to extrapolate from coal to coal. Given the crude reaction rate formulation used in such models, if temperature versus gas flux data are available at the conditions desired, they should be interpolated directly, rather than converting from temperature to kinetic parameters, then back to temperature. The interpolated temperature can then be used in the heat balance to evaluate the front velocity. This procedure is less prone to error than,

Figure 4.8: Plot of a dimensionless group that is proportional to the dimensional velocity versus dimensionless gas flux for a dimensionless activation energy of 10 and various dimensionless heats of reaction.

Figure 4.9: Plot of dimensionless front temperature versus dimensionless gas flux for a dimensionless activation energy of 20 and various dimensionless heats of reaction.

Figure 4.10: Plot of a dimensionless group that is proportional to the dimensional velocity versus dimensionless gas flux for a dimensionless activation energy of 20 and various dimensionless heats of reaction.

Figure 4.11: Plot of dimensionless front temperature versus dimensionless gas flux for a dimensionless activation energy of 20 and various dimensionless versus dimensionless gas flux for a dimensionless activation energy of 10, a dimensionless heat of combustion of 5, and various values of dimensionless tangential heat transport.

Figure 4.12: Plot of a dimensionless group that is proportional to the dimensional velocity versus dimensionless gas flux for a dimensionless activation energy of 10, a dimensionless heat of combustion of 5 and various values of dimensionless tangential heat transport.

Figure 4.13: Front temperature versus gas flux for various levels of tangential conduction.

Figure 4.14: Front propagation velocity versus gas flux for various levels of tangential conduction.

for example, that used by Visser *et al.* (1985).

By nondimensionalizing the equations before transforming to the moving coordinate system, the dimensionless parameters selected do not explicitly contain an eigenvalue (temperature or front velocity). This separates the dependent variables from the independent parameters. We can refer, for example, to a dimensionless activation energy of 5 or 50, without having to solve the problem. This effort is not completely successful because of the temperature dependence of physical properties and the combined temperature and velocity dependence of the gas compositions, but these effects are minor compared to the variation in gas flux.

This model employs the quasi-steady-state-approximation in a moving coordinate system. This assumption is not valid at extinction. Model results in this regime should be used with caution.

4.7 Application to Experimental Data Analysis

The first application of this model is to extract kinetic parameters from experimental data for use in this model. A set of experiments is conducted wherein a RC front propagates through a porous solid. During such an experiment the gas flux to some point on the propagating combustion front is calculated. There do not appear to be any experimental techniques that allow direct measurement of the gas flux inside a porous solid. The calculation of the gas flux presents no problem if the front is uniform, flat, and fills the entire cross-section. Otherwise, an equation of motion for porous media must be used to model the flow field inside of the solid. This is typically Darcy's law, although others exist. Solution of the flow field might be numerical, or for two-dimensional slab experiments by the electroconductive analog along the lines of Fuhrmann (1987), or by graphical methods. Taking a cross-sectional area of a growing channel and dividing the total gas flow by this area is not sufficiently accurate as study of Laplace's equation will show. For slab experiments, wall effects should be considered, since they may concentrate the flux at the center of the slab.

Equation 4.82 is the basis of the data-analysis. Two possible approaches can be used in applying this equation. The direct approach is to measure the temperature and use the calculated value of the gas flux for some point, and either measure or estimate the gas composition at the same point, for a variety of experimental conditions. An arbitrary value is assigned to the pre-exponential factor. The dimensionless group $\ln(Q/A)$ is then plotted versus $-1/T_f$. Neglecting the slight curvature caused by the presence of the temperature squared term in Equation 4.82, the data ideally should fall on a straight line. The slope is the dimensionless activation energy. From the definition of the dimensionless activation energy, the dimensional activation energy can be obtained. Equation 4.82 is solved for the dimensionless pre-exponential factor.

$$A = \frac{1}{2} Q \mathcal{E} T_{f1}^{-2} e^{\mathcal{E}/T_{f1}} \quad (4.88)$$

The value of the RHS is calculated for each experiment. The average is the dimensionless pre-exponential factor. From its definition, the dimensional pre-exponential factor can be obtained.

The indirect approach is to measure the front propagation and determine the gas flux and its derivatives (to estimate the tangential heat transport), then use Equation 4.87 to estimate the front temperature. The remainder of the procedure is as detailed above.

4.8 Application to Channel Growth Simulations

The second application of the model is as a propagation relationship in a simulation of the growth of a reverse combustion channel in a porous solid. In this application the front velocity is desired as a function of the components and derivatives of the gas flux. All other parameters such as composition of the injected gas, pressure at the burning front, and kinetic parameters are known, fixed values.

The overall solution technique is to apply damped successive substitution to a basic procedure to get converged values of the front temperature and velocity, and the derivative of the front temperature with respect to gas flux. This basic procedure is applied to both the actual gas flux desired and an incremented gas flux, to allow the guess for the derivative to be updated.

The basic procedure is as follows: Given a guess for the front temperature and front velocity, the gas composition at the combustion front is calculated, as well as the normal component of the gas flux after devolatilization. Next, the dimensionless parameters are calculated which requires a guess for the derivative of the front temperature with respect to the gas flux. The implicit equation for the front temperature, Equation 4.82 is solved by Wegstein's method (Ramirez, 1989), to give a new value of the front temperature. This allows a new value of the front speed to be determined with Equation 4.87. A FORTRAN 77 (with VAX extensions) subroutine, FULL COMBUST, which implements this algorithm is given in Appendix A. This appendix also includes the necessary supporting subroutines. Physical properties are from Kotowski and Gunn (1976), with conversion to SI units by Britten(1984).

4.9 Summary

A one-dimensional model for a weakly curved RC front propagating through a porous medium is formulated. This model presumes small tangential heat transport and complete combustion of the limiting reactant. The superabundance of fuel is not presumed. A boundary layer approach is used to incorporate the effects of tangential convection and conduction of heat on front propagation. The model is solved by using high activation energy asymptotics (HAEA). The characteristics of the solution are illustrated by a series of figures. Comments about the model are made, as well as precautions that should be observed in using this model. The application of this model to experimental data evaluation is outlined. The algorithm to implement this model is also outlined.

CHAPTER V

REVERSE COMBUSTION EXPERIMENTS IN SOLID COAL

5.1 Scope

This chapter begins with discussions of the objectives of the experimental program and the resulting design considerations. An overview of the apparatus and experiments is provided and key features of the design are discussed. The core of the chapter consists of a more complete description of the apparatus, the operating procedures, and the data-reduction procedures. The chapter ends with a summary. The experimental results are presented in Chapter 6.

5.2 Objectives of the Experimental Program

The objectives of the experimental program were established within constraints imposed by the research grant that supported this work. This study was funded by the United States Department of Energy (U.S.DOE). An important concern of this agency was that the experimental program should utilize solid coal, rather than a model material.

The first objective of the laboratory studies was to study the response of observed RC combustion front characteristics to changes in the process parameters. This was an empirical investigation of RC for the given range of conditions investigated.

The next objective was to test scale-down laws for RC channel propagation. Previous modeling work indicated that the size of the RC combustion channels created in coal could be changed by altering certain process parameters. Through this technique, it was hypothesized that RC could be conveniently studied in moderate-sized coal samples by operating at elevated pressures and/or enhanced oxygen content of the injected gas. The feasibility and limitations of this approach were investigated.

Another objective was to explore using combustion-tube data to extract global kinetic parameters for modeling RC in solid coal. These model parameters must be fit to RC experiments since techniques to measure these parameters independently have not been developed.

Another objective was to apply spatially-distributed flux estimations for correlating channel characteristics and for extraction of kinetic parameters. RC front propagation models depend on the local gas flux to the combustion surface, yet previous experiments in RC in solid coal were analyzed using the superficial gas flux through the entire cross-section. Extrapolation of combustion-tube data analyzed in this manner to field conditions is problematic.

The last objective was to test the concept of using promoters and inhibitors to alter RC channel characteristics as proposed in Chapter 3. Due to safety considerations, the initial goal was to investigate the effects of a combustion inhibitor.

These objectives imposed requirements on the experimental program. An experimental apparatus was needed for studying combustion of porous solids. In this apparatus, reverse combustion in solid, subbituminous coal samples had to be established. Naturally, an overriding requirement of the experimental program was that it be conducted in a safe manner, with proper concern for the safety of those directly involved in the research. Furthermore, these activities could not adversely impact others not directly involved in the research.

5.3 Design Considerations

The design considerations imposed by the objectives and requirements for the experimental program are presented in this section. The design considerations have been grouped into those involving controlled parameters, observation of channeling, coal samples, safety and reliability.

5.3.1 Controlled Parameters

A basic consideration in the design of the experimental apparatus was the parameters that had to be controlled and the systems to accomplish this. To test scale-down laws for RC, the experimental apparatus had to be capable of creating small RC channels in the laboratory. The scale-down laws suggested that this could be done by operating at elevated pressures and enhanced oxygen content. Consequently, the experimental apparatus had to be capable of creating a high, controlled and measured combustion back-pressure. A system capable of supplying oxidant gas at a controlled, measured flow rate and composition was needed. To test the idea of promoters and inhibitors, a candidate chemical agent had to be selected and a system for adding it to the injected oxidant gas was needed.

5.3.2 Observation of Channeling

A reverse-combustion channel is a temporally changing disturbance in the temperature, permeability, gas flux, and chemical reaction fields associated with a combustible porous media. To best pursue the objectives of this study, it would be desirable to observe all these aspects of a RC channel. This ideal is unobtainable given the extremely limited observational techniques available for investigation of porous solids. Instead, the aspects of RC channeling that are observed represent a compromise between the observational techniques available, their associated limitations, and the sometimes conflicting importance that different aspects have for different objectives.

In this study, primary emphasis has been placed on the macroscopic character of the channel (its temporal evolving shape). Thus, a major consideration in the design of the experimental apparatus was the method that would be used to observe the channel propagation. Three methods received serious consideration: infrared thermography (IRT), real-time

x-ray imaging (RTXI) , and temperature tracking (TT) from spot thermocouple measurements, coupled with post-burn sectioning. The advantages of IRT were that it would have a low cost (the necessary equipment was readily available) and that it would give a spatially-distributed picture of the temperature field. The disadvantage of IRT was that it would give a surface temperature measurement. Hence, it would require thin, large samples that would be difficult to prepare from coal. Thin samples would be required since the accuracy of estimating a temperature in the interior of a solid from a surface temperature distribution decreases with increasing thickness. Furthermore, the channels formed in such thin samples would be severely affected by heat losses, and hence would not faithfully simulate the characteristics of RC channels formed in-situ. The advantage of RTXI was that it would give a reasonably clear picture of transient channel evolution inside a comparatively thick coal sample (approximately 20 cm). Since it responds basically to the density changes occurring in the sample, it would give a reasonably clear picture of porosity and permeability changes that are the most important practical consequences of RC channel propagation. The disadvantages were that it would be relatively expensive (\$70,000 to \$100,000) and that it would require extensive, expensive safety precautions. The advantages of TT were that it would allow use of thick coal samples and that it was comparatively inexpensive. The disadvantage was that it does not give a clear picture of transient evolution of the channel. The apparatus design must balance the respective advantages and disadvantages of these approaches with the financial resources available for this research.

5.3.3 Coal Samples

This laboratory study was constrained to use solid coal as the combustible medium in which to study RC. Hence, the first design consideration for the sample acquisition program was the selection of a coal suitable for RC to use in this study. The next was the size and shape of the coal sample. Since RC propagates in large channels (approximately 1 m) at moderate operating conditions, large coal samples are desired to limit the severity of the process parameter changes needed to create channels smaller than the samples. However, small samples are desired to limit the difficulties and costs for sample acquisition, and to limit the size and expense of the experiment apparatus. The desired shape of the sample is influenced by the method used to observe channel propagation, by process and mechanical considerations for the design of apparatus, and by the method with which the sample is shaped (coring or sawing). The final consideration for the sample acquisition program was the number and quality of the coal samples that would be obtained. Clearly a large number of high quality samples is desired to allow an extensive experimental investigation of RC; however, this goal must be balanced against the costs (in terms of money, time and manpower) of sample acquisition.

5.3.4 Safety

Another important consideration in the design of the experimental apparatus was safety. Reverse combustion is inherently a relatively high temperature process. This high

temperature can lead to failure of mechanical systems. In order to create small channels in coal, it was felt necessary to use high pressures and/or high concentrations of oxygen. Two hazards are directly associated with the use of high pressure. The first is the danger of catastrophic failure of the pressure containment system, and the second is the danger of suffocation or poisoning from displacement of air with gases released from the pressure containment system. The chief hazards associated with use of high concentrations of oxygen are that it increases the likelihood of spontaneous ignition and that it intensifies combustion. The experimental design and operating procedures must recognize and minimize these dangers.

5.3.5 Reliability

The final design consideration is ensuring reliable operation of the apparatus. The greatest threat to reliable operation of laboratory coal-combustion apparatuses is plugging of the product gas handling equipment with tars and particulates (Bell, 1982). This part of the apparatus requires careful engineering design.

5.4 Design Overview and Discussion

This section begins with a brief description of the experimental program designed to meet the objectives and considerations presented in the previous sections. This is followed by a discussion of the most important aspects of the design.

5.4.1 Overview of Apparatus and Experiments

The apparatus consisted of a sample container, a pressure vessel, a gas-supply system, a gas clean-up system, and a data-acquisition system. The experiments consisted of burning a RC channel within a solid cylindrical coal sample, observing the progress of this burn, quenching the burn, and examining the resulting coal sample.

A roughly cylindrical solid coal sample was encased in a gas-tight container to allow oxidant gas to be forced through the sample at a controlled flow rate. To initiate combustion, the end of the coal sample was initially heated electrically. The gas-tight container was contained in a pressure vessel. The controlled process parameters were oxidant-gas-flow rate, oxidant-gas composition, and combustion pressure.

To observe the progress of the burn, measurements of 16 thermocouples emplaced in the coal sample were logged, along with gas composition as measured with a gas chromatograph. The burn was quenched by halting the gas flow. The resulting coal samples were examined by casting brightly pigmented resin in the channels, then slicing directly through the container and the enclosed sample. The shape of the channel(s) at the exposed surfaces was identified and recorded.

A digital approximation of the three-dimensional shape of the sample and channel(s) was constructed from the records. The gas flux to the channel surface at the end of the experiment then was estimated by three-dimensional finite-difference calculations. The

temperature-versus-time log was correlated with the shape of the channel to obtain an estimate of the propagation velocity of the tip of the most prominent channel at the end of the experiment.

5.4.2 Discussion of Key Design Features

The fundamental design decision was to use thermocouples to track the front (TT). IRT was rejected because Dindi's experience with sample preparation for electrolinking indicated that it would be infeasible to create coal samples of the dimensions needed for RC (Dindi, 1988). In addition, the creation of fully three-dimensional channels was felt to be important for studies using solid coal. Despite encouraging preliminary sample evaluations, RTX was rejected because the costs were simply too high to be absorbed by the existing research grant. A search for surplus X-ray imaging equipment was unsuccessful. Consequently, it was decided that the experimental apparatus would be designed to be compatible with the future addition of a real time X-ray imaging system. This requirement for compatibility primarily affected the design of the pressure vessel and sample container. The length of the pressure vessel was increased to allow translation of a 1 m long coal sample past a stationary X-ray imaging system. The pressure vessel was equipped with ports needed by an imaging system. In addition, aluminum was chosen for the construction of the sample container tube because of its small X-ray cross-section. This study, however, would use the more conventional approach of using spot thermocouples to measure the front temperature and propagation velocity and use post-burn sectioning to determine the final shape of the channel. To minimize the disturbance to the gas flow and temperature distribution, smaller thermocouples 16mm (1/16 inch) were used than in past studies (typically 32mm (1/8 inch)). To aid in identifying the channels, pigmented resin was cast in the channels. To give some insight into the chemical aspects of RC channeling, the product-gas composition was measured with a gas chromatograph. The average product-gas composition has limited utility for analysis of small-scale channeling experiments since the tip of the channel could be oxygen limited despite presence of oxygen in the product gas. It is primarily useful as a monitoring tool since it indicates the presence or absence of combustion.

Subbituminous, nonswelling coal was used for several reasons. In the western United States, UCG is envisioned to use such coals. This rank of coal is known to be reasonably well suited to RC, based on its volatile matter content. Various field test observations indicate that RC can propagate reasonably well in this rank of coal. The method chosen to obtain coal samples was horizontal coring at a mine site. The desire for large samples favored shaping the samples at the mine site, rather than obtaining the samples in rough form and shaping them in the laboratory, in order to limit the amount of coal that had to be transported and stored. Since coal has a much lower permeability across the bedding planes, and RC in field tests propagates basically parallel to the bedding planes, it was felt important that the major axis of the coal samples also be parallel to the bedding planes. Coring was selected over cutting and shaping chunks of coal for several reasons. First, this method was recommended by the U.S. DOE. It had the potential of providing coal samples with a relatively large aspect ratio to allow propagation of the channels for an extended period of time. Since it was more highly

mechanized, it had the potential of creating more coal samples in a shorter period of time. The time savings were desirable to minimize the interference with mine operations, and to provide more time to conduct the experiments. The sample diameter (17 cm) was primarily set by the availability of rentable coring tools. Coring at this diameter is an unusual practice. At larger diameters, the costs would be excessive since the tools would probably have to be purchased. Since the length of the obtainable coal cores was not known beforehand, the laboratory apparatus had to be adaptable to various length coal samples. In previously funded research, the U.S.DOE had provided coal samples through its contracts with various national laboratories. During the course of this study, the U.S.DOE substantially reduced its funding to both universities and the national laboratories for UCG research. Consequently, the national laboratories were unable to provide assistance for obtaining the coal samples. Since no additional funding was available directly from U.S.DOE, the funds for sample acquisition had to be obtained from previously allocated funds. The result of this situation was that only one attempt could be made to obtain coal samples; hence, their number and quality had to be accepted. Supplemental funding from other agencies was not available, since generally they consider this research to be within U.S.DOE's sphere of action.

Several design features were included in the apparatus to increase the safety of the apparatus in view of the potential hazards arising from operating a combustion process at high pressure and with enhanced oxygen. The sample containment system was a multi-layered design. The coal sample was immediately surrounded by a layer of insulating refractory concrete that prevented direct contact of the combustion front with the thin-walled gas-impermeable container. If the refractory failed to protect this container, it would be exposed to high temperatures. However, it was under low mechanical stress because of the partial balancing of pressure on each side of its wall. This container was enclosed in a thick-walled pressure vessel which was subject to the full stress required for operation at 40 bars, but was isolated from temperature excursions. The intermediate space between the two vessels was purged with nitrogen to moderate the shell temperature and to limit the concentration of combustible, corrosive gases that would be present if the inner pressure vessel failed. All equipment and instrumentation was conservatively pressure-rated and protected with pressure-relief valves. Furthermore, all components exposed to high concentrations of oxygen were cleaned to remove traces of combustible oils. The pressure vessel was located in the back room of a high pressure laboratory while the operators monitored and controlled the experiments from the front room, protected by a reinforced concrete wall. Unnecessary entry into the back room was avoided. Two operators were present whenever experiments were conducted outside normal working hours. The laboratory was equipped with an room-air exhaust fan, several fire extinguishers, and a carbon-monoxide detector. The laboratory was also equipped with an oxygen monitor to warn the operators in case of a leak of the high pressure gas cylinders. Operating and emergency procedures were developed, approved by the University's Department of Environmental and Safety, and posted.

Various preventive steps were taken to minimize the likelihood of plugging the effluent coolers with tars and particulates. First, the temperature difference between the heat exchanger surface and the bulk gas was controlled by using natural convection heat transfer to air, rather than to ice water as had been done in previous RC experiments. Second, the

coolers were arranged for down flow, so that both gravity and drag from the gas flow would assist in draining any condensed tars. Third, any liquid that condensed was periodically collected and removed from the heat exchanger. Fourth, the filters were paralleled to allow on-line cleaning.

5.5 Description of Apparatus

To aid the reader in obtaining an accurate understanding of the apparatus that has been constructed for this study, a schematic drawing of it has been provided in Figures 5.1 and 5.2. The oxygen and nitrogen used to carry out the reverse combustion were supplied by premixed cylinders of compressed gases. The gas used was either "Custom" grade oxygen/nitrogen mixes with a certified accuracy of 0.5 volume percent, or U.S.P. grade air supplied at 16,000 kPa in "T" size cylinders. The selection of premixed cylinders over on-line mixing was made for reasons of safety, operating reliability, and short-run cost savings. The gas could be discharged from cylinders at a pressure of up to 4000 kPa, as controlled by high discharge pressure regulators. The oxidant-flow rate was controlled by a mass-flow controller (Omega Engineering Model FMA-7108). The maximum controlled flow rate was 0.4 kg/hr, or equivalently 5 standard liters per minute (SLPM) The range of this instrument was over tenfold, with an accuracy of 2 percent of full scale. This instrument was factory calibrated. The factory calibration was checked for gross error against a rotameter. The feed gas could flow through a saturator at a controlled temperature, in order to allow addition of volatile liquids (such as combustion promoters or inhibitors into the feed gas; however, this feature was not used in these experiments. The saturator consisted of a stainless steel sample cylinder (1000 ml volume, 8.9 cm diameter), a stainless steel tube (1.27 cm diameter), and a nozzle assembly. The sample cylinder served as a reservoir for the liquid being added to the gas stream and as a disengaging space for entrainment. The tube, which was connected to the bottom of the sample cylinder, serves as a mass transfer contacting zone. The nozzle, which is located at the bottom of the tube, serves as a sparger to insure creation of small gas bubbles in the tube which is necessary for effective mass transfer. The saturator was placed in a large insulated, stirred water bath, which was of sufficient size that its temperature could be adequately controlled (± 0.25 C) simply by periodic addition of crushed ice. The gas supply system also provided inert gas for the pressure-vessel shell. The shell was maintained at a slightly higher pressure (+200kPa) than the feed gas with a positively-biased differential-back-pressure regulator (Tescom Model 269-375-242), so that the thin-walled sample container described below was always under compression.

The reverse combustion process was carried out in a solid coal block. Figure 5.3 is a drawing of the sample container built around the coal block. The sample container consists of a sample assembly that had to be created for each experiment, and two permanent, reusable end-cap assemblies.

The sample assembly consisted of a coal sample, an inner casting, a set of thermocouples and an outer casting, all contained in an aluminum tube. The inner casting provided mechanical support for the coal sample during sample preparation. It consisted of alternating layers of cement and resin cast into the annulus formed between the coal sample and a

Figure 5.1: Gas supply system and reverse combustion pressure vessel.

GAS CLEAN-UP SYSTEM

Figure 5.2: Gas clean-up system.

Figure 5.3: Reverse combustion sample container.

tube constructed of aluminum flashing. The castable refractory concrete used (Prekrete G-8, from Pocono Fabricators) was a moderately priced, high strength, "face-powder" fine, low permeability premixed concrete suitable for use at moderate temperatures (535 C). Despite selecting the cement for its low permeability character, it was still much more permeable than coal. The resin layers consisted of 20 ml of acrylic casting resin that partially permeated into the cement. The resin layers formed an impermeable barrier to gas flow, thus limiting the bypass-gas flow through the inner casting.

The thermocouples were fabricated from 16 mm (1/16 inch) stainless steel sheathed thermocouple stock. The thermocouples had to be fabricated in-house rather than purchased because of cost constraints. Although the thermocouples failed prematurely, (at temperatures below 300 C) ten to fifteen percent of the time, the material cost for a set of thermocouples was one-tenth of the cost of purchased thermocouple junctions. Sixteen type K thermocouples were used in each experiment. They could not be reused from experiment to experiment. The fabrication procedure is described under Operating Procedures. Four of these thermocouples were emplaced on or near the ignited surface of the coal. The remaining thermocouples were emplaced into the coal in three ranks spaced 3.8 cm apart. Each rank consisted of four thermocouples emplaced approximately 3 cm into the coal. The thermocouples were located top, bottom, east, and west, for the coal sample oriented as it was burned with its axis north and south. The thermocouple leads were routed toward the cold end of the sample.

The outer casting connected the outer aluminum tube to the inner casting. It shielded the outer aluminum tube from excessive heat and mechanically supported the tube against external pressure. It also prevented gas passage in the annulus through which the thermocouples pass. It consisted of layers of the same refractory cement used in the inner casting, and a single layer of the same gas impermeable resin cast in the annulus formed between the inner casting and the outer aluminum tube. Use of multiple layers of resin in the outer casting was abandoned after the unexplained rupture of the sample container in Experiment 3. By using only a single layer of resin, the creation of a completely enclosed, gas-tight annular ring was avoided. The latter configuration was undesirable since in considering an excessive internal pressure of unknown origin for this configuration, the outer aluminum was the weakest mechanical component since it was not supported by the extremely high compressive strength of the castable refractory. By avoiding this configuration, the sample container was made more resistant to rupture. No further failures occurred.

The aluminum tube served as the final gas-impermeable layer when connected with the end-cap assemblies. In making this connection, the aluminum must act as a gasket under hoop compression between a clamp and ridges around the circumference of the end-cap assembly. This was not a mechanically desirable configuration, but was chosen to avoid the necessity of fabricating flanges for each aluminum tube, or flaring the aluminum tubing. As purchased, the aluminum construction tube (6063-T6) was too hard to act as a gasket. Consequently it had to be annealed before use. To accomplish this, an annealing oven was constructed out of materials at hand. The oven chamber was fabricated from aluminum sheet. High temperature insulation was provided by FiberFrac ceramic-fiber batting held in place with a chicken-wire cage covered with aluminum foil. Low temperature insulation was

provided by residential fiberglass insulation, with the combustible paper-backing removed. The outer shell consisted of a 55-gallon drum extended and capped with an aluminum-foil covered chicken-wire cage. The heating elements consisted of two 400 watt and one 200 watt heavy duty fiberglass insulated heating tapes placed in direct contact with the aluminum tube, and held in place with fiberglass cords. A dual temperature controller was graciously donated by Dr. Hasan Dindi and was modified for this purpose by Mr. Norm Taylor. The annealing procedure (Van Horn, 1967) consisted of bringing the tube to 412 C as rapidly as possible, holding at that temperature for 2 to 3 hours, cooling at a controlled rate of 27 C/hr to 260 C, then shutting off the heaters. This procedure reduced the tube to approximately T-0 temper (completely annealed).

The cold end-cap assembly sealed the upstream end of the sample container and provided the inlet gas and the thermocouple feed-through connections. The principal component was a 304 stainless steel disk, 1.92 cm thick, and 19.93 cm in diameter. Holes were drilled into this disk and tapped for the inlet gas and the thermocouple feed-through connections. The inlet gas connection was simply a Swagelok fitting. A flexible hose was attached to this fitting. Oxidant feed gas passed through this hose which was inside the shell of the pressure vessel described below. The hose was stainless steel braided, with a teflon lining. This hose was protected from abrasion with a plastic tube and duct tape. The 16 thermocouple feed-through was purchased from Omega Engineering. It was modified to allow assembly completely from the outside. After assembly of the feed-through connection and construction of the thermocouple connectors, a thermocouple-lead-wire bundle was attached mechanically and electrically. The wire bundle was protected from abrasion by a plastic tube and duct tape. Initially, a heat-shielding layer of Saureisen cement approximate 3 cm thick was cast on the inner surface of the disk. It was held in place and reinforced with piano wire attached to No-8 screws. This heat shield was found to be unnecessary and was not repaired when it mechanically failed towards the end of the experiments. The outside edge of the stainless steel disk was machined down to form two sharp ridges 7.9 mm high. During assembly of the sample container, the end caps were slid into the aluminum tube. At each end, a clamp circumferentially squeezed the aluminum tube, causing it to plastically deform against the end cap. The ridges "bit" into the aluminum forming a strong mechanical seal. When combined with a silicone sealant, the ridges also formed an effective gas tight seal. Surprisingly, common silicone window caulking is heat resistant in short term service to 260 °C.

The hot-end-cap assembly sealed the downstream end of the sample assembly and provided the product-gas outlet connection. It included the electrical heater for igniting the coal. The construction was similar to the cold-end-cap. The electrical power for the heater was provided by two Conax EG-187-A-SS-L single conductor feed-throughs. The electrical heater consisted of two 4-meter-long parallel coils of beaded nichrome wire, capable of generating a total of 800 watts of heat, when supplied with 120 volts AC. Dual coils provided partial protection from breakage of the nichrome wire. None of the wires failed in service, but typically the nichrome wire would break during removal of the hot-end-cap after two or three experiments. Exposure to a hot oxidizing atmosphere, followed by a hot, corrosive reducing atmosphere embrittled the nichrome wire. The coils were held against

the face of the coal by pumice stones filling any void space between the coils and the hot-end-cap. The gas outlet consisted of a 0.63 cm Swagelok fitting which was bored through. The collector tube consisted of a stainless steel tube into which many thin slits were cut perpendicular to the axis of the tube. These slits function as a plugging resistant screen, to prevent large particles from entering the outlet tube. The end of the tube was crimped to prevent any large particles from entering and plugging the outlet tube. A stainless steel tube was connected to the outlet fitting. This tube conducted the product gas through the shell of the pressure vessel.

The sample container was supported by a sample-positioning system. The first component of the sample-positioning system was a set of rails secured to the interior of the pressure vessel described below. The sample container rested on two or three cars which rolled on these rails. The cars were secured to the sample container with stainless steel band clamps. A system of cords allowed the sample container, the oxidant gas hose and the thermocouple wire bundle to be pulled into place.

The thick-walled pressure vessel was designed to withstand internal pressures up to 40 bars, at temperatures up to 300 °C. carbon steel pipe and ANSI class 300 fittings by welding, for reasons of economy. It weighed approximately 700 kg. The internal diameter of the pressure vessel was 30.5 cm. Its length was 240 cm. This length, which is approximately twice that needed for these experiments, was set by design criteria associated with the future addition of a real-time X-ray imaging system. The pressure vessel was hydraulically tested at 6000 kPa. Rubber impregnated asbestos gaskets were used and performed well. These gaskets will not be available in the future, under government-mandated phase-out of the use of asbestos in a large number of products.

Upon exiting the pressure vessel, the gases passed through an effluent cooler. The cooled combustion gases and condensed tars entered a knockout pot. The tars were periodically collected. The gases entered a second effluent cooler. Any condensed tars or water were collected in a second knockout pot. The gases then entered a third effluent cooler. Any remaining tars or water were condensed, and were accumulated in a third knockout pot. These liquids were periodically collected. The cooled gas passed through a filter to remove entrained liquid, and then passed through the back-pressure regulator (Tescom 26-3224-24). The gases then were warmed back to room temperature to remove any self-cooling. Next, the mass flow rate was measured. A portion of the gases passed through a sample loop. Periodically the composition of these gases was measured by a gas chromatograph (GC), using a 10-port automatic sampling valve. The GC used a Porapak R precolumn to make a coarse separation of the gas. The condensible gases were then routed into a Porapak Q column. The order of elution was carbon dioxide, ethane, propane, and trace water. The other gases were routed to a molecular sieve MS5-A column. The order of elution from this column was hydrogen, oxygen, nitrogen, methane, and carbon monoxide. The carrier gas was argon. A thermal conductivity detector was used. It was maintained at 100 C, while the GC oven was maintained at 70 C. Product gases were discharged to an exhaust stack.

To allow the operators to concentrate on safe operation of the experiment, this apparatus featured a personal-computer-based data-acquisition and control system. The computer logged temperatures, pressures, flow rates, and the GC signal. It controlled the oxidant-gas-

Table 5.1
Ultimate analysis of coal from Upper Thompson Seam, Centralia, Washington,
near site of coring

Element	Mass Percent
Carbon	46.1
Hydrogen	5.9
Oxygen	8.7
Sulfur	2.9
Nitrogen	†
Balance	36.4

†Below detection limit.

flow rate and the GC. For use during start-up, the mass flow controller could be cascaded to a software-based PI differential pressure controller. The system hardware was based on a Scientific Solutions LabMaster 2000-PGH data-acquisition card and customized support circuitry. The software created for this system had two parts. The first was a TSR (terminate and stay resident) 8088 assembly-language program. On the prompting of the computer's timer-tick interrupt, this program collected data 18.2 times per second from the sixteen data-acquisition channels. These data were filtered and stored. The second part of the software was a collection of higher-level language (Microsoft QuickBASIC 4.0b) programs for display and interpretation of the collected data. This software is listed in Appendix B.

5.6 Operating Procedures

5.6.1 Sample Acquisition

The 17 cm diameter subbituminous coal samples were obtained at the Washington Irrigation and Development (Widco) strip mine in Centralia, Washington. The drilling site was selected in the upper Thompson seam, by Will Greenough, the chief geologist for the mine. Ultimate and proximate analyses of the coal are given in Tables 5.1 and 5.2 (Greenough, 1990). The samples were obtained by horizontal coring. The drilling contractor was Soil Sampling Services, of Puyallup, Washington. The drilling crew consisted of a driller and two helpers. In two-and-one-half days, 43 m of core were drilled. This yielded approximately 15.2 m of core of length greater than 15 cm. The longest sample was 86 cm. Only one drilling expedition could be conducted owing to budget limitations.

A double-walled core barrel was used with a carbide bit. As the core was cut, the core

Table 5.2
Proximate analysis of coal from Upper Thompson Seam, Centralia,
Washington, near site of coring

Fixed Carbon, mass percent	27
Volatile Matter, mass percent	29
Ash, mass percent	36.4
Moisture, inherent, mass percent	7.6
Sulfur, mass percent	2.9
Heating Value, BTU	7960

sample slid past a shoe into the inner barrel. The inner barrel was not driven, so its rotation with respect to the core was minimal. This reduced the stress on the coal core, compared to a single-walled core barrel. The outer rotating barrel drove the rotary coring bit. Water was pumped through the annulus between the two barrels to wash cuttings out of the hole. After 1.5 m of core were cut, the core barrel was retracted. The shoe held the core, preventing it from sliding back out of the barrel. This stress caused the end of the coal to fracture. The core barrel was retracted the travel distance of the "Ardvaark" horizontal drilling rig. A section of the drill pipe then was removed. The rotary motor that drove the drilling was advanced and the drill string was reconnected. The core barrel again was retracted. This process was repeated until the barrel was completely out of the hole. The core barrel was disassembled, removing the bit, the outer barrel and the shoe. The core then was slid out of the inner barrel under water pressure into a tray.

The core was rarely in one intact piece. The cores had variable amounts of shale. Cores that had no shale partings were weak and highly fragmented. Larger pieces tended to have more shale. The cores were wrapped in fiberglass insulation (which provided cushioning) and were slid into PVC tubes. The ends of the PVC tubes were sealed and the tubes partially filled with water. After completion of drilling, the cores were transported to Colorado in a rental truck.

5.6.2 Sample Preparation Procedure

The sample preparation procedure includes all steps taken prior to the actual experiment. It is described below:

1. **Selection of Coal Sample:** A coal sample was selected from the available cores. The primary criterion for choosing a sample was minimal thickness of shale partings. The secondary criterion was length. The external surface of the core was sketched while noting external indications of cleat orientation.

2. **Preparation of the Inner Casting:** The coal core was encased in a thin layer of castable refractory concrete. First, the ends of the sample were covered with aluminum foil and aluminum foil tape, to keep the ends of the sample free of concrete. The sample was placed in a cylinder of aluminum flashing material of 18 cm diameter. Alternating layers of refractory concrete (4-5 cm) and resin (10-20 ml) were cast into the annulus. Each layer was allowed to set before continuing.
3. **Fabrication of Thermocouples:** 16 sections of clad thermocouple stock were cut to a length of 80 cm with aviation snips. A length of 1 cm of the stainless steel sheathing was stripped from both ends of each piece, exposing two fine thermocouple wires. The wires were scraped with a razor blade to remove any traces of the ceramic insulation. The two wires were tightly twisted together with a needle-nose pliers. This helix was hit with a hammer to cold forge a thermocouple junction. The thermocouple junction was electrically insulated with a coating of glass. A capillary glass tube was slid over the junction and fused with a bunsen burner. The ceramic insulation inside the sheathing was protected from water infiltration by coating the base of the junction with acrylic resin. The thermocouples then were tested.
4. **Emplacement of Thermocouples:** For each thermocouple, a 3 mm carbide bit was used to pierce the surrounding concrete layer. A 2.3 mm bit was used to drill into the coal to the desired depth (3 cm). The thermocouple was coated with a layer of silicone sealant to provide an additional protective coating against water, then slid into the hole. The thermocouple lead was routed along the sample container toward the cold end. The lead was secured with aluminum foil tape, then labeled. The thermocouples then were tested.
5. **Preparation of the Outer Aluminum Tube:** The aluminum tube was cut to the length of the sample plus an additional 30 cm in a cutoff machine. Heater coils and thermocouples were secured to the tube using high temperature fiberglass cord (such as is used as a gasket in stoves). The tube was placed into the annealing oven. After electrical checks, the annealing procedure was carried out.
6. **Fabrication of the Outer Casting:** A casting form was covered with aluminum foil. The inner casting was placed on this form, with the coal face to be ignited facing down, and the thermocouples leads pointing up. The form set the necessary spacing between the ignited coal face and the end of the aluminum tube. The outer aluminum tube was slid over this assembly. The annulus was then filled with layers of refractory concrete. Approximately 8 cm from the cold end of the coal sample, a layer of acrylic resin (3mm) was cast, to prevent gas passage down annulus. Another layer of concrete was cast, to form the cavity at the cold end through which the thermocouple leads were routed to the thermocouple feed-through. After the last layer was set, the thermocouples were tested. The whole sample container was inverted. The casting form was removed, along with the aluminum foil on the face of the coal. A cavity for the ignitor coil then was cast.

7. **Cold-end-cap Installation:** The thermocouple cavity was filled with pumice rocks. All residual silicone sealant from the previous use was removed with a razor blade and an abrasive pad. The thermocouple leads were routed through the thermocouple feedthrough. A bead of silicone sealant was placed around the circumference of the cold end. The cold-end-cap was then lowered into the aluminum tube. After the circumferential ridges contacted the aluminum tube, the end cap was driven into the aluminum tube, with care taken so that it remained perpendicular to the axis of the tube. A circumferential clamp was placed around the outside of the tube. It was tightened to squeeze the aluminum tube against the circumferential ridges. An additional bead of silicone sealant was placed at the junction of the cold-end-cap and the aluminum tube then smoothed. The thermocouple feed-through then was assembled by sliding the pieces over the thermocouple leads. The feed-through was packed with silicone sealant. The thermocouple connectors then were fabricated. The thermocouples then were tested.
8. **Hot-end-cap Installation:** The heater coils were repaired or replaced as necessary. The end cap was cleaned as above. The exhaust line was removed, cleaned with acetone in an ultrasonic cleaner, then replaced. At the same time, the filters for the gas clean-up system were cleaned. The beaded nichrome wire heater coils were placed against the coal face. Any excess void space was filled with pumice rocks. The remainder of the procedure was the same as for the cold-end-cap. The heater coils then were tested.
9. **Loading of Sample Container into Pressure Vessel:** The cars were placed on a set of rails external to pressure vessel. The sample container was placed on cars and secured. The oxidant gas hose was attached to the sample container. The thermocouple wire bundle was secured to a strain relief strap attached to the cold-end-cap with wire. The thermocouple connections were made up. The connections were covered with electrical tape for insulation and strain relief. The entire thermocouple circuitry was tested. The sample was slid into the pressure vessel. The thermocouple wire bundle and the oxidant gas hose were kept from interfering with the cars by retracting them with a system of cords. The exhaust line was made up and covered with a layer of insulation. The power leads were connected to the heater coil and tested. The main blind flange for the pressure vessel then was secured. The laboratory then was prepared for safe operation of the experiment.

5.6.3 Experiment Procedure

The experiment procedure includes all operations with the combustion apparatus under pressure, as described below:

1. **Preparation for the Experiment:** The oxygen monitor was calibrated to room air then checked for operability by triggering the alarm by breathing directly into the sensor several times. The control panel was configured for pressurization. The data-acquisition programs were initialized. The gas chromatograph was started. During

some of the experiments, the condenser was filled with ice and water. This procedure was abandoned after episodes of erratic behavior of the pressure controller were correlated with periodic addition of ice into the water. Since the experiments were carried out at a minimum of 2500 kPa, the additional cooling was unnecessary, since the water content of the gas to the GC was small.

2. **Pressurization:** The back-pressure regulator set point was raised above the desired pressure. The regulator for the oxidant gas cylinder was set to 300 kPa above the shell pressure. The regulator for the nitrogen gas was set at 150 kPa above the shell pressure. The oxidant mass flow controller was cascaded to a software-implemented differential pressure controller. This controller allowed sufficient oxidant gas to enter the sample container to maintain a pressure differential between the shell pressure and product-gas outlet pressure of 80 kPa. Periodically, the regulators were adjusted to effect a steady increase in the shell pressure. The nitrogen cylinder was changed as necessary. One or two cylinders of nitrogen were used to bring the experiment up to pressure. Several hours were required to bring the apparatus to pressure. Once at pressure, the mass flow controller was returned to flow control service and the back pressure regulator set point was adjusted to the desired experiment pressure. The purge gas flow then was established.
3. **Calibration of Instrumentation:** The operation of the product gas flow meter was checked against the oxidant flow controller by using a bypass that caused flow to go directly from the mass flow controller to the mass flow meter. After verification of the consistency of the meters, the flow was redirected through the sample to check for measurable gas leakage. The calibration runs for the GC were made. After this the GC was configured to measure the product gas.
4. **Ignition of the Coal:** The oxidant gas flow rate was set to 0.5 SLPM (0.04 kg/hr). The variac that controls the power was gradually increased to full power over a period of 15 minutes. The temperature of the four thermocouples near or on the ignited coal face was monitored. The temperature would gradually increase and combustible volatiles would be released into the gas. Ignition of the gas would occur on the surface of the heater coil. This event did not correlate with the measured temperature. Frequently ignition was detected by an abrupt temperature rise to 550–700 C in one or more of the first-rank thermocouples. Another useful indication was a sudden temporary surge in the product gas flow rate. Upon ignition, the oxidant-gas-flow rate was increased, to initiate stable operation. After the GC confirmed carbon dioxide in the product gas, the variac was ramped down. The variac could not be abruptly turned off because that would lead to loss of product-gas flow, and hence pressure control, owing to the cooling and contraction of the gas in the heater coil cavity. The ignition period lasted approximately 40 minutes.
5. **Monitoring the Experiment:** During the course of the burn, the temperatures, flow rates, pressures, and GC runs were monitored with the aid of the data-acquisition software. The pressure regulators were monitored and adjusted to compensate for any

changes. If necessary, one set of product gas filters were taken off-line, disassembled, cleaned, reassembled then put back into service. Periodically, the knock-out pots were blown down and any liquid collected. This phase lasted on the order of six hours.

6. **Termination of the Experiment:** The experiments were intended to be halted before the combustion front burned through the sample. When the decision to halt an experiment was made, the oxidant-gas flow was abruptly stopped. The product-gas flow rate then stopped and the sample container pressure decreased by 100 kPa in less than one minute as the product gases in the ignition cavity cooled. This cooling would pull a slight flux of oxidant gas through the combustion front, but not enough to maintain combustion. The temperature would decay toward ambient. When all measured temperatures were below 60 C (one to two hours), the back-pressure-regulator-bypass needle valve was cracked open to commence active depressurization. The reduction in the sample container pressure would cause the shell pressure controller to release gas to maintain the set differential across the wall of the sample container. The bypass valve was adjusted to keep the shell pressure regulator in range. One to two hours were required for a controlled depressuring.

5.6.4 Post-burn Evaluation

1. **Removal of the Sample from the Pressure Vessel:** The procedure followed was basically the inverse of that used to install the sample container into the pressure vessel.
2. **Cold-end-cap Removal:** The thermocouple leads were severed at the feed-through and the feed-through packing nut was removed. The circumferential clamp was removed, then the cold-end-cap was pulled out of the aluminum tube using a special fixture. The base of the fixture was a metal ring 3 cm thick and 3 cm high. Attached to this ring were two pillars 13 cm high that supported a cross-piece. Two holes were drilled into this cross-piece. Threaded rods of 0.6 cm (1/4 inch) diameter went through these holes and were threaded into holes drilled and tapped into the face of the end cap. By tightening nuts on each of the threaded rods, the end cap was gradually pulled from the aluminum tube.
3. **Hot-end-cap Removal:** The hot-end-cap was pulled with the fixture described above. Care was needed to try to avoid breaking the heater coils.
4. **Resin Casting:** The sample tube was marked to indicate the location of the surface of the coal or ash, and where sections would be cut. The aluminum tube was placed with the cold end down on an aluminum plate. The joint between the aluminum plate and tube was sealed with a bead of silicone sealant, which was allowed to cure. A hole was drilled and tapped into the surface of the aluminum tube at cold end for a 1/8-NPT fitting. A hose barb was threaded into this hole. A vacuum line was attached to the hose barb and a partial vacuum pulled. Acrylic resin (0.75 liter) was colored with yellow lead chromate pigment, then catalyzed. This resin was poured into the hot end

of the sample, and was pulled into the channel by the vacuum, gravity, and capillarity. The resin was allowed to set for a week. For some samples, the resin did not harden as well as desired owing to an unidentified interaction with the coal or pyrolysis liquids.

5. **Sectioning of the Coal Sample:** Circumferential grooves were cut through the outer aluminum tube, the inner and outer castings, and into the coal. These grooves were 3.8 cm center-to-center, and approximately 4 mm wide. The grooves were cut with a circular saw equipped with a silicon carbide abrasive masonry cutting blade 17.2 cm diameter. The sections were then sliced off with a band saw. The coal surfaces were smoothed with a stationary belt sander. Axial cuts through the center of exposed channel were made by a similar technique. Appropriate protective equipment was used during these operations, including a half-face dust mask suitable for use with silica and coal dust, safety goggles, gloves and coveralls.
6. **Recording the Channel:** The surfaces were labeled by cut, face, and orientation. The exposed surfaces were examined to reveal the character of the RC channel. Photographs were taken of the surface for archival purposes. The usefulness of these photographs is variable. The photographs clearly show the channels in samples with hard set resin, owing to the contrast of the brightly pigmented resin with the black coal. The photographs do not clearly show the channels in samples with soft set resin, since the gummy resin was impregnated with coal dust during sample sectioning. The shapes of the channels were also recorded by placing mylar drafting film over the coal sample and then tracing and labelling the various regions.

5.7 Summary

The overall objective of experimental program was to study RC channel propagation in solid coal samples. The primary design consideration was the selection of the method to observe the channel propagation. The desire to observe three-dimensional channels, along with the limited funds to carry out this research, dictated the choice of the more conventional method of using spot thermocouples along with post-burn sectioning, rather than infrared thermography or real-time X-ray imaging. The costs of sample acquisition had to be obtained from the original allocated funding for this research owing to the general reduction of funding for UCG during the course of this study (in previous work, samples had been provided free of charge by U.S.DOE contractors). Consequently, only one expedition could be made to obtain coal samples and the resulting quantity and quality of samples had to be accepted. A complex experimental apparatus was constructed to allow creation and study of small RC channels in solid coal. The apparatus consisted of a sample container, a pressure vessel, a gas-supply system, a gas clean up system, and a data-acquisition system. Extensive sample preparation, operating, and post-burn analysis procedures were developed.

CHAPTER VI

EXPERIMENT DESIGN AND RESULTS

6.1 Scope

Experiment design is the selection of values for the operating parameters for a set of experiments. After acquisition of the coal samples, it was apparent that the number of experiments that could be conducted would be severely constrained by the number of coal samples and the availability of funds for operating supplies. The number of experiments needed to develop and refine technique was unknown. Furthermore, considering the experiences of Lu (1980), Bell (1982), and Schloemer *et al.* (1988), it was reasonable to assume that an unknown, but nonzero, number of experiments would fail owing to factors other than the selected values of their operating parameters. The experiment design had to be modified to some extent as the experiment results became available if the maximum useful information was to be obtained from these experiments. These changes in the experiment design in turn, impacted the subsequent experiment results. Thus, the experiment design and results were mutually interdependent. Consequently, both experiment design and results will be presented in this chapter.

This chapter begins with a description of the experiment design. Next, the observations of the experiment runs are reported and summarized. This is followed by qualitative analysis of the relationship between front speed and operating parameters. A comparison of experimental results to theory is made. The experiment design then is critiqued. This is followed by a discussion of experimental results. The chapter ends with a summary.

6.2 Experiment Design

Reverse Combustion is affected by many controlled and uncontrolled parameters. The parameters that are regularly considered as controlled parameters are oxygen content and flow rate of the injected gas and combustion back-pressure. An additional possible control parameter is the presence or absence of a combustion promoter or inhibitor in the injected gas. It is desirable to understand the effect of each of these parameters on combustion front speed and channel shape. It is also desirable to understand how well the current theories of RC describe this behavior. Given a set of coal cores, taken from the same location of a coal seam, there are still a number of uncontrolled parameters, such as spatial variation in volatiles content, ash, coal microstructure, shale partings, etc. It is necessary to decide how to obtain the information desired, in the face of the interference from these uncontrolled parameters. This is the subject of the experiment design.

Given the limited number of coal samples available, along with the high operating

costs, and length of time required to run and analyze the experiments, it was necessary to choose an experimental strategy. The decision was made to conduct a factorial experiment, on the premise that this would yield more information about the response of the propagating reverse combustion channel to the whole set of parameters of interest. Typically it is more efficient to conduct a factorial experiment considering all variables at once, rather than considering the response to each parameter independently (Diamond, 1981). In general it also is more efficient to conduct two-level rather than multi-level experiments in the initial stages of investigation.

The first concern of a factorial experiment is selection of the ranges of the independent variables. Ideally, these ranges should be chosen so that the relative uncertainty of response to each variable is approximately the same. This could not be determined prior to running the experiments. Britten's work, as well as previous applied linear stability analyses, suggest a monotonic response to pressure and oxygen. Consequently, since the response was likely to be noisy, the range for each variable was set as large as possible, consistent with equipment limitations. This would maximize the signal-to-noise ratio.

From the applied linear stability analyses of RC, it was felt that pressures in excess of 2000 kPa would be required to produce channels sufficiently small to be observed in the laboratory, possibly in conjunction with elevated levels of oxygen. The apparatus had been designed for operating at pressures from 1400 kPa to 4000 kPa. The upper limit was set by the design pressure of the pressure vessel. The range of 2100 to 3600 kPa was selected for investigation. The apparatus had been designed to use any level of oxygen in the injected gas. Since the initial tests showed sufficiently small channels with 50 percent oxygen, the range was selected as 75 percent and 21 percent (air). Since there was no apparent need, use of pure oxygen was avoided for safety reasons. The flow meter had been sized to yield channels about 4 cm in diameter, according to the rough guidelines of the applied linear stability theory. This flow meter was originally intended to be just the oxygen meter, with an additional larger flow meter to be purchased for mixing in nitrogen. When the DOE contract laboratories were unable to supply coal cores, this additional purchase was canceled, to free up funds for hiring a drilling contractor. Consequently, the flow rate was limited to that of the original flow meter. Since the required flow rate was unknown beforehand, the flow rate was increased by equal increments on a log scale, with the highest flow rate being near the maximum of the flow meter. This would allow easier adjustment of experiment design if the required flow rate was found to be less than that originally planned.

Prior to running the experiments, consideration was given to using a combustion inhibitor. This was highly speculative, since it is uncertain that a level could be found that affected combustion without quenching it. For this to be cost effective, it would have had to be effective at low concentrations. Thus, equipment was designed to allow saturation of the gas stream at a level of approximately 0.5 mole percent.

In the ideal model of experiment design for seeking an optimal response, given unknown uncertainties and an unknown region of practical interest, the initial phase of experimentation should follow a random strategy. Diamond (1981) suggests that the number of random trials should be limited if they are expensive or difficult. Consequently, the random phase of the experiments was kept to an absolute minimum. As soon as the sample preparation technique

and operating procedure were developed, the first systematic phase of the experiment began.

In the ideal model of experiment design, the number of experiments is determined by the required level of confidence and the uncertainty of the experiments. If the number of trials is unreasonably high and costs are an important consideration, the experiment should not be undertaken, since an unacceptably uncertain outcome would result from using too few trials. With these experiments, almost all the expense and most of the time would be expended before such a decision could be made. Consequently, the number of experiments was set by the number of coal cores of sufficient quality. It was recognized that the result of this strategy might be a null decision, i.e. that the quality of the data was not sufficient to allow statistically valid conclusions. It was felt that the potential gain from getting a fuller picture of RC justified this risk-laden strategy.

The implemented strategy had three phases. The first was a random experiment phase to develop the sample-preparation technique and the operating procedures. This used short cores, considered unacceptable for the actual experiments. It was terminated as soon as adequate experimental techniques were developed. The second phase was a half-factorial design in three variables at two-levels. In a full-factorial design, all of the possible combinations of the levels of experiment parameters are tested. A half-factorial design consists of one-half of all of the possible combinations of the levels of the experiment parameters. This subset is selected to fully span the parameter space. The fourth variable, inhibitor, was not used. The third phase was planned as a half-factorial design in three variables at two-levels. This would either complete a half-factorial design in four variables at two levels, or if the intermediate results appeared uncertain, it would complete a full-factorial design in three-variables, with the fourth variable, inhibitor, dropped. After the second phase preliminary results appeared erratic, the prospect of investigating inhibitor was canceled. Efforts were directed toward investigation at higher flow rates. In addition, the intention was to run longer time-duration experiments, to obtain longer channels in comparison to the startup period. Thus, the half-factorial design did not complete the full factorial design. After the middle two experiments of the design suffered mishaps as explained below, one of the remaining two experiments was repositioned so that it would complete the corners of the factorial design. The experiment design is shown in Figure 3.1. In this figure, the parameter values for a given experiment are shown by the location of the experiment (as indicated by a dot labeled with the experiment number) in the parameter space. The three dimensions of the parameter space in this figure are combustion back-pressure, gas-flow rate, and oxygen content of the injected gas. Note that the gas flow rate, as measured in standard liters per minute (SLPM), is displayed on a log scale.

6.3 Observations

6.3.1 Phase One - Technique

This phase consisted of five experiments of which three were successful burns. The last experiment in this phase was incorporated into the factorial design of Phase Two. During

Figure 6.1: Experiment design.

this phase, the heater coil was doubled in size to ensure successful ignition. In addition, it was recognized that a rank of thermocouples would have to be devoted to the ignited coal face, to monitor ignition and the preliminary stages of combustion. Design improvements were made in the outer casting. The inner casting technique was improved to minimize gas bypassing along the periphery of the core.

6.3.2 Phase Two - First Block of Experiments

The second phase of the experimental program consisted of six experiments, including one common to Phase One. All six experiments were successful burns. The shapes detected were complex as described below. The only change in the experiment technique was that a more stringent testing of the thermocouples was implemented after the occurrence of thermocouple failures at low temperature.

A single channel arose during Experiment 5. It was uniform compared to others observed during the course of these experiments. It propagated aligned with the cleat structure which was roughly aligned with the axis of the sample. The channel was filled with ash, not char as anticipated by the applied linear stability model. The tip of the channel consisted of char, with cracks radiating inward toward the channel. The cracks became wider toward the channel core. A photograph of a cross-section of the coal core near the tip of the channel is shown in Figure 6.2. The cracks are clearly visible because they are filled with pigmented resin.

Initially a single large channel arose in Experiment 6. This channel was wedge-shaped, narrowing toward the tip, with the boundaries aligned with the cleat structure which was at a 45 degree angle to the axis of the core. A smaller channel split off approximately half way down the wedge. It propagated perpendicular to the face of the main channel. The channel was ash-filled.

Initially a single channel arose in Experiment 7. The channel passed directly over several thermocouples. When doing so, the channel widened. Subsequently, the channel split into two, then three channels. The material inside the channel was irregular, being ash during ignition, char at places, and burned-out void spaces near the tip of the channel.

A single large channel arose in Experiment 8. The ignition face was large and the channel gradually decreased in cross-section. The channel had open void spaces.

In Experiment 9, ignition occurred at three spots. These three channels merged into two channels before splitting back to three. The channels had open void spaces in them.

In Experiment 10, a relatively uniform channel arose. The core of the channel was filled with ash, with the tip consisting of cracked char.

6.3.3 Phase Three - Second Block of Experiments.

The third phase of the experiments consisted of six experiments. Four of these were successful burns. One failed to ignite and another either had a large oxidant bypass, or its channel operated in a fuel-limited mode due to the high ash content. The cores were lower quality coal than the first block and typically had large claylike-shale partings up to 5

Figure 6.2: Photograph of a cross-section of coal core for Experiment 5 near the channel tip.

cm thick. Mishaps occurred during attempts to run experiments for longer duration, which resulted in two successful burns producing no usable data.

A vertically flattened single channel developed in Experiment 11. This sample had large shale partings. The channel was aligned with the bedding planes of the coal. The channel core consisted of ash.

A similar channel arose in Experiment 12. This experiment suffered from large amounts of uncombusted oxygen in the product gas.

In Experiment 13 after a successful burn, the filters began to plug. During changing out a set of filters, a relief valve popped, letting a greatly increased gas flow pass through the sample. The nature of this incident was not immediately recognized, and consequently the operator response was inappropriate. The channel burned the remaining length of the sample, rendering the data unanalyzable.

In an attempt to let a longer channel form, Experiment 14 was not immediately terminated after the combustion front reached the last rank of thermocouples. Upon opening the sample container, the channel was discovered to have burned through.

Experiment 15 was an unexpectedly short run, compared to the previous experiments. It was at the most extreme conditions of pressure, oxygen concentration, and flow rate. One channel formed. It was irregular, deforming as it passed over the thermocouples. The channel was filled with ash.

After repeated and prolonged efforts, ignition was not sustained for Experiment 16. Upon opening of the sample, it appeared that gas had preferentially passed through an incombustible claylike-shale parting.

Given the low quality of the remaining cores, the lack of funds to obtain additional coal cores, and the lack of funds for additional purchases of operating supplies, the experiment design was terminated after Experiment 16.

6.3.4 Summary of Observations

In summary, most of the channels were filled with ash or even had void spaces. Even when char was found, it had small cracks at the tip that increased in size toward the main channel. Thermocouples with a diameter of 16 mm in holes with 19.8 mm diameter affected the shape of the channel. The temperature versus time profiles typically declined nearly as abruptly as they rose, showing substantial conductive heat losses. Many of the channels propagated far slower than expected.

The experimental technique for sample preparation proceeded rapidly allowing a maximum number of the cores to be used for actual experiments. Twelve experiments were successful as RC burns. Nine of these yielded useful channel shapes. Two failed to ignite because of interference of a shale parting. One channel would probably have become extinct if the experiment were continued. Operation was more erratic during the second block of experiments than the first due to a combination of poorer quality coal and the attempt to run experiments longer.

The equipment performed well. The only major equipment problem was plugging of the product-gas filters with entrained tar. This occurred despite large size knock-out pots.

An essential feature of the apparatus was the ability to change out filters on-line. The heat exchangers showed no signs of plugging. They were oversized for these experiments since most of the combustion heat was absorbed by conductive losses to the coal. In retrospect, the maximum flow rate possible with the oxidant flow controller was too small, and more flexibility is needed in this area. The use of small thermocouples did not prevent them from affecting the front. The reason for this is a combination of flow distortions from drilling a hole into the coal which leads to a leakage path and to the higher thermal conductivity of the metal sheath. Future experiments possibly should use infrared thermography and accept the associated limitations. Another possibility is to analyze the mineral matter of the coke or ash for thermally induced changes. This technique has been recently applied to various aspects of UCG (Schloemer and Wagner, 1988; Wolf *et al.*, 1987) However, this technique does not yield gas phase combustion temperatures, nor does it give real-time temperature data. The sample container system performed well, and with appropriate modifications to the sample placement system it would be compatible with the use of a real time X-ray imaging system. The gas chromatograph performed well, however its data are of little value for channels as small as these, since it is impossible to accurately extract the gas composition at the channel tip from the bulk gas measurement. The data generated from gas chromatographs are more useful for monitoring field tests than for experiments, since in general it is not possible to excavate the channel and find its shape. Gas chromatograph data allow a material balance which can be translated into a channel shape only through the use of assumptions whose validity laboratory experiments are attempting to assess. The system is in good operating condition for future projects in reverse combustion in coal.

6.4 Results

The RC data are summarized in Table 6.1. The principal source of uncertainty for the superficial gas flux is variation from the desired set-point due to process upset. The magnitude of these excursions is less than 0.4 kg/hr/m^2 , and the time of process upset is no more than a few percent of the total experiment run. The principal source of uncertainty for the observed pressure is the accuracy of the pressure transducer which is approximately $\pm 50 \text{ kPa}$. The principal source of uncertainty for the oxygen content is the accuracy with which the gas were mixed. This is certified to be no more than 0.5 volume percent. The principal source of uncertainty for the front speed measurement is whether the front speed was uniform. It is debatable whether this type of uncertainty is most appropriately expressed as an absolute uncertainty of 3 cm/hr , or a relative uncertainty of 36% (based on the replicate experiments 7 and 8). To obtain more conservative conclusions, the assumption is made that the uncertainty is on an absolute basis. The front temperature is the highest observed temperature. Since there is no assurance that the hot tip of the channel passed through the thermocouple, the reported front temperature may be several hundred degrees cooler than the highest combustion front temperature. The principal source of uncertainty for the calculated dependent data is permeability characteristics of the coal. These uncertainties are difficult to quantify. The uncertainty of the surface Laplacian of the the gas flux is larger than that of the normal component of the gas flux at the channel tip, since evaluation of the surface

Table 6.1
Summary of reverse combustion experimental data
for the end of the run

Experiment	5	6	7
Independent Variables:			
Superficial Gas Flux, kg/hr/m ²	4.3 ± 0.4	16.7 ± 0.4	8.4 ± 0.4
Pressure, kPa	3540 ± 50	2170 ± 50	2850 ± 50
Oxygen Content, mole percent	75.0 ± 0.5	75.0 ± 0.5	50.0 ± 0.5
Directly Determined Dependent Data:			
Front Velocity, cm/hr	2.1 ± 3	2.9 ± 3	8.5 ± 3
Lateral Character:	Cross-section Decreasing	Cross-section Decreasing	Extremely Irregular
Front Temperature, K	725	873	915
Calculated Dependent Data:			
Gas Flux at Tip, kg/hr/m ²	11	45	21
Surface Laplacian of Gas Flux at Tip, MM kg/hr/m ⁴	-0.3	-1.7	-0.95

Table 6.1, Continued

Experiment	8	9	10
Independent Variables:			
Superficial Gas Flux, kg/hr/m ²	8.4 ± 0.4	16.7 ± 0.4	4.3 ± 0.4
Pressure, kPa	2900 ± 50	3620 ± 50	2220 ± 50
Oxygen Content, mole percent	50.0 ± 0.5	21.0 ± 0.5	21.0 ± 0.5
Directly Determined Dependent Data:			
Front Velocity, cm/hr	5.4 ± 3	3.5 ± 3	1.2 ± 3
Lateral Character:	Cross-section Decreasing	Cross-section Decreasing	Cross-section Uniform
Front Temperature, K	922	—	—
Calculated Dependent Data:			
Gas Flux at Tip, kg/hr/m ²	25	44	15
Surface Laplacian of Gas Flux at Tip, MM kg/hr/m ⁴	-1.3	-0.6	-0.4

Table 6.1, Continued

Experiment	11	12	15
Independent Variables:			
Superficial Gas Flux, kg/hr/m ²	16.7 ± 0.4	8.4 ± 0.4	16.7 ± 0.4
Pressure, kPa	2150 ± 50	2180 ± 50	3565 ± 50
Oxygen Content, mole percent	21.0 ± 0.5	75.0 ± 0.5	75.0 ± 0.5
Directly Determined Dependent Data:			
Front Velocity, cm/hr	1.7 ± 3	1.7 ± 3	15 ± 3
Lateral Character:	Cross-section Decreasing	Cross-section Decreasing	Irregular
Front Temperature, K	770	-	833
Calculated Dependent Data:			
Gas Flux at Tip, kg/hr/m ²	72	31	47
Surface Laplacian of Gas Flux at Tip, MM kg/hr/m ⁴	-1.4	-0.5	-2.5

Table 6.2
Product gas composition at end of experiment run

Experiment	5	6	7	8	9
Component, mole percent					
Carbon Dioxide	48.2	51.3	32.0	17.9	8.8
Hydrogen	0.6	0.5	1.0	4.9	0.4
Ethane	0.0	0.0	0.0	0.4	0.0
Oxygen	7.4	5.9	9.5	0.9	7.9
Nitrogen	39.5	40.4	55.5	67.8	80.4
Methane	1.1	0.3	0.8	5.0	0.2
Carbon Monoxide	3.2	1.6	1.3	3.1	0.8

Experiment	10	11	12	15
Component, mole percent				
Carbon Dioxide	8.8	8.5	10.0	51.6
Hydrogen	0.4	0.1	0.1	1.2
Ethane	0.0	0.0	0.0	0.0
Oxygen	7.6	9.1	57.1	2.6
Nitrogen	81.3	81.3	31.3	40.8
Methane	0.3	0.1	0.1	0.8
Carbon Monoxide	1.5	0.8	1.4	3.0

Laplacian requires an additional numerical differentiation. The gas chromatograph analyses of the product gas at the end of the experiments are given in Table 6.2. The uncertainty of these measurements with respect to the composition of the sampled gas is 0.1 percent, based on multiple measurements. The uncertainty of these measurements with respect to the composition of the gas at the tip of the combustion channel is so large that this information is not useful for obtaining an understanding of the combustion process. A representation of the response of the propagation velocity to the controlled parameters is given by Figure 6.3. In this figure, the location of each sphere is a given by the experiment design. The volume of the spheres is proportional to the propagation velocity, hence the propagation velocity of Experiment 15 is much greater than that of Experiment 10. This figure shows that there is a strong synergistic response between pressure, gas flow and oxygen content. Only the high levels of each of the parameters for Experiment 15 yielded propagation velocities compatible with a priori estimates based on the adapted linear stability estimations. The channels were too irregular in nature to justify an identification of a channel diameter.

To glean the maximum information possible from these data, appropriate pairwise

Figure 6.3: Response of propagation velocity.

comparisons of experimental runs were made to test for trends. The trends expected from theory were that higher gas flows, higher oxygen content, and higher pressures would all favor greater propagation velocity at elevated pressures. Due to adjustment of the experimental design to focus on higher fluxes and the occurrence of mishaps and failed experiments that were reasonably explained by behavior other than response to the experimental parameters, not as many pairwise comparisons could be made as with a full factorial design in 3 parameters using the same number of experiments. The possible pairs that had the same levels of two out of three parameters are listed below:

Variable gas flow rate: Comparing Experiments (6 vs. 12), (10 vs. 11), and (5 vs. 15) consistently indicates that higher gas flows will lead to higher propagation velocities, however, the variation observed between (10 vs. 11) and (6 vs. 12) are probably not statistically significant.

Variable oxygen content: Comparing Experiments (6 vs. 11), and (9 vs. 15) consistently indicates that higher oxygen content of the injected gas will lead to higher propagation velocities, however, the variation observed between (6 vs. 11) is probably not statistically significant.

Variable combustion pressure: Comparing Experiments (6 vs. 15), and (9 vs. 11) consistently indicates higher combustion pressure will lead to higher propagation velocities, however the variation observed between (9 vs. 11) is probably not statistically significant.

Thus, all of the postulated responses of the system were qualitatively confirmed without any conflicting pairs of experiments. However, the variation observed in many of these pairs is slight compared to the uncertainty in the data. Given this uncertainty, the observed response is better characterized by concluding that the speed was slow if any of the variables was at a low level. As stated earlier, the extremely strong response of experiment 15 suggests a synergistic, nonlinear response. Examination of the experiments that could not be directly compared is in agreement with this trend.

6.5 Comparison with Theory

Given the difficulty of determining a measure of the channel diameter, direct comparison of these experiments with lumped-parameter theories such as Mui and Miller's (1983) single large finger model or the applied linear stability theory of Gunn, Krantz and coworkers tests (Krantz and Gunn, 1977; Gunn and Krantz, 1980; Okutan, 1981; Krantz and Gunn, 1981; Britten *et al.*, 1982; Britten 1984) could not be made. Instead, a differentially-based analysis of the propagation of the reverse combustion channel tip was attempted, to permit direct comparison with the underlying planar models of reverse combustion from which these models were developed.

The purpose of thermocouples was to track the passage of the combustion front. There is no guarantee that a thermocouple will pass directly through the front, nor is there accurate means to correct the thermocouple response to the leading tip of the combustion front. In any case, the thermocouples frequently failed before the maximum combustion temperature was reached. Consequently, it is not possible to use the direct method of data-analysis outlined in Chapter 4. Thus, the indirect technique outlined in the same chapter was used.

The analysis was applied to the end of the experiment run.

To obtain an estimate of the flux to the channel tip, three-dimensional finite-difference flow calculations based on uniform zoned permeabilities were carried out. As a first step, a digitized representation of the channel shape was input into the computer based on the tracings made of the channel cross-section. The problem was solved iteratively using the successive over-relaxation method combined with a mesh-refinement procedure. The final uniform mesh of 32 by 32 by 64 corresponded to a spatial resolution of 0.5 cm. The computations were performed on a MacIntosh personal computer using a Microsoft QuickBasic program developed for this problem. The runs were carried out until there was no variation in the cross-section averaged flux. This took approximately 15 hours of computation per experiment. Most of the program involved the input of the channel shape, the graphical display of the flow profiles, and the post-processing of the resulting pressure profiles to obtain the flux to the channel surface. The programs are given in Appendix B.

Given this estimate of the flux, the assumption of small heat loss was made initially. The front temperatures then were determined by a one-dimensional heat balance at the tip. With the aid of the subroutines listed in Appendix A, an Arrhenius plot of these data could be made. This plot is given in Figure 6.4. It shows no apparent pattern from which an activation energy can be obtained, which was not unexpected given that this initial attempt was based on the assumption of negligible heat loss, whereas the channels were clearly dominated by heat loss. What is more disheartening is that the variation does not correlate with the estimates of heat loss qualitatively gained from application of the model in Chapter 4. Furthermore, estimation of the sensitivity of this analysis to the uncertainty associated with the estimated parameters indicates that it would be unable to discriminate between any of the previously postulated values for the activation energy. Consequently, further quantitative comparison of these experiments was not deemed worthwhile. Arrhenius kinetic parameters are a means of identifying the variation of the reaction temperature with process parameters. Without the ability to extract kinetic parameters, the most that can be identified is an approximate reaction temperature. These data are reasonably fit by a reaction temperature of 1100 C. Given the reaction temperature, the propagation velocity can be determined by an appropriate heat balance if heat losses are neglected. This body of experimental evidence is neither able to confirm nor reject the numerical results of existing continuum models of reverse combustion channel propagation. In addition it suggests that this could be accomplished only by an unreasonable number of experiments of this type for all the variables of practical interest.

6.6 Critique of Experiment Design

The factorial design allowed identification of the most promising region of interest for further studies, within the part of the parameter space searched. In addition it produced results that were able to qualitatively confirm expected trends. In this way, the experiment strategy was prudent, successful, and necessary.

The factorial design did not produce experiments that allowed confirmation or rejection of any quantitative aspect of the motivating theory. In fact, most of the experiments were

Figure 6.4: Arrhenius plot based on front speed.

characterized by slow channel growth. This regime is neither of interest to the practice of RC linking, nor is it particularly relevant to understanding the propagation of large channels.

With the number of samples secured, it was unrealistic to expect that both the goal of locating the region of feasible operation and the goal of exploring this region sufficiently to allow quantitative analysis could be met. Since finding the region of interest had to precede experiments designed to test the theory, it is disheartening, but not surprising that this goal was not reached. Sequential analysis of the data would have assisted in making a decision to terminate the experiments, had this alternative been seriously considered. Without a complete picture of the experimental results, construction of the tools for analysis would have been more difficult.

The inability to correlate the data does not invalidate the experiments. They did achieve the propagation of actual reverse combustion channels in solid coal. Although they can not address the quantitative results of RC theory, they do shed light on the assumptions underlying these theories as will be discussed in the next section.

6.7 Discussion

One of the most apparent observations about the reverse combustion channels was that for the most part they were filled with ash, with little char. The leading surface was char but cracks penetrated through this char toward the unaltered coal. This was also observed in the early experiments of Breidung (1982). The existing theory does not assist us in understanding this. For char-burning, Britten (1984) constructed a planar asymptotic analysis based on an infinite Lewis number with gas-solid kinetics. Despite whatever theoretical interest Britten's model has, it does not appear to address the points of most concern about char-burning for reverse combustion linking in coal. The essential differences between Britten's solid-gas combustion model and his homogeneous gas-phase combustion model were that the reaction contained a phase-change and that the kinetics were considered first-order in oxygen only, hence the pressure dependence was only first-order, rather than second. The assumption of uniform infinite Lewis number appears implausible given the fissured surface of the char. It would seem that with char-combustion, a model such as proposed by Corlett and Carson would be necessary. Britten's model provides no criteria for when only gas phase reaction should occur, or when both char-burning and gas-phase burning would occur. In fact, it suggests that operation at elevated pressures would lead to low temperatures, and hence only a mild pyrolysis. Hence, it does not aid in delimiting the range of accuracy of the adapted linear stability analyses. Based on these experiments and the recent work in Germany (Schlomer *et al.*, 1988), this range does not include operation at pressures above 2000 kPa. It appears that the premixed, planar models of RC need to be supplemented with a fissured, diffusion-limited combustion model. This is a topic worthy of future research.

The channel shape and path were highly dependent on small scale structures, such as the crack and coal microstructure. In fact a large part of the difficulty associated with correlating the experiments rests with the inaccuracy of the flow calculations based on homogeneous zoned permeability. It can be argued that channels formed under low pressure would be large enough that they would be insensitive to such small features, however it could

be countered that intermediate scale structures (cracks larger than cleats but smaller than identifiable fractures) and fractures in the coal seam would then determine the path. The literature on RC in coal does appear to have treated the distribution of crack structures in coal. The constancy of propagation velocity with respect to the total flow rate noted in field test was not observed in these experiments. Given the likelihood, or at least possibility that the differential flow patterns in coal are highly irregular, it would be useful to reflect on what characteristics of a reverse combustion front are necessary to provide this constancy in the face of large finite perturbations in the gas flux. This will be considered in Chapter 8. From the lack of constancy, it is reasonable to conclude that these channels are not dynamically similar to those created in low pressure field tests. Comparing these experiments to those of Schlomer *et al.*, (1988), suggests that the former experiments were conducted near the extinction limit for RC channel propagation.

6.8 Summary

The experiment design consisted of three phases. The first was a random design while experimental technique was developed. The second was a half-factorial design to quickly survey the parameters. The third was a half-factorial design to more fully explore the upper flow rate range of the experiments. No further phases of experimentation were carried out because of exhaustion of coal cores of sufficient quality. The experimental observations were reported and summarized. Most of the channels were ash-filled, irregular shaped and slowly propagating. A qualitative analysis of the experimental data provided support for the contention that higher pressure, oxygen content, and combustion back-pressure favor faster propagation of the channels. Quantitative analysis of the data by the indirect method described in Chapter 4 was not fruitful in yielding kinetic parameters. The most that can be said is that the combustion temperature was on the order of 1100 C. The experiment design was critiqued. The argument was advanced that better experiment design would not have made up for the small number of samples and the undersized-sized flow meter. The experimental results were then discussed. Several arguments were advanced regarding the interpretation of experimental evidence. The first was that existing theory of char burning provides scant guidance to the practice of reverse combustion linking. The second argument was that the effects of heterogeneities may be important for a broad range of length scales, given the existence of intermediate scale structures in the coal (larger than cleats but smaller than identifiable fractures). Finally, it was suggested that these RC channels did not achieve dynamic similarity with those created in field tests.

CHAPTER VII

FORMULATION OF A TRANSIENT, NUMERICAL SIMULATION FOR REVERSE COMBUSTION CHANNEL EVOLUTION

7.1 Scope

This chapter describes the formulation of a transient, numerical, two-dimensional simulation of reverse combustion channel evolution in a homogeneous porous medium. First, the motivation for such a simulation is given. Then the strategy pursued is outlined and the assumptions are discussed. The core of the chapter describes the implementation of the simulation. The simulation is appraised and directions for further development are indicated. The chapter ends with a summary. A case study using this simulation is given in Chapter 8.

7.2 Motivation

The nature of a planar RC front propagating through a porous medium has been analyzed by both numerical and analytical models. Linear stability analysis (LSA) of these models shows that a planar RC front is unstable under most conditions and therefore will not persist. The time-dependent shape of a channel is influenced by phenomena, such as lateral heat transport, extinction, and variation of gas flux with local channel geometry, that are not present in a planar model of reverse combustion, and that are represented only by linearized forms in LSA. Thus, much of the physics is lost in this linearization such that what remains is insufficient to describe the character of the long-term evolution of the channel. To advance the understanding of the evolution of RC channels, a multi-dimensional simulation of RC will be formulated and studied. It allows exploration of the effects of phenomena such as extinction that are not adequately understood. It serves as a foundation for the development of a more complete model of RC channel evolution. This simulation is transient, and can consider a broad class of boundary conditions on the flow. The chief advantage of this flexibility is that it leads to more robust models of RC since they can not rely upon specific characteristics of geometrically simpler configurations.

7.3 Strategy

Although in the future, given further understanding of the physics of RC, it may be desirable to develop a full three-dimensional model of RC channel evolution, at this time it is more reasonable to develop and exploit two-dimensional models of RC. This simula-

tion is restricted to exploring a two-dimensional slab geometry analog of RC. Thus, a two-dimensional cylindrical geometry (no variation in a cartesian z-coordinate) is considered, not a two-dimensional rotational geometry (no variation in a polar angular coordinate). The phrase "slab geometry" is used since cylindrical geometry might be misinterpreted as referring to a circular cylinder. The term "planar" is avoided since we wish to emphasize that the assumption being invoked is uniformity in the z-coordinate, not negligible thickness. Attention has been directed to the slab geometry for four reasons. First, the numerical implementation of this model is more straightforward in this geometry. Second, long-length-scale characteristics of a reverse combustion channel in a relatively thin coal seam divided by shale partings are better represented by a slab geometry. Third, laboratory experiments in particle board in this geometry (Visser *et al.*, 1985) generated interesting, geometrically complicated structures with which to compare simulation results. Fourth, the temporal evolution of an unstable, initially rotationally symmetric front would not be expected to preserve rotational symmetry, whereas variations in the z-direction could be limited by physical constraints.

Analytical solution of the full multi-dimensional model equations is not possible. Direct numerical solution of the multi-dimensional model equations is a difficult task. The computational effort would be large, even if attention is restricted to two-dimensional forms of the equations. The principal difficulty is that the preheat and combustion zones are very thin compared to the size of a channel. Consequently, this study has pursued a multi-scale approach to creating a simulation of RC channel propagation.

The flow distribution on the channel surface is calculated at each time step assuming that the channel shape is frozen. The combustion process is taken as a free boundary condition for the displacement of a low permeability region (coal) with a high permeability region (char). This boundary condition describes the deformation of the channel surface as a function of the gas-flux distribution. This approach has been taken in previous multi-dimensional models of RC channel evolution by Britten (1984), Monteiller (1986), and Fuhrmann (1987).

In this study, the flow distribution is determined numerically, by applying the boundary-element method. Numerical solution allows the transient study of the evolution of RC channels. It provides a great degree of flexibility in exploring the effects of the flow geometry. The boundary-element method provides advantages over spatial methods that will be discussed in Subsection 7.5.3.2 The channel shape as well as the flux and pressure distributions on the channel surface and other boundaries are represented with cubic hermetian shape functions. This allows (and to some degree forces) a smooth (continuous in first tangential derivative) representation of the channel surface, as would be expected to result from a continuum model for reverse combustion.

The specification of the model for RC, that is the combustion law describing the front propagation, is isolated from the rest of the simulation. The model can be easily changed without modification of the simulation itself. To a large degree, this simulation is not tied to RC, but could simulate, or be easily modified to simulate, a variety of other partially analogous problems, from solidification of microcrystalline wax from melt, to viscous displacement of oil with water.

The model is implemented in FORTRAN 77 (with VAX extensions). This language was chosen for this simulation because of the ease of porting code from one computer to

another, combined with the wide availability of IMSL MATH/LIBRARY routines to handle general numerical tasks, of which the most important was solution of sets of linear algebraic equations.

7.4 Assumptions

In regard to RC channel propagation for underground coal gasification, the principal assumptions of this simulation are that reverse combustion propagation is adequately represented by a continuum model for front propagation, that flow through coal is adequately represented by permeation flow rather than necessitating consideration of flow in discrete fractures, and that the channel surface has no contact resistance to flow. Contact resistance could occur if a thin, yet very low permeability layer formed at the channel surface through either condensation of tars or plastic deformation of the coal from a stress concentration at channel surface. These assumptions are not satisfied for RC at great depths, or in swelling coals. For RC in nonswelling, shallow coal seams typical of the western United States and Canada, these assumptions are reasonable, but have not been indisputably verified by experiments or field tests.

The following assumptions are made with regard to the flow calculations. The flow is essentially two-dimensional (Dupuit flow, i.e. smooth variation in seam thickness and orientation, resulting in negligible components of velocity in the direction perpendicular to the seam.) The permeability characteristics of the porous media are assumed to be governed by homogeneous zoned transmissivity (permeability times thickness). Thus, any spatial variation in viscosity, density, or permeability coefficient within a zone is neglected. As currently implemented, isotropic permeability is assumed, although orthotropic (orthogonal anisotropic) permeability could be handled within the framework of this simulation by suitable stretching and rotation of coordinates, as illustrated by Liggett and Liu (1983). The porous media are considered rigid, with no storativity. Storativity (specific storage times thickness) is a composite measure of fluid and porous medium compressibility. The degree of saturation of the porous media with water is assumed to be constant. Buoyancy effects are neglected. Darcy's law is assumed to apply. The flow calculations are done with time frozen. Given this set of assumptions, the pressure within a zone at any instant in time is governed by Laplace's equation, with the gas flux proportional to the negative of the gradient in the pressure. The compatibility conditions between zones are continuity of the pressure and the normal component of the flux. Thus, negligible contact resistance is assumed. The change in gas flux due to devolatilization is absorbed into the transmissivity of the char zone. Any spatial variation in this augmentation of the flux is ignored.

The following assumptions are made with regard to the propagation of the front. The propagation of the front is assumed to be adequately modeled by a free boundary condition model. The front is assumed to be smooth. The front propagation is normal, thus the front propagation model must determine the normal component of the front velocity. Any anisotropic effects must be handled by the combustion model.

The following restrictions apply to the geometric description of the porous media. The front and any internal or external boundaries are described by closed, non-intersecting con-

tours. The maximum number of zones is fixed, but the specific number can be easily adjusted by changing a parameter in the program code. The complete flow field must be surrounded by an external boundary. Zones need not be simply connected, but can contain holes. These holes can either enclose additional zones, or can be external boundaries. No provisions are made for the intersection of the propagating front with other zonal boundaries. Along any external boundaries, the flow distribution or pressure distribution can be prescribed. These prescribed distributions need not be spatially uniform. The different bounding surfaces can use flow or pressure boundary conditions, however, at least one surface must have a prescribed pressure distribution to generate a unique solution. The consequence of not observing this last restriction has not been tested. Mixed boundary conditions along a particular contour are not allowed. This restriction is not mathematical in nature; earlier versions of the simulation allowed mixed boundary conditions. Rather this restriction is imposed to simplify the expression of the boundary conditions.

7.5 Implementation

The transient, numerical simulation is implemented in a FORTRAN subroutine called SMOOTH. A block diagram for the simulation is given in Figure 7.1. The implementation of each of the tasks shown on this diagram is discussed in this section.

7.5.1 Reverse Combustion Model

The simulation handles the flux calculations, the propagation of the front, and graphical output of evolving channel shape. The actual model of RC channel evolution is given by the front-speed relationship. This section does not focus on the characteristics of a particular model, but rather the manner in which a model is implemented for use with this simulation. The features of the model used in Chapter 8 are discussed in that chapter.

The front-speed relationship must return the front speed at points along the front based on gas-flux distribution on the front, the pressure distribution on the front, and the shape of the front. No other information is available from the simulation. In general, the front speed at any point depends on shape, flux, and pressure distribution along the whole front, as well as the history of these variables. Although such an approach would be exact, it would be extremely difficult to implement and computationally intensive. Previous models of RC front propagation used in channel simulations have depended solely on the normal component of the flux at a particular point at that particular time. This approach is easily implemented, but is an inadequate representation of RC since it lacks any spatially dependent stabilizing effects. The behavior of such a front could never be accurately captured by a simulation, since it would not be smooth at any length scale. Here we propose a middle ground between these extremes. The front-speed relationship is localized, that is, it depends only on intrinsic variables evaluated at that point, but these variables are not restricted to only the normal component of the flux. In general, the front-speed relationship depends on the flux, the pressure, the radius of curvature of the front, and the tangential derivatives of each of these variables. The relationship of these components to physical processes and the relative

Figure 7.1: Block flow diagram for SMOOTH, a reverse combustion channel evolution simulation. The subroutine names are given in SMALL CAPS type style.

importance of these processes have already been discussed in Chapter 4 and will be further discussed in Chapter 8.

The model is implemented in the same manner as used by IMSL library routines. A driver routine is written that calls the simulation subroutine, SMOOTH, passing the names of the subroutines that define the model as external procedures. These routines are called by the model as needed. This organization is effective in isolating the statement of the model from the computational details required to implement the simulation. Any additional parameters needed by the model are passed from the driver routine to the external procedures through common blocks. The first routine, MODELINFOSUB, is called at the start of the front propagation process for each time step. This subroutine returns scale factors needed to convert from the dimensionless units used in the calculations to the dimensional units used by the model. It also carries out any initialization tasks needed by the model. The second subroutine, MODELSUB, is called with the number of points that are to be advanced. This subroutine actually calculates the front speed at these points and communicates this information to the simulation. A small set of interface functions is provided that allows the model to obtain the flux, pressure, or curvature at the various points. These interface functions allow the model to be formulated without concern for the internal workings of the simulation.

7.5.2 Simulation Procedure

The subroutine SMOOTH controls the overall process carried out to simulate RC channel evolution. After carrying out initialization activities, it enters the main loop. For each time step, the front is advanced. Periodically the shape of the front is recorded and the front is remeshed. The front advancement process consists of calculating the flux distribution to the channel surface, converting this information into front speed by invoking the front speed model, then moving the front. The main loop is repeated until either the desired ending time for the run is reached, or else the maximum number of nodes has been used, and hence the desired node spacing can no longer be maintained.

7.5.3 Flow Calculations

The purpose of the flow calculations is to determine the gas-flux distribution on the channel surface. Information away from the front is not needed.

7.5.3.1 Description of Flow in Porous Media

The gas flow in the porous media for reverse combustion is normally modeled using Darcy's law:

$$N_g = -\frac{\kappa \rho_g}{\mu_g} \nabla P \quad (7.1)$$

where N_g is the gas flux, κ is the permeability coefficient, ρ_g is the gas density, μ_g is the gas viscosity, and P is the pressure. For simplicity, we neglect any compressibility of the gas. The density is evaluated at the pressure of the production well. The principal effect of

this assumption is that the pressure required to force a given flow between two well bores would be overestimated. Some distortion of the flow field is also introduced. In addition, we have neglected hydrostatic and buoyancy effects. The porous media are divided into multiple zones (for example, char and virgin coal).

Generation or consumption of gas occurs only in the preheat and combustion zones. These zones are sufficiently thin that they may be taken as just the interface between the coal and char zones in modeling the flow. Therefore, the equation of continuity inside a flow zone takes a simple form:

$$\nabla \cdot \mathbf{N}_g = 0. \quad (7.2)$$

When the previous two equations are combined, the following equation arises:

$$\nabla \cdot \sigma \nabla P = 0 \quad (7.3)$$

where $\sigma = \frac{\kappa h_p}{\nu}$ is the transmissivity of the medium. Internal to each zone, it is reasonable to assume that the permeability of the porous media and the gas density and viscosity are approximately constant, although the permeability could be directionally dependent. Through suitable rotation and stretching of coordinates (Liggett and Liu, 1983), orthogonal anisotropic (orthotropic) transmittivity could be handled, however this feature is not implemented. The simulation is restricted to isotropic permeability. Hence, the problem reduces to solving Laplace's equation inside each zone:

$$\nabla^2 P = 0 \quad (7.4)$$

At the boundaries between zones, the pressure is continuous. The other necessary interzonal compatibility condition is that the normal component of the gas flux (in the global, unstrained coordinate system) is constant. Here we have absorbed the change in gas flux at the coal-char interface due to devolatilization into the effective transmissivity of the char. The error in doing so is not significant, but greatly simplifies the solution of the problem, since it completely decouples the flow problem from the combustion problem. When combined with the external boundary conditions, the problem is well-posed.

7.5.3.2 Choice of Numerical Method

To handle arbitrary coal seam geometry and initial channel shape, the problem must be solved numerically. Since only the gas flux distribution on the channel surface is needed, the Boundary Element Method (BEM), also known as the Boundary Integral Equation Method (BIEM), is well suited to this problem. In the BEM, only the boundaries and interfaces between the zones are discretized. Thus, with this model for which the shape of the channel surface changes with each time step, little additional overhead is required with the BEM, unlike with the Finite Element Method where a new two-dimensional mesh would have to be generated with each step. The Finite Difference Method would be poorly suited to this problem because of the irregular geometry of the channel surface and coal seam. It would require the use of a boundary conforming grid. The adaptive generation of such a grid could be as computationally expensive as the actual solution of Laplace's equation, depending on the mesh generation technique. Some adaptive grid generation methods involve solution

of a pair of elliptical differential equations, each of which is more complex than Laplace's equation. The selection of the BEM yielded benefits with respect to the time required for program development and execution.

7.5.3.3 The Boundary Element Method for Laplace's Equation

In the following paragraph the basis of the BEM for Laplace's equation will be briefly presented. The basic development is abstracted from Liggett and Liu (1983). A suitable starting point is Green's second identity

$$\int_{Domain} (U \nabla^2 V - V \nabla^2 U) d\tau = \int_{Surface} \left(U \frac{\partial V}{\partial n} - V \frac{\partial U}{\partial n} \right) dA \quad (7.5)$$

where we have employed the shorthand notation

$$\nabla V \cdot \mathbf{n} = \frac{\partial V}{\partial n} \quad (7.6)$$

$$\nabla U \cdot \mathbf{n} = \frac{\partial U}{\partial n}. \quad (7.7)$$

At this point we will restrict this development to two-dimensional problems (where there is no change along the z-coordinate direction). Select U as the pressure, P . We seek a functional form of P that satisfies Laplace's equation inside the domain. Therefore, we require that

$$\nabla^2 U = \nabla^2 P = 0 \quad (7.8)$$

inside the domain. Define $V \equiv \ln r$ where r is the distance to a singular point located on the surface (boundary) of the domain. From the definition of V , inside the domain,

$$\nabla^2 V = 0 \quad (7.9)$$

which is easily verified when the Laplacian operator is expressed in cylindrical coordinates. With these choices, the L.4.5. of Equation 7.5, (the volume integral), identically vanishes leaving the following boundary integral equation:

$$\int_{Surface} \left(\frac{P}{r} \frac{\partial r}{\partial n} - \ln r \frac{\partial P}{\partial n} \right) ds = 0. \quad (7.10)$$

Care must be paid in evaluating this integral because of the singularity at $r = 0$. The second term is an integrable singularity, but the first term must be evaluated by surrounding the singularity by an infinitesimally small circular arc, σ . The contribution from the first term at the singularity is then

$$\int_{\sigma} \frac{P}{r} \frac{\partial r}{\partial n} ds = \lim_{r_0 \rightarrow 0} \int_0^{\alpha} \frac{P_0}{r_0} (-1) r_0 d\Theta = -\alpha P_0 \quad (7.11)$$

where α is the angle between the tangents to the surface at the singularity, P_0 is the pressure at the singularity, and r_0 is the infinitesimally small radius of the arc. With this information we can write Equation 7.10 as

$$\int_{\Gamma} \left(\frac{P}{r} \frac{\partial r}{\partial n} - \ln r \frac{\partial P}{\partial n} \right) ds = \alpha P_0 \quad (7.12)$$

in which Γ is the surface with an infinitesimally small gap at the singularity. This boundary integral equation serves as the basis for the numerical technique.

Now let us consider approximating the boundary with linear elements. A suitable number of nodes are placed on the boundary. The pressure and normal derivative of the pressure are approximated by assuming that they vary linearly with respect to the arclength between the nodes. By applying Equation 7.12 with the singular point at a particular node, and expressing the boundary integral as a function of the pressure and its normal derivative at each of the nodes, we generate a linear algebraic equation. By doing so at each of N nodes, we generate N equations in $2 * N$ unknowns (the pressure and its derivative at each node). The remaining N equations must come from the boundary conditions. If the original boundary equations are well-posed, then the approximate linear algebraic problem is well-posed. For constant elements it is possible to integrate exactly over each element. For more complex elements, Gaussian integration may be used.

A prototype subprogram required to implement this algorithm was written. The kernel integration routines were adapted from Brebbia (1978). The basic program in this reference had to be rewritten because of the geometric complexities associated with multiple permeability regions, multiple surfaces, and surfaces that are not simply connected. The geometrical description of the problem is contained in tables that give nodal and segmental information. From this information, the program identified the appropriate equations to generate for each node, and extracted the relevant data needed for each equation. The assembled matrix equation then was solved using a standard linear equation solver. The matrix was full.

This subprogram was incorporated into a complete model of Reverse Combustion by linking it with an crude local combustion law and an associated front propagation algorithm. More of the details associated with this preliminary model are reported in Dobbs and Krantz (1988). Qualitatively, this model gave the expected behavior. Starting from a roughened circular initial channel shape, the front grew outward, breaking into fingers. These fingers only showed modest directional orientation toward the injection well. This preliminary model showed that the boundary-element method could be used to model the flow problem, but it also graphically demonstrated the deficiencies of a simple linear element implementation - the surfaces do not exhibit the smooth shape expected from reverse combustion in a homogeneous medium. To better represent the channel surface, and the flux distribution thereon, it was decided that a more sophisticated version of boundary-element method was warranted.

7.5.3.4 Cubic Hermetian Boundary Elements

Let us now consider the application of cubic hermetian boundary elements to the previous boundary-integral equation. Cubic hermetian boundary elements have been used by (Watson, 1980; Watson, 1982) for planar elastostatics. An arbitrary boundary variable g can be approximated using cubic hermetian shape functions in the following manner:

$$g(\xi) = g(-1)M_-(\xi) + \frac{\partial g}{\partial x}|_{-1}N_-(\xi) + g(+1)M_+(\xi) + \frac{\partial g}{\partial x}|_{+1}N_+(\xi) \quad (7.13)$$

Here, $M_{\pm}$ and $N_{\pm}$ are the following shape functions: (Watson, 1982)

$$\begin{aligned} M_+(\xi) &= \frac{1}{4}(1 + \xi)^2(2 - \xi) \\ N_+(\xi) &= -\frac{1}{4}(1 + \xi)^2(1 - \xi) \\ M_-(\xi) &= \frac{1}{4}(1 - \xi)^2(2 + \xi) \\ N_-(\xi) &= \frac{1}{4}(1 - \xi)^2(1 + \xi) \end{aligned} \quad (7.14)$$

We shall use isoparametric elements: both the geometry and the dependent variables will be expressed using these cubic hermetian shape functions. We shall require smoothness in all variables. For the geometric variables x and y this requires that

$$\left. \frac{\partial x}{\partial y} \right|_- = \left. \frac{\partial x}{\partial y} \right|_+ \quad (7.15)$$

Expanding

$$\begin{aligned} \left. \frac{\frac{\partial x}{\partial \xi}}{\frac{\partial y}{\partial \xi}} \right|_- &= \left. \frac{\frac{\partial x}{\partial \xi}}{\frac{\partial y}{\partial \xi}} \right|_+ \end{aligned} \quad (7.16)$$

We can automatically enforce this condition by expressing the derivatives as follows

$$\begin{aligned} \left. \frac{\partial x}{\partial \xi} \right|_- &= \frac{\Delta x}{\Delta \xi_-} \\ \left. \frac{\partial x}{\partial \xi} \right|_+ &= \frac{\Delta x}{\Delta \xi_+} \\ \left. \frac{\partial y}{\partial \xi} \right|_- &= \frac{\Delta y}{\Delta \xi_-} \\ \left. \frac{\partial y}{\partial \xi} \right|_+ &= \frac{\Delta y}{\Delta \xi_+} \end{aligned} \quad (7.17)$$

where we will further require that $\Delta \xi$ not change sign when passing through a node.

To automatically enforce the compatibility equations between adjacent regions, we will use the pressure P and the normal component of the flux, $N = -\sigma \frac{\partial P}{\partial n}$ as our independent variables. We wish these to vary smoothly from one element to the next, thus we demand that

$$\begin{aligned} \left. \frac{\partial P}{\partial S} \right|_- &= \left. \frac{\partial P}{\partial S} \right|_+ \\ \left. \frac{\partial N}{\partial S} \right|_- &= \left. \frac{\partial N}{\partial S} \right|_+ \end{aligned} \quad (7.18)$$

Expressing this in the local parameterization

$$\begin{aligned}\frac{\frac{\partial P}{\partial \xi}|_-}{\frac{\partial S}{\partial \xi}|_-} &= \frac{\frac{\partial P}{\partial \xi}|_+}{\frac{\partial S}{\partial \xi}|_+} \\ \frac{\frac{\partial N}{\partial \xi}|_-}{\frac{\partial S}{\partial \xi}|_-} &= \frac{\frac{\partial N}{\partial \xi}|_+}{\frac{\partial S}{\partial \xi}|_+}\end{aligned}\quad (7.19)$$

The Jacobian is

$$J = \frac{\partial S}{\partial \xi} = \sqrt{\left(\frac{\partial x}{\partial \xi}\right)^2 + \left(\frac{\partial y}{\partial \xi}\right)^2} \quad (7.20)$$

Consequently we can again enforce smoothness by expressing the derivatives at the nodes as

$$\begin{aligned}\frac{\partial P}{\partial \xi}|_- &= \frac{\Delta P}{\Delta \xi_-} \\ \frac{\partial P}{\partial \xi}|_+ &= \frac{\Delta P}{\Delta \xi_+} \\ \frac{\partial N}{\partial \xi}|_- &= \frac{\Delta N}{\Delta \xi_-} \\ \frac{\partial N}{\partial \xi}|_+ &= \frac{\Delta N}{\Delta \xi_+}.\end{aligned}\quad (7.21)$$

Each node has twice as many degrees of freedom as with linear elements. If application of our boundary-integral equation is limited to each node, we must generate a second independent boundary integral equation for each node, to have a solvable problem. We generate this equation by taking the tangential derivative of Equation 7.12 as follows:

$$\int_{\Gamma} \left(\frac{\partial}{\partial t} \left[\frac{P}{r} \frac{\partial r}{\partial n} \right] - \frac{\partial}{\partial t} \left[\ln r \frac{\partial P}{\partial n} \right] \right) ds = \pi \frac{\partial P}{\partial t} \quad (7.22)$$

Note that since we have enforced smoothness, the angle made by the two tangents at the singular point is π .

These two integral equations must be discretized into linear algebraic equations in terms of the nodal degrees of freedom. The method used is well-known and general, and hence a detailed description will not be given; instead the basic concepts will be outlined. The boundary integral is partitioned into a summation over all the elements of all the bounding surfaces for the given region. Each term of the summation thus is the integral over a particular element. Since the integral is linear with respect to both pressure and the normal component of pressure, and furthermore the approximation in terms of shape functions is linear with respect to nodal values, each term can be further partitioned into a linear combination of integrals each of which contains only one of the nodal degrees of freedom. Since the nodal degrees of freedom are constant with respect to the integration, they can be pulled out of the integration. The remaining integrals can be integrated with Gaussian

quadrature since they depend only on the geometry which is known. The contributions of each element that is dependent on a particular nodal degree of freedom are summed, to obtain a coefficient for that nodal degree of freedom. By considering each nodal degree of freedom, a linear algebraic equation is generated. This equation depends upon the location of the singular point on the boundary. A linear algebraic system is assembled from the equations generated by associating the singular point with each node on the surface of a given zone. To minimize the dimension of the matrix equation that must be solved, the nodal degrees of freedom that are specified by external boundary conditions are immediately multiplied by their corresponding coefficients and the products are brought into the forcing vector as the coefficients are generated.

7.5.3.5 Singular Integral Evaluation

The use of cubic hermetian elements leads to an additional complication in evaluating the singular integrals; that is, those integrals that arise from application of the boundary-integral equation for elements adjacent to the singular node. Unlike constant elements, these singular integrals can not be evaluated analytically. Unlike linear elements, the integrals are not immediately presented in a standard form amenable to application of a special quadrature rule. Rather than using such a rule, the singular integrals were evaluated by using singularity subtraction. The asymptotic form of the kernels at the singularities were derived and integrated analytically. The asymptotic shape then was subtracted from the actual kernels leaving a nonsingular integral that was evaluated by regular Gaussian quadrature. The integral of the asymptotic singular portion would then be added to get the integral of the actual kernel.

7.5.3.6 Implementation

The flow calculations are implemented in a FORTRAN 77 subroutine SMOOTH FLUX CALCS. This subroutine is a self-contained solution package for Laplace's equation in a homogeneous zoned media using cubic hermetian boundary elements. The large full matrix equation arising from the boundary element method is solved using the IMSL MATH/LIBRARY routine LSARG. This routine uses LU decomposition with scaled partial pivoting. The solution is improved with iterative refinement. The matrix arising from this simple-variable boundary-element formulation does not yield matrices suitable to iterative solution methods such as successive over-relaxation.

7.5.4 Front Propagation

The propagation process converts the front speed into a change in the shape of the front. Since the front propagates normal to the current shape of the front, the front velocity is a simple vector function of the front speed. Cubic hermetian elements have a uniquely defined normal direction across nodes, unlike linear elements, so determination of the normal direction presents no difficulty. Since the propagation relationship treats any interaction between the speed of propagation from point to point, this process is equiva-

lent to the numerical integration of a parameterized, vector first-order ordinary differential equation forward in time. Any explicit method suitable for integrating unparameterized scalar first-order ordinary differential equations can be used for this task. The program was written to allow either explicit Euler integration or the predictor-corrector method (Heun's method), depending on the values chosen for two constants. These methods were selected over multi-step methods because of the difficulties associated with retaining and using historical information if the front were remeshed. The predictor-corrector method has better numerical stability characteristics than Euler integration, however, in typical use of this simulation, the time step size is governed more by accuracy rather than stability considerations. Since the predictor-corrector is a second-order method, it should be possible to take larger time steps with this method than with first-order Euler integration while maintaining the same accuracy in the solution. However, it was not possible to realize these larger time steps because of discontinuous behavior around extinction. Consequently, most of the calculations were conducted using Euler integration, since it requires half the number of expensive function evaluations (assembly and solution of the large linear algebra problem associated with the boundary-element method flow calculations) if the same size time steps are taken.

In propagating a cubic hermetian element, it is necessary to propagate both the value and the tangential derivative of both the x and y coordinates forward in time. An earlier version of the code did this by evaluating the tangential derivative of the front propagation speed at the nodes and using this evaluation to propagate the derivatives by the same method used to propagate the corresponding values. This method led to ill-conditioned matrices, since the Jacobian could be quite small using this method. Consequently, the current code propagates the front at the nodes and at the mid points. After the front is propagated, a pair of periodic cubic splines is fit through the x and y coordinates. The values of the tangential derivatives at the nodes are then evaluated from the cubic splines. This procedure smooths the front since the splines resist high curvatures, but this effect is reduced as smaller node spacing is used.

The front propagation procedure is implemented using a FORTRAN 77 subroutine SMOOTH PROPAGATION. The cubic spline fits are created and evaluated using IMSL MATH/LIBRARY routines.

7.5.5 Remeshing

In remeshing, the node placement is adjusted to correct for deficiencies in the one-dimensional mesh. These deficiencies include nodes concentrating at convergent corners or nodes being stretched apart by an actively growing channel tip. The remeshing strategy used is to place nodes at equal arc-length increments. This strategy favors the creation of a well-conditioned matrices over the use of a minimal number of node points. The front remeshing procedure is implemented using a FORTRAN 77 subroutine SMOOTH REMESH.

7.5.6 Input and Output

The input to the simulation consists of a start-up file, which specifies data such as the desired node spacing and required time step, and an initial shape definition file, which describes the initial channel shape and other boundaries, as well as the boundary conditions.

The output from the program consists of a cumulative shape definition file. Each record of this file corresponds to the shape at a specific time step. Each record has the same structure as the initial shape definition file. The cumulative shape definition file then is translated into a visual representation of the evolution of the channel using another FORTRAN program SMOOTH GRAPH. The main body of the program is independent of a particular imaging system. To implement this program for a particular graphics system, a small number of interface routines must be written to access the appropriate graphics primitives. The output in Chapter 8 was produced using the Precision Visuals DI-3000 graphics system. Preliminary work was produced using the graphics library provided with Microsoft FORTRAN version 5.0.

7.6 Appraisal of the Simulation

This simulation provides an effective platform for exploring RC channel evolution. The features that are generic to a variety of RC channel propagation models, that is, the flow calculations and the front propagation of the combustion front, are packaged in an easily reusable form. Those features that are unique to each model, mostly the front speed relationship, are isolated to allow easy modification.

The flow calculations are implemented using the boundary-element method in a zoned media. A simple variable method is used. This method does not intrinsically lead to well-conditioned matrices (Karrila and Kim, 1989). A number of precautions were taken to produce acceptably well-conditioned matrices. First, the ratio of permeability between zones was limited to approximately 10. If a greater ratio was desired, the lower resistance zone was considered to present negligible resistance to flow and was not modeled. In general, this approach was taken, since it also reduces the number of linear algebraic equations that need be solved. The permeability ratio between coal and char is very large, but not accurately measured. This approach loses all information about lateral convection. Second, the length scale of the problem was chosen to make the overall size of the flow field of order 10, whereas the node spacing was limited to order 1/10th. This selection is dictated by the logarithmic nature of the Green's function for Laplace's equation in planar geometry. To restrict the number of variables, the outer boundaries were represented by a small number of large elements. Third, the total number of equations was restricted to order 100. With these restrictions, the linear equation solver reported no ill-conditioning and the solution is presumed to be an accurate solution of the matrix equation.

The fidelity of the matrix equation to the flow behavior of the problem is dependent on several factors. The most important is the ability of the cubic hermetian shape functions to represent the flow and pressure profiles. This is dependent on the number of elements

used to represent a given lobe of the front. As long as the front is sufficiently smooth that several elements are used to represent the tip of a lobe, the cubic hermetian elements will be able to accurately capture the flow distribution. The second factor is the accuracy of the numerical integrations to generate the coefficients of the matrix. Gaussian quadrature was used with seven function evaluations for each element which is a conservative choice for the number of function evaluations, given the large number of elements used to represent the front. The program was tested to ensure correct coding of the routines by generating various shaped regions with boundary conditions imposed that would lead to flow fields given by simple analytical solutions to Laplace's equation, such as uniform flow in various directions, or stagnation flow. The resulting flow fields represented the given flow field with an accuracy proportional to the accuracy of the imposed boundary conditions.

The cubic hermetian elements accurately represent the normal component of the flux as well as its first tangential derivative. The second tangential derivative is not continuously represented. A hermetian cubic spline was fit to the flux, which generates a continuous fit for the second derivative. This produced a extremely erratic value for the second derivative of flux. This is an important variable since it is related to tangential conduction in the propagation relationship. To generate an acceptably smooth value for the second derivative, a central finite-difference approximation was used. To get a more accurate measure of the second derivative, higher order hermetian elements would be useful and should be considered for future applications of the boundary-element method. The front shape is represented by cubic hermetian shape functions. This provides a satisfactorily smooth representation of the channel surface. The propagation procedure performed satisfactorily, provided sufficiently small time steps were taken.

7.7 Summary

Multi-dimensional simulations are needed to gain better understanding of the nature of RC channeling. The strategy employed for this simulation was to use a multi-scale approach, with the flow calculations via the boundary-element method and the front speed given as a free boundary condition for the displacement of coal with char. The assumptions for this model are that RC propagation is adequately modeled via the free boundary condition approach, and various assumptions that lead to modeling the flow using Laplace's equation in a zoned media. The accuracy of a complex simulation such as this is determined by the way it is used, and is not easily quantified. With a simple variable boundary element method, rather severe constraints must be placed on the geometric description of the problem, to insure a meaningful solution to the flow problem. A useful extension of this model would be to implement an intrinsically better conditioned boundary element method. In future work, careful attention should be directed toward implementing a method that gives a very accurate measure of the second tangential derivative of the flux since this higher derivative relates to an important physical mechanism in RC front propagation.

CHAPTER VIII

A CASE STUDY IN REVERSE COMBUSTION CHANNEL EVOLUTION USING BOUNDARY ELEMENT FLOW CALCULATIONS

8.1 Scope

In this chapter, a case study in reverse combustion channel evolution in homogeneous porous media is presented. This study explores the use of the boundary element flow calculations for modeling RC channel evolution. The two-dimensional, transient, numerical RC channel simulation described in Chapter 7 is used along with a model of front propagation partially fit to the experimental data of Visser *et al.* (1985). Three different configurations are considered: two-dimensional flow through a laterally bounded rectangular slab with a combustion front propagating along the major axis, two-dimensional flow inward from the periphery of disk to a radially expanding combustion front, and two-dimensional flow in a large disk from an injection well to a combustion front. For the laterally bounded slab, the effects on channel growth of gas-flow rate, tangential heat conduction, and the minimum gas flux necessary for front propagation are investigated. For the other two configurations, the effect of gas-flow rate is investigated. The features of the evolving channels are described. The discussion of results includes a qualitative comparison of the channel characteristics from these simulations with the observations of Visser *et al.* The discussion also considers the mathematical and numerical characteristics of RC channel evolution, and the implications of this study with regard to RC linking for underground coal gasification. The chapter ends with a summary.

8.2 Model Formulation

To use the RC channel simulation described in Chapter 7, a model of RC front propagation must be specified. The model formulated here is a variation on the model presented in Chapter 4. It is simplified by directly using experimental data to the largest extent possible. Additional simplifying assumptions are introduced that allow the front propagation model to be explicitly solved. The model is posed in dimensionless variables suitable to the overall channel-evolution problem.

As discussed in Chapter 4, an important aspect of a continuum RC front propagation model is a kinetic relationship between the front temperature and gas flux. At this time, this relationship must be obtained from experimental data. The experiments in solid coal cores for this study did not generate sufficiently accurate data from which to extract this relationship. The analysis of the experiments of Schloemer *et al.* (1988) in solid coal cores

did not include estimation of the gas flux to the channel; hence, this relationship could not be extracted from these experiments. Field tests yield neither accurate temperature measurements nor accurate flux estimation; hence, they are not very useful for this purpose. The variation of the observed temperature for the experiments of Lu (1980) for powdered coal prevents this relationship from being obtained from these experiments, even assuming that these data are relevant to RC in solid coal. No other data have been located that allow this relationship to be meaningfully identified for coal. Consequently, this model can not be fit to the kinetic characteristics of RC in coal, since they are not known sufficiently accurately.

Instead, the experiments of Visser *et al.* (1985) conducted in particle board will serve as the partial experimental foundation for the model formulated here. Unfortunately, the available data from these experiments are not sufficient to fit all the adjustable parameters for this model. The effects of the two adjustable parameters that can not be obtained empirically are explored numerically. Visser *et al.* conducted both one-dimensional and two-dimensional experiments. The one-dimensional experiments provide sufficient data from which the kinetic relationship can be directly fit. These data are fit by the empirical relationship

$$T_f = 1.8 + 1.4\mathcal{N} \quad (8.1)$$

where T_f is the dimensionless front temperature which has been scaled with the ambient temperature. $\mathcal{N}$ is a dimensionless gas flux to the channel surface defined by

$$\mathcal{N} = \frac{N_{g,a}}{N_{g,aS}} \quad (8.2)$$

with $N_{g,a}$ is the gas flux to the surface, and $N_{g,aS}$ is scale factor with the value 200 kg/hr/m². The data for $N_{g,a}$ ranged from approximately 125 kg/hr/m² to 550 kg/hr/m².

The second aspect of the front propagation model is a heat balance which yields the front-propagation speed. The governing relationship used is Equation 4.87 which is repeated here for the reader's convenience, with the largest root selected:

$$V = \frac{Q + \sqrt{Q^2 + 4(T_f - 1)S_f/g}}{2(T_f - 1)} - 1 \quad (8.3)$$

Here, V is the dimensionless front speed, Q is a dimensionless heat of combustion, defined by Equation 4.15, and S_f/g is a tangential heat transport strength multiplied by the corresponding boundary-layer thickness. As mentioned in Chapter 4, V , which was nondimensionalized in a manner appropriate to simplifying the front propagation model, is not directly proportional to the dimensional front-propagation speed because of the functional dependence of the length and time scales on the normal component of the gas flux. A new dimensionless front speed is introduced that is proportional to the dimensional front speed:

$$\mathcal{V} = \mathcal{N}V \quad (8.4)$$

The dimensionless heat of reaction, Q determines the value of the gas flux, $\mathcal{N}$, at which the front speed, $\mathcal{V}$ reaches a maximum for any particular strength of tangential heat

transport, S_f/g , for a specified functional dependence of the front temperature, T_f , on the gas flux, $\mathcal{N}$. The maximum in the propagation velocity versus gas flux relationship for the no-tangential-heat-transport case is an important aspect of the front propagation model, and it provides the most direct method to extract the dimensionless heat of reaction from the experimental data. A value for Q of 2.5 is obtained from the one-dimensional experiments of Visser *et al.* (1985), based on a dimensional gas flux of 300 kg/hr/m² at the maximum propagation velocity of 25 cm/hr. The dimensional front-speed relationship is illustrated in Figure 8.1. The triangles mark the one-dimensional data of Visser *et al.* The location of this maximum, or even its existence is not unambiguously indicated by direct consideration of the data of Visser *et al.* From a purely empirical perspective, these data would be reasonably fit by a linear relationship. However, a linear relationship would violate the principle of conservation of energy both at high and low fluxes. With no flux, the propagation velocity must be zero for the one-dimensional experiments, since no heat of combustion is released. Conversely, if the combustion-front temperature monotonically increases with gas flux, at some flux the temperature is sufficient that all the heat released is absorbed by gas-phase convection through the front and none is left for propagating the front.

With this model, the front speed is modified by tangential heat transport. Due to computational limitations discussed in Section 7.6, tangential convection was neglected to obtain more spatial resolution of the tangential conduction and normal convection. With this restriction, S_f is equal to S_d , which is defined by Equation 3.30. In addition, the lower order effect of the change in the gas flux due to devolatilization will be neglected. This allows the front propagation speed to be explicitly determined. Given these restrictions, the tangential heat transfer strength is given by

$$S_f = \frac{\partial T_f}{\partial N_{g,a}} \left(\frac{\lambda_s}{C_g N_{g,a}} \right)^2 \left(\frac{\partial^2 N_{g,a}}{\partial x_2^{*2}} \right) \quad (8.5)$$

In terms of dimensionless variables and parameters, the overall tangential heat transfer term can be written

$$S_f/g = \frac{\mathcal{H}}{\mathcal{N}^2} \frac{\partial^2 \mathcal{N}}{\partial \mathcal{X}^2} \quad (8.6)$$

The independent variable $\mathcal{X}$ is an arc length coordinate scaled with a length $x_s^* = 0.04$ m characteristic of the samples used by Visser *et al.* The parameter $\mathcal{H}$ is a dimensionless tangential heat transfer coefficient. It is defined by

$$\mathcal{H} = \frac{\partial T_f}{\partial N_{g,a}} \left(\frac{\lambda_s}{C_g N_{g,a}} \right)^2 \frac{N_{g,a} S}{x_s^{*2} g} \quad (8.7)$$

and is of order 0.006 for a tangential heat transport boundary-layer thickness of 5 mm at a gas flux of 200 kg/hr/m². This parameter could be extracted from a detailed point-wise analysis of the shape of the evolving front. The experiments of Visser *et al.* were not subjected to this level of scrutiny; hence, we will treat this as an adjustable parameter, and explore its effect on channel evolution.

As presented so far, this model does not feature extinction at low gas fluxes. Both modeling and experiments suggest that extinction occurs at finite gas fluxes. No one-dimensional

Figure 8.1: Combustion front propagation relationship compared to the data of Visser *et al* (1985).

experiments below 125 kg/hr/m² were reported. It is quite uncertain at which gas flux extinction does occur, hence we will treat this as an adjustable parameter, and explore its effect on channel evolution. The influence of extinction is approximated by including a multiplicative factor in the front-speed relationship.

$$\mathcal{V} = \mathcal{N}V\mathcal{F} \quad (8.8)$$

The multiplicative factor is of the following form

$$\mathcal{F} = \begin{cases} 0, & \mathcal{N} < \mathcal{N}_{min} \\ 1 - \exp\left(-\frac{\mathcal{N} - \mathcal{N}_{min}}{\mathcal{N}_{min}}\right), & \mathcal{N} > \mathcal{N}_{min} \end{cases} \quad (8.9)$$

It provides a smooth reduction of the front velocity to zero as the gas flux is reduced to the minimum gas flux for front propagation. The effects of tangential convection are absorbed into this empirical extinction factor.

8.3 Cases

In this section, the graphical results of the simulation cases are presented. The main output from a run is a recording of the combustion-front shape at constant time intervals. These time intervals correspond to every tenth time step. The time increment for front propagation was held constant for all the cases presented here, at a dimensional value of 33.5 seconds; thus, the combustion-front contours represent the channel evolution at 5.6 minute intervals. The node spacing was held constant at 8 mm. Since the front is defined with cubic hermetian shape functions, this is roughly equivalent to a node spacing of 4 mm with linear segments.

With this node spacing, the values of the normal component of the flux and its first tangential derivative are resolved with adequate accuracy for the channel shapes considered. The second derivative of the normal component of the flux at the tips of sharply pointed channels suffers from a degree of numerical smoothing with this node spacing. This smoothing reduces the stabilization owing to tangential heat transport. However, numerical smoothing of a sharply pointed channel tip arising from the front propagation and remeshing routines more than compensates for this reduction in stabilization. The numerical smoothing is not believed to qualitatively alter the trends with respect to the parameters varied. A smaller node spacing, and the associated reduction in the size of the time step, would reduce the length of time the channel could be propagated for a given computational effort. At this exploratory stage, a significantly greater use of computational resources is not felt justified. Consequently, the node spacing of 8 mm is accepted despite some numerical smoothing to allow the exploration of the longer term evolution of the channel shape. It should be noted that in previous two-dimensional models of RC, channel stabilization was wholly an artifact of the particular numerical (Thorsness *et al.*, 1977), analytical (Britten, 1984; Monteiller, 1986), or physical analog (Fuhrmann *et al.*, 1988) technique applied.

The simulations were normally run until the number of nodes on the combustion front reached 120, although some runs were terminated before this point because of contact of the

combustion surface with the external boundaries. The total number of degrees of freedom, and hence the size of the matrix that was inverted for the flow calculations, then was approximately 270, which includes the degrees of freedom associated with the other boundaries. The simulation runs each required between 20 and 120 minutes of CPU time on a VAX 6000-420 using the VMS V5.3-1 operating system. The CPU time chiefly depended on the flow configuration and the gas-flow rate.

8.3.1 Rectangular Slab Flow Configuration

The majority of the simulation cases used the rectangular slab flow configuration. This consists of a laterally bounded rectangular slab with a combustion front propagating along the major axis. This configuration provides for highly oriented channel growth owing to the confined nature of the flow field. The number of time steps that could be taken depended on the shape of the evolving channel since the maximum number of nodes was fixed. The strong directional orientation for the rectangular configuration allowed exploration of longer term behavior than was possible with unconfined flow configurations. This configuration is amenable to laboratory investigation, but the relationship between the resulting channel shapes and those occurring with unconfined flow fields must be carefully considered.

The flow configuration is shown in Figure 8.2. The corners of the rectangular slab are rounded to avoid introducing ill-conditioning into the boundary-element flow calculations. The scale of the rectangle is 24 cm X 80 cm, which is within the range considered by Visser *et al.* A prescribed flux boundary condition is imposed on the peripheral surfaces of the slab (a no-flux boundary condition for the top and bottom surfaces is implicitly imposed by the simulation). The normal component of the flux to the long sides of the rectangle is zero. The normal component of the flux at the two ends is equal in magnitude, but opposite in direction. The limitation imposed by the simulation that zone boundaries be nonintersecting closed curves requires some disparity between the configuration used by Visser *et al.* and that used in the simulation. To approximate the effect of a channel starting at one end of a slab, the initial channel shape is an eye. A constant pressure boundary condition is specified on this internal boundary. This is equivalent to a region with no resistance to flow. The flow streamlines are attracted to this region. However, there will be no net flow to the internal boundary, since there is no net flow to the rectangular boundary owing to the flux boundary conditions imposed on the rectangular surface. Conservation of mass for the region bounded by rectangular surface and the internal boundary requires that these flows sum to zero.

For this configuration, the effects of gas flow rate, tangential heat transfer coefficient, and the minimum flux for front propagation are investigated. Starting from a base case, each parameter is varied while holding constant values for the other two. The values of the parameters for the cases presented here are graphically displayed in Figure 8.3. For the reader's convenience, the channel shape record for the base case is included in each of the three series of figures.

Figure 8.2: The rectangular slab configuration

Figure 8.3: A graphical display of the parameter values for the cases presented.

8.3.1.1 Effect of Gas Flow Rate

The first series of cases illustrates the effect of the superficial gas flux on channel characteristics. The superficial gas flux is proportional to the gas flow rate for this geometry. The value of minimum gas flux for front propagation is held constant at 10 kg/hr/m^2 . The value of the dimensionless tangential heat transport coefficient is held constant at 0.005. This is equivalent to a tangential heat transport boundary layer of order 5 mm. The superficial gas fluxes for Figures 8.4, 8.5, 8.6, and 8.7 are 25, 50, 100, and 200 kg/hr/m^2 , respectively.

At high gas-flow rates, the leading tip of the front propagates at nearly the maximum front speed for the adiabatic propagation relationship, in accordance with the lumped-parameter model of Mul and Miller (1983). In this regime, the propagation speed of the leading tip is insensitive to the superficial gas flux, but the width of the channel increases with an increase in gas flux. At low gas-flow rates, both the propagation speed and the channel width increase with superficial gas flux.

8.3.1.2 Effect of Tangential Heat Transport

The second series of cases illustrates the effects of the tangential heat transport on channel characteristics. The value of superficial gas flux for front propagation is held constant

Figure 8.4: Record of channel evolution for Case 1 (rectangular slab configuration, superficial gas flux is 25 kg/hr/m^2 , $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m^2 , and $\mathcal{H}$ is 0.005).

Figure 8.5: Record of channel evolution for Case 2 (rectangular slab configuration, superficial gas flux is 50 kg/hr/m^2 , $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m^2 , and $\mathcal{H}$ is 0.005).

Figure 8.6: Record of channel evolution for Case 3 (rectangular slab configuration, superficial gas flux is 100 kg/hr/m^2 , $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m^2 , and $\mathcal{H}$ is 0.005).

Figure 8.7: Record of channel evolution for Case 4 (rectangular slab configuration, superficial gas flux is 200 kg/hr/m^2 , $\mathcal{N}_{\min}$ is equivalent to 10 kg/hr/m^2 , and $\mathcal{H}$ is 0.005).

at 50 kg/hr/m². The value of minimum gas flux for front propagation is held constant at 10 kg/hr/m². The tangential heat transport coefficient is varied in this series. This parameter is not an independent variable for laboratory reverse combustion experiments. Variation in this coefficient represents changes in the ratio of the tangential conduction boundary layer to the preheat zone thickness. In the model of RC used here, this parameter is postulated to be a constant value for a given combustible medium, at least for a given combustion back-pressure and oxygen content of injected gas. However, the experiments of Visser *et al.* were not subjected to the spatially distributed analysis of the gas flux to the channel required to extract this coefficient, so the correct value is not known. The values of the tangential heat transport coefficient for Figures 8.8, 8.9, 8.10, and 8.11 are 0, 0.005, 0.010, and 0.015, respectively.

At the given value for the superficial gas flux, increasing the tangential transport boundary layer thickness suppresses the formation of multiple fingers, but does not dramatically affect the propagation speed of the leading tip. The leading tip propagates at approximately 80 percent of the maximum propagation speed for the adiabatic case. For the case with no tangential heat transport, stabilization of the front is due solely to numerical, not physical factors. Since the numerical smoothing depends on the node spacing, if this were reduced, the characteristic length scale for the fingers would also be reduced. For nonzero values of the tangential heat transport, stabilization is due to both physical and numerical factors. For a fixed node spacing, the influence of numerical stabilization is greatest for the cases with the least physical stabilization. Hence, the response of the channel growth to changes in the tangential heat transport coefficient displayed in this series is dampened by the numerical smoothing. Still, the basic trend is clear: a thicker boundary layer for tangential heat transport would result in a smoother combustion front.

8.3.1.3 Effect of Extinction at a Minimum Gas Flux

The third series of cases illustrates the effects of the minimum gas flux for front propagation on channel characteristics. The value of superficial gas flux for front propagation is held constant at 50 kg/hr/m². The value of the tangential heat transport coefficient is held constant at 0.005. The value of minimum gas flux for front propagation is varied in this series. This parameter is not an independent variable for reverse combustion experiments or field tests. It is a phenomenological correction for stoichiometry, tangential convection, and others factors that influence extinction, but are not included explicitly in the model. The one-dimensional experiments of Visser *et al.* did not explore the low gas flux regime adequately to allow extraction of an accurate estimate of this parameter. The value of the minimum gas flux for front propagation for Figures 8.12, 8.13, 8.14, and 8.15 are 10, 20, 30, and 40 kg/hr/m², respectively.

At the given value for the superficial gas flux, the minimum gas flux for front propagation has a strong influence on the shape of the combustion front. At low values, the front propagates mainly as a broad, single finger, with small subsidiary fingers growing at the base of the main channel in a direction perpendicular to the main axis of the rectangular slab. As the minimum flux is increased, the channel tip breaks down into smaller fingers, with the subsidiary fingers suppressed. The fingers spread further apart from each other as this

Figure 8.8: Record of channel evolution for Case 5 (rectangular slab configuration, $\mathcal{H}$ is 0.0, superficial gas flux is 50 kg/hr/m^2 , and $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m^2).

Figure 8.9: Record of channel evolution for Case 6 (rectangular slab configuration, $\mathcal{H}$ is 0.005, superficial gas flux is 50 kg/hr/m², and $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m²).

Figure 8.10: Record of channel evolution for Case 7 (rectangular slab configuration, $\mathcal{H}$ is 0.010, superficial gas flux is 50 kg/hr/m^2 , and $\mathcal{N}_{\min}$ is equivalent to 10 kg/hr/m^2).

Figure 8.11: Record of channel evolution for Case 8 (rectangular slab configuration, $\mathcal{H}$ is 0.015, superficial gas flux is 50 kg/hr/m^2 , and $\mathcal{N}_{\min}$ is equivalent to 10 kg/hr/m^2).

Figure 8.12: Record of channel evolution for Case 9 (rectangular slab configuration, $\mathcal{N}_{min}$ is equivalent to 10 kg/hr/m^2 , the superficial gas flux is 50 kg/hr/m^2 , and $\mathcal{H}$ is 0.005).

Figure 8.13: Record of channel evolution for Case 10 (rectangular slab configuration, $\mathcal{N}_{min}$ is equivalent to 20 kg/hr/m², the superficial gas flux is 50 kg/hr/m², and $\mathcal{H}$ is 0.005).

Figure 8.14: Record of channel evolution for Case 11 (rectangular slab configuration, $\mathcal{N}_{\min}$ is equivalent to 30 kg/hr/m^2 , the superficial gas flux is 50 kg/hr/m^2 , and $\mathcal{H}$ is 0.005).

Figure 8.15: Record of channel evolution for Case 12 (rectangular slab configuration, $\mathcal{N}_{\min}$ is equivalent to 40 kg/hr/m², the superficial gas flux is 50 kg/hr/m², and $\mathcal{H}$ is 0.005).

parameter was increased.

8.3.2 Circular Flow Configuration

The circular flow configuration consists of a circular disk with an initially circular combustion front propagating outward from the center of the disk. The gas flows inward from the periphery to the channel surface. It represents the initial growth of a RC combustion front outward from the production well, assuming that the injection well is effectively at an infinite distance. This configuration is amenable to laboratory investigation. The relationship between the resulting channel shapes and those occurring with a injection well at a finite distance must be carefully considered.

The flow configuration is shown in each of the figures. The outer boundary is 30 cm in diameter. The initial combustion front is 6 cm in diameter. A prescribed pressure boundary condition is imposed on the periphery of the disk and on the channel surface. The resulting flux distribution to the combustion front is scaled to maintain a constant total gas flow rate. For this configuration, the effects of gas-flow rate are investigated. The total gas-flow rates for Figures 8.16, 8.17, 8.18, and 8.19 are chosen to obtain superficial gas fluxes for the initial channel shape of 50, 100, 150 and 200 kg/hr/m², respectively. The tangential heat transport coefficient is held constant at 0.0075. The minimum gas flux for front propagation is held constant at a dimensional value of 30 kg/hr/m². For these conditions, as the gas-flow rate is increased, the onset of fingering is delayed and the fingers that do form are broadened. The front-propagation speed increases with the total gas flux.

8.3.3 Two-well Flow Configuration

The two-well flow configuration consists of flow from an injection well to a combustion front. The second well is contained within the combustion front. It represents the initial growth of a RC combustion front outward from the production well, assuming that the injection well is at a finite distance. This configuration is the basic configuration used in isolated field tests.

The flow configuration is shown in Figures 8.20. The outer boundary is 160 cm in diameter. The two wells are spaced 56 cm apart, center-to-center. The initial combustion front is 6 cm in diameter. The injection well is modeled as a circle with a diameter of 6 cm. A prescribed pressure boundary condition is imposed on the surface of the injection well. The resulting flux distribution to the combustion front is scaled to maintain a constant total gas flow rate. For this configuration, the effects of gas flow rate are investigated. The total gas-flow rates for Figures 8.21, 8.22, 8.23, and 8.24 are chosen to obtain superficial gas fluxes for the initial channel shape of 50, 100, 150 and 200 kg/hr/m², respectively. The tangential heat transport coefficient is held constant at 0.0075. The minimum gas flux for front propagation is held constant at a dimensional value of 30 kg/hr/m². For these conditions, as the gas-flow rate is increased, the onset of fingering is delayed and the fingers that do form are broadened. The front-propagation speed increases with the total gas flux. The combustion front displays a distinct orientation in its growth toward the injection well.

Figure 8.16: Record of channel evolution for Case 13 (circular configuration, the superficial gas flux to the initial channel shape is 50 kg/hr/m^2 , $\mathcal{N}_{\min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).

Figure 8.17: Record of channel evolution for Case 14 (circular configuration, the superficial gas flux to the initial channel shape is 100 kg/hr/m^2 , $\mathcal{N}_{\min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).

Figure 8.18: Record of channel evolution for Case 15 (circular configuration, the superficial gas flux to the initial channel shape is 150 kg/hr/m^2 , $\mathcal{N}_{\min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).

Figure 8.19: Record of channel evolution for Case 16 (circular configuration, the superficial gas flux to the initial channel shape is 200 kg/hr/m^2 , $\mathcal{N}_{\min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).

Figure 8.20: The two-well flow configuration.

Figure 8.21: Record of channel evolution for Case 17 (two-well configuration, the superficial gas flux to the initial channel shape is 50 kg/hr/m^2 , $\mathcal{N}_{min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).

Figure 8.22: Record of channel evolution for Case 18 (two-well configuration, the superficial gas flux to the initial channel shape is 100 kg/hr/m^2 , $\mathcal{N}_{\min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).

Figure 8.23: Record of channel evolution for Case 19 (two-well configuration, the superficial gas flux to the initial channel shape is 150 kg/hr/m^2 , $\mathcal{N}_{\min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).

Figure 8.24: Record of channel evolution for Case 20 (two-well configuration, the superficial gas flux to the initial channel shape is 200 kg/hr/m^2 , $\mathcal{N}_{\min}$ is equivalent to 30 kg/hr/m^2 , and $\mathcal{H}$ is 0.0075).

8.4 Discussion

The model of front propagation used in this case study has three key features characteristic of RC in homogeneous porous combustible solids. These features arise from the interaction of the combustion-front temperature relationship with three physical processes: normal convection of heat, tangential conduction of heat, and extinction of combustion. First, the relationship between propagation speed and gas flux displays a maximum for any given amount of tangential heat transport. This feature of RC arises from the interaction of the combustion front temperature relationship with normal convection of heat through the combustion front. The heat released by combustion increases roughly linearly with the gas flux. The heat absorbed by convection increases with the product of the gas flux and the combustion-front temperature. The combustion front-temperature is basically a monotonically increasing function of the normal component of the gas flow to the channel surface. Thus, normal convection absorbs an ever-increasing proportion of the heat released as the gas flux is increased. Consequently, the residual heat that propagates the front reaches a maximum. Second, the propagation speed is dependent on the distribution of the normal component of the gas flux along the combustion front. This feature arises from the interaction of the temperature relationship with tangential conduction and convection of heat. The temperature distribution on the combustion front is closely related to the gas-flux distribution. Consequently, heat is transferred from areas where the gas flux is concentrated to those where it is not. This redistribution of heat has a smoothing effect on the variation of front-propagation speed with the gas distribution. Third, the propagation speed is negligible below a specific, nonzero gas flux. This feature of RC arises from the extinction of combustion. Extinction is basically a breakdown in the relationship between front temperature and gas flux.

The existence of these features in the propagation relationship led to recognizable evolutionary patterns in channel growth. At high gas-flow rates, the speed of the channel propagation was insensitive to the gas flow rate. The fingers were sufficiently broad that tangential heat transport did not significantly slow the propagation speed at the channel tip. Consequently, the channel tip propagated at nearly the maximum speed for the adiabatic case. If the channel shape near the tip were perturbed outward, attracting more flux at some point, the combined effect of increased tangential conduction and normal convection away from the perturbation slowed its growth. Alternatively, if the combustion front was perturbed so that less flux was received, tangential conduction toward the perturbation caused it to speed up, smoothing the front. However, perturbations in the channel shape farther away from the tip, where the gas flux was below that required for fast front propagation, were not necessarily suppressed, since normal convection was lessened. Instead, if the channel was sufficiently broad, these perturbations triggered channel bifurcations.

At lower flow rates, the influence of tangential conduction was displayed. As the gas flow was reduced, the width of the combustion finger decreased since the channel tip simply did not receive enough gas to supply a broad front with the gas flux required to propagate at the maximum propagation speed for adiabatic combustion. The variation of the gas-

flux distribution on the channel surface became more pronounced. This led to significant tangential conduction of heat from the channel tip, which slowed its propagation.

Changing the gas flux at which extinction occurred has a strong effect on the shape of the RC combustion front. The most obvious effect of extinction was to cause the fingers formed by a bifurcating channel tip to spread apart from each other. This is a consequence of the reduced range of flow rates for which front propagation was possible. Extinction did not directly decrease the propagation rate for a channel tip.

In the unconfined geometries (the two-well configuration or the circular configuration) a broad combustion front broke apart into fingers. The two-well configuration suggests that the formation of fingers proceeds slowly enough so that the oriented nature of the flow field influences the fingering pattern. The formation of two diametrically opposed fingers that subsequently bent toward the injection well, as is frequently used in illustrations of the onset of RC channels, was not observed. In contrast to the linear stability analyses of Puri (1979) and Bumb (1981), the number of channels observed increased with gas flux, although the onset of the instability was delayed.

The channel growth records from the simulation bear a strong resemblance to the channel evolution observed by Visser *et al.* via infrared thermography and post-burn sectioning. Figures 8.25 and 8.26 show representative channel shapes for the slab flow configuration and circular configuration for their particle board experiments. A basic trend observed in their work was that at high gas fluxes, the combustion front had a highly convoluted shape suggestive of a cross-section of a cauliflower. At low gas fluxes, the combustion front consisted of distinct fingers that propagated at some distance from each other. Comparing these observations to the simulation results suggests that these fronts were characterized by a relatively low degree of lateral stabilization and that extinction occurred at a comparatively high gas flux, although the data required to quantify these relationships are not available.

The mathematical model of RC used here is ill-conditioned in the sense that small variations in the initial channel shape lead to large variations in the channel shape at later times. This mathematical ill-conditioning is inherited from the ancestral problem of viscous displacement in the limit of no surface tension. The viscous displacement problem has received a great deal of attention from mathematicians. For our purposes, a key result is the observation of Howison (1986) that every smooth solution is embedded in an infinite family of solutions (that are close to the smooth solution in the sense of a polynomial approximation) that blow up (form a cusp) at a finite time. Indeed, every polynomial surface will form cusps at finite time. Our model includes stabilizing effects that prevent the formation of cusps. However, the stabilizing effects are not sufficient to completely remove the ill-conditioning inherent to the problem. Aside from any numerical effects, the location of the origination of combustion fingers arising from an initial channel shapes depends on the exact shape of the combustion front.

These effects carry over to the numerical simulation. As an example, the "hermetic circles" used as the initial combustion-front shape for the two-well configuration and the circular configuration were defined by four elements initially. The initial shapes were remeshed to 23 elements before the simulation commenced. Still, at low flow rates, the four-fold symmetry of the initial shape appeared in the fingers that formed many time steps

Figure 8.25: Representative channel shape from Visser *et al.* (1985) for a rectangular slab configuration.

Figure 8.26: Representative channel shape from Visser *et al.* (1985) for a circular flow configuration.

later.

An effect that is more numerical in nature is that the node spacing affected the minimum size feature observed for cases with a small degree of lateral stabilization. Refinement of the mesh below the cut-off size imposed by tangential heat transfer, along with the corresponding reduction in the size of the time step, would have greatly increased the computational resources required for this study, so this effect was tolerated. This problem could be partially ameliorated by more sophisticated front-propagation procedures. These modest difficulties with mesh size while using the boundary-element method illustrate the extreme problems that would arise from a fully two-dimensional model of RC. The direct numerical solution of the RC model equations via, for example, the finite element method would have to be restricted to a small portion of a RC front.

A consequence of these effects is that it would not be possible to exactly duplicate the long term evolution of observed channels regardless of how well the model parameters were fit to experimental data and how carefully the numerical simulations were conducted. However, it is reasonable to expect that given adequate experimental data, simulations such as these could capture channel characteristics in a statistical sense. The complex nature of the channels produced in these simulations, as well as those observed in the work of Visser *et al.*, illustrates that the measures used to characterize channels in previous work are not adequate to describe the structure of these channels.

Now let us consider the insight that these simulations give to RC linking for underground coal gasification. The sensitivity of the fingering pattern to the exact shape of the initial combustion front suggests that methods such as down-hole jet burners might strongly influence the location and number of fingers that are initiated at the production well. To do so, they would have to create a disturbance in the flow field sufficient to dominate existing natural preferred permeability paths. Less encouraging is the implication of these simulations that the number of fingers initiated at the production well might not strongly influence the number of surviving fingers exiting the vicinity (say 3 m) of the production well. Initiation of a large number of fingers could result just in more fingers becoming extinct. Alternatively, initiation of a single finger could just lead to its bifurcation into many fingers. Attempts to control the number of channels would have to be coordinated with the oxidant injection policy.

The tendency of a RC finger to bifurcate is exaggerated in two-dimensional experiments and simulations, compared to three-dimensional channels. Still, these results suggest that a RC finger would not persist as a single finger over the range of flow rates experienced during one of the Hanna field tests through the physics included in this model. Two trains of thought originate from this idea. The first is that perhaps the front did not propagate as two fingers as previously speculated, but rather it followed two basic paths with the number of actively advancing fingers varying over time with the total gas flow rate. The second is that perhaps the fingering pattern arose from the existence of preferred permeability paths, rather than completely from porous media stability considerations.

The simplest explanation of the insensitivity of propagation speed with respect to variations in the gas flow rate as observed in various field tests is that the channel tip is under a regime dominated by normal convection. Consequently, such data are only of modest

value in testing other aspects of RC front propagation models, such as the extent of lateral stabilization.

8.5 Summary

A simple model of RC front propagation was partially fit to the particle board experiments of Visser *et al.* This model was used in a case study of RC channel evolution. For a rectangular slab configuration, increasing the total gas flow rate increased both the breadth of the combustion front and the propagation speed of its tip. The propagation speed leveled off at high gas flow rates. For the same configuration, tangential heat transport suppressed the bifurcation of the combustion front, without greatly altering the propagation speed of the channel tip. Increasing the minimum gas flux for front propagation, which was used to emulate the effect of extinction, had a strong effect on the shape of the channel, as well as on how far RC fingers spread apart from each other. For the circular flow configuration, increasing the total flow rate delayed the onset of fingering. Higher flow rates led to a greater number of RC fingers. For the two-well configuration, similar results with respect to total gas flow rate were found, while the alteration in the fingering pattern caused by the existence of the injection well at finite distance was demonstrated. The channel fingering patterns created by these simulations bear a strong qualitative resemblance to those observed in the particle board experiments of Visser *et al.* These simulations suggest that the initial fingering pattern would be easily altered by methods such as down hole jet burners, but that this change in the fingering pattern might not have a strong effect on the long term channel growth.

CHAPTER IX

CONCLUSIONS AND RECOMMENDATIONS

9.1 Conclusions

9.1.1 Laboratory Studies

An important goal of the laboratory studies was to test scaling laws based on the applied linear stability analysis (ALSA) discussed in Chapter 2. The creation of channels of approximately the size anticipated provided qualitative confirmation of the hypothesized scaling relationships. It had been suggested that these scaling relationships could be used to aid experimental investigation of RC channeling through the establishment of dynamic similarity between small channels created in the laboratory and large channels found in field tests. The utility of these relationships was found to be limited by several factors. The principal complicating factor was that the physical structure of the coal, which did not scale, was found to strongly influence channel characteristics. The influence of heterogeneities in the coal, as well as the fracture, fissure, and cleat structures, was large for the small channels created in the laboratory apparatus. In addition, the orders of the combustion reaction with respect to pressure, fuel and oxygen have not been well established. Finally, phenomena not considered by ALSA, such as extinction, are not assured of scaling.

The variation of channel characteristics with process parameters (flow rate, oxygen content of injected gas, and combustion back-pressure) was investigated. The channel cross-section was highly variable, strongly affected by the fissure structure of the coal, and affected by the presence of small thermocouples. The irregular nature of the observed combustion channels made identification of a characteristic channel diameter difficult. Any trends between channel size and process parameters were obscured by the variation in the channel shape in the course of a single experiment. The propagation velocity of the channel tip increased at higher gas flow rates, combustion back-pressure, and oxygen content of the injected gas, in a synergistic, nonlinear, manner. The observed physical structure of the channels differs in several aspects from that envisioned by existing mathematical models of RC. Evidence of char-burning was found in many experiments. The char zone surrounding the channel core was penetrated by cracks that increased in size toward the channel core.

Extraction of global Arrhenius kinetic parameters from channeling experiments was unsuccessful. The correlation between the reaction rate and the reaction temperature from which these parameters would be extracted was obscured by the uncertainty of the gas flux (which enters into calculating the reaction rate) and the combustion-front temperature. In future work, kinetic parameters should be extracted from experiments optimized for this

purpose. Since RC field tests present even greater uncertainty with respect to determining the gas flux and the reaction temperature than did these laboratory experiments, the accuracy of kinetic parameters derived from field-test data is questionable.

The gas flux to the channel surface was estimated by numerical solution of Darcy's law in zoned media. This model of porous media flow is that used in most models of RC. The uncertainty of these estimates is large. The source of most of the uncertainty is the possibility that much of flow in the coal samples is via flow in discrete cracks, rather than via permeation. Still, such calculations are more accurate than the procedures used in previous analyses of RC experiments. The determination of the gas-flux distribution to the channel surface by numerical calculations for experiments using well characterized model porous media (such as particle board) would be quite beneficial for gaining insight into RC front propagation.

9.1.2 Modeling

The first task of the mathematical modeling was to consider the effects of combustion promoters and inhibitors on channel characteristics. In previous work, global kinetic parameters were judged to be properties of the combustible porous solid. Yet, since the composition of the reaction zone is influenced by the composition of the injected gas, the effective global kinetic parameters might be strongly influenced by combustion additives. This would provide an additional control parameter by which RC could be better optimized to field conditions. Modeling suggests that promoters would favor smaller, faster propagating channels. This would be economically advantageous for RC in shallow coal seams. Inhibitors would favor larger, slower channels, that create greater permeability enhancement. This would be advantageous for operation at greater depths where insufficient permeability enhancement is achieved with combustion additives.

The second task of the modeling was to formulate a model for the propagation of a curved reverse combustion front in homogeneous porous media. This model finds application to experimental data analysis and as a submodel for two-dimensional models of RC channels. The most important consequence of a curved front is that the flow field has tangential as well as normal components (with respect to the combustion front) and that they vary tangentially along the front. Consequently, the temperature of the combustion front is nonuniform along the front. Additional components of heat transport arise that are not present in geometrically idealized configurations. By using a boundary-layer approach, these additional components of heat transport were incorporated into the framework established by previous planar models. A simple solution to the model was found based on approximations arising from previous work. RC experiments in tar sands and previous nonadiabatic RC models have found the propagation velocity to vary markedly with heat losses, yet the front temperature was found to be insensitive to heat losses.

The final task of the modeling was to formulate and exploit a transient, numerical simulation of RC channel evolution. The boundary-element method with cubic-hermetian elements was found to be well suited for determining the gas flux through ideal porous media to the RC channel surface. The flux calculations were combined with the one-dimensional

model for the propagation of a curved RC front developed in this study. The model was partially fit to data from previous experiments with a model porous medium. The simulations qualitatively demonstrate the effects of tangential heat transport, extinction, and total gas flow on the evolution of the RC channel. Tangential heat transport suppresses the formation of fingers and slows down the propagation of sharply point combustion fingers. Higher gas-flow rates create larger channels that propagate at the maximum speed allowed by the combustion relationship. Higher gas-flow rates favor channel bifurcation. Extinction causes a cessation of lateral growth of a RC finger.

9.2 Recommendations

9.2.1 Laboratory Studies

Future laboratory studies using model combustible porous media (such as particle board) should be subjected to more exhaustive analyses than those done in the past. Specifically, these analyses should include flux calculations to the channel surface, as well a quantitative estimation of heat losses. This point-wise analysis of the channel should yield benefits both by improving the empirical understanding of the characteristics of RC and by providing the data needed to fully fit RC channel propagation models.

Extraction of kinetic parameters for solid coal should use experiments optimized for this task. For this purpose, accurate knowledge about the gas flux and combustion front is needed, whereas propagation speed is of secondary importance. This implies experiments that do not exhibit channeling. The experiments should utilize relatively small samples, involve a minimum of sample preparation, and involve straightforward, standardized data analyses so that a large number of experiments may be carried out. These experiments should be used to more fully explore the effects of oxygen, back-pressure, and gas flow rate. Enough experiments should be conducted to allow extraction of statistically significant values for the kinetic parameters, or conversely, to demonstrate that the experimental data are not well correlated by global Arrhenius kinetic parameters.

A real-time X-ray imaging system would be highly desirable for transient observation of three-dimensional channels. Such a system would provide valuable insight into the structure of a RC combustion front by giving a relatively clear picture of the spatial distribution of mass change from devolatilization and char combustion. It would allow more sophisticated experiments than possible with less powerful observational techniques. For example, the response of channel characteristics to operating parameters could be explored while minimizing the influence of coal sample variability by changing the operating parameter part of the way through a run. Since the same coal sample is being used, intersample variability is factored out.

Channeling experiments in solid coal should have a very limited role since they are expensive, time-consuming, and the data generated are difficult to analyze. In a comprehensive experimental investigation of RC, they are necessary to identify the magnitude of the influence of coal structure on channel character. In addition, they are useful to demonstrate

the applicability of novel control techniques to coal. The initial testing of such techniques, however, should utilize model porous media.

9.2.2 Modeling

Existing RC theory for front propagation provides no criterion for the transition between combustion of only volatiles, to combustion of both volatiles and fixed carbon. Such a criterion would be a valuable addition to RC theory. Such a criterion might arise from a volume-averaged, diffusion-flame model for RC. This model would provide a more fundamental basis for RC theory, since by taking the appropriate limit, the approximate premixed combustion model should be recovered.

The transient simulation presented here provides a foundation for further work in channel evolution. The first step of this work should be experimental, however, rather than modeling. A number of two-dimensional experiments similar to those of Visser *et al.* (1985) should be conducted to generate the data necessary to fit the semi-empirical model of RC channel propagation used in this study. With a smaller number of parameters to consider, sufficient computational resources could be focused on the simulations so that the interference of spatial and temporal discretization would be minimal. The effects of tangential convection, which were ignored in this study to allow better resolution of tangential conduction, should be considered. Further work in identifying the characteristics of extinction is needed.

Existing lumped-parameter RC theory for channel evolution provides no criteria for the bifurcation of a combustion finger. Prior to Visser's *et al.* experiments, the number of channels was believed to be determined by events at the onset of the instability around the production well. This assumption no longer appears to be valid, given the current experimental evidence for RC in coal and particle board, as well as the numerical simulation of channel growth presented in this study. Without this assumption, the lumped-parameter models of Krantz, Gunn and coworkers, and of Mui and Miller, are no longer predictive. Consequently, a criterion for channel bifurcation is needed. This might arise out of a linear stability analysis of a self-preserving channel shape generated using the curved front propagation model presented in Chapter 4.

9.2.3 Implications for Underground Coal Gasification

Several techniques were considered in this study that might directly impact the efficacy of RC for linking in Underground coal gasification. The use of promoters and inhibitors should be explored through laboratory experiments, and, if found to be effective, through RC field tests.

The effect of making small changes in the initial channel shape was explored implicitly in the numerical simulation. Although the numerical studies suggest that the number of channels formed at the production well could be altered, this does not appear to be an effective method to influence the long term evolution of the RC channel, given the possibility of channel bifurcation. The presence of cracks or fractures in a coal seam would exasperate any tendency to bifurcate. Consequently, research efforts should not be directed to this area.

A novel method of influencing channel growth is suggested by the effect that small thermocouples had on channel growth. The channel tended to align with the thermocouple. This directed growth could arise from either the distortion of the flow field or from the higher thermal conductivity of the thermocouple compared to the coal. The higher thermal conductivity of a small diameter metal conductor should direct the course of channel evolution by enhancing the conduction of heat from the combustion zone to the preheat zone. This would speed the propagation of the front near the metal conductor, leading to a local increase in the gas flux. This concept of directing channel growth should be tested in laboratory experiments in model porous media. Provided that the laboratory experiments confirm the effect, research efforts should be directed to develop a suitable micro-drilling technique to emplace a small diameter metal conductor in a coal seam. A possible micro-drilling method would cut the required hole in the coal with a small diameter oxygen lance. The lance itself would serve as the metal conductor during subsequent RC. Significant efforts would be required to develop a technique to drill a small diameter hole (5-20 mm) to a length of 3 to 20 meters at the bottom of a vertical well.

REFERENCES

- Abramowitz, M., and I.E. Stegun, **Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables** New York: Dover Publications, p. 304 (1964)
- Avasthi, J.M., and J.K.Lee, "Subsidence Modeling of Rawlins Underground Coal Gasification Tests 1 and 2" Proc. 10th Annual Underground Coal Gasification Symp., USDOE/METC-85/5 (DE85001956) p.502 (1984)
- Bartke, T.C. and R.D. Gunn, "The Hanna Wyoming Underground Coal Gasification Field Test Series" in **Underground Coal Gasification: The State of the Art** W. B. Krantz and R. D. Gunn, eds, AIChE Symp. Ser no 226, 79 pp. 4-14 (1983)
- Bell, G.J., "Comparison of Laboratory Combustion Tube Studies with Underground Coal Gasification Field Tests" M.S. thesis, University of Wyoming, Laramie, Wyoming (1982)
- Brebbia, C.A., **The Boundary Element Method for Engineers**, Pentech Press, London:Plymouth (1978)
- Breidung, K.P., "Stofftransport und Erhöhung der Gasdurchlässigkeit bei der in-situ-Vergasung Tiefliegender Steinkohleflöße", Ph.D. Dissertation, Rheinisch-Westfälische Technische Hochschule, Aachen, West Germany (1982)
- Britten, J.A., "Activation Energy Asymptotic Modeling of Reverse Combustion Instabilities in Porous Media: Application to In Situ Coal Gasification" Ph.D. Dissertation, University of Colorado, Boulder, Colorado (1984)
- Britten, J.A., and W.B. Krantz, "Linear Stability of Planar Reverse Combustion in Porous Media" *Combustion and Flame*, Vol. 60, pp. 125-40 (1985)
- Britten, J.A., W.B. Krantz, and R.D. Gunn, "Modeling Studies of Reverse Combustion Linking at High Pressure" Proc. 8th Underground Coal Conversion Symp., Report No. SAND82-2355, Sandia National Laboratories, pp. 515-24 (1982)
- Britten, J.A., W.B. Krantz, W. Visser, and R.D. Gunn, "Theoretical and Experimental Studies of Reverse Combustion" Proc. 10th Annual Underground Coal Gasification Symp., USDOE/METC-85/5 (DE85001956) pp. 357-366 (1984)
- Britten, J.A., C.B. Thorsness, "Further Development of an Axisymmetric Global UCG Cavity Growth Simulator" Proc. 13th Annual Underground Coal Gasification Symp., USDOE/METC-88/6095 (DE88010255) pp. 99-109 (1987)

- Bumb, A.C., "The Effects of Cylindrical and Spherical Geometry on the Stability of Reverse Combustion in Underground Coal Gasification" M.S. thesis, University of Wyoming, Laramie, Wyoming (1981)
- Carson, P.T., "An Investigation of Reverse Combustion Along Fissures in Pyrolyzing Fuels" M.S. thesis, University of Washington, Washington (1977)
- Cena, R.J., J.A. Britten, and C.B. Thorsness, "Excavation of the Partial Seam CRIP Underground Coal Gasification Test Site" Proc. 13th Annual Underground Coal Gasification Symp., USDOE/METC-88/6095 p. 382 (1987)
- Cena R.J., J.A. Britten, and C. B. Thorsness, "Resource Recovery and Cavity Growth during the Rocky Mountain I Field Test" Proc. 14th Annual Underground Coal Gasification Symp., USDOE/METC-88/6097 p. 202 (1988)
- Clavin, p. B. Denet, J. Monteillier, and P. Pelce, "Study of Reaction-Diffusion Waves Representative of Reverse Combustion in Porous Combustible Media" *Journal of Theoretical and Applied Mechanics* Numéro spécial, pp. 173-192 (1986)
- Cooney, D.O., "Final Technical Report: Experimental Aspects of Reverse Combustion" Unpublished internal report, University of Wyoming, Laramie, Wyoming (1979)
- Corlett, R.C., and C.F. Brandenburg, "Combustion Processes in In Situ Coal Gasification: Phenomena, Conceptual Models and Research Status. Part I - Overview and Continuum Wave Descriptions" 1977 Spring Meeting of the Western States Sections of the Combustion Institute, Seattle, Washington, April (1977)
- Corlett, R.C., and P.T. Carson, "Future Directions for Combustion Modeling of Underground Coal Conversion" Proc. 3rd Annual Underground Coal Gasification Symp., Conf-770652 p. 168 (1977)
- Corlett R.C. and D.M. Shearer, "Reverse Combustion Along Fissures in Shrinking Coals" Proc. 4th Annual Underground Coal Conversion Symp., SAND 78-0941 p. 231 (1978)
- Cupps, C.Q. , C.S. Land and L.C. Marchant, "Field Experiment of In-Situ Oil Recovery from a Utah Tar Sand by Reverse Combustion" AIChE Symp. Ser. no. 155, V72 pp. 61-76 (1976)
- Diamond, W. J., **Practical Experiment Designs for Scientists and Engineers** Lifetime Learning Publications, Belmont, California (1981)
- Dindi, H., "Dielectric Breakdown Front Propagation: Application to Electrolinking for In Situ Coal Gasification" Ph.D. dissertation, University of Colorado, Boulder, Colorado (1988)
- Dobbs II, R.L., W.B. Krantz, "Optimizing Reverse Combustion with Promoters and Inhibitors" Proc. 13th Annual Underground Coal Gasification Symp., USDOE/METC-88/6095 (DE88010255) pp. 202-210 (1987)

- Dobbs II, R.L., W.B. Krantz, "A Transient Model of Reverse Combustion Channel Propagation" Proc. 14th Annual Underground Coal Gasification Symp., USDOE/METC-88/6097 (DE88001093) pp. 102-106 (1988)
- Fuhrmann F., "Zur Technik der Gegenstromverbrennung für die flözzgäige Bohrlochverbindung in der Untertage-Kohlevergasung" Ph.D. Dissertation. Rheinisch-Westfälische Technische Hochschule, Aachen, West Germany, Aachen (1987)
- Fuhrmann, F., K.J. Guntermann, and H.W. Gudenau "Simulation of a Two-Dimensional Gas Flow in a Coal Seam - Modeling and Calculation in respect to Reverse Combustion Linking - " Proc. 14th Annual Underground Coal Gasification Symp., USDOE/METC-88/6097 (DE88001093) pp. 381-387 (1988)
- Gaudelle, C., D. Pavone, J.-F. Raffoux, A. Ternot, "Status of French Field Test at La Haute Deule" Proc. 11th Annual Underground Coal Gasification Symp., DOE/METC-85/6028 pp. 92-106 (1985)
- Gregg, D.W., "Relative Merits of Alternate Linking Techniques for Underground Coal Gasification and Their System Design Implications" *In-Situ* 4 pp. 207-236 (1980)
- Gregg, D.W., R.W. Hill, and D.U. Olness, "An Overview of the Soviet Effort in Underground Gasification of Coal" Lawrence Livermore Laboratory Rept. No. UCRL-50024 (1976)
- Greenough, W., personal communication, (1990)
- Gunn, R.D., W.B. Krantz, "Reverse Combustion Instabilities in Tar Sands and Coal" *SPE J* Vol. 20, pp. 267-277 (1980)
- Hele-Shaw, H.S. and A. Hay, "Lines of Induction in a Magnetic Field" *Phil. Trans. Roy. Soc.*, Vol. 195A, p. 303, Fig. 22. (1901)
- Howison, S.D., "Fingering in Hele-Shaw Cells" *J. Fluid Mech.* Vol. 167, pp. 439-453 (1986)
- Hubbert, M.K., **The Theory of Ground-Water Motion and Related Papers** Hafner, New York City, New York p. 37 (1969)
- IMSL, Inc., **MATH/LIBRARY(T.M.) - FORTRAN Subroutines for Mathematical Applications** IMSL, Inc., Houston, Texas (1982)
- IMSL, Inc., **MATH/LIBRARY(T.M.) - FORTRAN Subroutines for Mathematical Applications** IMSL, Inc., Houston, Texas (1987)
- Karrila, S.J., and S. Kim, "Integral Equations of the Second Kind for Stokes Flow: Direct Solution for Physical Variables and Removal of Inherent Accuracy Limitations" *Chem. Eng. Comm.* Vol 82, pp. 123-161 (1989)
- Kotowski, M.D., and R.D. Gunn, "Theoretical Aspects of Reverse Combustion in the Underground Gasification of Coal" Laramie Energy Research Center, LERC/RI-76/4 (1976)

- Krantz W.B., R.D. Gunn, "Linear Stability of Reverse Combustion for In Situ Coal Gasification" Proc. 3rd Annual Underground Coal Gasification Symp., pp. 146-159 (1977)
- Krantz, W.B., R.D. Gunn, "Use of Stability Theory in Interpreting UCG Field Test Data" *In Situ* Vol. 5(3) pp. 199-219 (1981)
- Krantz, W.B., R.D. Gunn, editors *Underground Coal Gasification: The State of the Art* AIChE Symposium Series, No. 226, Vol. 79 (1983)
- Kurth, M., K.P. Breidung, P. Mohtadi, and H. Gudenau, "Reverse Combustion Study of a Coal Block of 4m Length under Pressure" Proc. 7th Underground Coal Gasification Symp., Lawrence Livermore National Laboratory, Report No. CONF 810923, pp. 371-376 (1981)
- Ledent, P., T.K. Li, V. Chandelle, R. Fabry, M. Kurth, C. Sonntag, "First Results of the Belgian-German UCG-Test at Great Depth" Proc. 9th Underground Coal Gasification Symp., DOE/METC/84-7 (DE84003052) p. 30 (1984)
- Liggett, J.A., and P.L.-F. Liu, **The Boundary Integral Equation Method for Porous Media Flow** George Allen & Unwin, London (1983)
- Lovell, R.W., "Wet Reverse Combustion During Underground Coal Gasification" M.S. thesis, University of Wyoming, Laramie, Wyoming (1982)
- Lu, H.S., "Laboratory Simulation for Reverse Combustion of UCG" M.S. thesis, University of Wyoming, Laramie, Wyoming, (1980)
- Martin, J.W. and A.J. Liberatore "A Successful Underground Coal Gasification Field Test in a Thin Seam, Swelling Eastern Bituminous Coal" Proc. of the Sixth Underground Coal Conversion Symp., USDOE/Conf No. 800716 p. I-8 (1980)
- Monteillier, J., "Etude théorique de la propagation des fronts à combustion à contre courant dans une veine poreuse de charbon" Thèse, Docteur de 3e Cycle, Université de Provence, Provence, France (1986)
- Mui, K.C., C.A. Miller, "Stability of Displacement Fronts in Porous Media — Growth of Large Elliptical Finger" SPE of AIME Technical Paper 11542, February (1983)
- Okutan, H. "Stability Analysis of Wet and Dry Combustion in Coal" M.S. thesis University of Wyoming, Laramie, Wyoming (1981)
- Oliver, R.L., "Results from the Controlled Excavation of the TONO I UCG Cavity, Centralia, Washington" Proc. 13th Annual Underground Coal Gasification Symp., USDOE/METC-83/6095 p. 371 (1987)
- Pitin, R.N., "Effect of the Blast Flow Rate on the Displacement of the Zones in the Linking Stage During Underground Coal Gasification" PGU 1, 30-35 (1959) (UCRL Trans-11130, University of California Lawrence Livermore Laboratory, Livermore, California (1976))

- Puri, R., "Theoretical Modeling of Reverse Combustion Instabilities in Coal and Tar Sands", M.S. thesis, University of Colorado, Boulder, Colorado (1979)
- Ramirez, W.F. **Computational Methods for Process Simulation** Butterworth Publishers, Stoneham, MA (1989)
- Reed R.L., D.W. Reed, and J.H. Tracht, "Experimental Aspects of Reverse Combustion in Tar Sands" *Pet. Trans. AIME* Vol. 219, pp. 99-108 (1960)
- Roehl, A.A., R.A.S. Brown, N.C. Edward, and E.J. Jensen, "Underground Coal Gasification in Canada - An Assessment of the Alberta Test" Proc. 4th Underground Coal Conversion Symp., pp. 97-102 (1978)
- Saffman, P.G. and G.I. Taylor, "The Penetration of a Fluid into a Porous Media of Hele-Shaw Cell Containing a More Viscous Fluid " Proc. Roy. Soc. (London), A245, pp. 312-329 (1958)
- Schloemer, H.J., J. Wagner, "Reverse Channel Burning as a Method for Final Linking at Great Depths" Proc. 14th Annual Underground Coal Gasification Symp., DOE/METC-88/6097 (DE88001093) pp. 114-122 (1988)
- Shirey, D.L. and B.P. Engler " Development of a Cornering Water Jet Drill" Proc. 10th Annual Underground Coal Gasification Symp., DOE/METC-85/5 (DE85001956) pp. 79-91 (1984)
- Su, F.Y. and V.S. Engleman, "Reverse Combustion Link Phenomena in Bituminous Coal at High Pressure" Proc. 5th Annual Underground Coal Gasification Symp., CONF. NO: 790630 pp. 323-330 (1979)
- Tantekin, S.B., "Laboratory Characterization of the Spalling Properties of Overburden Rock in Underground Coal Gasification" M.S. thesis, University of Colorado, (1986)
- Thorsness, C.B., R.B. Rozsa, and R. Wong, "Two-Dimensional Modeling of In-Situ Coal Gasification" Proc. 3rd Annual Underground Coal Gasification Symp., Conf-770652 pp. 160-167 (1977)
- Van Horn, K.R., editor, **Aluminum, Volume Three, Fabrication and Finishing** American Society for Metals, Metals Park, Ohio p. 327 (1967)
- Visser, W., W.B. Krantz, J.A. Britten, and R.D. Gunn, "Experiments in Reverse Combustion" Proc. 11th Annual Underground Coal Gasification Symp., USDOE/METC-85/6028 (DE85013720) pp. 340-349 (1985)
- Watson, J.O., "Hermetian Cubic Boundary Elements for Plane Elastostatics" 2nd Int. Symp. on Innovative Numerical Analysis in Applied Engineering Science, University of Virginia (1980)
- Watson, J.O., "Hermetian Cubic Boundary Elements for Plane Problems of Fracture Mechanics" *Res Mechanica* Vol. 4 pp. 23-42 (1982)

Wilson L.A., R.L. Reed, D.W. Reed, R.R. Clay, and N.H. Harrison, "Some Effects of Pressure on Forward and Reverse Combustion" *AIME Trans.* 228 pp. 127-137 (1963)

Wolf K.-H.A.A., R.A. Kuhnel, and C.J. de Pater, "Thermal Behavior of Country Rock in Laboratory Experiments" Proc. 13th Annual Underground Coal Gasification Symp., DOE/METC-88/6095 (DE88010255) pp. 163-172 (1987)

NOMENCLATURE

Nomenclature for Chapter 3: Roman

A^*	Effective pre-exponential factor
A	Dimensionless pre-exponential factor
C_s	Heat capacity of the solid phase
C_g	Heat capacity of the gaseous phase
D	Effective molar diffusivity of the oxidant
E	Effective activation energy
erf	Error function
Le	Lewis number ($\frac{\lambda N_g}{D \rho_g \Sigma}$)
N_g	Gas flux
N_1	Dimensionless group defined by Equation 3.19
N_2	Dimensionless group ($\frac{\varphi \rho_g V}{N_g}$) defined by Equation 3.20
P	Absolute pressure
q	Effective molar heat of reaction
Q	Dimensionless heat of reaction ($\frac{Y_{-\infty} N_g q}{\Sigma E/R}$)
R	Universal gas constant
r^*	Dimensional volumetric reaction rate
r	Dimensionless reaction rate ($A e^{-1/T} Y$)
T^*	Absolute temperature
T	Scaled temperature variable ($\frac{T^*}{E/R}$)
$T_{-\infty}$	Dimensionless ambient temperature ($\frac{T_{-\infty}^*}{E/R}$)
$\frac{1}{T_{-\infty}}$	Dimensionless independent parameter proportional to activation energy and independent of pre-exponential factor
t^*	Time (seconds)
V	Velocity of combustion front w.r.t. stationary coordinates
x^*	Dimensional spatial coordinate
Y^*	Mole fraction of the oxidant
Y	Scaled oxidant concentration ($\frac{Y^*}{Y_{-\infty}^*}$)

Greek

β	Ajustable parameter in trial function
---------	---------------------------------------

η^*	Spatial coordinate centered on the combustion front
η	Scaled spatial coordinate w.r.t. moving coordinate system ($\frac{\eta^*}{\lambda/\Sigma}$)
Θ	Transformed temperature variable
λ	Effective thermal diffusivity
ρ_g	Density of gas phase
ρ_s	Density of solid phase
Σ	$C_s \rho_s V + C_g N_g$
τ	Scaled time variable ($\frac{\tau^*}{C_s \rho_s \lambda / \Sigma^2}$)
φ	Porosity

Subscripts

s	Solid phase
g	Gas phase
t	Trial function
r	Resulting
$-\infty$	Condition far upstream of combustion front

Superscripts

$*$	Dimensional
[none]	Dimensionless (generally)

Nomenclature for Chapter 4:

Roman

A	Arrhenius pre-exponential factor
C_g	Heat capacity of the gas phase
C_s	Heat capacity of the solid phase
c	Integration constant
D_{eff}	Effective mass diffusivity
D'	$\mathcal{P}d$
d	Integration constant
f_{vmr}	Fraction of volatile matter released
f	Spatial variation in tangential heat transport
F	$\int_{-\infty}^{\eta} f d\eta$
G_{Θ}	Decay constant for tangential heat transport in preheat zone
$G_{\oplus}$	Decay constant for tangential heat transport in post-combustion zone
g	$\frac{g'}{1+g'}$
g'	Ratio of the heat transport zone to the length scale of the preheat zone

$H_{\oplus}$	$\frac{G_{\oplus}}{1+V_0}$
$H_{\ominus}$	$\frac{G_{\ominus}}{1+V_0}$
$N_{g,x_1,f}$	Normal component of the gas flux at the front
$N_{g,x_2,f}$	Tangential component of the gas flux
$N_{g,x_3,f}$	Tangential components of the gas flux
N_g	Mass flux of gas
$N_{g,f}$	Mass gas flux at the combustion front
$N_{g,a}$	Mass gas flux prior to devolatilization
P	Absolute pressure
q	Heat of combustion
r^*	Reaction rate
R	Universal gas constant
t	Dimensionless time
t^*	Time
T	Dimensionless temperature
$T_{\odot}$	Inner zone temperature solution, $\eta = 0$
$T_{\ominus}$	Outer zone temperature solution, $\eta < 0$
$T_{\oplus}$	Outer zone temperature solution, $\eta > 0$
T^*	Absolute temperature
T_{amb}^*	Absolute ambient temperature far ahead of the front
T_f^*	Combustion front temperature
T_f	Dimensionless combustion front temperature
v_m	Weight percent volatile matter in the coal
V	Front propagation velocity
V_f	Propagation speed of the front
W	Scale function
x	Dimensionless spatial coordinate in the principal direction
x_1	Principle spatial coordinate
x_2	Tangential spatial coordinate
x_3	Tangential spatial coordinate
Y	Dimensionless concentration of the limiting reactant
$Y_{O_x}^*$	Oxygen mass fraction
$Y_{I_n}^*$	Inerts mass fraction
$Y_{F_u}^*$	Fuel mass fraction
Y_l^*	Mass fraction of the limiting reactant
Y_{nl}^*	Mass fraction of the nonlimiting reactant

$Y_{nl,f}^*$	Mass fraction of the nonlimiting reactant at onset of combustion
$Y_{l,f}^*$	Mass fraction of limiting reactant prior to combustion
z	$\frac{\partial \theta_1}{\partial \xi} - \gamma_\oplus$

Calligraphic

A	Dimensionless pre-exponential factor
B	Ratio of the specific heat of the gas phase to that of the solid
D_0	A Damkohler number defined by Equation 4.63
$\mathcal{E}$	Dimensionless activation energy
$\mathcal{L}$	Lewis number, dimensionless reciprocal diffusion coefficient
$\mathcal{M}$	$AWe^{-\mathcal{E}/(Tf)}$
$\mathcal{P}$	$\frac{\mathcal{E}z_1}{T_f}$
Q	Dimensionless heat of reaction
Q'_0	A rescaled dimensionless heat of reaction defined by Equation 4.64
S	Tangential heat transport source or sink term
S_d	Tangential conduction source or sink term
S_f	Total tangential heat transport source or sink term
S_v	Tangential convection source or sink term
S'_{f0}	Rescaled total tangential heat transport source or sink term defined by Equation 4.53
T	Rescaled dimensionless temperature

Greek

α	Ratio of combustion front temperature to ambient temperature
$\gamma_\oplus$	$\left. \frac{\partial \theta_1}{\partial \xi} \right _{\xi \rightarrow +\infty}$
$\gamma_\ominus$	$\left. \frac{\partial \theta_1}{\partial \xi} \right _{\xi \rightarrow -\infty}$
δ	Perturbation parameter, $1/\mathcal{E}$
η	Dimensionless moving spatial variable
λ_s	Thermal conductivity of the solid phase
φ	Porosity
ρ_s	Density of the solid phase
$\rho_{s,a}$	Coal density prior to devolatilization
ρ_g	Density of the gas phase
τ	Dimensionless temporal variable
ν_l	Stoichiometric coefficient of the limiting reactant
ξ	Stretched spatial coordinate
θ	Inner zone temperature
ζ	Rescaled stretched spatial coordinate

Symbols

∇^*	Dimensional gradient
∇^{*2}	Dimensional laplacian
$\asymp$	"is asymptotic to"

Subscripts

a	Conditions prior to devolatilization
f	Conditions at the front before combustion
l	Limiting reactant
nl	Nonlimiting reactant
$\ominus$	$\eta < 0$
$\odot$	$\eta = 0$
$\oplus$	$\eta > 0$

1, 2, etc. Order of perturbation quantities

Superscripts

+	Limit approaching zero from positive numbers
-	Limit approaching zero from negative numbers

Nomenclature for Chapter 7:

Roman

A	Dummy variable of integration
g	An arbitrary boundary variable
M_+	A cubic hermetian shape function
M_-	A cubic hermetian shape function
N_+	A cubic hermetian shape function
N_-	A cubic hermetian shape function
N	Normal component of the flux
N_g	Gas flux vector
n	Unit normal vector
P	Pressure
P_0	The pressure at the singular point
r	Distance from singular point
r_0	The radius of σ .
S	Jacobian (derivative of arclength w.r.t. ξ)
s	Dummy variable of integration
t	A parameterization variable for a boundary
U	An arbitrary function
V	An arbitrary function

x	Cartesian coordinate for the boundary
y	Cartesian coordinate for the boundary

Greek

α	The angle between the tangents to the surface at the singularity
Γ	The surface with a gap at the singularity
Δx	A variable used in defining the tangential derivative of x at a node
Δy	A variable used in defining the tangential derivative of y at a node
ΔP	A variable used in defining the tangential derivative of P at a node
ΔN	A variable used in defining the tangential derivative of N at a node
$\Delta \xi_-$	A variable used in defining tangential derivatives at a node
$\Delta \xi_+$	A variable used in defining tangential derivatives at a node
κ	Permeability coefficient tensor
μ_g	Gas viscosity
ρ_g	Gas density
σ	Transmissivity of porous media
σ	A circular arc surrounding the singular point
τ	Dummy variable of intergration
ξ	A local coordinate

Subscript

-	Evaluated for the negative side of a node
+	Evaluated for the positive side of a node
0	At the singular point

Nomenclature for Chapter 8:

Roman

C_g	Heat capacity of gas phase
g	Dimensionless tangential heat transport boundary layer thickness
$N_{g,a}$	Normal component of gas flux at front
$N_{g,aS}$	Scale factor for normal component of gas flux
T_f	Combustion front temperature
V	Dimensionless front velocity as defined in Chapter 4
x_2^*	Arc length coordinate
x_{2S}^*	Scale factor for arc length coordinate

Caligraphic

$\mathcal{F}$	Extinction factor
$\mathcal{H}$	Dimensionless tangential heat transfer coefficient

$\mathcal{N}$	Dimensionless normal component of gas flux
$\mathcal{N}_{\min}$	Minimum dimensionless gas flux for front propagation
$\mathcal{Q}$	See Chapter 4
S_f	See Chapter 4
$\mathcal{V}$	$\mathcal{NVF}$
$\mathcal{X}$	Dimensionless arc length coordinate
Greek	
λ_s	Thermal conductivity of the solid

APPENDIX A

COMPUTER PROGRAMS FROM CHAPTER 4

A.1 Program FEWCHT

```

C      PROGRAM FEWCHT ! Find Eigenvalue while considering heat transport
C
      REAL gPrime, ScriptQ, ScriptE, ScriptA, ScriptSf
      REAL Gibbit, VO, Tf
      gPrime = 1.
      ScriptQ = 5.
      ScriptE = 10.
      ScriptA = 10
C
C      WRITE (*,*) 'ScriptSf      Gibbit      Gibbit*VO,      Tf'
C
      DO ScriptSf = 1., -1., -0.5
        GibbitMax = 0.45 + 0.2 * ScriptSf
        DO Gibbit = 0.005, GibbitMax, GibbitMax / 12.
          ScriptA = 1. / (Gibbit * Gibbit)
          CALL GetEigen(ScriptQ, ScriptA, ScriptE, ScriptSf, gPrime,
&                      VO, Tf)
          WRITE (*,*) ScriptSf, Gibbit, Gibbit*VO, Tf
        END DO
      END DO
      WRITE (*,*)
      END DO
      RETURN
      END
C
C
C
C      SUBROUTINE GetEigen (ScrQ, ScrA, ScrE, ScrSf, gP, VO, Tf)
C
      INTEGER ItMax, N, iErrorSeverity, iPrintAction, iStopAction
      INTEGER IERCD, iErrorCode
      EXTERNAL IERCD
      REAL ScrQ, ScrA, ScrE, ScrSf, gP, VO, Tf, Cont
      REAL ScrQb, ScrAb, ScrEb, ScrSfb, gPb
      REAL ErrRel
      Parameter (N = 3)
      REAL Fnorm, X(N), XGuess(N)
      EXTERNAL Fcn, NEqnF, Adiabatic
      REAL ScriptA, ScriptQ, ScriptE, ScriptSf, gprime
      COMMON /BackDoor/ ScriptA, ScriptQ, ScriptE, ScriptSf, gprime
C
      ErrRel = 0.0001
      ItMax = 100
      iErrorSeverity = 4
      iPrintAction = 0
      iStopAction = 0
C
      CALL ErSet (iErrorSeverity, iPrintAction, iStopAction)
C
      ScrQb = 5.
      ScrEb = 10.
      ScrSfb = 0.
      ScrAb = 10.
      gPb = 1.

```

```

C
XGuess(1) = LOG(0.167)
XGuess(2) = LOG(5.)
ScriptQ = ScrQb
ScriptE = ScrEb
ScriptSf = ScrSfb
ScriptA = ScrAb
gprime = gPb

C
CALL NEqnF (Adiabatic, ErrRel, N, ItMax, XGuess, X, FNorm)
XGuess(1) = X(1)
XGuess(2) = X(2)

C
iErrorCode = 0.
DO Cont = 0., 1., 0.05

C
    ScriptQ = Cont * ScrQ + (1. - Cont) * ScrQb
    ScriptE = Cont * ScrE + (1. - Cont) * ScrEb
    ScriptSf = Cont * ScrSf + (1. - Cont) * ScrSfb
    ScriptA = Cont * ScrA + (1. - Cont) * ScrAb
    gprime = Cont * gP + (1. - Cont) * gPb

C
    CALL NEqnF (Fcn, ErrRel, N, ItMax, XGuess, X, FNorm)
    IF (IERCD( ) .NE. 0) THEN
        iErrorCode = 1
        EXIT
    END IF
    XGuess(1) = X(1)
    XGuess(2) = X(2)
END DO
IF (iErrorCode .eq. 1) THEN
    VO = 0.
    Tf = 1.
ELSE
    VO = exp(X(1))
    Tf = exp(X(2))
END IF

C
RETURN
END

C
C
C
SUBROUTINE Fcn (X, F, N)

C
INTEGER N
REAL X(N), F(N)
REAL C1, C2, DObyAlpha1Adia, Gamm, Gamp, SfOPrime, QOPrime
REAL ScriptA, ScriptQ, ScriptE, ScriptSf, gprime, VO, Tf
COMMON /BackDoor/ ScriptA, ScriptQ, ScriptE, ScriptSf, gprime

C
VO = exp(X(1))
Tf = exp(X(2))

C
SfOPrime = ScriptSf / (Tf * (1. + VO) * (1. + VO))
QOPrime = ScriptQ / (Tf / (1. + VO))

C
Gamp = SfOPrime
Gamm = 1. - SfOPrime / gprime

C
DObyAlpha1Adia = (ScriptA * Tf * Tf / (ScriptQ * ScriptE))
& * EXP (-ScriptE / Tf)

C
C1 = (1. - SfOPrime * (1./gprime + 1.))
C2 = Gamm + Gamp * LOG (Gamm - Gamp)
& - Gamp * LOG ( DObyAlpha1Adia * C1 - Gamp)
F(1) = 2.* DObyAlpha1Adia * C1 - C2
F(2) = VO - (ScriptQ / Tf - C1) / C1

```



```

C      RETURN
C      END
C
C      SUBROUTINE Adiabatic (X, F, N)
C
C      INTEGER N
C      REAL X(N), F(N)
C      REAL DObyAlpha1Adia, Tf, VO
C
C      REAL ScriptA, ScriptQ, ScriptE, ScriptSf, gprime
C      COMMON /BackDoor/ ScriptA, ScriptQ, ScriptE, ScriptSf, gprime
C
C      VO = exp(X(1))
C      Tf = exp(X(2))
C
C      DObyAlpha1Adia = (ScriptA * Tf * Tf / (ScriptQ * ScriptE))
C      & * EXP (- ScriptE / Tf)
C
C      F(1) = 2.* DObyAlpha1Adia - 1
C      F(2) = VO - (ScriptQ / Tf - 1.)
C
C      RETURN
C      END

```

A.2 Subroutine FULLCOMBUST

```

SUBROUTINE FullCombust (
&   Nga, LapNga, ConvTerm, Pres, Yoxa, Yfa, A, EbyR,
&   TfStarNew, VfStarNew, Ylf, Ynlf, Betal,
&   iErrorCode, ScriptSf, ScriptQ, lFixedScriptSf )
C
C *** Input Variables
C   Set 1 : These are spatial dependent along the front.
C   Nga      Normal gas flux before devol., kg/hr/m/m
C   LapNga    2d Laplacian of normal gas flux before devol.
C             kg/hr/m/m/m/m
C   ConvTerm  Tangential Gas Flux Dotted into the 2d Gradient
C             of the normal gas flux before devol.,
C             (kg/hr/m/m) (kg/hr/m/m) / m
C
C   Set 2 : These are independent of the location on the front.
C   Pres      Pressure, kPa
C   Yoxa      Oxygen mass fraction before devol.
C   Yfa      Fuel mass fraction before devol.
C   A        Pre-Exponential factor,
C   EbyR      Activation energy / Gas Constant, K
C
C *** Output Variables:
C   Tfstar    Front temperature, K
C   Vfstar    Front velocity, K
C   YOxf      Oxygen mass fraction after devol.
C   Yff       Fuel mass fraction after devol.
C   Betal     Stoichiometric Coefficient of Limiting Reactant
C   iErrorCode
C
C   REAL Nga, LapNga, NgaPlus
C   REAL ConvTerm, Pres, Yoxa, Yfa, A, EbyR
C   REAL TfStar, VfStar, Ylf, Ynlf, Betal
C   REAL TfStarNew, VfStarNew, h
C   REAL TfStarPlus, VfStarPlus
C   REAL TfStarNewPlus, VfStarNewPlus
C   REAL dTfstarNga
C   LOGICAL lFixedScriptSf
C
C   INTEGER i, iErrorCode
C
C   INCLUDE 'LOCALCMN.FI'      ! Common block definitions
C
C   iErrorCode = 0
C   If (Nga .le. 0.) THEN
C     VfStarNew = 0.
C     TfStarNew = Tamb
C     iErrorCode = 254
C   ELSE
C     CALL LoadBackDoor
C     &   (Nga, LapNga, ConvTerm, Pres, YOxa, Yfa, EbyR, A)
C     h = 0.5
C     Tfstar = 750 + 1.7 * Nga ! Degrees Kelvin
C     TfstarPlus = Tfstar
C
C     Vfstar = 0.08 ! m / hr
C     VfstarPlus = Vfstar
C
C     NgaPlus = Nga + 2.
C     DO i = 0, 250
C
C       CALL OnceThrough
C       &   (dTfstarNga, NgaPlus, TfstarPlus, VfstarPlus,
C       &   TfstarNewPlus, VfstarNewPlus, Ylf, Ynlf, Betal,

```

```

C      &      ScriptSf, ScriptQ, lFixedScriptSf)
C
C      CALL OnceThrough
C      &      (dTfistardNga, Nga, Tfistar, Vistar,
C      &      TfistarNew, VistarNew, Ylf, Ynlf, Betal,
C      &      ScriptSf, ScriptQ, lFixedScriptSf)
C
C      IF ((ABS(Tfistar - TfistarNew) .lt. .01)
C      &      .AND. (ABS(Vistar - VistarNew) .lt. 1.e-5)) THEN
C          EXIT
C      END IF
C      Tfistar = (1. - h) * Tfistar + h * TfistarNew
C      Vistar = (1. - h) * Vistar + h * VistarNew
C      TfistarPlus = (1. - h) * TfistarPlus + h * TfistarNewPlus
C      VistarPlus = (1. - h) * VistarPlus + h * VistarNewPlus
C      IF ( ((TfistarNew .gt. 1.2 * Tamb)
C      &      .and. (TfistarNewPlus .gt. 1.2 * Tamb ))
C      &      .and. (TfistarPlus .gt. Tfistar) ) THEN
C          dTfistardNga = (TfistarPlus - Tfistar) /
C          &      (NgaPlus - Nga)
C      END IF
C
C      END DO
C      END IF
C
C      IF ((VistarNew .lt. 0.) .or. (i .gt. 200)) THEN
C          TfistarNew = Tamb
C          VistarNew = 0.
C          iErrorCode = 255
C      END IF
C
C      RETURN
C      END

```

A.2.1 Support subroutines COMBSUP.FOR

```

C
C
C
C      SUBROUTINE LoadBackDoor
C      &      (Nga, LapNga, ConvTerm, Pres, YOxa, Yfa, EbyR, A)
C
C      This routine passes parameters to the support
C      routines since they can not be passed through the
C      calling list because of the limitations of the
C      intervening IMSL routine NEQNF.
C
C      INCLUDE 'LOCALCMN.FI'      ! Common block definitions
C
C      REAL Nga, LapNga
C
C      bdNga = Nga
C      bdLapNga = LapNga
C      bdConvTerm = ConvTerm
C      bdPres = Pres
C      bdYOxa = YOxa
C      bdYfa = Yfa
C      bdEbyR = EbyR
C      bdA = A
C
C      RETURN
C      END
C
C      SUBROUTINE GetDimensionlessGroups (
C      &      Tfistar, Vistar, Ngf, Ylf, Ynlf, Betal,

```

```

&      Tf, V,
&      ScriptA, ScriptB, ScriptQ, ScriptE,
&      ScriptSd, ScriptSv )
C
C      Input variables:
C      Refer to SUBROUTINE GetGasCompositionAtFront
C
C      Output variables:
C      Tf, V, ScriptE, ScriptA, ScriptQ, ScriptSd, ScriptSv, ScriptT
C      are all dimensionless quantities defined in the nomenclature
C      of the thesis.
C
C      IMPLICIT REAL (a - h, o - z)
C      REAL Ngf, Lambda
C      INCLUDE 'LOCALCMN.FI'          ! Common block definitions
C
C      Lambda = CharThermalDiffusivity (Tfstar)
C      Cg = GasHeatCapacity ((Tfstar + Tamb) * 0.5)
C      Cs = SolidHeatCapacity ((Tfstar + Tamb) * 0.5)
C      xScale = Lambda / (Cg * Ngf)
C      TimeScale = Cs * RhoS * xScale * xScale / Lambda
C      dTfdNga = bddTfstar * dNga / Tamb
C
C      Now we can convert the dimensional eigenvalue to the
C      dimensionless one:
C      V = Vfstar * TimeScale / xScale
C      Tf = Tfstar / Tamb
C
C      Calculate the various dimensionless groups
C      ScriptA =
&      Betal * Lambda * Phi * bda * bdPres * bdPres * Ynlf
&      / (Cg * Ngf * Ngf)
C
C      ScriptB = Phi * RhoG(Tfstar, bdPres) * Cg / (RhoS * Cs)
C
C      ScriptQ = HeatOfCombustionPerMassOfOxygen(Tfstar) * Ylf
&      / (Cg * Tamb * Betal)
C
C      ScriptE = bdEbyR / Tamb
C
C      Prefactor = xScale * xScale * dTfdNga
C      ScriptSd = Prefactor * bdLapNga
C
C      ScriptSv = Prefactor * bdConvTerm * Cg / Lambda
C
C      RETURN
C      END
C
C
C      SUBROUTINE GetGasCompositionAtFront (
&      Tfstar, Vfstar,
&      Ngf, Ylf, Ynlf, Betal)
C
C      Input Variables:
C      Refer to SUBROUTINE Combust
C
C      Output Variables:
C      Ngf      Gas Flux at front after Devolatilization
C      Ylf      Mass fraction of the limiting reactant at front
C      Ynlf     Mass fraction of the nonlimiting reactant at front
C      Betal    Stoichiometric coefficient for limiting reactant
C
C      IMPLICIT REAL (a - h, o - z)
C      REAL Ngf
C      INCLUDE 'LOCALCMN.FI'          ! Common block definitions
C
C      ** Calculate composition at front:

```

```

O      YIna = 1. - bdYOxa - bdYfa
      fvmr = FractionVolatileMatterReleased (Tfstar)
C
C *** Debugging Code:
      SolidFlux = (Vfstar * RHOs * (1. - (1. - fvmr) * Vm))
C
      Ngf = bdNga + (Vfstar * RHOs * Vm * fvmr)
C
      YOxf = bdYOxa * (bdNga / Ngf)
      YInf = YIna * (bdNga / Ngf)
      Yff = 1. - YOxf - YInf
C
      IF (Yff / stoich .lt. YOxf) THEN
C        Fuel is the limiting reactant
          Ylf = Yff
          Ynlf = YOxf
          Betal = stoich
      ELSE
C        Oxygen is the limiting reactant
          Ylf = YOxf
          Ynlf = Yff
          Betal = 1.
      ENDIF
C
      RETURN
      END
C
C
C
      BLOCK DATA DefineCoalProperties
C
      IMPLICIT REAL (a - h, o - z)
      INCLUDE 'LOCALCMN.FI'          ! Common block definitions
C
      DATA Vm /0.326/, RHOs /1400./, Stoich /0.541/,
      *          PHI /.40/, Tamb /298/
C
      END
C
C
C
      FUNCTION RhoG (TempInKelvin, PressureInkPa)
C
C *** Ideal Gas Law:
C *** [RhoG] = kg/m/m/m
C
      IMPLICIT REAL (a - h, o - z)
      WeightMolecular = 29.
      R = 8.14          ! Gas Constant in kPa * m * m * m / kg mol
      RhoG = PressureInkPa * WeightMolecular / (R * TempInKelvin)
C
      RETURN
      END
C
C
C
      FUNCTION SolidHeatCapacity (TempInKelvin)
C
C *** As Given in Britten's Thesis: (From Kowtowski & Gunn)
C *** [GasHeatCapacity] = kJ/kg/K
C
      IMPLICIT REAL (a - h, o - z)
      SolidHeatCapacity =
      * 4.183 * (0.06 + 0.00087984 * TempInKelvin )
C
      RETURN
      END
C

```

```

C
C
C      FUNCTION GasHeatCapacity (TempInKelvin)
C
C *** As Given in Britten's Thesis: (From Kowtowski & Gunn)
C *** [GasHeatCapacity] = kJ/kg/K
C
C      IMPLICIT REAL (a - h, o - z)
C      GasHeatCapacity =
C      &      4.183 * (0.2531 + 0.0000702 * TempInKelvin)
C
C      RETURN
C      END
C
C
C
C      FUNCTION HeatOfCombustionPerMassOfOxygen (TempAtFrontInKelvin)
C
C *** This routine encodes the correlation used by Kowtowski & Gunn
C *** for the heat of combustion per mass of Oxygen.
C *** Note that J.A. Britten incorrectly considered that
C *** this was on a per unit mass of fuel, not oxygen.
C *** [HeatOfCombustion] = kJ / kg
C
C      IMPLICIT REAL (a - h, o - z)
C      HeatOfCombustionPerMassOfOxygen =
C      &      11400. + 1.47 * TempAtFrontInKelvin
C
C      RETURN
C      END
C
C
C
C      FUNCTION CharThermalDiffusivity (TempInKelvin)
C
C *** As Given in Britten's Thesis: (From Kowtowski & Gunn)
C *** [CharThermalDiffusivity] = kJ/kg/K
C
C      IMPLICIT REAL (a - h, o - z)
C      IF (TempInKelvin .lt. 673.) THEN
C          CharThermalDiffusivity = 0.8939
C      ELSE
C          CharThermalDiffusivity = 6.707 *
C          &      (1.4382 - 0.0041413 * TempInKelvin
C          &      + 3.2724E-6 * TempInKelvin * TempInKelvin)
C      ENDIF
C
C      RETURN
C      END
C
C
C
C      FUNCTION FractionVolatileMatterReleased (TempAtFrontInKelvin)
C
C *** This Routine encodes the correlation used by Kowtowski & Gunn
C *** for the fraction of the volatile matter released as a function
C *** of Temperature
C
C      IMPLICIT REAL (a - h, o - z)
C      Tf = TempAtFrontInKelvin
C      Td = ((TempAtFrontInKelvin*1.8) - 460. - 617.) / 1035.
C      IF (Td .lt. 0.) THEN
C          fvmr = 0.
C      ELSEIF (Td .ge. 1.) THEN
C          fvmr = 1.
C      ELSEIF (Td .lt. 0.65217) THEN
C          fvmr = (1.0187 * (7.81976 * Td * Td) / (1 + 23.457 * Td * Td))
C          1 / 0.3264

```

```

ELSE
  fvmr = (0.3264 - 1.0187 * 0.16986 * (1. - Td)*(1. - Td)
1  /((1 + 1.50389 * (1. - Td)*(1-Td)) ) / 0.3264
ENDIF
FractionVolatileMatterReleased = fvmr

C
RETURN
END

C
C
C
SUBROUTINE OnceThrough
&      (dTfstartdNga, Nga, Tfstar, Vistar,
&      TfstarNew, VistarNew, Ylf, Ynlf, Betal,
&      ScriptSf, ScriptQ, lFixedScriptSf)

C
  INCLUDE 'LOCALCMN.FI'      ! Common block definitions

C
  REAL Ngf, Vnew, ScriptSf, Tfnew, QEbyA, ScriptSv, ScriptSd
  REAL ScriptE, ScriptA, ScriptQ, ScriptB, V, Tf
  REAL dTfstartdNga, Nga
  LOGICAL lFixedScriptSf

C
  bddTfstartdNga = dTfstartdNga
  bdNga = Nga

C
  CALL GetGasCompositionAtFront (
&      Tfstar, Vistar,
&      Ngf, Ylf, Ynlf, Betal)

C
  CALL GetDimensionlessGroups (
&      Tfstar, Vistar, Ngf, Ylf, Ynlf, Betal,
&      Tf, V,
&      ScriptA, ScriptB, ScriptQ, ScriptE,
&      ScriptSd, ScriptSv )

C
  QEbyA = ScriptQ * ScriptE / ScriptA

C
  CALL GetConvergedTemperature (QEbyA, ScriptE, Tfnew)

C
  IF (.not. lFixedScriptSf) THEN
    ScriptSf = ScriptSd + ScriptSv
  END IF

C
  CALL GetConvergedVelocity ( Tfnew, ScriptQ, ScriptSf, Vnew )

C
  VistarNew = Vnew * Vistar / V
  TfstarNew = Tfnew * Tfstar / Tf

C
  RETURN
  END

C
C
C
SUBROUTINE GetConvergedVelocity
&      ( Tfnew, ScriptQ, ScriptSf, Vnew )

C
  Input Variables:  See SUBROUTINE GetDimensionlessGroups
  Output Variables:
    Vnew           Converged value for dimensionless front velocity

C
  IMPLICIT REAL (a - h, o - z)
  REAL g, ScriptSf, Tfnew, Vnew
  REAL Deter, Denom

C
  g = 1.
  Deter = ScriptQ * ScriptQ
&      + 4. * (Tfnew - 1.) * ScriptSf / g

```

```

Denom = 2. * (Tfnew - 1.)
IF ((Deter .ge. 0.) .and. (Denom .gt. 0.0)) THEN
  Vnew = (ScriptQ + SQRT(Deter) ) / Denom - 1.
ELSE
  Vnew = 0.
END IF
IF (Vnew .lt. 0.) THEN
  Vnew = 0.0
END IF
RETURN
END

C
C
C
SUBROUTINE GetConvergedTemperature (QEbyA, ScriptE, Tfnew)
C
C   Input Variables:  See SUBROUTINE GetDimensionlessGroups
C   Output Variables:
C       Tfnew        Converged value for dimensionless front temp.
C
C   IMPLICIT REAL (a - h, o - z)
C   REAL x1, x2, x3, denom, g1, g2, Tol, Titer, T
C   INTEGER i
C
C   LOGICAL Converged
C
C   *** Statement Function Definitions
C
C   Titer(T) =  ScriptE / (2. * log(T) - log(QEbyA))
C
C
C   Tol = 1e-6
C
C   IF (QEbyA .le. 0.) THEN
C     Tfnew = 1.
C   ELSE
C ***   Calculate the eigenvalue in terms of the front temperature
C ***   Uses Wegstein Accelerator:
C
C     x1 = - ScriptE / log(0. * QEbyA)      ! Initial Guess
C     IF (x1 .le. 0) THEN
C       x1 = ScriptE
C     END IF
C     g1 = Titer(x1)
C     x2 = g1
C     Converged = .FALSE.
C     DO I = 1, 40
C       IF (x2 .lt. 1) THEN
C         x2 = 1.
C       END IF
C       g2 = Titer(x2)
C       denom = x1 - x2 - g1 + g2
C       IF (abs(g2 - x2) .lt. tol) THEN
C         Converged = .TRUE.
C         EXIT
C       ELSEIF (abs(x1 - x2) .lt. tol) THEN
C         x3 = (g1 + g2) * 0.5
C       ELSEIF (abs(denom) .lt. 0.01 * tol) THEN
C         x3 = (x2 + g1 + g2) * .3333333
C       ELSE
C         x3 = (x1 * g2 - x2 * g1) / denom
C       END IF
C       g1 = g2
C       x1 = x2
C       x2 = x3
C     END DO
C     IF (Converged) THEN
C       Tfnew = x2
    
```



```

      ELSE
        Tfinew = 1.
      END IF
C
      IF (Tfinew .le. 1.) THEN
        Tfinew = 1.
      END IF
C
      RETURN
    END

```

A.2.2 Include file LOCALCMN.FI

```

C
C   This include file defines the common blocks for the
C   Combust subroutine (In the file Combust.for).
      REAL      bdNga, bdLapNga, bdConvTerm, bdTinfstar,
&
&      bddTfStardNga,
&      bdPres, bdYOxa, bdYfa,
&      bdEbyR, bda
      REAL      Vm, RHOs, stoich, PHI, Tamb
C
      COMMON /Backdoor/ bdNga, bdLapNga, bdConvTerm, bdTinfstar,
&
&      bddTfStardNga,
&      bdPres, bdYOxa, bdYfa,
&      bdEbyR, bda
C      COMMON /CoalProp/ Vm, RHOs, stoich, PHI, Tamb

```

APPENDIX B

DATA ACQUISITION AND DISPLAY PROGRAMS

B.1 QuickBASIC Program DATA LOG

```

DECLARE SUB DLControl (SetPoint!, ec%)
DECLARE SUB DLPresControlSettings (ec%)
DECLARE SUB DLMsrSystemMenu ()
' $INCLUDE: 'NameCmn.bas'
' $INCLUDE: 'qb.bi'
DECLARE SUB LotusMenu (name$, short$(), long$(), items%, choice%, frgd%,
bkgrd%, case.sens%)
DECLARE SUB DLMainMenu (ExitCode%)
DECLARE SUB DLShutDown ()
DECLARE SUB DLStartUp ()
DECLARE SUB DLTable (ec%)
' Datalog - a program for logging data from the Labmaster Data
' Acquisition Board and displaying it.
CALL DLStartUp
CALL DLMainMenu(ExitCode%)
CALL DLShutDown
END
SUB DLMainMenu (ExitCode%)
' $DYNAMIC
DIM short$(5), long$(5)
' $STATIC
name$ = "Data Log - Main Menu"
short$(1) = "Units"
long$(1) = "Display in Metric or Standard units"
short$(2) = "Table"
long$(2) = "Work with tables of logged data"
short$(3) = "Control"
long$(3) = "Change the setpoint of the flowrate"
short$(4) = "Pressurize"
long$(4) = "Change settings for Pressurizing Routine"
short$(5) = "Quit"
long$(5) = "Exit Program"
items% = 5
sens% = 0
frgd% = 7
bkgrd% = 0
DO
CALL LotusMenu(name$, short$(), long$(), items%, choice%, frgd%,
bkgrd%, sens%)
SELECT CASE choice%
CASE 0                                'Escape key pressed
BEEP
CASE 1
CALL DLMsrSystemMenu
ec% = 0
CASE 2
CALL DLTable(ec%)
CASE 3
CALL DLControl(SetPoint!, ec%)
CASE 4
CALL DLPresControlSettings(ec%)

```

```

CASE 5
LOCATE 2, 4
PRINT SPACES$(50)
LOCATE 3, 4
PRINT SPACES$(50)
LOCATE 2, 4
PRINT "Are you sure? (y,n) "
DO
    akey$ = INKEY$
    LOOP WHILE LEN(akey$) = 0
    IF akey$ = "y" OR akey$ = "Y" THEN
        ec% = -1
    ELSE
        ec% = 0
    END IF
    CASE ELSE
        ec% = 0
    END SELECT
    LOOP UNTIL ec% = -1
    CLS
    ERASE short$, long$
    name$ = ""
    END SUB
    SUB DLShutDown
    CLS
    PRINT FRE(-1)
    DO
        akey$ = INKEY$
        LOOP WHILE akey$ = ""
    END SUB
DECLARE SUB DLBackgroundTask (TaskTimer!)
DECLARE SUB DLPresController (TaskTimer!)
DECLARE FUNCTION ElapsedTime! (TO!)
DECLARE SUB GetChan (Row%, Channel%, Gain%, Value%, ErrorCode%)
DECLARE SUB GetTCCh (Row%, Channel%, Gain%, Value%, ErrorCode%)
DECLARE FUNCTION EngUnitsTable% (ChannelCode%)
DECLARE FUNCTION GainValue! (Gain%)
DECLARE FUNCTION KTC! (v!)
DECLARE FUNCTION JTC! (v!)
DECLARE FUNCTION DTime$ (Row%)
DECLARE SUB DLGoTo (frow%, BegRow%, lastrow%, stepp%)
DECLARE SUB DLGetData (Row%, ChanCode%, DataCode%, RtnValue!, ErrorCode%)
DECLARE SUB DLReadColData (frow%, fcol%, stepp%, nmcol%, ColData%(), frgd%,
    bkgd%, crow%, ccol%, nmrow%, BegRow%, maxrow%, lastrow%)
DECLARE SUB DLWrite (BegRow%, FirstRow%, FirstCol%, ColData%(),
    stepp%, frgd%, bkgd%, akey$)
DECLARE SUB DLTCursor (cursorrow%, cursorcol%, fcolor%, bcolor%)
DECLARE SUB DLTKyHndlr (crow%, ccol%, frow%, fcol%, nmcol%, nmrow%, ec%,
    akey$)
DECLARE SUB DLTInsert (ColData%(), Col%, nmcol%)
DECLARE SUB DLDelete (ColData%(), Col%, nmcol%)
DECLARE SUB DLTEdit (ColData%(), Col%, nmcol%)
DECLARE SUB DLTNameUse (ec%, frow%, fcol%, stepp%, nmcol%, ColData%(),
    frgd%, bkgd%, crow%, ccol%, nmrow%, BegRow%, maxrow%,
    lastrow%) DECLARE SUB DLTNameSave (ec%, frow%, fcol%, stepp%, nmcol%,
    ColData%(), frgd%,
    bkgd%, crow%, ccol%, nmrow%, BegRow%, maxrow%, lastrow%)
DECLARE SUB DLTRow (BegRow%, stepp%, lastrow%, nmrow%, maxrow%)
DECLARE SUB DLWriteColData (frow%, fcol%, stepp%, nmcol%, ColData%(),
    frgd%, bkgd%, crow%, ccol%, nmrow%, BegRow%, maxrow%, lastrow%)
DECLARE SUB DLTEChannelMenu (ec1%, ChannelCode%, MenuName$)
DECLARE SUB DLTEUnitsMenu (ec%, DataCode%)
DECLARE SUB DLFormatMenu (ec%, formatcode%, places%, name$)
DECLARE SUB DLTEUEngUnitsMenu (ec%, DataCode%)
DECLARE SUB DLTAAdjust (stepp%, cursorr%, windoww%, widthhh%, maxx%,
    minn%, ec%)
DECLARE SUB DLTMenu (ec%)
DECLARE SUB DLTMCColumnMenu (ec%)

```

```

DECLARE SUB DLTMenuNameMenu (ec%)
DECLARE SUB DLTWGetFormat (Col%, format$, ColData%())
DECLARE SUB DLTWGetName (Col%, ColData%(), name$)
REM $INCLUDE: 'DataLogC.bi'
REM $INCLUDE: 'NameCmn.bas'
REM $INCLUDE: 'MsrUnCmn.bas'
COMMON SHARED /DLT/ DLTmenuflag%
REM $INCLUDE: 'Constant.bas'
REM $INCLUDE: 'ExpCmn.bi'
REM $INCLUDE: 'qb.bi'
CONST AbsTChNumber = 9
DIM SHARED long$(10), short$(10), Prompt$, name$, frgd%, bkgd%
DIM ColData%(4, 256)
SUB DLTTable (ec%)
  STATIC frow%, fcol%, stepp%, nmcol%, flag%, crow%, ccol%,
    nmrow%, BegRow%, maxrow%, lastrow%
  SHARED ColData%()
  frgd% = 7
  bkgd% = 0
  'Read in column data values if needed
  IF flag% = 0 THEN
    filename$ = "default.col"
    FileMode$ = "INPUT"
    FileNum% = 101
    OPEN filename$ FOR INPUT AS FileNum%
    ec% = 0
    'CALL SafeFileOpener(filename$, FileMode$, filenum%, ec%)
    IF ec% <> 0 THEN STOP
    CALL DLTReadColData(frow%, fcol%, stepp%, nmcol%, ColData%(),
      frgd%, bkgd%,
      crow%, ccol%, nmrow%, BegRow%,
      maxrow%, lastrow%)
    flag% = 1
  END IF
  'Draw screen for first time
  akey$ = ""
  CLS
  CALL DLTWrite(BegRow%, frow%, fcol%, ColData%(), stepp%, frgd%,
    bkgd%, akey$)
  CALL DLTCursor(crow%, ccol%, bkgd%, frgd%)
  DO
    oldcrow% = crow%
    oldccol% = ccol%
    CALL DLTKeyHndlr(crow%, ccol%, frow%, fcol%, nmcol%, nmrow%, ec%,
      akey$)
    SELECT CASE ec%
      CASE Ins, Del, F2, F3, F4, F5, F7
        'This section of code
        'bumps the screen
        'if needed for
        'for a menu
        IF DLTmenuflag% = FALSE THEN
          CALL Scroll(-4, 0, 0, 23, 79)
          DLTmenuflag% = TRUE
        END IF
        IF oldcrow% > 15 THEN
          CALL DLTCursor(oldcrow%, oldccol%, bkgd%, frgd%)
        END IF
      CASE ELSE
        END SELECT
      CASE 0
        'Do nothing
      CASE -1
        'Cursor moved
        CALL DLTCursor(oldcrow%, oldccol%, frgd%, bkgd%)
        CALL DLTCursor(crow%, ccol%, bkgd%, frgd%)
        ec% = 0
      CASE -2, F9
        'Page moved
        CALL DLTCursor(oldcrow%, oldccol%, frgd%, bkgd%)
        CALL DLTWrite(BegRow%, frow%, fcol%, ColData%(), stepp%, frgd%,
          bkgd%, akey$)
    
```

```

CALL DLTCursor(crow%, ccol%, bkgd%, frgd%)
ec% = 0
    CASE Ins
        'Insert pressed, >> insert column
        Col% = fcol% + ccol%
        CALL DLTInsert(ColData%(), Col%, nmcol%)
        CALL DLTCursor(oldcrow%, oldccol%, frgd%, bkgd%)
        CALL DLTWrite(BegRow%, frow%, fcol%, ColData%(), stepp%, frgd%,
            bkgd%, akey$)
        CALL DLTCursor(crow%, ccol%, bkgd%, frgd%)
        ec% = 0
    CASE Del
        'Delete pressed, >> delete column
        Col% = fcol% + ccol%
        CALL DLTDelete(ColData%(), Col%, nmcol%)
        CALL DLTCursor(oldcrow%, oldccol%, frgd%, bkgd%)
        CALL DLTWrite(BegRow%, frow%, fcol%, ColData%(), stepp%, frgd%,
            bkgd%, akey$)
        CALL DLTCursor(crow%, ccol%, bkgd%, frgd%)
        ec% = 0
    CASE F2
        'Edit key pressed, >> change
        column data
        Col% = fcol% + ccol%
        CALL DLTEdit(ColData%(), Col%, nmcol%)
        CALL DLTCursor(oldcrow%, oldccol%, frgd%, bkgd%)
        CALL DLTWrite(BegRow%, frow%, fcol%, ColData%(), stepp%, frgd%,
            bkgd%, akey$)
        CALL DLTCursor(crow%, ccol%, bkgd%, frgd%)
        ec% = 0
    CASE F3
        'Name Use Pressed'
        CALL DLTNameUse(ec%, frow%, fcol%, stepp%, nmcol%, ColData%(),
            frgd%, bkgd%,
crow%, ccol%, nmrow%, BegRow%, maxrow%, lastrow%)
        IF ec% = 27 THEN
            'do nothing here
        ELSEIF ec% <> 0 THEN
            BEEP
            BEEP
            BEEP
        END IF
        CALL DLTCursor(oldcrow%, oldccol%, frgd%, bkgd%)
        CALL DLTWrite(BegRow%, frow%, fcol%, ColData%(), stepp%, frgd%,
            bkgd%, akey$)
        CALL DLTCursor(crow%, ccol%, bkgd%, frgd%)
        ec% = 0
    CASE F4
        'Name Save Pressed
        CALL DLTNameSave(ec%, frow%, fcol%, stepp%, nmcol%, ColData%(),
            frgd%, bkgd%,
crow%, ccol%, nmrow%, BegRow%, maxrow%, lastrow%)
        IF ec% = 27 THEN
            'do nothing here
        ELSEIF ec% <> 0 THEN
            BEEP
            BEEP
            BEEP
        END IF
        CALL DLTCursor(oldcrow%, oldccol%, frgd%, bkgd%)
        CALL DLTWrite(BegRow%, frow%, fcol%, ColData%(), stepp%, frgd%,
            bkgd%, akey$)
        CALL DLTCursor(crow%, ccol%, bkgd%, frgd%)
        ec% = 0
    CASE F5
        CALL DLTGoTo(frow%, BegRow%, lastrow%, stepp%)
        CALL DLTCursor(oldcrow%, oldccol%, frgd%, bkgd%)
        CALL DLTWrite(BegRow%, frow%, fcol%, ColData%(), stepp%, frgd%,
            bkgd%, akey$)
        CALL DLTCursor(crow%, ccol%, bkgd%, frgd%)
        ec% = 0
    CASE F7
        CALL DLTRow(BegRow%, stepp%, lastrow%, nmrow%, maxrow%)

```

```

CALL DLTCursor(oldcrow%, oldccol%, frgd%, bkgd%)
CALL DLTWrite(BegRow%, frow%, fcol%, ColData%(), stepp%, frgd%,
bkgd%, akey$)
CALL DLTCursor(crow%, ccol%, bkgd%, frgd%)
ec% = 0
CASE ESC 'Escape pressed
'Do nothing, will exit with ec% = 27
CASE ELSE
ec% = 0
akey$ = ""
END SELECT
LOOP WHILE ec% = 0
END SUB
SUB DLTAdjust (stepp%, cursorr%, windoww%, widthh%, maxx%,
minn%, ec%)
oldec% = ec%
oldcursor% = cursorr%
oldwindow% = windoww%
cstep% = stepp% MOD (widthh% + 1)
windoww% = windoww% + stepp% - cstep% 'Get correct page
cursorr% = cursorr% + cstep% 'DLTAdjust cursor
remainder% = 0 'Force cursor to be
in page
IF cursorr% < 0 THEN
remainder% = cursorr%
cursorr% = 0
ELSEIF cursorr% > widthh% THEN
remainder% = cursorr% - widthh%
cursorr% = widthh%
END IF
windoww% = windoww% + remainder% 'Force window to
fit in 'range
IF windoww% < minn% THEN
windoww% = minn%
BEEP
ELSEIF windoww% > maxx% - widthh% THEN
windoww% = maxx% - widthh%
BEEP
END IF
ec% = 0 'Determine correct exit code%
IF oldcursor% <> cursorr% THEN ec% = -1
IF oldwindow% <> windoww% THEN ec% = -2
IF oldec% < ec% THEN ec% = oldec%
END SUB
SUB DLTCursor (cursorrow%, cursorcol%, fcolor%, bcolor%)
COLOR fcolor%, bcolor%
scol% = 16 + cursorcol% * 13
srow% = 5 + cursorrow%
LOCATE srow%, scol%, 0
FOR i% = 0 TO 11
a% = SCREEN(srow%, scol% + i%)
PRINT CHR$(a%);
NEXT
END SUB
SUB DLDelete (ColData%(), Col%, nmcol%)
CALL LotusMenuClear
name$ = "Data Log - Table - Delete"
short$(1) = "No"
long$(1) = "Do not delete column under cursor, return to viewing table"
short$(2) = "Yes"
long$(2) = "Delete column under cursor, then return to viewing table"
items% = 2
case.sens% = FALSE
CALL LotusMenu(name$, short$(), long$(), items%, choice%,
frgd%, bkgd%, case.sens%)
SELECT CASE choice%
CASE 0, 1
CASE 2

```

```

        'Shift Stack
        FOR i% = Col% TO nmcol% - 1
FOR j% = 0 TO 3
    ColData%(j%, i%) = ColData%(j%, i% + 1)
NEXT
    NEXT
    nmcol% = nmcol% - 1
    IF nmcol% < 5 THEN nmcol% = 5
CASE ELSE
    BEEP
END SELECT
CALL LotusMenuClear
END SUB
SUB DLTEChannelMenu (ec1%, ChannelCode%, MenuName$)
'$DYNAMIC
DIM dummylist$(16)
'$STATIC
CALL LotusMenuClear
name$ = MenuName$ + " - Channel"
short$(1) = "Regular"
long$(1) = "Select displayed channel from list of regular channels"
short$(2) = "TC Mux"
long$(2) = "Select displayed channel from list of TC Mux channels"
items% = 2
case.sens% = FALSE
DO
    CALL LotusMenu(name$, short$(), long$(), items%, choice%, frgd%,
bkgd%, case.sens%)
    SELECT CASE choice%
    CASE 0
        ec1% = ESC
        CASE 1
            ' Selected Regular Channels
            listcount% = 16
            menuname2$ = name$ + " - Regular"
            Prompt$ = "Please select channel:"
            FOR i% = 1 TO 16
                dummylist$(i%) = AtoDName$(i%)
            NEXT
            CALL LotusListSelector(Prompt$, dummylist$(), listcount%, menuname2$,
listptr%, ec2%)
            SELECT CASE ec2%
            CASE -1
                ChannelCode% = 1000 + listptr% - 1
                ec1% = TRUE
            CASE ESC
                ec1% = FALSE
            CASE ELSE
                ec1% = ec2%
            END SELECT
            CASE 2
                listcount% = 16
                menuname2$ = name$ + " - Thermocouple"
                Prompt$ = "Please select channel:"
                FOR i% = 1 TO 16
                    dummylist$(i%) = TCName$(i%)
                NEXT
                CALL LotusListSelector(Prompt$, dummylist$(), listcount%, menuname2$,
listptr%, ec2%)
                SELECT CASE ec2%
                CASE -1
                    ChannelCode% = 2000 + listptr% - 1
                    ec1% = TRUE
                CASE ESC
                    ec1% = FALSE
                CASE ELSE
                    ec1% = ec2%
                END SELECT
            CASE ELSE

```

```

BEEP
ec1% = FALSE
    END SELECT
    LOOP WHILE ec1% = FALSE
    ERASE dummylist$
    menuname2$ = ""
    CALL LotusMenuClear
    END SUB
    SUB DLTEdit (ColData%(), Col%, nmcol%)
    ChannelCode% = ColData%(0, Col%)
    DataCode% = ColData%(1, Col%)
    formatcode% = ColData%(2, Col%)
    places% = ColData%(3, Col%)
    DO
        name$ = "Data Log - Table - Column - Edit"
        short$(1) = "Channel"
        long$(1) = "Select channel type and number"
        short$(2) = "Units"
        long$(2) = "Choose units for display"
        short$(3) = "Format"
        long$(3) = "Integer, Fixed Format, or Scientific Notation"
        short$(4) = "Quit"
        long$(4) = "Return to viewing table"
        items% = 4
        case.sens% = FALSE
        CALL LotusMenu(name$, short$(), long$(), items%, choice%, frgd%,
bkgd%, case.sens%)
        SELECT CASE choice%
        CASE 0
        ec% = ESC
        CASE 1
        LotusMenuClear
        CALL DLTEChannelMenu(ec%, ChannelCode%, name$)
        LotusMenuClear
        IF ec% = TRUE THEN
            ColData%(0, Col%) = ChannelCode%
            ec% = 0
        END IF
        CASE 2
        LotusMenuClear
        CALL DLTEUnitsMenu(ec%, DataCode%)
        LotusMenuClear
        IF DataCode% = 5 THEN
            DataCode% = EngUnitsTable%(ColData%(0, Col%))
        END IF
        IF ec% = 0 THEN
            ColData%(1, Col%) = DataCode%
        END IF
        CASE 3
        LotusMenuClear
        CALL DLFormatMenu(ec%, formatcode%, places%, name$)
        LotusMenuClear
        IF ec% = 0 THEN
            ColData%(2, Col%) = formatcode%
            ColData%(3, Col%) = places%
        END IF
        CASE 4
        ec% = ESC
        CASE ELSE
        BEEP
        ec% = 0
        END SELECT
        LOOP WHILE ec% = 0
        LotusMenuClear
        END SUB
        SUB DLTEUnitsMenu (ec%, DataCode%)
        name$ = "Data Log - Table - Units"
        short$(1) = "Counts"

```



```

long$(1) = "Display 12 bit signal from A to D"
short$(2) = "Gain"
long$(2) = "Display gain"
short$(3) = "Scaled"
long$(3) = "Display counts*500/gain"
short$(4) = "Voltage"
long$(4) = "Display transducer voltage"
short$(5) = "Engineering"
long$(5) = "Display in appropriate engineering units for channel"
items% = 5
case.sens% = FALSE
DO
    CALL LotusMenu(name$, short$(), long$(), items%, choice%, frgd%,
bkgd%, case.sens%)
    SELECT CASE choice%
    CASE 0
        ec% = ESC
        CASE 1, 2, 3, 4, 5
            DataCode% = choice%
            ec% = 0
        CASE ELSE
            BEEP
            ec% = 1
    END SELECT
    LOOP WHILE ec% = 1
    CALL LotusMenuClear
END SUB
SUB DLTGoTo (frow%, BegRow%, lastrow%, stepp%)
    frgd% = 7
    bkgd% = 0
    name$ = "Data Log - Table - Go To"
    normal% = 7
    CALL LFMMakeBorder(name$, normal%)
    Prompt$ = "Go to row:"
    DO
        number! = 0
        LotusNumberPrompt Prompt$, number!, frgd%, bkgd%
        LOOP UNTIL number! >= BegRow% AND number! <= lastrow%
        frow% = (number! - BegRow%) stepp%
        LotusMenuClear
    END SUB
    SUB DLTInsert (ColData%(), Col%, nmc0l%) STATIC
        ' Minimal insert column routine
        ' Shift stack
        FOR i% = nmc0l% TO Col% + 1 STEP -1
            FOR j% = 0 TO 3
                ColData%(j%, i%) = ColData%(j%, i% - 1)
            NEXT
        NEXT
        nmc0l% = nmc0l% + 1
        CALL DLTEdit(ColData%(), Col%, nmc0l%)
    END SUB
    SUB DLTKeyHndlr (crow%, ccol%, frow%, fcol%, nmc0l%, nmrow%, ec%,
akey$) STATIC
        ' DataLog Table Key Handler
        IF LEN(akey$) = 0 THEN ec% = 0 ELSE ec% = -2
    DLTKEa:
        IF LEN(akey$) = 0 THEN
            ScreenUpdateTimer = TIMER
            BackgroundTaskTimer = TIMER
            DO
                BackgroundTaskTimer = TIMER
                CALL DLBBackgroundTask(BackgroundTaskTimer)
                akey$ = INKEY$
                LOOP UNTIL LEN(akey$) <> 0 OR ((ElapsedTime!(ScreenUpdateTimer) >
15!) AND (Mode$ = "TAKE"))
            END IF
            IF LEN(akey$) = 2 THEN

```

```

      'Handle extended keycodes'
      g% = ASC(MID$(akey$, 2)) * 100
    ELSEIF LEN(akey$) = 0 THEN
      g% = F9
    ELSE
      g% = ASC(akey$)
    END IF
    SELECT CASE g%
    CASE Up
      ' Cursor Up Key
      CALL DLTAdjust(-1, crow%, frow%, 19, nmrow%, 0, ec%)
    CASE Lft
      ' Cursor Left Key
      CALL DLTAdjust(-1, ccol%, fcol%, 4, nmcol% - 1, 0, ec%)
    CASE Rgt
      ' Cursor Right Key
      CALL DLTAdjust(1, ccol%, fcol%, 4, nmcol% - 1, 0, ec%)
    CASE Down
      ' Cursor Down Key
      CALL DLTAdjust(1, crow%, frow%, 19, nmrow%, 0, ec%)
    CASE PgDn
      ' Page Down Key
      CALL DLTAdjust(20, crow%, frow%, 19, nmrow%, 0, ec%)
    CASE PgUp
      ' Page Up Key
      CALL DLTAdjust(-20, crow%, frow%, 19, nmrow%, 0, ec%)
    CASE ShTab
      ' Shift Tab Key
      CALL DLTAdjust(-5, ccol%, fcol%, 4, nmcol% - 1, 0, ec%)
    CASE TabKey
      ' Tab Key Pressed
      CALL DLTAdjust(5, ccol%, fcol%, 4, nmcol% - 1, 0, ec%)
    CASE Home
      ccol% = 0
      fcol% = 0
      crow% = 0
      frow% = 0
      ec% = -2
    CASE EndKey
      ccol% = 4
      crow% = 19
      frow% = nmrow% - 19
      fcol% = nmcol% - 5
      ec% = -2
    CASE Slash
      ' Slash Key pressed - Bring up menu
      CALL DLTMenu(ec%)
    CASE ELSE
      ec% = g%
    END SELECT
    IF ec% <= 0 THEN
      akey$ = INKEY$
      IF LEN(akey$) > 0 THEN GOTO DLTKHa
    END IF
  END SUB
  SUB DLTMColumnMenu (ec%)
    ' This routine sets up and executes the menus for Data Log
    Table Column
    name$ = "Data Log - Table - Column "
    short$(1) = "Insert"
    long$(1) = "Insert a column before the cursor"
    short$(2) = "Delete"
    long$(2) = "Delete the column under the cursor"
    short$(3) = "Edit"
    long$(3) = "Edit the column under the cursor"
    items% = 3
    sens% = 0
    LotusMenuClear
    DO
      CALL LotusMenu(name$, short$(), long$(), items%, choice%, frgd%,
        bkgd%, sens%)
      ec% = -1
      SELECT CASE choice%
      CASE 0
        'Escape Key pressed - exit menu,
        ' return to previous level
      CASE 1
        'Insert
      ec% = Ins

```

```

CASE 2                                'Delete
ec% = Del
CASE 3                                'Edit
ec% = F2
CASE ELSE                             'Intercept function keys, throw in
ec% = -1                             ' bit bucket for now.
END SELECT
LOOP WHILE ec% = -1
LotusMenuClear
END SUB
SUB DLTMenu (ec%)
' This routine sets up and executes the menus for Data Log Table
IF DLTMenuflag% = FALSE THEN
CALL Scroll(-4, 0, 0, 0, 23, 79)
END IF
DLTMenuflag% = TRUE
name$ = "Data Log - Table"
short$(1) = "Column"
long$(1) = "Insert, delete, or edit a column"
short$(2) = "Row"
long$(2) = "Change beginning row, ending rows, or the step between
rows"
short$(3) = "Name"
long$(3) = "Save or Use a named table configuration"
short$(4) = "Print"
long$(4) = "Output a table to printer or a file"
short$(5) = "View"
long$(5) = "View the the current table"
short$(6) = "Quit"
long$(6) = "Return to previous menu"
items% = 6
sens% = 0
CALL LotusMenuClear
DO
CALL LotusMenu(name$, short$(), long$(), items%, choice%, frgd%,
bkgd%, sens%)
SELECT CASE choice%
CASE 0                                'Escape Key pressed - exit menu
ec% = -2                              'update screen
CASE 1                                'Column - Insert, delete or edit?
CALL DLTMColumnMenu(ec%)
CASE 2                                'Row
ec% = F7
CASE 3                                'Name - Save or use?
CALL DLTMNameMenu(ec%)
CASE 4                                'Print pressed
ec% = F8
CASE 5                                'View
ec% = -2
CASE 6                                'Quit
ec% = ESC
CASE ELSE
ec% = 0
END SELECT
LOOP WHILE ec% = 0
LotusMenuClear
END SUB
SUB DLTMNameMenu (ec%)
' This routine sets up and executes the menus for Data Log Table
Column
name$ = "Data Log - Table - Name "
short$(1) = "Use"
long$(1) = "Use a named table arrangement"
short$(2) = "Save"
long$(2) = "Save the current table arrangement"
items% = 2
sens% = 0
CALL LotusMenuClear

```

```

DO
  CALL LotusMenu(name$, short$(), long$(), items%, choice%, frgd%,
bkgd%, sens%)
  ec% = -1
  SELECT CASE choice%
    CASE 0
      'Escape Key pressed - exit menu,
      ' return to previous level
    CASE 1
      'Use
    CASE 2
      'Save
    CASE 4
      'Intercept function keys, throw in
    CASE ELSE
      ' bit buckets for now.
  END SELECT
  LOOP WHILE ec% = -1
  END SUB
  SUB DLTNameSave (ec%, frow%, fcol%, stepp%, nmcol%, ColData%(),
frgd%, bkgd%,
crow%, ccol%, nmrow%, BegRow%, maxrow%, lastrow%) STATIC
  'This routine Saves the channel names in a file
  CALL GetDefDrv(drive$)
  CALL GetCurDir(drive$, pathname$)
  FileSpec$ = "*.col"
  FileMode$ = "OUTPUT"
  FileNum% = 101
  MenuName$ = "Data Log - Table - Save"
  Prompt$ = "Enter file name for saving table configuration:"
  CALL LotusFileGetter(pathname$, FileSpec$, MenuName$, Prompt$,
FileMode$, FileNum%, ec%)
  IF ec% = 0 THEN
    'File sucessfully opened, no special
    keys pressed
    CALL DLTWriteColData(frow%, fcol%, stepp%, nmcol%, ColData%(),
frgd%, bkgd%,
crow%, ccol%, nmrow%, BegRow%, maxrow%, lastrow%)
  END IF
  END SUB
  SUB DLTNameUse (ec%, frow%, fcol%, stepp%, nmcol%, ColData%(),
frgd%, bkgd%,
crow%, ccol%, nmrow%, BegRow%, maxrow%, lastrow%) STATIC
  'This routine Saves the channel names in a file
  CALL GetDefDrv(drive$)
  CALL GetCurDir(drive$, pathname$)
  FileSpec$ = "*.col"
  FileMode$ = "INPUT"
  FileNum% = 101
  MenuName$ = "Data Log - Table - Use"
  Prompt$ = "Enter file name for table configuration to use:"
  CALL LotusFileGetter(pathname$, FileSpec$, MenuName$, Prompt$,
FileMode$, FileNum%, ec%)
  IF ec% = 0 THEN
    'File sucessfully opened, no special
    keys pressed
    CALL DLTReadColData(frow%, fcol%, stepp%, nmcol%, ColData%(),
frgd%, bkgd%,
crow%, ccol%, nmrow%, BegRow%, maxrow%, lastrow%)
  END IF
  END SUB
  SUB DLTReadColData (frow%, fcol%, stepp%, nmcol%, ColData%(), frgd%,
bkgd%,
crow%, ccol%, nmrow%, BegRow%, maxrow%, lastrow%)
  ' This routine reads in a configuration of a DataLog Table
  ,
  , Output
  ,   frow%           First row to be displayed ( a counter in
  ,                   the data storage arrays
  ,   fcol%           First column to be displayed ( in the list
  ,                   stored in coldata%()
  ,   stepp%          Between rows displayed
  ,   nmcol%          Number of columns displayed

```

```

      coldata%(i,column_number )
      i = 0:      channel code
      i = 1:      data code
      i = 2:      format code
      i = 3:      number of digits for format code
INPUT #101, frow%, fcol%, stepp%
INPUT #101, crow%, ccol%
INPUT #101, nmrow%, BegRow%, maxrow%, lastrow%
INPUT #101, frgd%, bkgd%
INPUT #101, nmcol%
FOR i% = 0 TO nmcol% - 1
  FOR j% = 0 TO 3
INPUT #101, ColData%(j%, i%)
  NEA.
  NEXT
CLOSE #101
END SUB
SUB DLTRow (BegRow%, stepp%, lastrow%, nmrow%, maxrow%) STATIC
beg% = BegRow%
delta% = stepp%
last% = lastrow%
name$ = "Data Log - Table - Row"
short$(1) = "Start"
long$(1) = "Select row to start table"
short$(2) = "Interval"
long$(2) = "Select interval between rows"
short$(3) = "End"
long$(3) = "Select last row in table"
short$(4) = "Go"
long$(4) = "DLTAdjust values, return to viewing table"
short$(5) = "Quit"
long$(5) = "Return to viewing table"
items% = 4
case.sens% = FALSE
DO
  CALL LotusMenu(name$, short$(), long$(), items%, choice%,
frgd%, bkgd%, case.sens%)
  SELECT CASE choice%
    CASE 0
      ec% = ESC
      CASE 1
        number! = beg%
        Prompt$ = "Start at row:"
        DO
          number! = FIX(number!)
          CALL LotusNumberPrompt(Prompt$, number!, frgd%, bkgd%)
          LOOP UNTIL number! >= 0 AND number! < maxrow%
          beg% = number!
          CALL LotusMenuClear
          ec% = 0
          CASE 2
            number! = delta%
            Prompt$ = "Interval between rows:"
            DO
              number! = FIX(number!)
              CALL LotusNumberPrompt(Prompt$, number!, frgd%, bkgd%)
              LOOP UNTIL number! >= 0 AND number! < maxrow% 20 + 1
              delta% = number!
              CALL LotusMenuClear
              ec% = 0
              CASE 3
                number! = last%
                Prompt$ = "End at row:"
                DO
                  number! = FIX(number!)
                  CALL LotusNumberPrompt(Prompt$, number!, frgd%, bkgd%)
                  LOOP UNTIL number! >= 0 AND number! < maxrow%

```

```

last% = number!
CALL LotusMenuClear
ec% = 0
CASE 4
BegRow% = beg%
stepp% = delta%
nmrow% = ((last% - beg%) delta%)
lastrow% = last%
ec% = -1
CASE 5
ec% = ESC
END SELECT
LOOP WHILE ec% = 0
CALL LotusMenuClear
END SUB
SUB DLTWGetFormat (Col%, format$, ColData%()) STATIC
formatcode% = ColData%(2, Col%)
digits% = ColData%(3, Col%)
SELECT CASE formatcode%
CASE 1
format$ = " ##### "
CASE 2
IF digits% < 0 THEN digits% = 0
IF digits% > 8 THEN digits% = 8
format$ = " " + STRING$(10 - digits%, "#") + "." + STRING$(dig
its%, "#") + " "
CASE 3
IF digits% < 0 THEN digits% = 0
IF digits% > 4 THEN digits% = 4
format$ = " ##." + STRING$(digits%, "#") + "---- "
a$ = SPACE$(13)
RSET a$ = format$
SWAP format$, a$
CASE ELSE
format$ = " ##### "
END SELECT
END SUB
SUB DLTWGetName (Col%, ColData%(), name$) STATIC
ChanCode% = ColData%(0, Col%)
ErrorCode% = 0
SELECT CASE ChanCode%
CASE 1000 TO 1015
Channel% = ChanCode% - 1000
name$ = AtoDName$(Channel% + 1)
CASE 2000 TO 2015
Channel% = ChanCode% - 2000
name$ = TCName$(Channel% + 1)
CASE ELSE
ErrorCode% = 2
END SELECT
a$ = SPACE$(13)
RSET a$ = name$
name$ = a$
END SUB
SUB DLTWrite (BegRow%, FirstRow%, FirstCol%, ColData%(),
stepp%, frgd%, bkgd%, akey$) STATIC
DIM format$(4)
DLTmenuflag% = FALSE
'Write Headers
LOCATE 1, 1, 0
PRINT Heading1$;
LOCATE 1, 70, 0
IF Mode$ = "TAKE" THEN PRINT DATE$; ELSE PRINT ExpDate$;
LOCATE 2, 1, 0
PRINT Heading2$;

```

```

LOCATE 2, 70, 0
PRINT Mode$;
LOCATE 3, 1, 0
head$ = " Row Time "
PRINT head$;
IF MsrSystem$ = "METRIC" THEN
    unitflag% = 2
ELSE
    unitflag% = 1
END IF
FOR i% = 0 TO 4
    Col% = i% + FirstCol%
    LOCATE 3, 15 + 13 * i%, 0
    CALL DLTWGetName(Col%, ColData%(), colname$)
    CALL QPrint(colname$, 15)
    LOCATE 4, 18 + 13 * i%, 0
    DataCode% = ColData%(1, Col%)
    CALL QPrint(MsrUnit$(DataCode%, unitflag%), 15)
    CALL DLTWGetFormat(Col%, format$(i%), ColData%())
NEXT
COLOR frgd%, bkgd%
FOR i% = 0 TO 19
    'Write entry number and time
    LOCATE 5 + i%, 1, 0
    Row% = BegRow% + (FirstRow% + i%) * stepp%
    PRINT USING "#### " Row%, DTime$(Row%);
    FOR j% = 0 TO 4
        LOCATE 5 + i%, 15 + j% * 13
        Col% = j% + FirstCol%
        ChanCode% = ColData%(0, Col%)
        DataCode% = ColData%(1, Col%)
        DLGetData Row%, ChanCode%, DataCode%, RtnValue, ErrorCode%
        IF ErrorCode% = 0 THEN
            PRINT USING format$(j%); RtnValue;
        ELSE
            PRINT USING " Error:#### " ErrorCode%;
        END IF
    NEXT
    BackgroundTaskTimer = TIMER
    CALL DLBackgroundTask(BackgroundTaskTimer)
    akey$ = INKEY$
    IF LEN(akey$) <> 0 THEN EXIT FOR
NEXT
END SUB
SUB DLTWriteColData (frow%, fcol%, stepp%, nmcol%, ColData%(),
    frgd%, bkgd%,
    crow%, ccol%, nmrow%, BegRow%, maxrow%, lastrow%) STATIC
    ' This routine writes a configuration of a DataLog Table
    ' See DLTReadColData for variables
    PRINT #101, frow%, fcol%, stepp%
    PRINT #101, crow%, ccol%
    PRINT #101, nmrow%, BegRow%, maxrow%, lastrow%
    PRINT #101, frgd%, bkgd%
    PRINT #101, nmcol%
    FOR i% = 0 TO nmcol% - 1
        FOR j% = 0 TO 3
            PRINT #101, ColData%(j%, i%); " ";
        NEXT
        PRINT #101, " "
    NEXT
    CLOSE #101
END SUB
REM $INCLUDE: 'NameCmn.bas'
REM $INCLUDE: 'Page1Cmn.bas'
DATA 4,30,CHANNEL NAMES
DATA 6,1,Analog to Digital
DATA 7,1,

```

DATA 6,28,Digital Input/Output
 DATA 7,28,""
 DATA 6,55,Digital to Analog
 DATA 7,55,""
 DATA 8,1," 1 1 1"
 DATA 9,1," 2 2 2"
 DATA 10,1," 3 3"
 DATA 11,1," 4 4"
 DATA 12,1," 5 5"
 DATA 13,1," 6 6"
 DATA 14,1," 7 7"
 DATA 15,1," 8 8"
 DATA 16,1," 9 9 17"
 DATA 17,1,"10 10 18"
 DATA 18,1,"11 11 19"
 DATA 19,1,"12 12 20"
 DATA 20,1,"13 13 21"
 DATA 21,1,"14 14 22"
 DATA 22,1,"15 15 23"
 DATA 23,1,"16 16 24"
 DATA 8, 4
 DATA 9, 4
 DATA 10, 4
 DATA 11, 4
 DATA 12, 4
 DATA 13, 4
 DATA 14, 4
 DATA 15, 4
 DATA 16, 4
 DATA 17, 4
 DATA 18, 4
 DATA 19, 4
 DATA 20, 4
 DATA 21, 4
 DATA 22, 4
 DATA 23, 4
 DATA 8,31
 DATA 9,31
 DATA 10,31
 DATA 11,31
 DATA 12,31
 DATA 13,31
 DATA 14,31
 DATA 15,31
 DATA 16,31
 DATA 17,31
 DATA 18,31
 DATA 19,31
 DATA 20,31
 DATA 21,31
 DATA 22,31
 DATA 23,31
 DATA 16, 58
 DATA 17, 58
 DATA 18, 58
 DATA 19, 58
 DATA 20, 58
 DATA 21, 58
 DATA 22, 58
 DATA 23, 58
 DATA 8, 58
 DATA 9, 58
 DATA 40, 2, 41, 17, 1, 16
 DATA 40, 3, 42, 18, 1, 1
 DATA 40, 4, 19, 19, 1, 2
 DATA 40, 5, 20, 20, 1, 3
 DATA 40, 6, 21, 21, 1, 4
 DATA 40, 7, 22, 22, 1, 5


```

DATA 40, 8, 23, 23, 1, 6
DATA 40, 9, 24, 24, 1, 7
DATA 40, 10, 33, 25, 1, 8
DATA 40, 11, 34, 26, 1, 9
DATA 40, 12, 35, 27, 1, 10
DATA 40, 13, 36, 28, 1, 11
DATA 40, 14, 37, 29, 1, 12
DATA 40, 15, 38, 30, 1, 13
DATA 40, 16, 39, 31, 1, 14
DATA 40, 1, 40, 32, 1, 15
DATA 40, 18, 1, 41, 1, 32
DATA 40, 19, 2, 42, 1, 17
DATA 40, 20, 3, 3, 1, 18
DATA 40, 21, 4, 4, 1, 19
DATA 40, 22, 5, 5, 1, 20
DATA 40, 23, 6, 6, 1, 21
DATA 40, 24, 7, 7, 1, 22
DATA 40, 25, 8, 8, 1, 23
DATA 40, 26, 9, 33, 1, 24
DATA 40, 27, 10, 34, 1, 25
DATA 40, 28, 11, 35, 1, 26
DATA 40, 29, 12, 36, 1, 27
DATA 40, 30, 13, 37, 1, 28
DATA 40, 31, 14, 38, 1, 29
DATA 40, 32, 15, 39, 1, 30
DATA 40, 17, 16, 40, 1, 31
DATA 40, 34, 25, 9, 1, 42
DATA 40, 35, 26, 10, 1, 33
DATA 40, 36, 27, 11, 1, 34
DATA 40, 37, 28, 12, 1, 35
DATA 40, 38, 29, 13, 1, 36
DATA 40, 39, 30, 14, 1, 37
DATA 40, 40, 31, 15, 1, 38
DATA 40, 41, 32, 16, 1, 39
DATA 40, 42, 17, 1, 1, 40
DATA 40, 33, 18, 2, 1, 41
SUB DLStartUp
FOR i% = 1 TO 42 'Define format strings
fcpage1$(i%) = " "
NEXT
FOR i% = 1 TO 23
READ ispage1%(i%, 1), ispage1%(i%, 2), tspage1$(i%)
NEXT
'Define locations of editable fields
FOR i% = 1 TO 42
READ icpage1%(i%, 1), icpage1%(i%, 2)
NEXT
'Define interconnectivity
FOR i% = 1 TO 42
FOR j% = 1 TO 6
READ ccpage1%(i%, j%)
NEXT
NEXT
END SUB
SUB DLTEUEngUnitsMenu (ec%, datacode%)
DIM short$(10), long$(10)
name$ = "Data Log - Table - Units - Engineering"
short$(1) = "J TC"
long$(1) = "J type thermocouple with cold junction compensation"
short$(2) = "K TC"
long$(2) = "K type thermocouple with cold junction compensation"
short$(3) = "Abs Temp"
long$(3) = "Absolute temperature transducer"
short$(4) = "PT 0-1000"
long$(4) = "Pressure transducer, range 0-1000psig == 0-10v"
short$(5) = "Flow 0-10"
long$(5) = "Mass Flow Meter, range 0 - 10 slpm == 0-5v"
short$(6) = "Flow 0-5"

```

```

long$(6) = "Mass Flow Controller; range 0 - 5 slpm == 0-5v"
items% = 6
case.sens% = FALSE
frgd% = 7
bkgd% = 0
DO
CALL LotusMenu(name$, short$(), long$(), items%, choice%,_
frgd%, bkgd%, case.sens%)
SELECT CASE choice%
CASE 0
ec% = ESC
CASE 1, 2, 3, 4, 5, 6
datacode% = 4 + choice%
ec% = 0
CASE ELSE
BEEP
ec% = 1
END SELECT
LOOP WHILE ec% = 1
CALL LotusMenuClear
END SUB
DECLARE SUB DLGetData (Row%, ChanCode%, DataCode%, RtnValue!, ErrorCode%)
DECLARE FUNCTION ElapsedTime! (TO!)
DECLARE FUNCTION KTC! (v!)
DECLARE FUNCTION GainValue! (Gain%)
DECLARE FUNCTION JTC! (v!)
DECLARE FUNCTION EngUnitsTable% (ChannelCode%)
DECLARE SUB GetTCCh (Row%, Channel%, Gain%, Value%, ErrorCode%)
DECLARE SUB GetChan (Row%, Channel%, Gain%, Value%, ErrorCode%)
DECLARE SUB DLPresController (TaskTimer!)
DECLARE SUB DLStGetBaseName (BaseName$, ExpInfo AS ANY, ExpHeader() AS ANY)
REM $INCLUDE: 'NameCmn.bas'
REM $INCLUDE: 'MsrUnCmn.bas'
REM $INCLUDE: 'qb.bi'
REM $INCLUDE: 'datalogc.bi'
REM $INCLUDE: 'ExpCmn.bi'
DECLARE FUNCTION DTime$ (a%)
DECLARE SUB DLGetBaseName (BaseName$, ExpInfo AS ANY, ExpHeader() AS ANY)
DIM PCPropBand!, PCTauIntegral!, PCdifSetPoint!, PCErrorSum!, PresControl
Flag%, OldTaskTimer!
CONST AbsTChNumber = 9
DATA " Counts ", " Counts "
DATA " Gain ", " Gain "
DATA " S Counts ", " S Counts "
DATA " Volts ", " Volts "
DATA " F ", " C "
DATA " F ", " C "
DATA " R ", " K "
DATA " Psig ", " KPa "
DATA " Scfh ", " Slpm "
DATA " Scfh ", " Slpm "
DATA " F ", " C "
DATA " F ", " C "
DATA " Scfh ", " Slpm "
DIM SHARED long$(10), short$(10), Prompt$, name$, frgd%, bkgd%, items%
DIM SHARED choice%, case.sens%, sens%
SUB DLBackgroundTask (TaskTimer)
' Currently the only possible background task is the pressurization
' controller (PI Master Controller with Anti-Reset Windup)
SHARED PresControlFlag%
IF PresControlFlag% = -1 THEN
CALL DLPresController(TaskTimer)
END IF
END SUB
SUB DLControl (SetPoint!, ec%)
Prompt$ = "Enter new flow rate for oxidant gas (range 0 - 5 slpm):"
NewSetPoint! = SetPoint!
DO

```

```

CALL LotusNumberPrompt(Prompt$, NewSetPoint!, frgd%, bkgd%)
LOOP UNTIL NewSetPoint! >= 0 AND NewSetPoint! < 5.01
CALL LotusMenuClear
SetPoint! = NewSetPoint!
' This is code to program the labmaster D to A Channel 1 (see p.65 of Manual)
BaseAdr% = 1808
decimal! = (SetPoint! * 819) - 2048
decimal! = INT(decimal!)
high% = INT(decimal! / 256)
low% = decimal! - 256 * high%
OUT BaseAdr% + 1, high%
OUT BaseAdr%, low%
ec% = 0
END SUB

SUB D:\FormatMenu (ec%, formatcode%, places%, name$)
localname$ = name$ + " - Format"
short$(1) = "Integer"
long$(1) = "Integer format, no decimal point"
short$(2) = "Fixed"
long$(2) = "Fixed format, with decimal point"
short$(3) = "Scientific"
long$(3) = "Scientific Notation"
items% = 3
case.sens% = FALSE
CALL LotusMenu(localname$, short$(), long$(), items%, choice%,
frgd%, bkgd%, case.sens%)
SELECT CASE choice%
CASE 0
ec% = ESC
CASE 1
formatcode% = 1
ec% = 0
CASE 2, 3
formatcode% = choice%
number! = places%
Prompt$ = "Number of places after decimal point:"
DO
number! = FIX(number!)
CALL LotusNumberPrompt(Prompt$, number!, frgd%, bkgd%)
LOOP UNTIL number! >= 0 AND number! < 10
places% = number!
CALL LotusMenuClear
ec% = 0
CASE ELSE
BEEP
ec% = ESC
END SELECT
CALL LotusMenuClear
localname$ = ""
END SUB

SUB DLGetData (Row%, ChanCode%, DataCode%, RtnValue, ErrorCode%)
' This routine retrieves data from the data base (in raw form) and
' converts it to the desired units selected with DataCode%
ErrorCode% = 0
SELECT CASE ChanCode%
CASE 1000 TO 1015
Channel% = ChanCode% - 1000
CALL GetChan(Row%, Channel%, Gain%, Value%, ErrorCode%)
CASE 2000 TO 2015
Channel% = ChanCode% - 2000
CALL GetTCCh(Row%, Channel%, Gain%, Value%, ErrorCode%)
CASE ELSE
ErrorCode% = 2
END SELECT
Value = Value%
to real
IF ErrorCode% <> 0 THEN EXIT SUB
Gain = GainValue(Gain%)

```

```

IF Gain < 0 THEN ErrorCode% = 3
IF ErrorCode% <> 0 THEN EXIT SUB
SELECT CASE DataCode%
'procedure
CASE 1
    RtnValue = Value
    'Counts
CASE 2
    RtnValue = Gain
    'Gain
CASE 3
    RtnValue = Value * (500 / Gain)
    'Scaled counts
CASE 4
    RtnValue = Value / (Gain * 409.5)
    'Volts
CASE 5, 11
    RawVolt = Value / (Gain * 409.5)
    'J TC, super and sub
    GetChan Row%, (AbsTChNumber), Gain%, Value%, errcode%
    'ambient respectively
    IF errcode% <> 0 THEN
        ErrorCode% = errcode%
EXIT SUB
    END IF
    Value = Value%
    Gain = GainValue(Gain%)
    IF Gain < 0 THEN
        ErrorCode% = 7
EXIT SUB
    END IF
    RefTemp = Value / (Gain * 8.19) - 273.16
    RefVolt = (1.019 + ((RefTemp - 20!) / 5) * (1.277 - 1.019)) / 1000!
    IF DataCode% = 5 THEN
        CorVolt = RawVolt + RefVolt
    ELSE
        CorVolt = -RawVolt + RefVolt
    END IF
    TempC = JTC(CorVolt)
    IF MsrSystem$ = "METRIC" THEN
        RtnValue = TempC
    ELSE
        RtnValue = (TempC * 1.8) + 32
    END IF
CASE 6, 12
    RawVolt = Value / (Gain * 409.5)
    'K TC super and sub
    GetChan Row%, (AbsTChNumber), Gain%, Value%, errcode%
    'ambient respectively
    IF errcode% <> 0 THEN
        ErrorCode% = errcode%
EXIT SUB
    END IF
    Value = Value%
    Gain = GainValue(Gain%)
    IF Gain < 0 THEN
        ErrorCode% = 7
EXIT SUB
    END IF
    RefTemp = Value / (Gain * 8.19) - 273.16
    RefVolt = (.798 + ((RefTemp - 20!) / 5) * (1! - .798)) / 1000!
    IF DataCode% = 6 THEN
        CorVolt = RawVolt + RefVolt
    ELSE
        CorVolt = -RawVolt + RefVolt
    END IF
    TempC = KTC(CorVolt)
    IF MsrSystem$ = "METRIC" THEN
        RtnValue = TempC
    ELSE
        RtnValue = (TempC * 1.8) + 32
    END IF
CASE 7
    IF MsrSystem$ = "METRIC" THEN
        RtnValue = Value / (Gain * 8.19)
    ELSE
        RtnValue = Value * (500 / Gain)
    END IF
    'Return with degrees C
    'Return with degrees F
    'Degrees K with gain of 2

```

```

RtnValue = 1.8 * Value / (Gain * 8.19) 'Degrees R with gain of 2
END IF
CASE 8 'Pressure transducer
  IF MsrSystem$ = "METRIC" THEN
    RtnValue = ((Value / (Gain * 4.095)) + 14.7) * 6.89476 'KPa
  ELSE
    RtnValue = Value / (Gain * 4.095) '0 -1000 Psig
  END IF
CASE 9 'Flow Controller
  IF MsrSystem$ = "METRIC" THEN
    RtnValue = Value / (Gain * 409.5) '0-5 slpm
  ELSE
    RtnValue = (Value / (Gain * 409.5)) * .03532 * 60! ' scfh
  END IF
CASE 10 'Flow Meter linearized
  IF MsrSystem$ = "METRIC" THEN
    RtnValue = Value / (Gain * 204.75) - .2385 '0-10 slpm
  ELSE
    RtnValue = (Value / (Gain * 204.75) - .2385) * .03532 * 60! ' scfh
  END IF
CASE 13 'Flow Meter nonlinearized output
  Vlt = Value / (Gain * 409.5)
  DIM RCf(4)
  RCf(1) = .064229
  RCf(2) = .255158
  RCf(3) = -.00356
  RCf(4) = .005996
  SLPM = Vlt * (RCf(1) + Vlt * (RCf(2) + Vlt * (RCf(3) + Vlt *
RCf(4))))
  IF MsrSystem$ = "METRIC" THEN
    RtnValue = SLPM
  ELSE
    RtnValue = SLPM * .03532 * 60! 'scfh
  END IF
CASE ELSE
  ErrorCode% = 4
END SELECT
END SUB

SUB DLMsrSystemMenu
CALL LotusMenuClear
name$ = "Data Log - Measurement System"
short$(1) = "Metric"
long$(1) = "Display in Metric units"
short$(2) = "Standard"
long$(2) = "Display in Standard units"
items% = 2
sens% = 0
CALL LotusMenu(name$, short$(), long$(), items%, choice%, frgd%, bkgd%,
sens%)
SELECT CASE choice%
CASE 0 'Escape key pressed
CASE 1
  MsrSystem$ = "METRIC"
CASE 2
  MsrSystem$ = "STANDARD"
CASE ELSE
END SELECT
CLS
ERASE short$, long$
name$ = ""
END SUB

SUB DLPresController (TaskTimer!)
' This routine implements a PI Master Controller
' to control the flow into the sample container to allow pressurization
' at the maximum possible rate while holding the differential pressure
' at the control point.
SHARED PCPropBand!, PCTauIntegral!, PCDiffSetPoint!, PCErrSum!,
OldTaskTimer!

```

```

DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
'Get Current Counts
InRegs.ax = 1
CALL INTERRUPTX(&HA1, InRegs, OutRegs)
IF OutRegs.cx >= 0 THEN
    Count% = OutRegs.cx
ELSE
    Count% = 65536 + OutRegs.cx
END IF
Count% = OutRegs.dx * 65536 + Count%
Row% = Count% / 256
'Retrieve Data
CurMsrSystem$ = MsrSystem$
MsrSystem$ = "METRIC"
CALL DLGetData(Row%, 1000, 8, PresSample!, ErrorCode%)
IF ErrorCode% <> 0 THEN
    CALL DLGetData(Row% - 1, 1000, 8, PresSample!, ErrorCode%)
END IF
IF ErrorCode% <> 0 THEN
    MsrSystem$ = CurMsrSystem$
    EXIT SUB
END IF
CALL DLGetData(Row%, 1001, 8, PresShell!, ErrorCode%)
IF ErrorCode% <> 0 THEN
    CALL DLGetData(Row% - 1, 1001, 8, PresShell!, ErrorCode%)
END IF
MsrSystem$ = CurMsrSystem$
IF ErrorCode% <> 0 THEN EXIT SUB
'Process Data
PCErrorSignal! = (PresShell! - PresSample!) - PCDiffSetPoint!
PCErrorSum! = PCErrorSum! + PCErrorSignal! * (TaskTimer - OldTaskTimer)
IF ABS(PCErrorSum!) > ABS(PCPropBand * PCTauIntegral) THEN
    PCErrorSum! = SGN(PCErrorSum!) * ABS(PCPropBand * PCTauIntegral)
END IF
PCFlowSetPoint! = 5! * (PCErrorSignal! + PCErrorSum! / PCTauIntegral!) /
PCPropBand!
IF PCFlowSetPoint! < 0! THEN
    PCFlowSetPoint! = 0!
    PCErrorSum! = ErrorSum! - PCErrorSignal! * (TaskTimer - OldTaskTimer)
ELSEIF PCFlowSetPoint! > 5! THEN
    PCFlowSetPoint! = 5!
    PCErrorSum! = PCErrorSum! - PCErrorSignal! * (TaskTimer! - OldTaskTimer!)
END IF
'Set Flow Control Set Point
'This is code to program the labmaster D to A Channel 1 (see p.65 of Manual)
BaseAdr% = 1808
decimal! = (PCFlowSetPoint! * 819) - 2048
decimal! = INT(decimal!)
high% = INT(decimal! / 256)
low% = decimal! - 256 * high%
OUT BaseAdr% + 1, high%
OUT BaseAdr%, low%
OldTaskTimer! = TaskTimer!
END SUB
SUB DLPresControlSettings (ec%)
,
SHARED PCPropBand!, PCTauIntegral!, PCDiffSetPoint!, PCErrorSum!,
PresControlFlag%, OldTaskTimer!
CALL LotusMenuClear
name$ = "Data Log - Pressurization Control Settings"
short$(1) = "Start"
long$(1) = "Start Pressurization Routine"
short$(2) = "Stop"
long$(2) = "Stop Pressurization Routine"
short$(3) = "SetPoint"
long$(3) = "Change Setpoint for Differential Pressure between Shell and
Sample, kPA"
short$(4) = "Prop. Band"

```

```

long$(4) = "Change value of Proportional Band, kPa"
short$(5) = "Reset Time"
long$(5) = "Change value of Reset Time, Seconds"
short$(6) = "Quit"
long$(6) = "Return to Previous Menu"
items% = 8
sens% = 0
exitcode% = 0
DO
  CALL LotusMenu(name$, short$(), long$(), items%, choice%, frgd%, bkgd%,
sens%)
  SELECT CASE choice%
    CASE 0                                'Escape key pressed
    CASE 1                                'Start
      PCErrSum! = 0
      PresControlFlag% = -1
    CASE 2                                'Stop
      PresControlFlag% = 0
    CASE 3                                'SetPoint
      number! = PCdifSetPoint!
      Prompt$ = "Differential Pressure between Shell and Sample, kPa: "
      DO
        CALL LotusNumberPrompt(Prompt$, number!, frgd%, bkgd%)
        LOOP UNTIL number! >= 0! AND number! < 200!
        PCdifSetPoint! = number!
        CALL LotusMenuClear
      CASE 4                                'Prop Band
        number! = PCPropBand!
        Prompt$ = "Proportional Band, kPa: "
        DO
          CALL LotusNumberPrompt(Prompt$, number!, frgd%, bkgd%)
          LOOP UNTIL number! >= .01 AND number! < 200!
          PCPropBand! = number!
          CALL LotusMenuClear
        CASE 5                                'Reset Time
          number! = PCTauIntegral!
          Prompt$ = "Reset Time, seconds: "
          DO
            CALL LotusNumberPrompt(Prompt$, number!, frgd%, bkgd%)
            LOOP UNTIL number! >= .01 AND number! < 200!
            PCPTauIntegral! = number!
            CALL LotusMenuClear
          CASE 6                                'Quit
            exitcode% = -1
        END SELECT
      LOOP UNTIL exitcode% = -1
      ec% = 0
      LotusMenuClear
      OldTaskTimer! = TIMER
    END SUB
    SUB DLStartUp
      SHARED PCPropBand!, PCTauIntegral!, PCdifSetPoint!, PCErrSum!,
PresControlFlag%
    '$DYNAMIC
    DIM ExpInfo AS ExpInfoType, ExpHeader(0 TO 5) AS ExpHeaderType
    '$STATIC
    CALL DLStGetBaseName(BaseName$, ExpInfo, ExpHeader())
    Mode$ = ExpInfo.Mode
    Heading1$ = ExpHeader(0).s
    Heading2$ = ExpHeader(1).s
    IF ExpInfo.Mode = "USE " THEN
      ChanName$ = "A:" + BaseName$ + ".ChN "
      EngUnit$ = "A:" + BaseName$ + ".EnU"
    ELSE
      ChanName$ = "DataLog.ChN"
      EngUnit$ = "DataLog.EnU"
    END IF
    Month$ = RIGHT$("00" + LTRIM$(STR$(ExpInfo.Month)), 2)

```

```

Day$ = RIGHT$("00" + LTRIM$(STR$(ExpInfo.Day)), 2)
Year$ = RIGHT$(STR$(ExpInfo.Year), 2)
ExpDate$ = Month$ + "/" + Day$ + "/" + Year$
g$ = DTime$(-1)
CLS
'-----
'               DEFINE CHANNEL NAMES:
OPEN ChanName$ FOR INPUT AS 105
INPUT #105, NullString$
FOR i% = 1 TO 16
    INPUT #105, AtoDName$(i%)
NEXT
FOR i% = 1 TO 24
    INPUT #105, DIOName$(i%)
NEXT
FOR i% = 1 TO 2
    INPUT #105, DtoAName$(i%)
NEXT
FOR i% = 1 TO 16
    INPUT #105, TCName$(i%)
NEXT
CLOSE 105
'-----
'Define unit names for STD and METRIC systems
FOR i% = 1 TO 13
    FOR j% = 1 TO 2
        READ MsrUnit$(i%, j%)
    NEXT
NEXT
'-----
OPEN EngUnit$ FOR INPUT AS 105
FOR i% = 1 TO 50
    INPUT #105, EngUnitsList%(i%, 1), EngUnitsList%(i%, 2)
    IF EngUnitsList%(i%, 1) = 0 THEN EXIT FOR
NEXT
CLOSE 105
'Default differential pressure controller settings
PCPropBand! = 20!      'kPa
PCTauIntegral! = 60!   'Seconds
PCDifSetPoint! = 95!   'kPa
PCErrorSum! = 0!
PresControlFlag% = 0
frgd% = 7
bkgd% = 0
END SUB
SUB DLStGetBaseName (BaseName$, ExpInfo AS ExpInfoType, ExpHeader() AS
ExpHeaderType)
DIM InRegs AS RegTypeI, OutRegs AS RegTypeI
DIM FixedLengthString AS STRING * 64
'Get Current Counts
InRegs.ax = 1
CALL INTERRUPTX(&HA1, InRegs, OutRegs)
IF OutRegs.cx >= 0 THEN
    Count& = OutRegs.cx
ELSE
    Count& = 65536 + OutRegs.cx
END IF
ExpInfo.Counts = OutRegs.dx * 65536 + Count&
'Get date
InRegs.ax = 2
InRegs.bx = 0
CALL INTERRUPTX(&HA1, InRegs, OutRegs)
ExpInfo.Year = OutRegs.cx
ExpInfo.Month = OutRegs.dx 256
ExpInfo.Day = OutRegs.dx AND &HFF
'Get time
InRegs.ax = 3
InRegs.bx = 0

```



```

CALL INTERRUPTX(&HA1, InRegs, OutRegs)
ExpInfo.Hour = OutRegs.cx 256
ExpInfo.Minute = OutRegs.cx AND &HFF
ExpInfo.Second = OutRegs.dx 256
ExpInfo.Hundredth = OutRegs.dx AND &HFF
InRegs.ax = 4
CALL INTERRUPTX(&HA1, InRegs, OutRegs)
IF OutRegs.bx = 0 THEN
    ExpInfo.Mode = "TAKE"
ELSE
    ExpInfo.Mode = "USE "
END IF
'Determine BaseName
BaseName& = (ExpInfo.Month) * 100 + ExpInfo.Day
BaseName& = (BaseName& * 100) + ExpInfo.Hour
BaseName& = (BaseName& * 100) + ExpInfo.Minute
BaseName$ = "00000000000" + LTRIM$(STR$(BaseName&))
BaseName$ = RIGHT$(BaseName$, 8)
FOR i% = 0 TO 5
    InRegs.ax = 5
    InRegs.bx = i%
    InRegs.cx = VARSEG(FixedLengthString)
    InRegs.dx = VARPTR(FixedLengthString)
    CALL INTERRUPTX(&HA1, InRegs, OutRegs)
    ExpHeader(i%).s = FixedLengthString
NEXT
END SUB
FUNCTION DTime$ (Row%)
    ' This function returns the data time that corresponds to a given
    ' row in the database
    STATIC InRegs AS RegType, OutRegs AS RegType, StartTicks&
    IF Row% < 0 THEN
        InRegs.ax = 3
        CALL INTERRUPT(&HA1, InRegs, OutRegs) 'Get starting time
        dx& = OutRegs.dx
        cx& = OutRegs.cx
        IF dx& < 0 THEN dx& = dh& + 65536
        IF cx& < 0 THEN cx& = cx& + 65536
        Hrs% = cx& 256
        Min% = cx& MOD 256
        Sec% = dx& 256
        Hdrs% = dx& MOD 256
        StartTicks& = Hrs% * 65536! + Min% * 1092! + Sec% * 18.2 + Hdrs% *
        18.2 / 100!
        InRegs.ax = 0
        DTime$ = "Initialized"
    ELSE
        InRegs.bx = Row%
        CALL INTERRUPT(&HA1, InRegs, OutRegs) 'Get Ticks since start
        IF OutRegs.cx < 0 THEN Tick& = OutRegs.cx + 65536! ELSE Tick& =
        OutRegs.cx
        IF OutRegs.dx < 0 THEN HighCount& = OutRegs.dx + 65536! ELSE
        HighCount& = OutRegs.dx
        Tick& = Tick& + 65536! * HighCount& + StartTicks&
        IF OutRegs.ax = 0 THEN
            Hrs% = FIX(Tick& / 65536#)
            Tick& = Tick& - Hrs% * 65536#
            Min% = FIX(Tick& / 1092.6667#)
            Tick& = Tick& - Min% * 1092.6667#
            Sec! = FIX(Tick& / .18204#) / 100#
            Sec% = FIX(Sec!)
            Hdr% = 100 * (Sec! - Sec%)
            Hrs$ = RIGHT$(" " + STR$(Hrs%), 2)
            Min$ = RIGHT$("00" + LTRIM$(STR$(Min%)), 2)
            Sec$ = RIGHT$("00" + LTRIM$(STR$(Sec%)), 2)
            Hdr$ = RIGHT$("00" + LTRIM$(STR$(Hdr%)), 2)
            DTime$ = Hrs$ + ":" + Min$ + ":" + Sec$ + "." + Hdr$
        ELSE

```

```

    DTime$ = "Not Available"
END IF
END IF
END FUNCTION
FUNCTION ElapsedTime! (TO!)
ET! = TIMER - TO!
IF ET! < 0 THEN ET! = ET! + 86400 - TO!
ElapsedTime! = ET!
END FUNCTION
FUNCTION EngUnitsTable% (ChannelCode%)
FOR i% = LBOUND(EngUnitsList%, 1) TO UBOUND(EngUnitsList%, 1)
    IF EngUnitsList%(i%, 1) = ChannelCode% THEN
        EngUnitsTable% = EngUnitsList%(i%, 2)
    EXIT FUNCTION
END IF
NEXT
EngUnitsTable% = 1
END FUNCTION
FUNCTION GainValue (Gain%)
SELECT CASE Gain%
CASE 0
    GainValue = 1
CASE 1
    GainValue = 10
CASE 2
    GainValue = 100
CASE 3
    GainValue = 500
CASE ELSE
    GainValue = -1
END SELECT
END FUNCTION
SUB GetChan (Row%, Channel%, Gain%, Value%, ErrorCode%)
' This subroutine provides an interface to the assembly level
' GetChan routine that is loaded as a TSR (Terminate and Stay Resident)
' program before the basic program is run.
DIM InRegs AS RegType, OutRegs AS RegType
InRegs.ax = Channel%
InRegs.bx = Row%
CALL INTERRUPT(&HA3, InRegs, OutRegs)
Gain% = OutRegs.cx
Value% = OutRegs.bx
ErrorCode% = OutRegs.ax
END SUB
SUB GetTCCh (Row%, Channel%, Gain%, Value%, ErrorCode%)
' This subroutine provides an interface to the assembly level
' GetTCCh routine that is loaded as a TSR (Terminate and Stay Resident)
' program before the basic program is run.
DIM InRegs AS RegType, OutRegs AS RegType
InRegs.ax = Channel%
InRegs.bx = Row%
CALL INTERRUPT(&HA4, InRegs, OutRegs)
Gain% = OutRegs.cx
Value% = OutRegs.bx
ErrorCode% = OutRegs.ax
END SUB
FUNCTION JTC! (v!)
v# = v!
a# = (11569199.78# + v# * (-264917531.4# + v# * 2018441314))
JTC = -.048868252# + v# * (19873.14503# + v# * (-218614.5353# + v# * a#))
END FUNCTION
FUNCTION KTC! (v!)
v# = v!
g# = v# * (-860963914.9# + v# * (48350600000# + v# * (-1184520000000# +
v# * (1.3869E+13 + v# * -6337080000000#))))
KTC! = .226584602# + v# * (24152.109# + v# * (67233.42479999999# + v# *
(2210340.682# + g#)))
END FUNCTION

```

B.2 QuickBASIC Program DLSAVE

```

' $INCLUDE: 'qb.bi'
' $INCLUDE: 'datalgc.bi'
DECLARE SUB DLSMakeExpFile (BaseName$, ExpInfo AS ANY, ExpHeader() AS ANY)
DECLARE SUB DLSGetBaseName (BaseName$, ExpInfo AS ANY, ExpHeader() AS ANY)
DECLARE SUB DLSaveTm (BaseName$)
DECLARE SUB DLSaveGC (BaseName$)
DECLARE SUB DLSaveTCs (BaseName$)
DECLARE SUB DataLogSave ()
DECLARE SUB DLSaveChannel (BaseName$)
CALL DataLogSave
SUB DataLogSave
' Prompt user to place floppy disk in drive A
PRINT "Place empty data disk in drive A:"
PRINT "Press any key when done."
DO
LOOP UNTIL LEN(INKEY$) <> 0
PRINT
'Determine "BaseName" for files
DIM ExpInfo AS ExpInfoType, ExpHeader(0 TO 9) AS ExpHeaderType
CALL DLSGetBaseName(BaseName$, ExpInfo, ExpHeader())
'Save data for channels
CALL DLSaveChannel(BaseName$)
'Save data for tc channels
CALL DLSaveTCs(BaseName$)
'Save data for gc channel
CALL DLSaveGC(BaseName$)
'Save data for time
CALL DLSaveTm(BaseName$)
'Save channel names to disk
Comm$ = "copy datalog.EnU a:" + BaseName$ + ".EnU"
SHELL Comm$
'Save engineering units to disk
Comm$ = "copy datalog.ChN a:" + BaseName$ + ".ChN"
SHELL Comm$
'Save auxillary information in .exp file
CALL DLSMakeExpFile(BaseName$, ExpInfo, ExpHeader())
PRINT
END SUB
SUB DLSaveChannel (BaseName$)
'
'   This routine uses the BSAVE statement to conveniently save
'   an image of ch_buf to disk for archival and post processing.
'
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
'Get ch_buf addressing information
InRegs.ax = 0
InRegs.bx = 0
CALL INTERRUPTX(&HA3, InRegs, OutRegs)
ChanSeg% = OutRegs.es
ChanOff% = OutRegs.dx
ChanSize% = OutRegs.si
'Save file to disk
DEF SEG = ChanSeg%
BSAVE "A:" + BaseName$ + ".ChD", ChanOff%, ChanSize%
END SUB
SUB DLSaveGC (BaseName$)
'
'   This routine uses the BSAVE statement to conveniently save
'   an image of gc_buf to disk for archival and post processing.
'
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
'Get gc_buf addressing information
InRegs.ax = 0

```

```

InRegs.bx = 0
CALL INTERRUPTX(&HA5, InRegs, OutRegs)
GCseg% = OutRegs.es
GCoff% = OutRegs.dx
GCsize% = OutRegs.si
'Save file to disk
DEF SEG = GCseg%
BSAVE "A:" + BaseName$ + ".GCD", GCoff%, GCsize%
END SUB
SUB DLSaveTCs (BaseName$)
,
'   This routine uses the BSAVE statement to conveniently save
'   an image of tc_buf to disk for archival and post processing.
,
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
'Get tc_buf addressing information
InRegs.ax = 0
InRegs.bx = 0
CALL INTERRUPTX(&HA4, InRegs, OutRegs)
TCseg% = OutRegs.es
TCoff% = OutRegs.dx
TCsize% = OutRegs.si
'Save file to disk
DEF SEG = TCseg%
BSAVE "A:" + BaseName$ + ".TCD", TCoff%, TCsize%
END SUB
SUB DLSaveTm (BaseName$)
,
'   This routine uses the BSAVE statement to conveniently save
'   an image of td_tm_buf to disk for archival and post processing.
,
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
'Get td_tm_buf addressing information
InRegs.ax = 0
InRegs.bx = 0
CALL INTERRUPTX(&HA1, InRegs, OutRegs)
Tmseg% = OutRegs.es
Tmoff% = OutRegs.di
Tmsize% = OutRegs.si
'Save file to disk
DEF SEG = Tmseg%
BSAVE "A:" + BaseName$ + ".TmD", Tmoff%, Tmsize%
END SUB
SUB DLGetBaseName (BaseName$, ExpInfo AS ExpInfoType, ExpHeader() AS
    ExpHeaderType)
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
DIM FixedLengthString AS STRING * 64
'Get Current Counts
InRegs.ax = 1
CALL INTERRUPTX(&HA1, InRegs, OutRegs)
IF OutRegs.cx >= 0 THEN
    Count% = OutRegs.cx
ELSE
    Count% = 65536 + OutRegs.cx
END IF
ExpInfo.Counts = OutRegs.dx * 65536 + Count%
'Get date
InRegs.ax = 2
InRegs.bx = 0
CALL INTERRUPTX(&HA1, InRegs, OutRegs)
ExpInfo.Year = OutRegs.cx
ExpInfo.Month = OutRegs.dx 256
ExpInfo.Day = OutRegs.dx AND &HFF
'Get time
InRegs.ax = 3
InRegs.bx = 0
CALL INTERRUPTX(&HA1, InRegs, OutRegs)
ExpInfo.Hour = OutRegs.cx 256

```

```

ExpInfo.Minute = OutRegs.cx AND &HFF
ExpInfo.Second = OutRegs.dx 256
ExpInfo.Hundredth = OutRegs.dx AND &HFF
'Determine BaseName
BaseName# = (ExpInfo.Month) * 100 + ExpInfo.Day
BaseName# = (BaseName# * 100) + ExpInfo.Hour
BaseName# = (BaseName# * 100) + ExpInfo.Minute
BaseName$ = "0000000000" + LTRIM$(STR$(BaseName#))
BaseName$ = RIGHT$(BaseName$, 8)
FOR i% = 0 TO 9
    InRegs.ax = 5
    InRegs.bx = i%
    InRegs.cx = VARSEG(FixedLengthString)
    InRegs.dx = VARPTR(FixedLengthString)
    CALL INTERRUPTX(&BA1, InRegs, OutRegs)
    ExpHeader(i%).s = FixedLengthString
NEXT
END SUB
SUB DL$MakeExpFile (BaseName$, ExpInfo AS ExpInfoType, ExpHeader() AS
ExpHeaderType)
g% = FREEFILE
OPEN "A:" + BaseName$ + ".Exp" FOR OUTPUT AS g%
PRINT #g%, "Base Name = " + BaseName$ + ""
Month$ = RIGHT$("00" + LTRIM$(STR$(ExpInfo.Month)), 2)
Day$ = RIGHT$("00" + LTRIM$(STR$(ExpInfo.Day)), 2)
Year$ = RIGHT$(STR$(ExpInfo.Year), 2)
StDate$ = Month$ + "/" + Day$ + "/" + Year$
PRINT #g%, "Date = " + StDate$ + ""
Hour$ = RIGHT$("00" + LTRIM$(STR$(ExpInfo.Hour)), 2)
Minute$ = RIGHT$("00" + LTRIM$(STR$(ExpInfo.Minute)), 2)
Second$ = RIGHT$("00" + LTRIM$(STR$(ExpInfo.Second)), 2)
Hundrth$ = RIGHT$("00" + LTRIM$(STR$(ExpInfo.Hundredth)), 2)
StTime$ = Hour$ + ":" + Minute$ + ":" + Second$ + "." + Hundrth$
PRINT #g%, "Time = " + StTime$ + ""
PRINT #g%, "Counts = " + STR$(ExpInfo.Counts) + ""
FOR i% = 0 TO 9
    PRINT #g%, "Header " + STR$(i%) + " = " + ExpHeader(i%).s + ""
NEXT
CLOSE (g%)
END SUB

```

B.3 QuickBASIC Program DLLOAD

```

'$INCLUDE: 'qb.bi'
'$INCLUDE: 'datalogc.bi'
TYPE BsaveHeaderType
    Fchar AS STRING * 1
    FSeg AS INTEGER
    Foff AS INTEGER
    FSize AS INTEGER
END TYPE
DECLARE SUB DLLoadExpInfo (BaseName$, ExpInfo AS ANY, ExpHeader() AS ANY)
DECLARE SUB DLGetExpInfo (BaseName$, ExpInfo AS ExpInfoType,
ExpHeader() AS ExpHeaderType)
DECLARE SUB DataLogLoad ()
DECLARE SUB DLLoadTCs (BaseName$)
DECLARE SUB DLLoadTm (BaseName$)
DECLARE SUB DLLoadGC (BaseName$)
DECLARE SUB DLLoadChannel (BaseName$)
CALL DataLogLoad
SUB DataLogLoad
    DIM ExpInfo AS ExpInfoType
    DIM DirList$(100)
    DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
    DIM ExpHeader(0 TO 9) AS ExpHeaderType
    'Prompt User to place floppy disk in drive A:

```

```

PRINT "Place data disk in drive A:."
PRINT "Press any key to continue."
DO
LOOP UNTIL LEN(INKEY$) <> 0
'Retrieve BaseName
FileSpec$ = "A:*.Exp"
Attrib% = 0
GetDirectory FileSpec$, Attrib%, DirList$(), DirCount%
BaseName$ = DirList$(1)
Period% = INSTR(BaseName$, ".")
IF Period% > 0 THEN BaseName$ = LEFT$(BaseName$, Period% - 1)
'Halt TakeData
InRegs.ax = 4
InRegs.bx = &HFF
CALL INTERRUPTX(&HA0, InRegs, OutRegs)
CALL DLGetExpInfo(BaseName$, ExpInfo, ExpHeader())
CALL DLLoadChannel(BaseName$)
CALL DLLoadTCs(BaseName$)
CALL DLLoadGC(BaseName$)
CALL DLLoadTm(BaseName$)
'This should be after the BLoads so that the original info is overwritten
CALL DLLoadExpInfo(BaseName$, ExpInfo, ExpHeader())
END SUB
SUB DLGetExpInfo (BaseName$, ExpInfo AS ExpInfoType, ExpHeader() AS
ExpHeaderType)
g% = FREEFILE
OPEN "A:" + BaseName$ + ".Exp" FOR INPUT AS g%
'Scan through .Exp file for keywords and there respective values
DO
LINE INPUT #g%, a$
EqLoc% = INSTR(a$, "=")
IF EqLoc% > 0 THEN
Keyword$ = RTRIM$(LTRIM$(LEFT$(a$, EqLoc% - 1)))
Keyword$ = UCASE$(Keyword$)
a$ = MID$(a$, EqLoc% + 1)
RightMark% = INSTR(a$, "'")
a$ = LEFT$(a$, RightMark% - 1)
LeftMark% = INSTR(a$, "'")
KeywordValue$ = MID$(a$, LeftMark% + 1)
SELECT CASE Keyword$
CASE "BASE NAME"
FileBaseName$ = KeywordValue$
CASE "DATE"
StDate$ = KeywordValue$
CASE "TIME"
StTime$ = KeywordValue$
CASE "COUNTS"
Counts$ = KeywordValue$
CASE "HEADER 0" TO "HEADER 9"
i% = VAL(RIGHT$(Keyword$, 1))
ExpHeader(i%).s = KeywordValue$
CASE ELSE
END SELECT
END IF
LOOP WHILE NOT EOF(g%)
CLOSE (g%)
'Cut apart date:
Slash1% = INSTR(StDate$, "/")
Month$ = LEFT$(StDate$, Slash1% - 1)
ExpInfo.Month = VAL(Month$)
Slash2% = INSTR(Slash1% + 1, StDate$, "/")
Day$ = MID$(StDate$, Slash1% + 1, Slash2% - Slash1% - 1)
ExpInfo.Day = VAL(Day$)
Year$ = "19" + MID$(StDate$, Slash2% + 1)
ExpInfo.Year = VAL(Year$)
Colon1% = INSTR(StTime$, ":")
Hour$ = LEFT$(StTime$, Colon1% - 1)
ExpInfo.Hour = VAL(Hour$)

```

```

Colon2% = INSTR(Colon1% + 1, StTime$, ":")
Minute$ = MID$(StTime$, Colon1% + 1, Colon2% - Colon1% - 1)
ExpInfo.Minute = VAL(Minute$)
Period% = INSTR(StTime$, ".")
Second$ = MID$(StTime$, Colon2% + 1, Period% - Colon2% - 1)
ExpInfo.Second = VAL(Second$)
Hndrth$ = MID$(StTime$, Period% + 1, 2)
ExpInfo.Hundredth = VAL(Hndrth$)
ExpInfo.Counts = VAL(Count$)
END SUB
SUB DLLoadChannel (BaseName$)
DIM Header AS BsaveHeaderType
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
'Get ch_buf addressing information
InRegs.ax = 0
InRegs.bx = 0
CALL INTERRUPTX(&HA3, InRegs, OutRegs)
ChanSeg% = OutRegs.es
ChanOff% = OutRegs.dx
ChanSize% = OutRegs.si
g% = FREEFILE
OPEN "A:" + BaseName$ + ".Chd" FOR BINARY ACCESS READ WRITE AS #g%
GET #g%, 1, Header
OrgSize% = Header.FSize
IF OrgSize% > ChanSize% THEN
    Header.FSize = ChanSize%
    PUT #g%, 1, Header
END IF
'Retrieve file from disk
DEF SEG = ChanSeg%
BLOAD "A:" + BaseName$ + ".Chd", ChanOff%
IF OrgSize% > ChanSize% THEN
    Header.FSize = OrgSize%
    PUT #g%, 1, Header
END IF
CLOSE (g%)
END SUB
SUB DLLoadExpInfo (BaseName$, ExpInfo AS ExpInfoType, ExpHeader() AS
    ExpHeaderType)
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
DIM FixedLengthString AS STRING * 64, FixedLengthString2 AS STRING * 64
'Load Current Counts
InRegs.ax = 1
InRegs.dx = ExpInfo.Counts / 65536
Count% = ExpInfo.Counts MOD 65536
IF Count% > 32767 THEN
    InRegs.cx = 65536 - Count%
ELSE
    InRegs.cx = Count%
END IF
CALL INTERRUPTX(&HA0, InRegs, OutRegs)
InRegs.ax = 2 'Function Load Start Date
InRegs.cx = ExpInfo.Year
InRegs.dx = &H100 * ExpInfo.Month + ExpInfo.Day
CALL INTERRUPTX(&HA0, InRegs, OutRegs)
InRegs.ax = 3
InRegs.cx = &H100 * ExpInfo.Hour + ExpInfo.Minute
InRegs.dx = &H100 * ExpInfo.Second + ExpInfo.Hundredth
CALL INTERRUPTX(&HA0, InRegs, OutRegs)
FOR i% = 0 TO 9
    FixedLengthString = ExpHeader(i%).s
    InRegs.ax = 5
    InRegs.bx = i%
    InRegs.cx = VARSEG(FixedLengthString)
    InRegs.dx = VARPTR(FixedLengthString)
    CALL INTERRUPTX(&HA0, InRegs, OutRegs)
    InRegs.ax = 5
    InRegs.bx = 0

```

```

InRegs.cx = VARSEG(FixedLengthString2)
InRegs.dx = VARPTR(FixedLengthString2)
CALL INTERRUPTX(&HA1, InRegs, OutRegs)
PRINT FixedLengthString2
NEXT
InRegs.ax = 5
END SUB
SUB DLLoadGC (BaseName$)
DIM Header AS BsaveHeaderType
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
'Get gc_buf addressing information
InRegs.ax = 0
InRegs.bx = 0
CALL INTERRUPTX(&HA5, InRegs, OutRegs)
GCseg% = OutRegs.es
GCoff% = OutRegs.dx
GCsize% = OutRegs.si
g% = FREEFILE
OPEN "A:" + BaseName$ + ".GCD" FOR BINARY ACCESS READ WRITE AS #g%
GET #g%, 1, Header
OrgSize% = Header.FSize
IF OrgSize% > GCsize% THEN
    Header.FSize = GCsize%
    PUT #g%, 1, Header
END IF
'Retrieve file to disk
DEF SEG = GCseg%
BLOAD "A:" + BaseName$ + ".GCD", GCoff%
IF OrgSize% > GCsize% THEN
    Header.FSize = OrgSize%
    PUT #g%, 1, Header
END IF
CLOSE (g%)
END SUB
SUB DLLoadTCs (BaseName$)
DIM Header AS BsaveHeaderType
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
'Get tc_buf addressing information
InRegs.ax = 0
InRegs.bx = 0
CALL INTERRUPTX(&HA4, InRegs, OutRegs)
TCseg% = OutRegs.es
TCoff% = OutRegs.dx
TCsize% = OutRegs.si
g% = FREEFILE
OPEN "A:" + BaseName$ + ".TCD" FOR BINARY ACCESS READ WRITE AS #g%
GET #g%, 1, Header
OrgSize% = Header.FSize
IF OrgSize% > TCsize% THEN
    Header.FSize = TCsize%
    PUT #g%, 1, Header
END IF
'Retrieve file from disk
DEF SEG = TCseg%
BLOAD "A:" + BaseName$ + ".TCD", TCoff%
IF OrgSize% > TCsize% THEN
    Header.FSize = OrgSize%
    PUT #g%, 1, Header
END IF
CLOSE (g%)
END SUB
SUB DLLoadTm (BaseName$)
DIM Header AS BsaveHeaderType
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
'Get td_tm_buf addressing information
InRegs.ax = 0
InRegs.bx = 0
CALL INTERRUPTX(&HA1, InRegs, OutRegs)

```



```

TmSeg% = OutRegs.es
TmOff% = OutRegs.di
TmSize% = OutRegs.si
g% = FREEFILE
OPEN "A:" + BaseName$ + ".TmD" FOR BINARY ACCESS READ WRITE AS #g%
GET #g%, 1, Header
OrgSize% = Header.FSize
IF OrgSize% > TmSize% THEN
    Header.FSize = TmSize%
    PUT #g%, 1, Header
END IF
'Save file to disk
DEF SEG = TmSeg%
BLOAD "A:" + BaseName$ + ".TmD", TmOff%
IF OrgSize% > TmSize% THEN
    Header.FSize = OrgSize%
    PUT #g%, 1, Header
END IF
CLOSE (#g%)
END SUB

```

B.4 QuickBASIC Program GCCTRL

```

' $INCLUDE: 'QB.BI'
DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
DIM FixedLengthString AS STRING * 64
INPUT "Run Time, in Minutes: ", RunTime!
InRegs.bx = RunTime! * 60! * 18.2 / 2!
IF InRegs.bx < 0 THEN InRegs.bx = &H2000 'About 15 Minutes
INPUT "Switch Valve Reset Time, in Seconds: ", SwitchTime!
InRegs.cx = SwitchTime! * 18.2 / 2!
IF InRegs.cx < 0 THEN InRegs.cx = &H280 'About 70 Seconds
INPUT "Continuous Runs (y/n): ", y$
IF UCASE$(y$) = "N" THEN
    InRegs.ax = &H0
ELSE
    InRegs.ax = &HFF
END IF
CALL INTERRUPTX(&HA2, InRegs, OutRegs)
FixedLengthString = "Run: " + STR$(RunTime!) + "Min. Switch: " +
STR$(SwitchTime!) + "Sec. Reload: " + y$
InRegs.ax = 5
InRegs.bx = 3
InRegs.cx = VARSEG(FixedLengthString)
InRegs.dx = VARPTR(FixedLengthString)
CALL INTERRUPTX(&HA0, InRegs, OutRegs)
FixedLengthString = SPACE$(64)
InRegs.ax = 5
InRegs.bx = 3
InRegs.cx = VARSEG(FixedLengthString)
InRegs.dx = VARPTR(FixedLengthString)
CALL INTERRUPTX(&HA1, InRegs, OutRegs)
PRINT FixedLengthString
END

```

B.5 QuickBASIC Program GCINTEG

```

DECLARE SUB GCBasicPlot (BegOfPlot%, EndOfPlot%, BaseLine%)
DECLARE SUB GCMakeHeader (RunPtr%, Tab1%, Tab2%)
DECLARE FUNCTION GCTime$ (Now%)
DECLARE SUB GCMainLoop ()
DECLARE SUB DLSGetBaseName (BaseName$, ExpInfo AS ANY, ExpHeader() AS ANY)
DECLARE SUB GCAPTable (RunPtr%)

```

```

DECLARE SUB GCMaKeRHSGraph (BegOfRun%, EndOfRun%)
DECLARE SUB GCMaKeGraphBasic (BegOfRun%, EndOfRun%)
DECLARE SUB GCERaseEvents (RunPtr%)
DECLARE SUB GCDrawEvents (RunPtr%)
DECLARE SUB GCAEFetchPoint (RunPtr%, name$, Prompt$, CursorValues
AS ANY, Count%, Now%, exitcode%)
DECLARE FUNCTION GCBaseLine% (RunPtr%, Now%)
DECLARE SUB GCStartUp ()
DECLARE SUB GCMaKeWholeScreenState (WholeScreen AS ANY)
DECLARE SUB GCMaKeGraphRightHandSide (RightSideGraph AS ANY,
BegOfRun%, EndOfRun%)
DECLARE SUB GCAdjustPeaks (RunPtr%)
DECLARE SUB GCElimNoisePeaks (RunPtr%)
DECLARE SUB GCElimLocalPeaks (RunPtr%)
DECLARE SUB GCAEReview (RunPtr%)
DECLARE SUB GCOrderEvents (RunPtr%)
DECLARE SUB GCEventTableQuickSort (Low%, High%, RunPtr%)
DECLARE FUNCTION GCRandInt% (lower%, upper%)
DECLARE SUB GCSortEventTable (RunPtr%)
DECLARE SUB GCAEAdd (RunPtr%)
DECLARE SUB GCAEDelete (RunPtr%)
DECLARE SUB GCAdjustEvents (RunPtr%)
' $INCLUDE: 'qb.bi'
' $INCLUDE: 'GrafUtil.bi'
' $INCLUDE: 'Constant.bas'
' $INCLUDE: 'DataLogC.bi'
TYPE RunTableType
    BeginRun AS LONG
    EndRun AS LONG
    Analyzed AS INTEGER
    BaseLineBeg AS INTEGER
    BaseLineEnd AS INTEGER
    PeakPtr AS INTEGER
END TYPE
TYPE EventTableType
    Event AS INTEGER
    Counts AS INTEGER
    Time AS LONG
END TYPE
TYPE EventDataType
    Ptr AS INTEGER
    Status AS INTEGER
    Peak AS INTEGER
    LastValue AS INTEGER
    LastTime AS LONG
END TYPE
TYPE IntegTableType
    ElutTime AS LONG
    Counts AS LONG
    Component AS INTEGER
    Concentration AS SINGLE
END TYPE
TYPE CompIDType
    CompName AS STRING * 5
    ElutTime AS LONG
    Counts AS LONG
    Range AS LONG
    CalibFactor AS SINGLE
    IDd AS INTEGER
END TYPE
DECLARE FUNCTION GCscaled! (Count%, BaseLine%)
DECLARE FUNCTION GCInvScaled% (Now%, y!, RunPtr%)
DECLARE SUB GCadjustBaseLine (RunPtr%)
DECLARE SUB GCSetUpGCScreen (BegOfRun%, EndOfRun%, RunPtr%)
DECLARE SUB GCIntegRunMenu (RunPtr%, exitcode%)
DECLARE SUB GCHighLevelPlot (RunPtr%, AnalMode$, ErrorCode%)
DECLARE SUB GCAnalyze (RunPtr%, ErrorCode%)
DECLARE SUB GCCalcComp (RunPtr%, ErrorCode%)

```

```

DECLARE SUB GCProcess (RunPtr%, PlotMode$, ErrorCode%)
DECLARE SUB GCPeakID (RunPtr%, ErrorCode%)
DECLARE SUB GCFindAnalMode (RunPtr%, AnalMode$)
DECLARE SUB GCIntegTableWrite (FileName$)
DECLARE SUB GCIntegTableLoad (FileName$)
DECLARE SUB GCIntegInitial ()
DECLARE SUB GCIntegrate (RunPtr%, GCMinPeakWidth%, GCMinNumOfCounts%,
    ErrorCode%)
DECLARE SUB GCEventId (Now$, Counts%, BaseLine%, RunPtr%)
DECLARE SUB GCFindBaseLine (BegOfRange$, EndOfRange$, BaseLine%)
DECLARE SUB GCUpdateRunTable (RunPtr%)
DECLARE SUB GCFindEndOfRun (RunPointer$, EndOfRun$, ErrorCode%)
DECLARE SUB GCEventMonitor (Now$, Count%, Mark$, EvCount%, Event%())
DECLARE SUB GetGC (Row%, Channel%, Gain%, Value%, ErrorCode%)
DECLARE FUNCTION GCPeakInteg$ (RunPtr%, PeakStart$, PeakStop$)
CONST LOAD = 0, INJECT = 3
' $DYNAMIC
DIM SHARED CompId(-1 TO 20) AS CompIDType
DIM SHARED IncompRunPtr%, SearchedToPtr$
DIM SHARED GCNoise%, GCMinPeakSize%, GCMinNumOfCount%,
    GCMinPeakWidth%
DIM SHARED ElutWt!, CountWt!, MinLike!, NumOfComponents%
DIM SHARED EventName$(-3 TO 3)
DIM SHARED RunTable(40) AS RunTableType
DIM SHARED EventData(40) AS EventDataType
DIM SHARED IntegTable(40, 20) AS IntegTableType
DIM SHARED EventTable(40, 60) AS EventTableType
DIM SHARED WholeScreen AS ScreenStateType, RightSideGraph AS ScreenStateType
DIM SHARED BasicGraph AS ScreenStateType
DIM SHARED ExpInfo AS ExpInfoType, ExpHeader(0 TO 9) AS ExpHeaderType
DIM SHARED BaseName$
CALL GCMainLoop
END
REM $STATIC
SUB DLSGetBaseName (BaseName$, ExpInfo AS ExpInfoType, ExpHeader() AS
    ExpHeaderType)
    DIM InRegs AS RegTypeX, OutRegs AS RegTypeX
    DIM FixedLengthString AS STRING * 64
    'Get Current Counts
    InRegs.ax = 1
    CALL INTERRUPTX(&HA1, InRegs, OutRegs)
    IF OutRegs.cx >= 0 THEN
        Count$ = OutRegs.cx
    ELSE
        Count$ = 65536 + OutRegs.cx
    END IF
    ExpInfo.Counts = OutRegs.dx * 65536 + Count$
    'Get date
    InRegs.ax = 2
    InRegs.bx = 0
    CALL INTERRUPTX(&HA1, InRegs, OutRegs)
    ExpInfo.Year = OutRegs.cx
    ExpInfo.Month = OutRegs.dx 256
    ExpInfo.Day = OutRegs.dx AND &HFF
    'Get time
    InRegs.ax = 3
    InRegs.bx = 0
    CALL INTERRUPTX(&HA1, InRegs, OutRegs)
    ExpInfo.Hour = OutRegs.cx 256
    ExpInfo.Minute = OutRegs.cx AND &HFF
    ExpInfo.Second = OutRegs.dx 256
    ExpInfo.Hundredth = OutRegs.dx AND &HFF
    'Determine BaseName
    BaseName$ = (ExpInfo.Month) * 100 + ExpInfo.Day
    BaseName$ = (BaseName$ * 100) + ExpInfo.Hour
    BaseName$ = (BaseName$ * 100) + ExpInfo.Minute
    BaseName$ = "00000000000" + LTRIM$(STR$(BaseName$))
    BaseName$ = RIGHT$(BaseName$, 8)

```

```

FOR i% = 0 TO 9
  InRegs.ax = 5
  InRegs.bx = i%
  InRegs.cx = VARSEG(FixedLengthString)
  InRegs.dx = VARPTR(FixedLengthString)
  CALL INTERRUPTX(&HA1, InRegs, OutRegs)
  ExpHeader(i%).s = FixedLengthString
NEXT
END SUB
SUB GCAdjustBaseLine (RunPtr%)
DIM CursorValues AS CursorValueType
CALL GraphicCursorSelect(0)
CursorValues.xMin = RunTable(RunPtr%).BeginRun
CursorValues.yMin = -.9
CursorValues.xMax = RunTable(RunPtr%).EndRun - 31
CursorValues.yMax = .9
CursorValues.xInc = 1!
CursorValues.yInc = .008
Changed% = FALSE
GLMMakeBorder "Baseline", 0
LOCATE 2, 8
PRINT "Mark beginning location of baseline."
LOCATE 3, 8
PRINT "Press Return when done."
x! = CursorValues.xMin
y! = 0!
CALL GraphicCursor(x!, y!, CursorValues, KeyCode%)
IF KeyCode% = CR THEN
  Now% = x!
  BegBase% = GCInvScaled%(Now%, y!, RunPtr%)
  Changed% = TRUE
END IF
GLMMakeBorder "Baseline", 0
LOCATE 2, 8
PRINT "Mark ending location of baseline."
LOCATE 3, 8
PRINT "Press Return when done."
x! = CursorValues.xMax
y! = 0!
CALL GraphicCursor(x!, y!, CursorValues, KeyCode%)
IF KeyCode% = CR THEN
  Now% = x!
  EndBase% = GCInvScaled%(Now%, y!, RunPtr%)
  Changed% = TRUE
END IF
IF Changed% THEN
  RunTable(RunPtr%).BaseLineBeg = BegBase%
  RunTable(RunPtr%).BaseLineEnd = EndBase%
  RunTable(RunPtr%).Analyzed = 1 ' Status Base Line defined
END IF
END SUB
SUB GCAdjustEvents (RunPtr%)
' $DYNAMIC
DIM short$(10), long$(10)
' $STATIC
'CALL LotusMenuClear
name$ = "GC-Integrate-Runs-Events"
short$(1) = "Review"
long$(1) = "Review the events currently used in identifying peaks."
short$(2) = "Add"
long$(2) = "Add an event."
short$(3) = "Delete"
long$(3) = "Delete an event."
short$(4) = "Clean"
long$(4) = "Eliminate local extrema and noise peaks."
short$(5) = "Quit"
long$(5) = "Return to Previous Menu."
items% = 5

```

```

sens% = 0          ' 0 = Not case sensitive
AnalMode$ = "Editing Events"
DO
  CALL GraphicLotusMenu(name$, short$(), long$(), items%, choice%, sens%)
  SELECT CASE choice%
    CASE 0          'Escape key pressed
      BEEP
      exitcode% = 1
    CASE 1
      CALL GCAEReview(RunPtr%)
      exitcode% = 1
    CASE 2          'Add Event
      CALL GCAEAdd(RunPtr%)
      exitcode% = 1
    CASE 3          'Delete Event
      CALL GCAEDelete(RunPtr%)
      exitcode% = 1
    CASE 4          'Clean
      CALL GCEraseEvents(RunPtr%)
      CALL GCElimLocalPeaks(RunPtr%)
      CALL GCElimNoisePeaks(RunPtr%)
      CALL GCDrawEvents(RunPtr%)
      exitcode% = 1
    CASE 5          'Quit
      exitcode% = -1
  END SELECT
  'IF ExitCode% = 0 THEN
  '  CLS
  '  BegOfRun# = RunTable(RunPtr%).BeginRun
  '  EndOfRun# = RunTable(RunPtr%).EndRun
  '  GCSetUpGCScreen BegOfRun#, EndOfRun#, RunPtr%
  '  CALL GCHighLevelPlot(RunPtr%, AnalMode$, ErrorCode%)
  ' END IF
  LOOP UNTIL exitcode% = -1
  RunTable(RunPtr%).Analyzed = 2
END SUB
SUB GCAdjustPeaks (RunPtr%)
IF RunPtr% >= IncompRunPtr% THEN
  BEEP
  EXIT SUB
END IF
SetScreenState WholeScreen
CLS
CALL GraphicCursorSelect(1)
CALL GCMakeHeader(RunPtr%, 2, 58)
CALL GCAPTable(RunPtr%)
BegOfRun# = RunTable(RunPtr%).BeginRun
EndOfRun# = RunTable(RunPtr%).EndRun
LOCATE 25, 40
PRINT "0";
LOCATE 25, 58
PRINT "Minutes";
LOCATE 25, 77
Min% = FIX((EndOfRun# - BegOfRun#) * 16 / (18.2 * 60!) + .5)
PRINT Min%;
GCMakeRESGraph BegOfRun#, EndOfRun#
SetScreenState RightSideGraph
FOR i! = BegOfRun# TO EndOfRun# STEP 68.25
  LINE (i!, -1.06)-(i!, -1.01)
NEXT
GCHighLevelPlot RunPtr%, "Adjust Peaks", ErrorCode%
CurPeakPtr% = 0
DO
  IF CurPeakPtr% = 0 THEN
    CALL GraphicCursorPtr(x!, y!, 0)
  ELSE
    x# = IntegTable(RunPtr%, CurPeakPtr%).ElutTime + BegOfRun#
    x! = x#
  
```

```

LargeStep% = x* 16
SmallStep% = x* MOD 16
CALL GetGC(LargeStep%, SmallStep%, ValvePos%, Count%, ErrorCode%)
y! = GCscaled(Count%, GCBaseLine%(RunPtr%, x*))
CALL GraphicCursorPtr(x!, y!, 1)
END IF
IF CurPeakPtr% > 0 AND CurPeakPtr% < 19 THEN
  SetScreenState WholeScreen
  b$ = RTRIM$(LTRIM$(STR$(CurPeakPtr%)))
  LOCATE CurPeakPtr% + 5, 5 - LEN(b$)
  InversPrint b$
  SetScreenState RightSideGraph
END IF
DO
  AKey$ = INKEY$
  LOOP WHILE LEN(AKey$) = 0
  ' Test for extended ascii
  IF LEN(AKey$) = 2 THEN
    g% = ASC(RIGHT$(AKey$, 1)) * 100
  ELSE
    g% = ASC(AKey$)
  END IF
  IF CurPeakPtr% > 0 AND CurPeakPtr% <= 19 THEN
    SetScreenState WholeScreen
    LOCATE CurPeakPtr% + 5, 2
    PRINT USING "###"; CurPeakPtr%;
  END IF
  SetScreenState RightSideGraph
  SELECT CASE g%
    CASE ESC
      exitcode% = ESC
    CASE Lft, Up
      CurPeakPtr% = CurPeakPtr% - 1
    CASE Rgt, Down
      CurPeakPtr% = CurPeakPtr% + 1
    CASE CR
      exitcode% = CR
    CASE Del
      IntegTable(RunPtr%, CurPeakPtr%).Component = -1 'Reset to unknown
      RunTable(RunPtr%).Analyzed = 4
      CALL GCProcess(RunPtr%, "No", ErrorCode%)
      CALL GCAPTable(RunPtr%)
      SetScreenState RightSideGraph
    CASE ASC("+"), ASC("-")
      IF g% = ASC("+") THEN Inc% = 1 ELSE Inc% = -1
      IntegTable(RunPtr%, CurPeakPtr%).Component = IntegTable(RunPtr%,
CurPeakPtr%).Component + Inc%
      RunTable(RunPtr%).Analyzed = 4
      CALL GCProcess(RunPtr%, "No", ErrorCode%)
      CALL GCAPTable(RunPtr%)
      SetScreenState RightSideGraph
    CASE ELSE
  END SELECT
  IF CurPeakPtr% < 0 THEN CurPeakPtr% = 1
  IF CurPeakPtr% > RunTable(RunPtr%).PeakPtr THEN CurPeakPtr% = 0
  LOOP UNTIL exitcode% = CR OR exitcode% = ESC
  CALL GraphicCursorPtr(x!, y!, 0)
END SUB
SUB GCAEAdd (RunPtr%)
' $DYNAMIC
DIM short$(10), long$(10)
' $STATIC
DIM CursorValues AS CursorValueType
name$ = "GC-Integrate-Runs"
short$(1) = "Start+"
long$(1) = "Mark the start of a positive peak."
short$(2) = "Peak+"
long$(2) = "Mark the extreme point of a positive peak."

```

```

short$(3) = "End+"
long$(3) = "Mark the end of a positive peak."
short$(4) = "Start-"
long$(4) = "Mark the start of a negative peak."
short$(5) = "Peak-"
long$(5) = "Mark the extreme point of a negative peak."
short$(6) = "End-"
long$(6) = "Mark the end of a negative peak."
short$(7) = "Quit"
long$(7) = "Return to the previous menu."
items% = 7
sens% = 0      ' 0 = Not case sensitive
CALL GraphicCursorSelect(0)
CursorValues.xMin = RunTable(RunPtr%).BeginRun
CursorValues.yMin = -.9
CursorValues.xMax = RunTable(RunPtr%).EndRun - 30
CursorValues.yMax = .9
CursorValues.xInc = 1!
CursorValues.yInc = .008
Now& = (CursorValues.xMin + CursorValues.xMax) / 2!
Count% = GCBaseLine%(RunPtr%, Now&)
DO
  CALL GraphicLotusMenu(name$, short$(), long$(), items%, choice%, sens%)
  SELECT CASE choice%
    CASE 0
      CASE 1 'Start Positive
        Ev% = 1
        Prompt$ = "Mark start of a positive peak"
      CASE 2 'Positive Peak
        Ev% = 2
        Prompt$ = "Mark extreme point of a positive peak"
      CASE 3 'End Positive
        Ev% = 3
        Prompt$ = "Mark the end of a positive peak"
      CASE 4 'Start Neg
        Ev% = -1
        Prompt$ = "Mark the start of a negative peak"
      CASE 5 'Negative Peak
        Ev% = -2
        Prompt$ = "Mark the extreme point of a negative peak"
      CASE 6 'End Negative
        Ev% = -3
        Prompt$ = "Mark the end of a negative peak"
      CASE ELSE
        EXIT SUB
    END SELECT
    exitcode% = 0
    CALL GCAEFetchPoint(RunPtr%, "Add", Prompt$, CursorValues, Count%, Now&,
    exitcode%)
    IF exitcode% = CR OR exitcode% = Ins THEN
      EvPtr% = EventData(RunPtr%).Ptr + 1
      EventData(RunPtr%).Ptr = EvPtr%
      EventTable(RunPtr%, EvPtr%).Event = Ev%
      EventTable(RunPtr%, EvPtr%).Time = Now&
      LargeStep% = Now& 16
      SmallStep% = Now& MOD 16
      CALL GetGC(LargeStep%, SmallStep%, ValvePos%, Count%, ErrorCode%)
      EventTable(RunPtr%, EvPtr%).Counts = Count%
      'Now update screen:
      BaseLine% = GCBaseLine%(RunPtr%, Now&)
      LINE (Now&, GCScalcd(Count%, BaseLine%) - .05)-(Now&, GCScalcd(Count%,
      BaseLine%) + .05)
      GCOrderEvents RunPtr%
    END IF
  LOOP UNTIL exitcode% = ESC OR exitcode% = CR
  ERASE long$, short$
END SUB
SUB GCAEDelete (RunPtr%)

```

```

GraphicCursorSelect (0)
GLMMakeBorder "Delete", 0
LOCATE 2, 8
PRINT "Use Right and Left cursor keys to select event. "
LOCATE 3, 8
PRINT "Commands: [Del],[Esc],[Return]"
EvPtr% = 1
DO
  IF EvPtr% > EventData(RunPtr%).Ptr THEN EvPtr% = EventData(RunPtr%).Ptr
  x! = EventTable(RunPtr%, EvPtr%).Time
  x% = x!
  BaseLine% = GCBaseLine%(RunPtr%, x%)
  y! = GCScalEd(EventTable(RunPtr%, EvPtr%).Counts, BaseLine%)
  CALL GraphicCursorPtr(x!, y!, 1)
  LOCATE 3, 40
  PRINT "Event: "; EvPtr%; " - "; EventName$(EventTable(RunPtr%, EvPtr%).Event);
  DO
    AKey$ = INKEY$
    LOOP WHILE LEN(AKey$) = 0
    ' Test for extended ascii
    IF LEN(AKey$) = 2 THEN
      g% = ASC(RIGHT$(AKey$, 1)) * 100
    ELSE
      g% = ASC(AKey$)
    END IF
    SELECT CASE g%
      CASE ESC
        exitcode% = ESC
      CASE Lft
        EvPtr% = EvPtr% - 1
      CASE Rgt
        EvPtr% = EvPtr% + 1
      CASE CR
        exitcode% = CR
      CASE Del
        CALL GraphicCursorPtr(x!, y!, 0)
        LINE (x!, y! - .05)-(x!, y! + .05), 0
        LINE (x!, 0)-(x!, y!)
        CALL GraphicCursorPtr(x!, y!, 1)
        FOR i% = EvPtr% TO EventData(RunPtr%).Ptr
          EventTable(RunPtr%, i%).Event = EventTable(RunPtr%, i% + 1).Event
          EventTable(RunPtr%, i%).Time = EventTable(RunPtr%, i% + 1).Time
          EventTable(RunPtr%, i%).Counts = EventTable(RunPtr%, i% + 1).Counts
        NEXT
        EventData(RunPtr%).Ptr = EventData(RunPtr%).Ptr - 1
      CASE ELSE
    END SELECT
    IF EvPtr% < 1 THEN EvPtr% = 1
  LOOP UNTIL exitcode% = CR OR exitcode% = ESC
  CALL GraphicCursorPtr(x!, y!, 0)
  IF exitcode% = CR THEN
    LINE (x!, y! - .05)-(x!, y! + .05), 0
    LINE (x!, 0)-(x!, y!)
    FOR i% = EvPtr% TO EventData(RunPtr%).Ptr
      EventTable(RunPtr%, i%).Event = EventTable(RunPtr%, i% + 1).Event
      EventTable(RunPtr%, i%).Time = EventTable(RunPtr%, i% + 1).Time
      EventTable(RunPtr%, i%).Counts = EventTable(RunPtr%, i% + 1).Counts
    NEXT
    EventData(RunPtr%).Ptr = EventData(RunPtr%).Ptr - 1
  END IF
END SUB
SUB GCAEFetchPoint (RunPtr%, name$, Prompt$, CursorValues AS CursorValueType,
Count%, Now%, exitcode%)
GLMMakeBorder name$, 0
LOCATE 2, 8
PRINT Prompt$
LOCATE 3, 8
PRINT "[Ins], [Return] or [Esc]"

```



```

x! = Now&
y! = GCscaled(Count%, GCBaseLine(RunPtr%, Now&))
CALL GraphicCursor(x!, y!, CursorValues, KeyCode%)
exitcode% = KeyCode%
Now& = x!
Count% = GCInvScaled%(Now&, y!, RunPtr%)
END SUB
SUB GCAEReview (RunPtr%)
CALL GraphicCursorSelect(0)
GLMMakeBorder "Review", 0
LOCATE 2, 8
PRINT "Use Right and Left cursor keys to select event."
LOCATE 3, 8
PRINT "Press Return when done."
EvPtr% = 1
DO
  IF EvPtr% > EventData(RunPtr%).Ptr THEN EvPtr% = EventData(RunPtr%).Ptr
  x! = EventTable(RunPtr%, EvPtr%).Time
  x& = x!
  y! = GCscaled(EventTable(RunPtr%, EvPtr%).Counts, GCBaseLine%(RunPtr%, x&))
  CALL GraphicCursorPtr(x!, y!, 1)
  LOCATE 3, 40
  PRINT "Event: "; EvPtr%; " - "; EventName$(EventTable(RunPtr%, EvPtr%).Event);
  DO
    AKey$ = INKEY$
    LOOP WHILE LEN(AKey$) = 0
    ' Test for extended ascii
    IF LEN(AKey$) = 2 THEN
      g% = ASC(RIGHT$(AKey$, 1)) * 100
    ELSE
      g% = ASC(AKey$)
    END IF
    SELECT CASE g%
      CASE ESC
        exitcode% = ESC
      CASE Lft
        EvPtr% = EvPtr% - 1
      CASE Rgt
        EvPtr% = EvPtr% + 1
      CASE CR
        exitcode% = CR
      CASE ELSE
      END SELECT
    IF EvPtr% < 1 THEN EvPtr% = 1
  LOOP UNTIL exitcode% = CR OR exitcode% = ESC
  CALL GraphicCursorPtr(x!, y!, 0)
END SUB
SUB GCAnalyze (RunPtr%, ErrorCode%)
BegOfRun& = RunTable(RunPtr%).BeginRun
EndOfRun& = RunTable(RunPtr%).EndRun
EndBaseLine% = RunTable(RunPtr%).BaseLineEnd
BegBaseLine% = RunTable(RunPtr%).BaseLineBeg
Drift! = (BegBaseLine% - EndBaseLine%) / (BegOfRun& - EndOfRun&)
FOR Now& = BegOfRun& TO EndOfRun&
  BaseLine% = BegBaseLine% + Drift! * (Now& - BegOfRun&)
  LargeStep% = Now& \ 16
  SmallStep% = Now& MOD 16
  CALL GetGC(LargeStep%, SmallStep%, ValvePos%, Count%, ErrorCode%)
  IF ErrorCode% <> 0 THEN EXIT SUB
  CALL GCEventId(Now&, Count%, BaseLine%, RunPtr%)
NEXT
END SUB
SUB GCAPTable (RunPtr%)
SetScreenState WholeScreen
LOCATE 5, 2
PRINT "Peak Time,m   Counts Comp. Conc,mol%"
FOR PeakPtr% = 1 TO RunTable(RunPtr%).PeakPtr
  LOCATE PeakPtr% + 5, 2

```

```

PRINT USING "###"; PeakPtr%;
PRINT USING " ###.###"; IntegTable(RunPtr%, PeakPtr%).ElutTime * 16!
/ (18.2 * 60!);
PRINT USING " #####"; IntegTable(RunPtr%, PeakPtr%).Counts;
PRINT USING "      "; CompId(IntegTable(RunPtr%, PeakPtr%).Component).
CompName;
PRINT USING " ###.###"; IntegTable(RunPtr%, PeakPtr%).Concentration * 100!;
NEXT
END SUB
FUNCTION GCBaseLine% (RunPtr%, Now%)
BegOfRun% = RunTable(RunPtr%).BeginRun
EndOfRun% = RunTable(RunPtr%).EndRun
EndBaseLine% = RunTable(RunPtr%).BaseLineEnd
BegBaseLine% = RunTable(RunPtr%).BaseLineBeg
Drift! = (BegBaseLine% - EndBaseLine%) / (BegOfRun% - EndOfRun%)
GCBaseLine% = BegBaseLine% + Drift! * (Now% - BegOfRun%)
END FUNCTION
SUB GCBasicPlot (BegOfPlot%, EndOfPlot%, BaseLine%)
FOR Now% = BegOfPlot% TO EndOfPlot%
LargeStep% = Now% 16
SmallStep% = Now% MOD 16
CALL GetGC(LargeStep%, SmallStep%, ValvePos%, Count%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
Diff = (Count% - BaseLine%) / 2047
IF Diff > .03 THEN
ScaledDiff = .5 + .5155 * (Diff - .03)
ELSEIF Diff < -.03 THEN
ScaledDiff = -.5 + .5155 * (Diff + .03)
ELSE
ScaledDiff = 16.667 * Diff
END IF
IF ValvePos% = 0 THEN PSET (Now%, -1.05) ELSE PSET (Now%, -1.02)
PSET (Now%, ScaledDiff)
NEXT
END SUB
SUB GCCalcComp (RunPtr%, ErrorCode%)
'Apply weighting factors:
Accum! = 0
FOR Peak% = 1 TO RunTable(RunPtr%).PeakPtr
a! = ABS(IntegTable(RunPtr%, Peak%).Counts)
b! = ABS(CompId(IntegTable(RunPtr%, Peak%).Component).CalibFactor)
IntegTable(RunPtr%, Peak%).Concentration = a! * b!
Accum! = Accum! + a! * b!
NEXT
'Normalize Composition
IF Accum! <> 0! THEN
FOR Peak% = 1 TO RunTable(RunPtr%).PeakPtr
a! = IntegTable(RunPtr%, Peak%).Concentration
IntegTable(RunPtr%, Peak%).Concentration = a! / Accum!
NEXT
END IF
END SUB
SUB GCDrawEvents (RunPtr%)
FOR EvPtr% = 1 TO EventData(RunPtr%).Ptr
x! = EventTable(RunPtr%, EvPtr%).Time
x% = x!
BaseLine% = GCBaseLine%(RunPtr%, x%)
y! = GCScalcd(EventTable(RunPtr%, EvPtr%).Counts, BaseLine%)
LINE (x!, y! - .05)-(x!, y! + .05)
NEXT
END SUB
SUB GCElimLocalPeaks (RunPtr%)
,
, This routine checks for multiple extrema during a peak.
, If it finds one it eliminates the first one.
,
Ptr% = 0
DO

```

```

Ptr% = Ptr% + 1
DeleteEvent% = FALSE
IF EventTable(RunPtr%, Ptr%).Event = 2 THEN
  IF EventTable(RunPtr%, Ptr% + 1).Event = 2 THEN
    DeleteEvent% = TRUE
  END IF
ELSEIF EventTable(RunPtr%, Ptr%).Event = -2 THEN
  IF EventTable(RunPtr%, Ptr% + 1).Event = -2 THEN
    DeleteEvent% = TRUE
  END IF
END IF
IF DeleteEvent% = TRUE THEN
  FOR i% = Ptr% TO EventData(RunPtr%).Ptr
    EventTable(RunPtr%, i%).Event = EventTable(RunPtr%, i% + 1).Event
    EventTable(RunPtr%, i%).Time = EventTable(RunPtr%, i% + 1).Time
    EventTable(RunPtr%, i%).Counts = EventTable(RunPtr%, i% + 1).Counts
  NEXT
  EventData(RunPtr%).Ptr = EventData(RunPtr%).Ptr - 1
  Ptr% = Ptr% - 1
END IF
LOOP UNTIL Ptr% = EventData(RunPtr%).Ptr
END SUB
SUB GCElimNoisePeaks (RunPtr%)
FOR EvPtr% = 1 TO EventData(RunPtr%).Ptr
  DeletePeak% = FALSE
  SELECT CASE EventTable(RunPtr%, EvPtr%).Event
    CASE 0
      Status% = 0
      'Start of Run
    CASE 1
      'Start Positive Peak
      SELECT CASE Status%
        CASE 0, 3, -3
          Status% = 1
          PeakStart% = EventTable(RunPtr%, EvPtr%).Time
          PeakStartPtr% = EvPtr%
        CASE ELSE
          ErrorCode% = -1
          Status% = 3
          END SELECT
          CASE 2
            'Max Positive Peak
            SELECT CASE Status%
              CASE 1, 2
                Status% = 2
              CASE ELSE
                ErrorCode% = -1
                Status% = 3
                END SELECT
                CASE 3
                  'End of Positive Peak
                  SELECT CASE Status%
                    CASE 2
                      Status% = 3
                      PeakStop% = EventTable(RunPtr%, EvPtr%).Time
                      IF PeakStop% - PeakStart% > GCMinPeakWidth% THEN
                        Counts% = GCPeakInteg%(RunPtr%, PeakStart%, PeakStop%)
                        IF ABS(Counts%) < GCMinNumOfCounts% THEN
                          DeletePeak% = TRUE
                        END IF
                      ELSE
                        DeletePeak% = TRUE
                      END IF
                    CASE ELSE
                      Status% = 3
                      END SELECT
                      CASE -1
                        'Start of Negative Peak
                        SELECT CASE Status%
                          CASE 0, 3, -3
                            Status% = -1
                            PeakStart% = EventTable(RunPtr%, EvPtr%).Time
                            PeakStartPtr% = EvPtr%

```

```

CASE ELSE
  ErrorCode% = -1
  Status% = -3
  END SELECT
  CASE -2
    SELECT CASE Status%
CASE -1, -2
  Status% = -2
CASE ELSE
  ErrorCode% = -1
  Status% = -3
  END SELECT
  CASE -3
    SELECT CASE Status%
CASE -2
  Status% = -3
  PeakStop% = EventTable(RunPtr%, EvPtr%).Time
  IF PeakStop% - PeakStart% > GCMinPeakWidth% THEN
    Counts% = GCPeakInteg%(RunPtr%, PeakStart%, PeakStop%)
    IF ABS(Counts%) < GCMinNumOfCounts% THEN
      DeletePeak% = TRUE
    END IF
  ELSE
    DeletePeak% = TRUE
  END IF
  CASE ELSE
    Status% = -3
    END SELECT
  END SELECT
  IF DeletePeak% = TRUE THEN
    FOR j% = EvPtr% TO PeakStartPtr% STEP -1
      FOR i% = j% TO EventData(RunPtr%).Ptr
        EventTable(RunPtr%, i%).Event = EventTable(RunPtr%, i% + 1).Event
        EventTable(RunPtr%, i%).Time = EventTable(RunPtr%, i% + 1).Time
        EventTable(RunPtr%, i%).Counts = EventTable(RunPtr%, i% + 1).Counts
      NEXT
      EventData(RunPtr%).Ptr = EventData(RunPtr%).Ptr - 1
    NEXT
  END IF
NEXT
END SUB
SUB GCERaseEvents (RunPtr%)
FOR EvPtr% = 1 TO EventData(RunPtr%).Ptr
  x! = EventTable(RunPtr%, EvPtr%).Time
  x% = x!
  BaseLine% = GCBaseLine%(RunPtr%, x%)
  y! = GCScalEd(EventTable(RunPtr%, EvPtr%).Counts, BaseLine%)
  LINE (x!, y! - .05)-(x!, y! + .05), 0
  LINE (x!, 0)-(x!, y!)
NEXT
END SUB
SUB GCEventId (Now%, Counts%, BaseLine%, RunPtr%)
r% = RunPtr%
p% = EventData(r%).Ptr
s% = EventData(r%).Status
IF p% = UBOUND(EventTable, 2) AND s% <> 0 THEN EXIT SUB
k% = EventData(r%).Peak
t% = EventData(r%).LastTime
v% = EventData(r%).LastValue
SELECT CASE s%
CASE 0
  'StartRun
  s% = 1
  'BaseLine
  p% = 1
  'First Event
  EventTable(r%, p%).Time = Now%
  'Record Time
  EventTable(r%, p%).Event = 0
  'StartRun
  EventTable(r%, p%).Counts = Counts%
  k% = 0
CASE 1
  'BaseLine"

```

```

IF Counts% - BaseLine% > GCMinPeakSize% THEN
    s% = 2                                'PositivePrePeak'
    p% = p% + 1
    EventTable(r%, p%).Time = t%          'Record Time
    EventTable(r%, p%).Event = 1          'Start Positive Peak
    EventTable(r%, p%).Counts = v%
    k% = 0
ELSEIF Counts% - BaseLine% < -GCMinPeakSize% THEN
    s% = -2                                'NegativePrePeak'
    p% = p% + 1
    EventTable(r%, p%).Time = t%
    EventTable(r%, p%).Event = -1
    EventTable(r%, p%).Counts = v%
    k% = 0
END IF
CASE 2                                    'PositivePrePeak
IF Counts% < v% THEN
    s% = 3                                'PositivePostPeak'
    k% = v%
    p% = p% + 1
    EventTable(r%, p%).Event = 2
    EventTable(r%, p%).Counts = v%
    EventTable(r%, p%).Time = t%
END IF
CASE -2                                    'NegativePrePeak
IF Counts% > LastValue% THEN
    s% = -3                                'NegativePostPeak'
    k% = v%
    p% = p% + 1
    EventTable(r%, p%).Event = -2
    EventTable(r%, p%).Counts = v%
    EventTable(r%, p%).Time = t%
END IF
CASE 3                                    'PositivePostPeak
IF Counts% < v% AND v% > k% THEN
    s% = 3                                'PositivePostPeak'
    k% = v%
    p% = p% + 1
    EventTable(r%, p%).Event = 2
    EventTable(r%, p%).Counts = v%
    EventTable(r%, p%).Time = t%
END IF
IF Counts% - BaseLine% < GCNoise% THEN
    s% = 1                                'BaseLine'
    p% = p% + 1
    EventTable(r%, p%).Event = 3
    EventTable(r%, p%).Time = Now%
    EventTable(r%, p%).Counts = Counts%
END IF
CASE -3                                    'NegativePostPeak
IF Counts% > v% AND v% < k% THEN
    s% = -3                                'NegativePostPeak'
    k% = v%
    p% = p% + 1
    EventTable(r%, p%).Event = -2
    EventTable(r%, p%).Counts = v%
    EventTable(r%, p%).Time = t%
END IF
IF Counts% - BaseLine% > -GCNoise% THEN
    s% = 1                                'BaseLine'
    p% = p% + 1
    EventTable(r%, p%).Event = -3
    EventTable(r%, p%).Time = Now%
    EventTable(r%, p%).Counts = Counts%
END IF
END SELECT
EventData(r%).Ptr = p%
EventData(r%).Status = s%

```

```

EventData(r%).Peak = k%
EventData(r%).LastTime = Now%
EventData(r%).LastValue = Counts%
END SUB
DEFINT A-Z
' ===== QuickSort =====
' =====
' QuickSort works by picking a random "pivot" element in SortArray, then
' moving every element that is bigger to one side of the pivot, and every
' element that is smaller to the other side. QuickSort is then called
' recursively with the two subdivisions created by the pivot. Once the
' number of elements in a subdivision reaches two, the recursive calls end
' and the array is sorted.
' =====
' =====
SUB GCEventTableQuickSort (Low, High, RunPtr%)
  IF Low < High THEN
    ' Only two elements in this subdivision; swap them if they are out of
    ' order, then end recursive calls:
    IF High - Low = 1 THEN
      IF EventTable(RunPtr%, Low).Time > EventTable(RunPtr%, High).Time THEN
        SWAP EventTable(RunPtr%, Low), EventTable(RunPtr%, High)
      END IF
    ELSE
      ' Pick a pivot element at random, then move it to the end:
      RandIndex = GCRandInt%(Low, High)
      SWAP EventTable(RunPtr%, High), EventTable(RunPtr%, RandIndex)
      Partition = EventTable(RunPtr%, High).Time
      DO
        ' Move in from both sides towards the pivot element:
        i = Low: j = High
        DO WHILE (i < j) AND (EventTable(RunPtr%, i).Time <= Partition)
          i = i + 1
        LOOP
        DO WHILE (j > i) AND (EventTable(RunPtr%, j).Time >= Partition)
          j = j - 1
        LOOP
        ' If we haven't reached the pivot element, it means that two
        ' elements on either side are out of order, so swap them:
        IF i < j THEN
          SWAP EventTable(RunPtr%, i), EventTable(RunPtr%, j)
        END IF
      LOOP WHILE i < j
      ' Move the pivot element back to its proper place in the array:
      SWAP EventTable(RunPtr%, i), EventTable(RunPtr%, High)
      ' Recursively call the QuickSort procedure (pass the smaller
      ' subdivision first to use less stack space):
      IF (i - Low) < (High - i) THEN
        GCEventTableQuickSort Low, i - 1, RunPtr%
        GCEventTableQuickSort i + 1, High, RunPtr%
      ELSE
        GCEventTableQuickSort i + 1, High, RunPtr%
        GCEventTableQuickSort Low, i - 1, RunPtr%
      END IF
    END IF
  END IF
END SUB
DEFSNG A-Z
SUB GCFindAnalMode (RunPtr%, AnalMode$)
IF RunTable(RunPtr%).Analyzed = 1 THEN
  AnalMode$ = "Analyze"
  EventData(RunPtr%).Ptr = 0
  EventData(RunPtr%).Status = 0
  EventData(RunPtr%).Peak = 0
  EventData(RunPtr%).LastValue = 0
  EventData(RunPtr%).LastTime = 0
ELSEIF RunTable(RunPtr%).Analyzed = 2 THEN

```

```

AnalMode$ = "Integrate"
FOR i% = 1 TO 20
    IntegTable(RunPtr%, i%).ElutTime = -1
    IntegTable(RunPtr%, i%).Counts = 0
    IntegTable(RunPtr%, i%).Component = -1
    IntegTable(RunPtr%, i%).Concentration = -1
NEXT
ELSEIF RunTable(RunPtr%).Analyzed = 3 THEN
    AnalMode$ = "IDPeak"
    FOR i% = 1 TO 20
        IntegTable(RunPtr%, i%).Component = -1
        IntegTable(RunPtr%, i%).Concentration = -1
    NEXT
ELSEIF RunTable(RunPtr%).Analyzed = 4 THEN
    AnalMode$ = "CalcComp"
    FOR i% = 1 TO 20
        IntegTable(RunPtr%, i%).Concentration = -1
    NEXT
ELSE
    AnalMode$ = "Complete"
END IF
END SUB
SUB GCFindBaseLine (BegOfRange%, EndOfRange%, BaseLine%)
FOR Ptr1% = BegOfRange% TO EndOfRange% - 11 STEP 15
    Ptr2% = Ptr1% + 5
    Ptr3% = Ptr1% + 10
    LargeStep% = Ptr1% 16
    SmallStep% = Ptr1% MOD 16
    CALL GetGC(LargeStep%, SmallStep%, ValvePos%, Count1%, ErrorCode1%)
    LargeStep% = Ptr2% 16
    SmallStep% = Ptr2% MOD 16
    CALL GetGC(LargeStep%, SmallStep%, ValvePos%, Count2%, ErrorCode2%)
    LargeStep% = Ptr3% 16
    SmallStep% = Ptr3% MOD 16
    CALL GetGC(LargeStep%, SmallStep%, ValvePos%, Count3%, ErrorCode3%)
    IF ErrorCode1% = 0 AND ErrorCode2% = 0 AND ErrorCode3% = 0 THEN
        Mean! = (Count1% + Count2% + Count3%) / 3!
        Sigma! = (Count1% - Mean!) * (Count1% - Mean!)
        Sigma! = Sigma! + (Count2% - Mean!) * (Count2% - Mean!)
        Sigma! = Sigma! + (Count3% - Mean!) * (Count3% - Mean!)
        Sigma! = SQR(Sigma! / 3)
        IF Sigma! < GCNoise% THEN
            BaseLine% = Mean!
            ErrorCode% = 0
            EXIT SUB
        END IF
    END IF
NEXT
Ptr% = (BegOfRange% + EndOfRange%) / 2
LargeStep% = Ptr% 16
SmallStep% = Ptr% MOD 16
CALL GetGC(LargeStep%, SmallStep%, ValvePos%, Count%, ErrorCode%)
BaseLine% = Count%
ErrorCode% = -1
END SUB
SUB GCFindBegOfRun (RunPointer%, BegOfRun%, ErrorCode%)
LargeStep% = RunPointer% 16
SmallStep% = RunPointer% MOD 16
CALL GetGC(LargeStep%, SmallStep%, SubValvePos%, Count%, ErrorCode%)
IF ErrorCode% <> 0 THEN
    ErrorCode% = 1
    EXIT SUB
END IF
FOR Ptr% = RunPointer% TO 1 STEP -1
    LargeStep% = Ptr% 16
    SmallStep% = Ptr% MOD 16
    CALL GetGC(LargeStep%, SmallStep%, ThisValvePos%, Count%, ErrorCode%)
    IF ErrorCode% <> 0 THEN

```

```

        ErrorCode% = 2
    EXIT SUB
END IF
ErrorCode% = -1
IF ThisValvePos% = LOAD AND SubValvePos% = INJECT THEN
    BegOfRun% = Ptr% + 1
    ErrorCode% = 0
    EXIT FOR
END IF
SubValvePos% = ThisValvePos%
NEXT
IF ErrorCode% <> 0 THEN BegOfRun% = 0
END SUB
SUB GCFindEndOfRun (RunPointer%, EndOfRun%, ErrorCode%)
LargeStep% = RunPointer% 16
SmallStep% = RunPointer% MOD 16
CALL GetGC(LargeStep%, SmallStep%, PrevValvePos%, Count%, ErrorCode%)
IF ErrorCode% <> 0 THEN
    ErrorCode% = 1
    EXIT SUB
END IF
ErrorCode% = -1
FOR Ptr% = RunPointer% TO RunPointer% + 4000
    LargeStep% = Ptr% 16
    SmallStep% = Ptr% MOD 16
    CALL GetGC(LargeStep%, SmallStep%, ThisValvePos%, Count%, ErrorCode%)
    IF ErrorCode% <> 0 THEN
        EndOfRun% = Ptr% - 1
        ErrorCode% = 2
        EXIT SUB
    END IF
    ErrorCode% = -1
    IF ThisValvePos% = INJECT AND PrevValvePos% = LOAD THEN
        EndOfRun% = Ptr% - 1
        ErrorCode% = 0
        EXIT FOR
    END IF
    PrevValvePos% = ThisValvePos%
NEXT
IF ErrorCode% <> 0 THEN EndOfRun% = RunPointer% + 4000
END SUB
SUB GCHighLevelPlot (RunPtr%, AnalMode$, ErrorCode%)
BegOfRun% = RunTable(RunPtr%).BeginRun
EndOfRun% = RunTable(RunPtr%).EndRun
'EndBaseLine% = RunTable(RunPtr%).BaseLineEnd
'BegBaseLine% = RunTable(RunPtr%).BaseLineBeg
'Drift! = (BegBaseLine% - EndBaseLine%) / (BegOfRun% - EndOfRun%)
FOR Now% = BegOfRun% TO EndOfRun%
    ' BaseLine% = BegBaseLine% + Drift! * (Now% - BegOfRun%)
    BaseLine% = GCBaseLine%(RunPtr%, Now%)
    LargeStep% = Now% 16
    SmallStep% = Now% MOD 16
    CALL GetGC(LargeStep%, SmallStep%, ValvePos%, Count%, ErrorCode%)
    IF ErrorCode% <> 0 THEN EXIT SUB
    IF AnalMode$ = "Analyze" THEN
        CALL GCEventId(Now%, Count%, BaseLine%, RunPtr%)
    END IF
    'Plot Data here
    FOR g% = 1 TO EventData(RunPtr%).Ptr
        IF EventTable(RunPtr%, g%).Time = Now% - 1 THEN
            LINE -STEP(0, .04), , , &HFFFF
            LINE -STEP(0, -.08), , , &HFFFF
            LINE -STEP(0, .04), , , &HFFFF
        END IF
    END IF
NEXT
LINE (Now%, 0) - (Now%, GCscaled!(Count%, BaseLine%))
IF ValvePos% = 0 THEN PSET (Now%, -1.05) ELSE PSET (Now%, -1.02)

```



```

PSET (Now$, GCScalcd!(Count%, BaseLine%))
NEXT
END SUB
SUB GCIntegInitial
'
' This routine takes care of initializing arrays appropriately
FOR i% = 0 TO 40
  RunTable(i%).BeginRun = -1
  RunTable(i%).EndRun = -1
  RunTable(i%).Analyzed = -1
  RunTable(i%).BaseLineBeg = 2048
  RunTable(i%).BaseLineEnd = 2048
NEXT
RunTable(0).BeginRun = 0
IncompRunPtr% = 0
SearchedToPtr% = 0
REDIM EventData(40) AS EventDataType
REDIM IntegTable(40, 20) AS IntegTableType
REDIM EventTable(40, 50) AS EventTableType
CompId(-1).CompName = "Unkn"
CompId(-1).CalibFactor = 67.7 / 14089!
CompId(1).CompName = "N2"
CompId(1).ElutTime = 4.982 * 60! * 18.2 / 16!
CompId(1).Counts = -13450
CompId(1).Range = 150000
CompId(1).CalibFactor = -67.7 / 14089!
CompId(2).CompName = "H2S"
CompId(2).ElutTime = -1! * 60! * 18.2 / 16!
CompId(2).Counts = 100
CompId(2).Range = 1000
CompId(2).CalibFactor = 67.7 / 14089!
CompId(3).CompName = "CO2"
CompId(3).ElutTime = 1.934 * 60! * 18.2 / 16!
CompId(3).Counts = 3725
CompId(3).Range = 5535
CompId(3).CalibFactor = 18! / 3725!
CompId(4).CompName = "CO"
CompId(4).ElutTime = 12.557 * 60! * 18.2 / 16!
CompId(4).Counts = -547
CompId(4).Range = 547
CompId(4).CalibFactor = -3! / 547!
CompId(5).CompName = "H2"
CompId(5).ElutTime = 2.989 * 60! * 18.2 / 16!
CompId(5).Counts = -6897
CompId(5).Range = 6897
CompId(5).CalibFactor = -4.9 / 6897!
CompId(6).CompName = "C1"
CompId(6).ElutTime = 7.033 * 60! * 18.2 / 16!
CompId(6).Counts = -2876
CompId(6).Range = 2876
CompId(6).CalibFactor = -5! / 2876!
CompId(7).CompName = "C2-"
CompId(7).ElutTime = -1! * 60! * 18.2 / 16!
CompId(7).Counts = 100!
CompId(7).Range = 100
CompId(7).CalibFactor = .4 / 200!
CompId(8).CompName = "C2"
CompId(8).ElutTime = 3.487 * 60! * 18.2 / 16!
CompId(9).Counts = 200
CompId(8).Range = 200
CompId(8).CalibFactor = .4 / 200!
CompId(9).CompName = "C3-"
CompId(9).ElutTime = -1 * 60! * 18.2 / 16!
CompId(9).Counts = 100
CompId(9).Range = 1000
CompId(9).CalibFactor = .4 / 200!
CompId(10).CompName = "C3"
CompId(10).ElutTime = 9! * 60! * 18.2 / 16!

```

```

CompId(10).Counts = 100
CompId(10).Range = 100
CompId(10).CalibFactor = .4 / 200
CompId(11).CompName = "C4-"
CompId(11).ElutTime = -1! * 60! * 18.2 / 16!
CompId(11).Counts = 100
CompId(11).Range = 100
CompId(11).CalibFactor = .4 / 200
CompId(12).CompName = "iC4"
CompId(12).ElutTime = -1! * 60! * 18.2 / 16!
CompId(12).Counts = 50
CompId(12).Range = 50
CompId(12).CalibFactor = .4 / 200
CompId(13).CompName = "nC4"
CompId(13).ElutTime = -1! * 60! * 18.2 / 16!
CompId(13).Counts = 40
CompId(13).Range = 40
CompId(13).CalibFactor = .4 / 200
CompId(14).CompName = "C4+"
CompId(14).ElutTime = -1! * 60! * 18.2 / 16!
CompId(14).Counts = 30
CompId(14).Range = 30
CompId(14).CalibFactor = .4 / 200
CompId(15).CompName = "H2O"
CompId(15).ElutTime = 15! * 60! * 18.2 / 16!
CompId(15).Counts = 200
CompId(15).Range = 1000
CompId(15).CalibFactor = 0!
CompId(16).CompName = "O2"
CompId(16).ElutTime = 3.839 * 60! * 18.2 / 16!
CompId(16).Counts = -982
CompId(16).Range = 10000
CompId(16).CalibFactor = -.9 / 982!
NumOfComponents% = 16
EventName$(-3) = "End of negative peak"
EventName$(-2) = "Extreme of negative peak"
EventName$(-1) = "Start of negative peak"
EventName$(0) = "Start of GC run"
EventName$(1) = "Start of positive peak"
EventName$(2) = "Extreme of positive peak"
EventName$(3) = "End of positive peak"
GCNoise% = 2
GCMinPeakSize% = 4
GCMinNumOfCount% = 50
GCMinPeakWidth% = 6
ElutWt! = 1!
CountWt! = .001
END SUB
SUB GCIntegrate (RunPtr%, GCMinPeakWidth%, GCMinNumOfCounts%, ErrorCode%)
'
' This routine interprets the event table to identify the peaks and
' then it calls a subroutine the carries out the actual integrations.
'
RunStart% = RunTable(RunPtr%).BeginRun
PeakPtr% = 0
FOR EvPtr% = 1 TO EventData(RunPtr%).Ptr
    SELECT CASE EventTable(RunPtr%, EvPtr%).Event
        CASE 0
            Status% = 0
            'Start of Run
        CASE 1
            Status% = 1
            'Start Positive Peak
        SELECT CASE Status%
            CASE 0, 3, -3
                Status% = 1
                PeakStart% = EventTable(RunPtr%, EvPtr%).Time
            CASE ELSE
                ErrorCode% = -1
                Status% = 3
        END SELECT
    END SELECT

```

```

CASE 2                                     'Max Positive Peak
  SELECT CASE Status%
CASE 1, 2
  PeakTime% = EventTable(RunPtr%, EvPtr%).Time
  Status% = 2
CASE ELSE
  ErrorCode% = -1
  Status% = 3
  END SELECT
CASE 3                                     'End of Positive Peak
  SELECT CASE Status%
CASE 2
  Status% = 3
  PeakStop% = EventTable(RunPtr%, EvPtr%).Time
  IF PeakStop% - PeakStart% > GCMInPeakWidth% THEN
    Counts% = GCPeakInteg%(RunPtr%, PeakStart%, PeakStop%)
    IF ABS(Counts%) > GCMInNumOfCounts% THEN
      PeakPtr% = PeakPtr% + 1
      IntegTable(RunPtr%, PeakPtr%).Counts = Counts%
      IntegTable(RunPtr%, PeakPtr%).ElutTime = PeakTime% - RunStart%
    END IF
  END IF
CASE ELSE
  Status% = 3
  END SELECT
CASE -1                                     'Start of Negative Peak
  SELECT CASE Status%
CASE 0, 3, -3
  Status% = -1
  PeakStart% = EventTable(RunPtr%, EvPtr%).Time
CASE ELSE
  ErrorCode% = -1
  Status% = -3
  END SELECT
CASE -2                                     'Max of Negative Peak
  SELECT CASE Status%
CASE -1, -2
  PeakTime% = EventTable(RunPtr%, EvPtr%).Time
  Status% = -2
CASE ELSE
  ErrorCode% = -1
  Status% = -3
  END SELECT
CASE -3                                     'End of Negative Peak
  SELECT CASE Status%
CASE -2
  Status% = -3
  PeakStop% = EventTable(RunPtr%, EvPtr%).Time
  IF PeakStop% - PeakStart% > GCMInPeakWidth% THEN
    Counts% = GCPeakInteg%(RunPtr%, PeakStart%, PeakStop%)
    IF ABS(Counts%) > GCMInNumOfCounts% THEN
      PeakPtr% = PeakPtr% + 1
      IntegTable(RunPtr%, PeakPtr%).Counts = Counts%
      IntegTable(RunPtr%, PeakPtr%).ElutTime = PeakTime% - RunStart%
    END IF
  END IF
CASE ELSE
  Status% = -3
  END SELECT
END SELECT
NEXT
RunTable(RunPtr%).PeakPtr% = PeakPtr%
END SUB
SUB GCIntegRunMenu (RunPtr%, exitcode%)
' $DYNAMIC
DIM short$(10), long$(10)
' $STATIC
CALL LotusMenuClear

```

```

name$ = "GC-Integrate-Runs"
short$(1) = "Run"
long$(1) = "Change the run being reviewed."
short$(2) = "Baseline"
long$(2) = "Adjust the baseline position."
short$(3) = "Events"
long$(3) = "Add, Delete, or Review events (start, end or elution time)."
short$(4) = "Peaks"
long$(4) = "Examine information about peaks after integration."
short$(5) = "Calibration"
long$(5) = "Examine or change calibration factors."
short$(6) = "Save"
long$(6) = "Save the analysis to disk"
short$(7) = "Quit"
long$(7) = "Return to the previous menu."
items% = 7
sens% = 0      ' 0 = Not case sensitive
CALL GraphicLotusMenu(name$, short$(), long$(), items%, choice%, sens%)
SELECT CASE choice%
CASE 0
    'Escape key pressed
    exitcode% = 1
CASE 1
    GLMMakeBorder "", 0
    LOCATE 2, 8
    INPUT "GC Run to review next: "; RunPtr%
    IF RunPtr% = 0 THEN RunPtr% = IncompRunPtr%
    exitcode% = 0
CASE 2
    CALL GCAdjustBaseLine(RunPtr%)
    exitcode% = 0
CASE 3
    CALL GCAdjustEvents(RunPtr%)
    exitcode% = 1
CASE 4
    CALL GCAdjustPeaks(RunPtr%)
    exitcode% = 0
CASE 5
    'CALL GCChangeCalibration
    exitcode% = 0
CASE 6
    GLMMakeBorder "GC Analysis - Save File", 0
    LOCATE 2, 8
    INPUT "Name of File in which to store analysis"; FileName$
    IF FileName$ = "" THEN FileName$ = "A:" + BaseName$ + "GCA"
    CALL GCIntegTableWrite(FileName$)
    exitcode% = 1
CASE 7
    exitcode% = -1
CASE ELSE
    exitcode% = 1
END SELECT
CLS 2
ERASE short$, long$
END SUB
SUB GCIntegTableLoad (FileName$)
g% = FREEFILE
OPEN FileName$ FOR INPUT AS g%
INPUT #g%, IncompRunPtr%, SearchedToPtr%
FOR RunPtr% = 0 TO IncompRunPtr%
    INPUT #g%, RunTable(RunPtr%).BeginRun
    INPUT #g%, RunTable(RunPtr%).EndRun
    INPUT #g%, RunTable(RunPtr%).Analyzed
    INPUT #g%, RunTable(RunPtr%).BaseLineBeg
    INPUT #g%, RunTable(RunPtr%).BaseLineEnd
    INPUT #g%, RunTable(RunPtr%).PeakPtr
NEXT
FOR RunPtr% = 0 TO IncompRunPtr%
    INPUT #g%, EventData(RunPtr%).Ptr

```

```

INPUT #g%, EventData(RunPtr%).Status
INPUT #g%, EventData(RunPtr%).Peak
INPUT #g%, EventData(RunPtr%).LastValue
INPUT #g%, EventData(RunPtr%).LastTime
NEXT
FOR RunPtr% = 0 TO IncompRunPtr%
  FOR EvPtr% = 1 TO EventData(RunPtr%).Ptr
    INPUT #g%, EventTable(RunPtr%, EvPtr%).Event
    INPUT #g%, EventTable(RunPtr%, EvPtr%).Counts
    INPUT #g%, EventTable(RunPtr%, EvPtr%).Time
  NEXT
NEXT
FOR RunPtr% = 0 TO IncompRunPtr%
  FOR PeakPtr% = 1 TO 20
    INPUT #g%, IntegTable(RunPtr%, PeakPtr%).ElutTime
    INPUT #g%, IntegTable(RunPtr%, PeakPtr%).Counts
    INPUT #g%, IntegTable(RunPtr%, PeakPtr%).Component
    INPUT #g%, IntegTable(RunPtr%, PeakPtr%).Concentration
  NEXT
NEXT
CLOSE (g%)
END SUB
SUB GCIntegTableWrite (FileName$)
g% = FREEFILE
IF UCASE$(FileName$) = "NUL" THEN EXIT SUB
OPEN FileName$ FOR OUTPUT AS g%
PRINT #g%, IncompRunPtr%, SearchedToPtr%
FOR RunPtr% = 0 TO IncompRunPtr%
  PRINT #g%, RunTable(RunPtr%).BeginRun,
  PRINT #g%, RunTable(RunPtr%).EndRun,
  PRINT #g%, RunTable(RunPtr%).Analyzed,
  PRINT #g%, RunTable(RunPtr%).BaseLineBeg,
  PRINT #g%, RunTable(RunPtr%).BaseLineEnd,
  PRINT #g%, RunTable(RunPtr%).PeakPtr
NEXT
FOR RunPtr% = 0 TO IncompRunPtr%
  PRINT #g%, EventData(RunPtr%).Ptr,
  PRINT #g%, EventData(RunPtr%).Status,
  PRINT #g%, EventData(RunPtr%).Peak,
  PRINT #g%, EventData(RunPtr%).LastValue,
  PRINT #g%, EventData(RunPtr%).LastTime
NEXT
FOR RunPtr% = 0 TO IncompRunPtr%
  FOR EvPtr% = 1 TO EventData(RunPtr%).Ptr
    PRINT #g%, EventTable(RunPtr%, EvPtr%).Event,
    PRINT #g%, EventTable(RunPtr%, EvPtr%).Counts,
    PRINT #g%, EventTable(RunPtr%, EvPtr%).Time
  NEXT
NEXT
FOR RunPtr% = 0 TO IncompRunPtr%
  FOR PeakPtr% = 1 TO 20
    PRINT #g%, IntegTable(RunPtr%, PeakPtr%).ElutTime,
    PRINT #g%, IntegTable(RunPtr%, PeakPtr%).Counts,
    PRINT #g%, IntegTable(RunPtr%, PeakPtr%).Component,
    PRINT #g%, IntegTable(RunPtr%, PeakPtr%).Concentration
  NEXT
NEXT
CLOSE (g%)
END SUB
FUNCTION GCInvScaled% (Now$, y!, RunPtr%)
Diff = y! / 16.667
IF Diff > .03 THEN
  Diff = (y! - .5) / .5155 + .03
ELSEIF Diff < -.03 THEN
  Diff = (y! + .5) / .5155 - .03
END IF
Count% = 2048! * Diff + GCBaseLine%(RunPtr%, Now$)
GCInvScaled% = Count%

```

```

END FUNCTION
SUB GCMainLoop
CALL GCIntegInitial
CALL GCStartUp
GLMMakeBorder "", 0
LOCATE 2, 8
INPUT "GC Run to review first: ", RunPtr%
PlotMode$ = "Yes"
DO
    GLMMakeBorder "Updating Run Table", 0
    LOCATE 2, 8
    PRINT "Please be patient. This may take a few minutes."
    CALL GCUpdateRunTable(RunPtr%)
    BegOfRun% = RunTable(RunPtr%).BeginRun
    EndOfRun% = RunTable(RunPtr%).EndRun
    IF EndOfRun% > BegOfRun% THEN
        'Make Graph Outline
        IF PlotMode$ = "Yes" THEN
            GCSsetUpGCScreen BegOfRun%, EndOfRun%, RunPtr%
        END IF
        CLS 2
        CALL GCMakeHeader(RunPtr%, 8, 58)
        CALL GCProcess(RunPtr%, PlotMode$, ErrorCode%)
    ELSEIF BegOfRun% >= 0 THEN
        EndOfRun% = SearchedToPtr%
        BaseLine% = 2047
        GCSsetUpGCScreen BegOfRun%, EndOfRun%, RunPtr%
        CALL GCMakeHeader(RunPtr%, 8, 58)
        CALL GCBasicPlot(BegOfRun%, EndOfRun%, BaseLine%)
    ELSE
        BegOfRun% = 1
        EndOfRun% = SearchedToPtr%
        BaseLine% = 2047
        GCSsetUpGCScreen BegOfRun%, EndOfRun%, RunPtr%
        CALL GCMakeHeader(RunPtr%, 8, 58)
        CALL GCBasicPlot(BegOfRun%, EndOfRun%, BaseLine%)
    END IF
DO
    AKey$ = INKEY$
    LOOP UNTIL AKey$ = "/" OR AKey$ = CHR$(13)
    CALL GCIntegRunMenu(RunPtr%, exitcode%)
    SELECT CASE exitcode%
        CASE 0
            PlotMode$ = "Yes"
        CASE 1
            PlotMode$ = "No"
        CASE ELSE
            PlotMode$ = "Yes"
    END SELECT
    LOOP UNTIL exitcode% = -1
    'PRINT FRE(-1), FRE(-2), FRE(4), FRE(" ")
    GLMMakeBorder "GC Analysis - Save File", 0
    LOCATE 2, 8
    INPUT "Name of File in which to store analysis"; FileName$
    IF FileName$ = "" THEN FileName$ = "A:" + BaseName$ + "GCA"
    CALL GCIntegTableWrite(FileName$)
    CLOSE
END SUB
SUB GCMakeGraphBasic (BegOfRun%, EndOfRun%)
BasicGraph.SSViewPrint.Initialize = FALSE
BasicGraph.SSViewPrint.TopLine = 1
BasicGraph.SSViewPrint.BottomLine = 4
BasicGraph.SSViewPrint.Columns = 80
BasicGraph.SSViewPrint.Lines = 25
BasicGraph.SSView.Initialize = FALSE
BasicGraph.SSView.Coordinates = ""
BasicGraph.SSView.x1 = 60
BasicGraph.SSView.y1 = 60

```

```

BasicGraph.SSView.x2 = 700
BasicGraph.SSView.y2 = 310
BasicGraph.SSView.ViewColor = -1
BasicGraph.SSView.Border = TRUE
BasicGraph.SSWindow.Initialize = FALSE
BasicGraph.SSWindow.Coordinates = ""
BasicGraph.SSWindow.x1 = BegOfRun&
BasicGraph.SSWindow.y1 = -1.06
BasicGraph.SSWindow.x2 = EndOfRun&
BasicGraph.SSWindow.y2 = 1.03
BasicGraph.SSScreenMode = 3
END SUB
SUB GCMakeHeader (RunPtr%, Tab1%, Tab2%)
LOCATE 1, Tab1%
PRINT ExpHeader(0).s
LOCATE 1, Tab2%
Month$ = RIGHT$("00" + RTRIM$(LTRIM$(STR$(ExpInfo.Month))), 2)
Day$ = RIGHT$("00" + RTRIM$(LTRIM$(STR$(ExpInfo.Day))), 2)
Year$ = RIGHT$(STR$(ExpInfo.Year), 2)
ExpDate$ = Month$ + "/" + Day$ + "/" + Year$
PRINT "Exp. Date: "; ExpDate$
LOCATE 2, Tab1%
PRINT ExpHeader(1).s
LOCATE 2, Tab2%
PRINT "Start Time: "; LEFT$(GCTime$(0), 8)
LOCATE 3, Tab1%
PRINT "Run Number: "; RunPtr%;
LOCATE 3, Tab2%
PRINT "Sample Time: "; LEFT$(GCTime$(RunTable(RunPtr%).BeginRun), 8)
END SUB
SUB GCMakeRESGraph (BegOfRun&, EndOfRun&)
GCMakeGraphBasic BegOfRun&, EndOfRun&
RightSideGraph = BasicGraph
RightSideGraph.SSView.x1 = 350
RightSideGraph.SSView.y1 = 60
RightSideGraph.SSView.x2 = 700
RightSideGraph.SSView.y2 = 325
END SUB
SUB GCMakeWholeScreenState (WholeScreen AS ScreenStateType)
WholeScreen.SSViewPrint.Initialize = FALSE
WholeScreen.SSViewPrint.TopLine = 1
WholeScreen.SSViewPrint.BottomLine = 25
WholeScreen.SSViewPrint.Columns = 80
WholeScreen.SSViewPrint.Lines = 25
WholeScreen.SSView.Initialize = FALSE
WholeScreen.SSView.Coordinates = "SCREEN"
WholeScreen.SSView.x1 = 0
WholeScreen.SSView.y1 = 0
WholeScreen.SSView.x2 = 719
WholeScreen.SSView.y2 = 347
WholeScreen.SSView.ViewColor = -1
WholeScreen.SSView.Border = FALSE
WholeScreen.SSWindow.Initialize = FALSE
WholeScreen.SSWindow.Coordinates = "SCREEN"
WholeScreen.SSWindow.x1 = 1
WholeScreen.SSWindow.y1 = 1
WholeScreen.SSWindow.x2 = 80 + 8! / 9!
WholeScreen.SSWindow.y2 = 25 + 11! / 14!
WholeScreen.SSScreenMode = 3
END SUB
SUB GCOrderEvents (RunPtr%)
GCEventTableQuickSort 1, EventData(RunPtr%).Ptr, RunPtr%
DO
  Exchanged% = FALSE
  FOR i% = 1 TO EventData(RunPtr%).Ptr - 1
    IF EventTable(RunPtr%, i%).Time = EventTable(RunPtr%, i% + 1).Time THEN
      IF ABS(EventTable(RunPtr%, i%).Event) <
        ABS(EventTable(RunPtr%, i% + 1).Event)

```

```

THEN
SWAP EventTable(RunPtr%, i%), EventTable(RunPtr%, i% + 1)
Exchanged% = TRUE
END IF
END IF
NEXT
LOOP UNTIL NOT Exchanged%
END SUB
SUB GCPeakID (RunPtr%, ErrorCode%)
DIM Like!(RunTable(RunPtr%).PeakPtr, NumOfComponents%)
FOR Peak% = 1 TO RunTable(RunPtr%).PeakPtr
ETP% = IntegTable(RunPtr%, Peak%).ElutTime
CTP% = IntegTable(RunPtr%, Peak%).Counts
FOR Comp% = 1 TO NumOfComponents%
ETC% = CompId(Comp%).ElutTime
CTC% = CompId(Comp%).Counts
RTC% = CompId(Comp%).Range
a! = SGN(CTP%) * SGN(CTC%)
b! = ElutWt! * ABS((ETP% - ETC%) / ETC%)
c! = CountWt! * ABS((CTP% - CTC%) / RTC%)
Like!(Peak%, Comp%) = a! / (1! + b! + c!)
NEXT
NEXT
FOR i% = 1 TO NumOfComponents%
CompId(i%).IDd = 0
NEXT
DO
LikeMax! = 0
FOR Peak% = 1 TO RunTable(RunPtr%).PeakPtr
IF IntegTable(RunPtr%, Peak%).Component = -1 THEN
FOR Comp% = 1 TO NumOfComponents%
IF CompId(Comp%).IDd = 0 THEN
IF Like!(Peak%, Comp%) > LikeMax! THEN
LikeMax! = Like!(Peak%, Comp%)
PeakMax% = Peak%
CompMax% = Comp%
END IF
END IF
END IF
NEXT
END IF
NEXT
IF LikeMax! > MinLike! THEN
CompId(CompMax%).IDd = -1
IntegTable(RunPtr%, PeakMax%).Component = CompMax%
END IF
LOOP WHILE LikeMax! > MinLike!
ErrorCode% = 0
END SUB
FUNCTION GCPeakInteg% (RunPtr%, PeakStart%, PeakStop%)
Accum% = 0
BegOfRun% = RunTable(RunPtr%).BeginRun
EndOfRun% = RunTable(RunPtr%).EndRun
EndBaseLine% = RunTable(RunPtr%).BaseLineEnd
BegBaseLine% = RunTable(RunPtr%).BaseLineBeg
Drift! = (BegBaseLine% - EndBaseLine%) / (BegOfRun% - EndOfRun%)
FOR Now% = PeakStart% TO PeakStop%
LargeStep% = Now% \ 16
SmallStep% = Now% MOD 16
CALL GetGC(LargeStep%, SmallStep%, ValvePos%, Count%, ErrorCode%)
BaseLine% = BegBaseLine% + Drift! * (Now% - BegOfRun%)
IF ErrorCode% <> 0 THEN
Count% = BaseLine%
END IF
Accum% = Accum% + Count% - BaseLine%
NEXT
GCPeakInteg = Accum%
END FUNCTION
SUB GCProcess (RunPtr%, PlotMode$, ErrorCode%)

```



```

CALL GCFindAnalMode(RunPtr%, AnalMode$)
IF PlotMode$ = "Yes" THEN
    CALL GCHighLevelPlot(RunPtr%, AnalMode$, ErrorCode%)
    IF AnalMode$ = "Analyze" AND ErrorCode% = 0 THEN
        RunTable(RunPtr%).Analyzed = 2
        CALL GCFindAnalMode(RunPtr%, AnalMode$)
    END IF
ELSEIF AnalMode$ = "Analyze" THEN
    CALL GCAnalyze(RunPtr%, ErrorCode%)
    IF ErrorCode% = 0 THEN
        RunTable(RunPtr%).Analyzed = 2
        CALL GCFindAnalMode(RunPtr%, AnalMode$)
    END IF
END IF
IF AnalMode$ = "Integrate" THEN
    CALL GCOrderEvents(RunPtr%)
    CALL GCIntegrate(RunPtr%, GCMinPeakWidth%, GCMinNumOfCounts%, ErrorCode%)
    IF ErrorCode% = 0 THEN
        RunTable(RunPtr%).Analyzed = 3
        CALL GCFindAnalMode(RunPtr%, AnalMode$)
    END IF
END IF
IF AnalMode$ = "IDPeak" THEN
    CALL GCPeakID(RunPtr%, ErrorCode%)
    IF ErrorCode% = 0 THEN
        RunTable(RunPtr%).Analyzed = 4
        CALL GCFindAnalMode(RunPtr%, AnalMode$)
    END IF
END IF
IF AnalMode$ = "CalcComp" THEN
    CALL GCCalcComp(RunPtr%, ErrorCode%)
    IF ErrorCode% = 0 THEN
        RunTable(RunPtr%).Analyzed = 5
        CALL GCFindAnalMode(RunPtr%, AnalMode$)
    END IF
END IF
END SUB
DEFINT A-Z
' ===== GCRandInt% =====
'
' Returns a random integer greater than or equal to the Lower parameter
' and less than or equal to the Upper parameter.
' =====
'
FUNCTION GCRandInt% (lower, Upper) STATIC
    GCRandInt% = INT(RND * (Upper - lower + 1)) + lower
END FUNCTION
DEFSNG A-Z
FUNCTION GCScaled! (Count%, BaseLine%)
    Diff = (Count% - BaseLine%) / 2048!
    IF Diff > .03 THEN
        ScaledDiff = .5 + .5155 * (Diff - .03)
    ELSEIF Diff < -.03 THEN
        ScaledDiff = -.5 + .5155 * (Diff + .03)
    ELSE
        ScaledDiff = 16.667 * Diff
    END IF
    GCScaled! = ScaledDiff
END FUNCTION
SUB GCSetUpGCScreen (BegOfRun$, EndOfRun$, RunPtr%)
    GCMakeWholeScreenState WholeScreen
    SetScreenState WholeScreen
    CLS
    LOCATE 24, 7
    PRINT "0"
    LOCATE 24, 40
    PRINT "Minutes"

```

```

LOCATE 24, 77
Min% = FIX((EndOfRun% - BegOfRun%) * 16 / (18.2 * 60!) + .5)
PRINT Min%
GCMakeGraphBasic BegOfRun%, EndOfRun%
SetScreenState BasicGraph
LINE (BegOfRun%, 0)-(EndOfRun%, 0), , , &H5555
PSET (BegOfRun%, 0)
FOR i! = BegOfRun% TO EndOfRun% STEP 68.25
    LINE (i!, -1.06)-(i!, -1.01)
NEXT
END SUB
SUB GCStartUp
CALL DLGGetBaseName(BaseName$, ExpInfo, ExpHeader())
' $DYNAMIC
DIM short$(10), long$(10)
' $STATIC
CLS
name$ = "Preliminary Menu"
short$(1) = "Analyze"
long$(1) = "Conduct a new analysis of the GC data"
short$(2) = "Review"
long$(2) = "Use an existing analysis of the data stored on disk"
items% = 2
sens% = 0 ' 0 = Not case sensitive
CALL GraphicLotusMenu(name$, short$(), long$(), items%, choice%, sens%)
SELECT CASE choice%
    CASE 0 'Escape key pressed
        STOP
    CASE 1
        'Nothing needs to be done
    CASE 2
        GLMMakeBorder "Load Analysis", 0
        LOCATE 2, 8
        INPUT "Name for data file containing analysis"; FileName$
        IF FileName$ = "" THEN FileName$ = "A:" + BaseName$ + ".GCA"
        CALL GCIntegTableLoad(FileName$)
    CASE ELSE
        'Nothing needs to be done
END SELECT
MakeAsTextScreenState
GCMakeWholeScreenState WholeScreen
END SUB
FUNCTION GCTime$ (Now%)
' This function returns the data time that corresponds to a GC Tock
'
STATIC InRegs AS RegType, OutRegs AS RegType, StartTicks%, Flag%
IF Flag% = 0 THEN
    Flag% = -1
    InRegs.ax = 3
    CALL INTERRUPT(&HA1, InRegs, OutRegs) 'Get starting time
    dx% = OutRegs.dx
    cx% = OutRegs.cx
    IF dx% < 0 THEN dx% = dx% + 65536
    IF cx% < 0 THEN cx% = cx% + 65536
    Hrs% = cx% \ 256
    Min% = cx% MOD 256
    Sec% = dx% \ 256
    Hdrs% = dx% MOD 256
    StartTicks% = Hrs% * 65536! + Min% * 1092! + Sec% * 18.2 + Hdrs% * 18.2 / 100!
END IF
Tick% = Now% * 16 + StartTicks%
Hrs% = FIX(Tick% / 65536#)
Tick% = Tick% - Hrs% * 65536#
Min% = FIX(Tick% / 1092.6667#)
Tick% = Tick% - Min% * 1092.6667#
Sec! = FIX(Tick% / .18204#) / 100#
Sec% = FIX(Sec!)

```

```

Hdr% = 100 * (Sec! - Sec%)
Hrs$ = RIGHT$(" " + STR$(Hrs%), 2)
Min$ = RIGHT$("00" + LTRIM$(STR$(Min%)), 2)
Sec$ = RIGHT$("00" + LTRIM$(STR$(Sec%)), 2)
Hdr$ = RIGHT$("00" + LTRIM$(STR$(Hdr%)), 2)
GCTime$ = Hrs$ + ":" + Min$ + ":" + Sec$ + "." + Hdr$
END FUNCTION
SUB GCUpdateRunTable (RunPtr%)
' This routine updates the run table listing for beginning of run and
' end of run for all runs up to RunPtr% or end of available data
DO UNTIL RunTable(RunPtr%).EndRun <> -1
  IF RunTable(IncompRunPtr%).BeginRun = -1 THEN
    CALL GCFindEndOfRun(SearchedToPtr%, EndOfRun%, ErrorCodes%)
    IF ErrorCodes% = 0 THEN
      SearchedToPtr% = EndOfRun%
      RunTable(IncompRunPtr%).BeginRun = EndOfRun% + 1
    ELSE
      SearchedToPtr% = EndOfRun%
      EXIT DO
    END IF
  END IF
  CALL GCFindEndOfRun(SearchedToPtr% + 1, EndOfRun%, ErrorCodes%)
  IF ErrorCodes% = 0 THEN
    SearchedToPtr% = EndOfRun%
    RunTable(IncompRunPtr%).EndRun = EndOfRun%
    BegOfRange% = RunTable(IncompRunPtr%).BeginRun
    EndOfRange% = BegOfRange% + 50
    CALL GCFindBaseLine(BegOfRange%, EndOfRange%, BaseLine%)
    RunTable(IncompRunPtr%).BaseLineBeg = BaseLine%
    EndOfRange% = RunTable(IncompRunPtr%).EndRun
    BegOfRange% = EndOfRange% - 50
    CALL GCFindBaseLine(BegOfRange%, EndOfRange%, BaseLine%)
    RunTable(IncompRunPtr%).BaseLineEnd = BaseLine%
    RunTable(IncompRunPtr%).Analyzed = 1
    IncompRunPtr% = IncompRunPtr% + 1
  ELSE
    SearchedToPtr% = EndOfRun%
    EXIT DO
  END IF
LOOP
END SUB
SUB GetGC (Row%, Channel%, Gain%, Value%, ErrorCodes%)
' This subroutine provides an interface to the assembly level
' GetGC routine that is loaded as a TSR (Terminate and Stay Resident)
' program before the QuickBASIC program is run.
DIM InRegs AS RegType, OutRegs AS RegType
InRegs.ax = Channel%
InRegs.bx = Row%
CALL INTERRUPT(&H5, InRegs, OutRegs)
Gain% = OutRegs.cx
Value% = OutRegs.bx
ErrorCodes% = OutRegs.ax
END SUB

```

B.6 QuickBASIC Program DLGRAPH

```

DECLARE SUB RenderFlowGraph (BegRow%, EndRow%, MinValPres!,
MaxValPres!, MinValFlow!, MaxValFlow!, ExitCode%)
DECLARE SUB DLGDisplay (BegRow%, EndRow%, ChanCode%, ExitCode%)
DECLARE SUB DLGTemp (Redraw%, ExitCode%)
DECLARE SUB DLGFlow (Redraw%, ExitCode%)
DECLARE SUB DLGHeader ()
REM $INCLUDE: 'ExpCmn.bi'
DECLARE SUB DLGTimeTicks (BegRow%, EndRow%, ScreenRow%, MinVal!,
MaxVal!)
DECLARE SUB DLStartUp ()

```

```

DECLARE SUB DLGetData (Row%, ChanCode%, DataCode%, RtnValue!,
ErrorCode%)
DECLARE FUNCTION EngUnitsTable% (ChannelCode%)
DECLARE FUNCTION DTime$ (Row%)
DECLARE FUNCTION ElapsedTime! (To!)
SCREEN 3
INPUT "Refresh Screen:(y,n): ", Refresh$
Refresh$ = LEFT$(UCASE$(Refresh$), 1)
IF Refresh$ = "Y" THEN
    RefreshRate! = 1!
END IF
CALL DLStartUp
DO
    CLS 0
    PRINT
    PRINT "DataLog Graph:"
    PRINT
    PRINT "Options:"
    PRINT "  Temperatures"
    PRINT "  Flows and pressures"
    PRINT "  Quit"
    PRINT
    PRINT "Select Option: ";
    DO
        aKey$ = INKEY$
        LOOP WHILE aKey$ = ""
        aKey$ = UCASE$(aKey$)
        SELECT CASE aKey$
            CASE "T"
                PRINT "Temperatures"
                DLGTemp 0, ExitCode%
                ActiveGraph$ = "Temp"
                IF UCASE$(CHR$(ExitCode%)) = "F" THEN
                    SCREEN 3, , 1, 1
                    ActiveGraph$ = "Flow"
                    ExitCode% = 0
                END IF
            CASE "F"
                PRINT "Flows and pressures"
                DLGFlow 0, ExitCode%
                ActiveGraph$ = "Flow"
                IF UCASE$(CHR$(ExitCode%)) = "T" THEN
                    SCREEN 3, , 0, 0
                    ActiveGraph$ = "Temp"
                    ExitCode% = 0
                END IF
            CASE "Q"
                EXIT DO
            CASE ELSE
                END SELECT
        RefreshTimer! = TIMER
    DO
        IF ElapsedTime!(RefreshTimer!) > RefreshRate! AND Refresh$ = "Y" THEN
            RefreshTimer! = TIMER
            IF ActiveGraph$ = "Flow" THEN
                DLGFlow -1, ExitCode%
                IF UCASE$(CHR$(ExitCode%)) = "T" THEN
                    SCREEN 3, , 0, 0
                    ActiveGraph$ = "Temp"
                    ExitCode% = 0
                END IF
            ELSEIF ActiveGraph$ = "Temp" THEN
                DLGTemp -1, ExitCode%
                IF UCASE$(CHR$(ExitCode%)) = "F" THEN
                    SCREEN 3, , 1, 1
                    ActiveGraph$ = "Flow"
                    ExitCode% = 0
                END IF
            END IF
        END IF
    END IF

```

```

        END IF
    END IF
    LOOP UNTIL LEN(INKEY$) > 0 OR ExitCode% <> 0
LOOP
END
SUB DLGDisplay (BegRow%, EndRow%, ChanCode%, ExitCode%)
FOR Row% = BegRow% TO EndRow%
    DataCode% = EngUnitsTable%(ChanCode%)
    CALL DLGetData(Row%, ChanCode%, DataCode%, RtnValue, ErrorCode%)
    IF ErrorCode% = 0 THEN
        PSET (Row%, RtnValue)
    END IF
    A$ = INKEY$
    IF A$ <> "" THEN
        ExitCode% = ASC(A$)
        EXIT SUB
    END IF
NEXT Row%
ExitCode% = 0
END SUB
SUB DLGFlow (Redraw%, ExitCode%) STATIC
IF Redraw% = 0 THEN
    IF EnteredAtLeastOnce% THEN
        CLS 0
        PRINT "Use previous values (y/n): ",
        DO
            UsePreviousValues$ = INKEY$
            LOOP WHILE LEN(UsePreviousValues$) = 0
            UsePreviousValues$ = UCASE$(UsePreviousValues$)
            CLS 0
        ELSE
            UsePreviousValues$ = "N"
        END IF
    ELSE
        IF EnteredAtLeastOnce% THEN
            UsePreviousValues$ = "Y"
        ELSE
            UsePreviousValues$ = "N"
        END IF
    END IF
    IF UsePreviousValues$ = "N" THEN
        CLS 0
        PRINT "Pressures and Flows"
        PRINT
        INPUT "Beginning Row: ", BegRow%
        INPUT "Ending Row: ", EndRow%
        INPUT "Minimum value for pressures, PSIG: ", MinValPres
        INPUT "Maximum value for pressures, PSIG: ", MaxValPres
        INPUT "Minimum value for flows, SCFH: ", MinValFlow
        INPUT "Maximum value for flows, SCFH: ", MaxValFlow
        EnteredAtLeastOnce% = -1
        CLS 0
    END IF
    RenderFlowGraph BegRow%, EndRow%, MinValPres, MaxValPres,
    MinValFlow, MaxValFlow, ExitCode%
END SUB
SUB DLGHeader
LOCATE 1, 1, 0
PRINT Heading1$;
LOCATE 1, 70, 0
IF Mode$ = "TAKE" THEN PRINT DATES$; ELSE PRINT ExpDates$;
LOCATE 2, 1, 0
PRINT Heading2$;
LOCATE 2, 70, 0
PRINT Mode$;
END SUB
SUB DLGTemp (Redraw%, ExitCode%) STATIC
IF Redraw% = 0 THEN

```

```

IF EnteredAtLeastOnce% THEN
  CLS 0
  PRINT "Use previous values (y/n): ",
  DO
    UsePreviousValues$ = INKEY$
    LOOP WHILE LEN(UsePreviousValues$) = 0
    CLS 0
    UsePreviousValues$ = UCASE$(UsePreviousValues$)
  ELSE
    UsePreviousValues$ = "N"
  END IF
ELSE
  IF EnteredAtLeastOnce% THEN
    UsePreviousValues$ = "Y"
  ELSE
    UsePreviousValues$ = "N"
  END IF
END IF
IF UsePreviousValues$ = "N" THEN
  CLS 0
  PRINT "Temperatures"
  PRINT
  INPUT "Beginning Row: ", BegRow%
  INPUT "Ending Row: ", EndRow%
  INPUT "Minimum Value: ", MinVal
  INPUT "Maximum Value: ", MaxVal
  INPUT "Display by Cross Sections or By Axial Slices (C/A): ", DispOpt$
  DispOpt$ = UCASE$(DispOpt$)
  EnteredAtLeastOnce% = -1
  CLS 0
END IF
SCREEN 3, , 0, 0
CALL DLGHeader
LOCATE 3, 1, 0
PRINT USING "#####"; MaxVal;
LOCATE 8, 1, 0
PRINT USING "#####"; MinVal;
LOCATE 9, 1, 0
PRINT USING "#####"; MaxVal;
LOCATE 13, 1, 0
PRINT USING "#####"; MinVal;
LOCATE 14, 1, C
PRINT USING "#####"; MaxVal;
LOCATE 18, 1, 0
PRINT USING "#####"; MinVal;
LOCATE 19, 1, 0
PRINT USING "#####"; MaxVal;
LOCATE 24, 1, 0
PRINT USING "#####"; MinVal;
FOR i% = 0 TO 3
  Delta% = (330! - 30!) / 4!
  VIEW (50, 32 + i% * Delta%)-(690, 30 + (i% + 1) * Delta%), , 1
  WINDOW (BegRow%, MinVal)-(EndRow%, MaxVal)
  FOR j% = 0 TO 3
    SELECT CASE i%
      CASE 0
        LOCATE 4, j% * 12 + 12, 0
      CASE 1
        LOCATE 9, j% * 12 + 12, 0
      CASE 2
        LOCATE 14, j% * 12 + 12, 0
      CASE 3
        LOCATE 20, j% * 12 + 12, 0
    END SELECT
    IF DispOpt$ = "C" THEN
      ChanCode% = 4 * i% + j% + 2000
    ELSE
      ChanCode% = 4 * j% + i% + 2000
    END IF
  NEXT j%
NEXT i%

```

```

    END IF
    PRINT USING "TC:##"; ChanCode% - 2000;
NEXT
FOR j% = 0 TO 3
    IF DispOpt$ = "C" THEN
        ChanCode% = 4 * i% + j% + 2000
    ELSE
        ChanCode% = 4 * j% + i% + 2000
    END IF
    CALL DLGDisplay(BegRow%, EndRow%, ChanCode%, ExitCode%)
    IF ExitCode% <> 0 THEN EXIT SUB
NEXT
NEXT
CALL DLGTimeTicks(BegRow%, EndRow%, 25, MinVal, MaxVal * 2)
END SUB
SUB DLGTimeTicks (BegRow%, EndRow%, ScreenRow%, MinVal, MaxVal)
SELECT CASE EndRow% - BegRow%
    CASE IS >= 1024
        LittleStep% = 256
    CASE 512 TO 1023
        LittleStep% = 128
    CASE 256 TO 511
        LittleStep% = 64
    CASE IS < 256
        LittleStep% = (EndRow% - BegRow%) / 4
END SELECT
BigStep% = 256
FOR i% = BegRow% TO EndRow% STEP BigStep%
    LINE (i%, MinVal)-(i%, MinVal + (MaxVal - MinVal) * .08)
NEXT
FOR i% = BegRow% TO EndRow% STEP LittleStep%
    LINE (i%, MinVal)-(i%, MinVal + (MaxVal - MinVal) * .04)
NEXT
IF (EndRow% - BegRow%) / LittleStep% > 6 THEN
    TimeLen% = 5
ELSE
    TimeLen% = 8
END IF
FOR i% = BegRow% TO EndRow% STEP LittleStep%
    A$ = LEFT$(DTime$(i%), TimeLen%)
    ScreenCol% = ((i% - BegRow%) / (EndRow% - BegRow%)) * 72 + 6 - LEN(A$) / 2
    IF ScreenCol% <= 0 THEN ScreenCol% = 1
    IF ScreenCol% + LEN(A$) > 80 THEN ScreenCol% = 80 - LEN(A$)
    LOCATE ScreenRow%, ScreenCol%, 0
    PRINT A$;
NEXT
END SUB
SUB RenderFlowGraph (BegRow%, EndRow%, MinValPres, MaxValPres, MinValFlow,
MaxValFlow, ExitCode%)
Delta% = (330! - 30!) / 2!
SCREEN 3, , 1, 1
CALL DLGHeader
LOCATE 3, 1, 0
PRINT USING "#####"; MaxValPres
LOCATE 8, 1, 0
PRINT "PSIG"
LOCATE 13, 1, 0
PRINT USING "#####"; MinValPres
VIEW (50, 30)-(690, 30 + Delta%), , 1
WINDOW (BegRow%, MinValPres)-(EndRow%, MaxValPres)
FOR ChanCode% = 1000 TO 1002
    CALL DLGDisplay(BegRow%, EndRow%, ChanCode%, ExitCode%)
    IF ExitCode% <> 0 THEN EXIT SUB
NEXT
LOCATE 14, 1, 0
PRINT USING "###.#"; MaxValFlow
LOCATE 19, 1, 0
PRINT "SCFH"

```

```

LOCATE 24, 1, 0
PRINT USING "###.#"; MinValFlow;
VIEW (50, 30 + Delta%)-(690, 30 + 2 * Delta%), , 1
WINDOW (BegRow%, MinValFlow)-(EndRow%, MaxValFlow)
CALL DLGTimeTicks(BegRow%, EndRow%, 25, MinValFlow, MaxValFlow)
FOR ChanCode% = 1004 TO 1005
    CALL DLGDisplay(BegRow%, EndRow%, ChanCode%, ExitCode%)
    IF ExitCode% <> 0 THEN EXIT SUB
NEXT
END SUB

```

B.7 8088 Assembly Language Program TAKEDATA

B.7.1 TAKEDATA.ASM

```

        TITLE      TakeData Version 5 - The Workhorse
; Revision History
; [1] RLD 12-21-87   Add Jump tables to simplify control structure.
; [2] RLD 12-22-87   Move Channel routines, TC routines, and GC
;                   routines to separate modules so that each can
;                   have separate fardata segments.
; [3] RLD 1-19-88    Fardata segments didn't work with QB3.0 or QB4.0
;                   So try to incorporate data arrays into code segments
;                   Note: As of 1-20-88, this seems to work fine
; [4] RLD 4-14-89    Add auxilliary functions to allow TakeData to
;                   operate synchronously as a background task
; [5] RLD 4-16-89    Adjust Timing, Convert to TSR
; [6] RLD 4-25-89    Add additional functions to return information
;                   need to store data to disk
; [7] RLD 4-26-89    Cause Attach Take Data to rezero timer.
        PAGE       ,132
        DOSSEG
        .MODEL      small
        EXTRN       ChStore:PROC, TCStore:PROC, ChStart:PROC, TCStart:PROC
        EXTRN       GCStore:PROC, GCStart:PROC, GetGC:PROC
        EXTRN       AttachGetChan:PROC, AttachGetTCCh:PROC, AttachGetGC:PROC
        EXTRN       ChClear:PROC, TCClear:PROC, GCClear:PROC
        EXTRN       ch_buf_size:ABS, gc_buf_size:ABS, tc_buf_size:ABS
base_adr EQU        1808                ; Base address of Lab Master Board
td_tm_buf_size EQU 800h
;res_size1 EQU      (ch_buf_size/8 + tc_buf_size/8 + gc_buf_size/8)
;res_size2 EQU      td_tm_buf_size/4
;                   ; space for buffers
res_size3 EQU       800h                ; space for code & misc data
;tot_size EQU       res_size1 + res_size2 + res_size3
        .STACK     100h
        .DATA
        PUBLIC     td_ct_hw
td_count DW         0                  ; Number of times interrupt has be called
td_ct_hw DW         0                  ; High word of td_count
td_start_date DD    0
td_start_time DD    0
td_status DB        0
        .FARDATA
td_tm_buf DW        td_tm_buf_size dup (0101h, 0101h) ; Code for data not taken
td_string DB        650 dup ('-')
        .CODE
;=====
; First Step is to install interrupts
;
Start:
        mov        ax,@data
        mov        ds,ax

```



```

        call    AttachGetChan
        call    AttachGetTCCh
        call    AttachGetGC
        call    AttachTakeData
;Terminate and stay resident:
        mov     dx, 800h
        mov     cl, 3
        mov     ax, ch_buf_size
        shr     ax, cl
        add     dx, ax
        mov     cl, 3
        mov     ax, tc_buf_size
        shr     ax, cl
        add     dx, ax
        mov     cl, 3
        mov     ax, gc_buf_size
        shr     ax, cl
        add     dx, ax
        mov     cl, 4
        mov     ax, td_tm_buf_size
        shr     ax, cl
        add     dx, ax
; At last we have the correct
; size of ram to reserve in dx
        mov     ax, 3100h
; Error Code = 00
        int     21h
; Transfer to DOS
=====
        PUBLIC  TakeData
TakeData PROC
        sti
        push    bp
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        push    ss
        push    ds
        push    es
        mov     ax, @data
        mov     ds, ax
; Load start of DGROUP
; Addressing established for MS conv.
        cmp     td_status, 0
        je      TD_0
; if zero then take data
; else immediately return
        jmp     td_ot_tb LABEL
        DW      TD_g
        DW      TD_h
        DW      TD_i
        DW      TD_j
        td_in_tb LABEL WORD
; Construct jump table:
        DW      td_b
        DW      td_c
        DW      td_d
        DW      td_e
TD_0:      mov     bp, td_count
; Retrieve value of counter
        mov     dx, base_adr + 4
; Select status port address
        mov     cx, 0FFFh
; Load max number of loops
        in      al, dx
; Read status port
        test    al, 100000000b
; Test bit 7
        jnz     TD_a1
; When done, continue
        loop    TD_a
; if even then loop, else continue
        jmp     TD_f
; If time out occurs, then skip
; recording data.
        TD_a1:    inc     dx
; Select data port address
        in      ax, dx
; Read both data bits in one instruction
; data in ax, tdcoun in bp
        mov     bx, bp
; Select branch
        and     bx, 3

```

```

        shl     bx,1
        jmp     td_in_tb[bx]      ; Jump to proper branch
TD_b:   call    ChStore           ; ax = data, bp = td_count
        jmp     TD_f
TD_c:   call    GCStore
        jmp     TD_f
TD_d:   call    TCStore
        jmp     TD_f
TD_e:   call    GCStore
        jmp     TD_f
;-----
TD_f:   add     bp,1              ; Increment counter
        adc     td_ct_hw,0        ; Increment high word of counter
        mov     td_count,bp      ; Store counter to memory also
        mov     bx,bp
        and     bx,07Fh          ; Is this 1 out of 128?
        jnz     TD_f2           ; If not, continue
        ; If yes, then
        mov     ax,@fardata      ; Establish addressing to data
        mov     es,ax
        Assume  es:@fardata
        mov     bx,bp            ; bp contains counter
        mov     si,td_ct_hw      ; get high word
        mov     ax,si            ; store td_ct_hw in reg
        ; shift over 8
        mov     bl,bh
        mov     bh,al
        and     bx,td_tm_buf_size-1 ; Implement circular buffer
        shl     bx,1             ; Adjust for double word
        shl     bx,1
        mov     td_tm_buf[bx][2],ax ; Store high word of count in array
        mov     td_tm_buf[bx][0],bp ; Store low word of count in array
        mov     bx,bp            ; restore count to bx
TD_f2:  and     bx,3
        shl     bx,1
        jmp     td_ot_tb[bx]     ; Jump to correct branch
;-----
TD_g:   call    ChStart
        jmp     TD_k             ; Now finish up
TD_h:   call    GCStart
        jmp     TD_k             ; Now finish up
TD_i:   call    TCStart
        jmp     TD_k             ; Now finish up
TD_j:   call    GCStart
        jmp     TD_k             ; Now finish up
;-----
TD_k:   pop     es
        pop     ds
        pop     ss
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     bp
        iret
TakeData ENDP
;=====
; This routine installs interrupt vector 1Ch to TakeData,
; Logs in starting date and time,
; and installs vector for A1h Retrieve Take Data Info
;
; CALL AttachTakeData
PUBLIC AttachTakeData
AttachTakeData PROC
        push    ds

```

```

mov     ax, @data
mov     ds, ax
; Log starting time and date using DOS function calls
mov     ax, 2A00h
int     21h
mov     WORD PTR td_start_date[2], cx
mov     WORD PTR td_start_date[0], dx
mov     ax, 2C00h
int     21h
mov     WORD PTR td_start_time[2], cx
mov     WORD PTR td_start_time[0], dx
mov     ax, 0
mov     td_count, ax
mov     td_ct_hw, ax
; Initialize LabMaster:
mov     al, 128                ; make control byte
mov     dx, base_adr+4        ; select control port
out     dx, al                ; write control byte
inc     dx                    ; select channel port
mov     al, 0                  ; get channel number
out     dx, al                ; write channel number
inc     dx                    ; select start port
out     dx, al                ; start conversion of data
mov     dx, base_adr+15       ; Point to pp control port
mov     al, 80h               ; 80h is all ports mode 0, with Output
out     dx, al                ; Write control byte for pp
; Install TakeData to Software Timer Tick using MS-DOS function:
mov     ax, cs
mov     ds, ax
mov     dx, OFFSET TakeData
mov     ax, 251Ch
int     21h
; Install Take Data Control to Int A0
mov     ax, cs
mov     ds, ax
mov     dx, OFFSET TDCControl
mov     ax, 25A0h
int     21h
; Install Get Take Data Info to Int A1
mov     ax, cs
mov     ds, ax
mov     dx, OFFSET GetTDInfo
mov     ax, 25A1h
int     21h
pop     ds
ret

```

AttachTakeData ENDP

=====

```

; Get Take Data Information (Int A1)
PUBLIC  GetTDInfo

```

```

GetTDInfo PROC
sti
mov     di, ax
mov     si, bx
mov     ax, @data
mov     ds, ax
mov     ax, @fardata
mov     es, ax
ASSUME  ES:@fardata
jmp     gtdi_a

```

```

; Jump Table for function selection:

```

gtdi_fn	tb LABEL	WORD	
DW	gtdi_b		; function 0 Get Data Time
DW	gtdi_c		; 1 Get Current Counts
DW	gtdi_d		; 2 Get Start Date
DW	gtdi_e		; 3 Get Start Time
DW	gtdi_f		; 4 Get Status

```

      DW      gtdi_g      ;      5 Get String
;-----
gtdi_a:  mov     bx, di
        cmp     bx, 5      ; Restrict to functions 1 to 4
        jnle    gtdi_a1
        shl     bx, 1
        jmp     gtdi_fn_tb[bx]
gtdi_a1: mov     ax, 2      ; Set error code to 2
        jmp     gtdi_exit
gtdi_b:  ;Function (0) Get Data Time (or Get td_buf Info [bx = 0] )
        mov     bx, si      ; Row is in bx
        cmp     bx, td_tm_buf_size-1 ; Test to see if it is in range
        jle     gtdi_b1
        mov     ax, bx      ; out of range return error code
        jmp     gtdi_exit   ; of out of range value
gtdi_b1: shl     bx, 1
        shl     bx, 1      ; Adjust for double words
        mov     dx, WORD PTR td_tm_buf[bx][2] ; Read Data high word
        mov     ah, dh      ; If data taken (dh = 0) => ax = 0
        sub     ah, ah      ; else (dh = 01h) => ax = 1
        mov     cx, WORD PTR td_tm_buf[bx][0] ; Read Data low word
        mov     si, td_tm_buf_size SHL 2 + 640 ; Buffer size in si
        mov     di, OFFSET td_tm_buf      ; Plus size of strings
        jmp     gtdi_exit
gtdi_c:  ; Function (1) Get Current Counts
        mov     dx, td_ct_hw
        mov     cx, td_count
        mov     ax, 0
        jmp     gtdi_exit
gtdi_d:  ; Function (2) Get Start Date
        mov     cx, WORD PTR td_start_date[2] ; year
        mov     dx, WORD PTR td_start_date[0] ; dh = month, dl = day
        mov     ax, 0      ; error code = 0
        jmp     gtdi_exit
gtdi_e:  ; Function (3) Get Start Time
        mov     cx, WORD PTR td_start_time[2] ; ch = hour, cl = min
        mov     dx, WORD PTR td_start_time[0] ; dh = sec, dl =
        ; hundredths of sec
        ; error code = 0
        mov     ax, 0
        jmp     gtdi_exit
gtdi_f:  ; Function (4) Get Status
        mov     bl, td_status
        sub     bh, bh
        jmp     gtdi_exit
gtdi_g:  ; Function (5) Get String
        mov     ax, es
        mov     ds, ax      ; ds now contains seg of string space
        mov     es, cx      ; es now contains seg of calling string
        mov     di, dx
        mov     bx, si
        mov     cl, 8
        shl     bx, cl      ; Adjust to point at 64 byte strings
        mov     si, OFFSET td_string ; Point to beginning of string space
        add     si, bx      ; Point to start of string
        mov     cx, 64
        cld
        rep     movsb
        jmp     gtdi_exit
gtdi_exit: ;Don't pop anything every register can have
        iret      ;data in it
GetTDInfo ENDP
;=====
PUBLIC   TDControl ; Install as Interrupt A0
TDControl PROC
        sti
        push    ds
        push    es

```

```

        push    di
        push    si
        mov     di, ax
        mov     si, bx
        mov     ax, @data
        mov     ds, ax
        mov     ax, @fardata
        mov     es, ax
        ASSUME  ES:@fardata
        jmp     tdc_a
;-----
;      Jump Table for function selection:
tdc_fn_tb LABEL WORD
        DW      tdc_b           ; function 0 Clear Data
        DW      tdc_c           ;          1 Load Counts
        DW      tdc_d           ;          2 Load Start Date
        DW      tdc_e           ;          3 Load Start Time
        DW      tdc_f           ;          4 Load Take Data Status
        DW      tdc_g           ;          5 Load String
;-----
tdc_a:   mov     bx, di
        cmp     bx, 6           ; Restrict to functions 0 to 6
        jnl     tdc_a1
        shl     bx, 1
        jmp     tdc_fn_tb[bx]
tdc_a1:  mov     ax, 2           ; Set error code to 2
        jmp     gtdi_exit
tdc_b:   ; Function (0)      Clear Data Buffers
        mov     td_status, OFFh ; Make sure Take Data is halted.
        mov     cx, td_tm_buf_size
        mov     ax, 0
        mov     di, OFFSET td_tm_buf
        cld
        rep     stosw
        mov     ax, 0
        mov     WORD PTR td_start_date[2], ax
        mov     WORD PTR td_start_date[0], ax
        mov     WORD PTR td_start_time[2], ax
        mov     WORD PTR td_start_time[0], ax
        call    ChClear
        call    GCClear
        call    TCClear
        jmp     tdc_exit
tdc_c:   ; Function (1)      Load Take Data Count
        mov     td_ct_hw, dx
        mov     td_count, cx
        jmp     tdc_exit
tdc_d:   ; Function (2)      Load Start Date
        mov     WORD PTR td_start_date[2], cx ; year
        mov     WORD PTR td_start_date[0], dx ; dh = month, dl = day
        mov     ax, 0                     ; error code = 0
        jmp     tdc_exit
tdc_e:   ; Function (3)      Load Start Time
        mov     WORD PTR td_start_time[2], cx ; ch = hour, cl = min
        mov     WORD PTR td_start_time[0], dx ; dh = sec,
                                                ; dl = hundredths of sec
                                                ; error code = 0
        mov     ax, 0
        jmp     tdc_exit
tdc_f:   ; Function (4)      Halt Take Data
        mov     bx, si
        mov     td_status, bl
        jmp     tdc_exit
tdc_g:   ; Function (5)      Get String
        mov     ds, cx           ; ds = seg of string to be filled
        mov     bx, si           ; Returns String Num to bx
        mov     cl, 6
        shl     bx, cl           ; BX points at 64 byte strings
        mov     di, OFFSET td_string ; beginning of string space

```

```

        add     di,bx                ; Point to start of string
        mov     si,dx
        mov     cx,64
        cld
        rep     movsb
        jmp     tdc_exit
tdc_exit:
        pop     si
        pop     di
        pop     es
        pop     ds
        iret
TDCControl ENDP
        END     Start

```

B.7.2 CHROUTIN.ASM

```

        TITLE   Channel Routines Version 1
; Revision History
; [1] RLD 12-22-87 Extracted from take data
; [2] RLD 1-19-88 Make ch_buf in code segment
; [3] RLD 1-20-88 Implement full buffer addressing
; [4] RLD 1-21-88 Implement adjustable gain
; [5] RLD 1-22-88 Implement adjustable filtering. Note: This is
;               adjustable only during assembly via change the
;               value of the ch_filter equate.
; [6] RLD 4-16-89 Turn into TSR that runs below QB program.
;               Readjust to avoid necessity of reprogramming timer
;               Return ch_buf to data segment
;               Make data extraction into an software interrupt
; [7] RLD 4-25-89 Change GetCh to allow retrieval of ch_buf addressing
;               and size information with a call of ax, bx = 0
; [8] RLD 4-26-89 Add ChClear to clear ch_buf
        PAGE    ,132
        DOSSEG
        .MODEL   small
        PUBLIC   ch_buf_size
base_adr    EQU    1808                ; Base address of Lab Master Board
ch_buf_size EQU    8000h              ; Max value = 8000h
ch_filter   EQU    2                  ; Range 0 - 4
ch_fltr_mul EQU    1 SHL ch_filter    ; 0=>1, 1=>2, 2=>4, 3=>8, 4=>16
ch_ov_fl    EQU    0666h*ch_fltr_mul ; 0=>666h, etc.
ch_un_fl    EQU    066h*ch_fltr_mul  ; 0=>66h, etc.

        .DATA
        EXTRN    td_ct_hw:WORD
ch_cv       DW     16 DUP (0)
ch_gn       DB     16 DUP (0)
ch_av       DW     16 DUP (0)
ch_gn_av    DB     16 DUP (0)
        .FARDATA
ch_buf      DW     ch_buf_size DUP (0)
        .CODE
;=====
; Procedure: ChStore
; Purpose:   This routine is called by TakeData. It stores the
;           normal channel data in a large array called Ch_Buf.
;           This data can then be accessed by a high level language
;           through a software interrupt
;
;           This routine is an integral part of TakeData.
;           Revision of this routine should be carefully coordinated with
;           that of TakeData.
;           On entry bp = td_count
;                   ax = data from LabMaster
;=====

```

```

ChStore PUBLIC ChStore
ChStore Proc
    mov     di, bp                ; Handle Normal channels here
    and     di, 00111100b        ; Identify Channel
    shr     di, 1                ; Now have Channel Number in di
    mov     si, di               ; di points for words
    shr     si, 1                ; si points for bytes
    mov     ch_cv[di], ax        ; Store current value
;-----
; Digital Filtering:
    mov     dx, ch_av[di]        ; Retrieve old avg
    mov     bx, dx
    mov     cl, ch_filter
    shr     bx, cl               ; Adjust it to 12 digits
    sub     ax, bx               ; Calculate correction to filter value
    add     dx, ax               ; ch_av new = ch_av old + new - ch_av/16
    jmp     CS_a                 ; skip over jump table
;-----
; Adjustable Gain:
cs_bg_tb LABEL WORD
    DW      CS_g                 ; Do nothing
    DW      CS_b                 ; Divide by 8
    DW      CS_b                 ; Divide by 8
    DW      CS_c                 ; Divide by 4
cs_sm_tb LABEL WORD
    DW      CS_e                 ; Multiply by 8
    DW      CS_e                 ; Multiply by 8
    DW      CS_f                 ; Multiply by 4
    DW      CS_g                 ; Do nothing
CS_a:     mov     bl, ch_gn[si]   ; Get gain for current channel
    sub     bh, bh
    shl     bx, 1
    cmp     dx, ch_ov_f1         ; Test for danger of overflow
    jna     CS_d                 ; If no then continue
    jmp     cs_bg_tb[bx]         ; Jump depending on gain
CS_b:     shr     dx, 1           ; Case 0,1
CS_c:     shr     dx, 1           ; Case 2
    shr     dx, 1
    shr     bx, 1
    dec     bx
    mov     ch_gn[si], bl        ; Set new value of gain
    shl     bx, 1
    jmp     CS_g
CS_d:     cmp     dx, ch_un_f1    ; Test for underflow
    jnb     CS_g                 ; If no continue
    jmp     cs_sm_tb[bx]         ; Jump depending on gain
CS_e:     shl     dx, 1           ; Case 1,2
CS_f:     shl     dx, 1           ; Case 3
    shl     dx, 1
    shr     bx, 1
    inc     bx
    mov     ch_gn[si], bl        ; Set new value of gain
    shl     bx, 1
;-----
; Store Data in Buffer:
CS_g:     mov     ch_av[di], dx   ; Store new filter value in static array
    mov     cl, ch_filter
    shr     dx, cl               ; Truncate to 12 significant digits
    mov     cl, 4
    ror     bx, cl               ; Move gain to high byte
    add     dx, bx               ; Insert gain into space
    or      dx, 1000h           ; Set flag bit to show data taken
    ; Calculate row number:
    mov     bx, bp               ; bp contains counter
    and     bx, 0FF00h           ; mask off low bits - branch, channel,
    ; iteration
    mov     si, td_ct_hw         ; get high word
    mov     cx, 4                ; Shift double word over

```

```

CS_h:  shr     si,1
       rcr     bx,1
       loop    CS_h
       and     bx,ch_buf_size-1    ; Implement circular buffer
       shl     bx,1                ; Adjust for words
       mov     ax,@fardata
       mov     es,ax
       ASSUME  es:@fardata
       mov     ch_buf[bx][di],dx ; Store data in buffer

```

```

CS_i:  ret
ChStore ENDP

```

```

=====
; Procedure: ChStart
; Purpose:
; PUBLIC    ChStart
ChStart PROC
       mov     bx,bp                ; Handle Normal channels here
       and     bx,00111100b        ; Identify Channel
       shr     bx,1
       shr     bx,1                ; bx points for bytes now
       mov     al,ch_gn[bx]        ; Get gain from array
       or      al,128              ; Make control byte
       mov     dx,base_adr+4       ; Select control port
       out     dx,al               ; Write control byte
       inc     dx                  ; Select channel port
       mov     al,bl               ; Get channel number
       out     dx,al               ; Write channel number
       inc     dx                  ; Select start port
       out     dx,al               ; Start conversion of data
       ret

```

```

ChStart ENDP

```

```

=====
; This routine is called by QuickBasic through a software interrupt
; to retrieve the previously collected data.

```

```

; in basic:
; call GetChan (itime%,channel%,gain%,value%,error%)
; GetChan > int A3h

```

```

; Int A3h

```

```

; -----
; Call with
; ax = channel%
; bx = time displacement
; -----

```

```

; Returns
; ax = ErrorCode    0 = no error
; bx = Counts
; cx = Gain Code 0 = Gain 1, 1 = Gain 10, 2 = Gain 100, 3 = Gain 500
; es = Segment of ch_buf
; si = size of ch_buf in bytes
; dx = Offset of ch_buf

```

```

; Nothing preserved.

```

```

PUBLIC    GetChan

```

```

GetChan PROC

```

```

       sti
       ; ax is channel num
       mov     di,ax                ; mov channel num to di
       and     di,0Fh              ; Restrict to 0 to 15
       shl     di,1                ; Adjust for words
       mov     ax,@data             ; Establish addressing
       mov     ds,ax
       ; Time displacement is in bx
       cmp     bx,(ch_buf_size/16)-1 ; Test to see if it is in range
       jnl     GtCh_b              ; if not in range return error code
       mov     cl,5

```



```

        shl     bx,cl                ; Adjust for 16 channels + words
        mov     ax,@fiardata
        mov     es,ax
        ASSUME  es:@fiardata
        mov     dx,ch_buf[bx][di]   ; Read Data
        mov     ax,dx
        and     ax,1000h             ; Test Data Read flag
        jz      GtCh_c
        mov     ax,dx
        and     ax,6000h             ; Separate out gain
        mov     cl,3
        rol     ax,cl
        mov     cx,ax               ; Return gain
        mov     bx,dx
        and     bx,0FFFh            ; Separate out data
        mov     ax,0                ; Return error code of zero
GtCh_a:  mov     si,(ch_buf_size-1) SHL 1 ; Info used for storing data
                                           ; to file
        mov     dx, OFFSET ch_buf   ;
        mov     iret
GtCh_b:  mov     ax,bx               ; Not in range, return error code
        jmp     GtCh_a
                                           ; = bad range
GtCh_c:  mov     ax,1               ; Return error code of 1
        jmp     GtCh_a
GetChan  ENDP
;=====
;      This routine installs interrupt vector A3h to GetChan
;
;      CALL AttachGetChan
;      PUBLIC  AttachGetChan
AttachGetChan PROC
        push    ds
        mov     ax, @data
        mov     ds, ax
;      Chain in GetChan using MS-DOS function:
        mov     ax, cs
        mov     ds, ax
        mov     dx, OFFSET GetChan
        mov     ax, 25A3h
        int     21h
        pop     ds
        ret
AttachGetChan ENDP
;=====
;      This routine uninstalls interrupt vector A3h to GetChan
;
;      CALL DetachGetChan
;      PUBLIC  DetachGetChan
DetachGetChan PROC
        push    ds
;      Remove GetChan using MS-DOS function:
        mov     dx, 0
        mov     ds, dx
        mov     ax, 25A3h
        int     21h
        pop     ds
        ret
DetachGetChan ENDP
;=====
;      Procedure is called from TDControl to clear ch_buf.
;      PUBLIC  ChClear
ChClear  PROC
        mov     ax, @fiardata       ; Establish addressing
        mov     es, ax
        ASSUME  es:@fiardata
        mov     cx,ch_buf_size      ; Zero buffer

```

```
                mov     ax,0
                mov     di, OFFSET ch_buf
                cld
                rep     stosw
ChClear:        ENDP
                END
```

B.7.3 TCRROUTIN.ASM

```

        TITLE      Thermocouple Routines Version 5
; Revision History
; [1] RLD 12-22-87  Extract Routines from TakeData
; [2] RLD  1-28-88  Update to same level as ChRoutines
; [3] RLD  2-29-88  Change tc_mux_ch because of hardware change
; [4] RLD  4-16-89  General revisions to convert to TSR for synchronous
;                   data acquisition in the background
; [5] RLD  4-25-89  Revised GetTCCh to allow extraction of tc_buf
;                   addressing and size info.
; [6] RLD  4-26-89  Added TCClear (used to clear tc_buf)
        PAGE      ,132
        DOSSEG
        .MODEL    small
        PUBLIC    tc_buf_size
base_adr    EQU    1808                ; Base address of Lab Master Board
tc_mux_ch   EQU    8                  ; A to D channel of TC mux box
tc_buf_size EQU    8000h
tc_filter   EQU    1
tc_fltr_mul EQU    1 SHL tc_filter
tc_ov_fl    EQU    1900*tc_fltr_mul
tc_un_fl    EQU    170*tc_fltr_mul

        .DATA
        EXTRN     td_ct_hw:WORD
tc_cv       DW     16 DUP (0)
tc_gn       DB     16 DUP (0)
tc_av       DW     16 DUP (0)
tc_gn_av    DB     16 DUP (0)
        .FARDATA
tc_buf       DW     tc_buf_size DUP (0)
        .CODE
;=====
; This routine is called by TakeData. It stores the tc mux
; channel data in a large array called tc_buf. This data
; then is accessed by a high level language through a software
; interrupt.
;
; This routine is an integral part of TakeData.
; Revision of this routine should be carefully coordinated with
; that of TakeData.
;
        PUBLIC    TCStore
TCStore     PROC
        mov       di,bp                ; Handle Normal channels here
        and       di,00111100b         ; Identify Channel
        shr       di,1                 ; Now have Channel Number in di
        mov       si,di                ; di points for words
        shr       si,1                 ; si points for bytes
        mov       tc_cv[di],ax         ; Store current value
;-----
; Digital Filtering:
        mov       dx,tc_av[di]         ; Retrieve old avg
        mov       bx,dx
        mov       cx,tc_filter
        shr       bx,cx                 ; Adjust it to 12 digits
        sub       ax,bx                ; Calculate correction to filter value
        add       dx,ax                 ; ch_av new = ch_av old + new - ch_av/16
        jmp       TCS_a
;-----
; Adjustable Gain:
tcs_bg_tb   LABEL    WORD
        DW        TCS_g                ; Do nothing
        DW        TCS_b                ; Divide by 8
        DW        TCS_b                ; Divide by 8
        DW        TCS_c                ; Divide by 4

```

```

tcs_sm_tb LABEL      WORD
                    DW      TCS_e      ; Multiply by 8
                    DW      TCS_e      ; Multiply by 8
                    DW      TCS_f      ; Multiply by 4
                    DW      TCS_g      ; Do nothing
TCS_a:  mov     bl,tc_gn[si]      ; Get gain for current channel
        sub     bh,bh
        shl     bx,1
        cmp     dx,tc_ov_fl      ; Test for danger of overflow
        jna     TCS_d            ; If no then continue
        jmp     tcs_bg_tb[bx]    ; Jump depending on gain
TCS_b:  shr     dx,1              ; Case 0,1
TCS_c:  shr     dx,1              ; Case 2
        shr     dx,1
        shr     bx,1
        dec     bx
        mov     tc_gn[si],bl     ; Set new value of gain
        shl     bx,1
        jmp     TCS_g
TCS_d:  cmp     dx,tc_un_fl      ; Test for underflow
        jnb     TCS_g            ; If no continue
        jmp     tcs_sm_tb[bx]    ; Jump depending on gain
TCS_e:  shl     dx,1              ; Case 1,2
TCS_f:  shl     dx,1              ; Case 3
        shr     bx,1
        inc     bx
        mov     tc_gn[si],bl     ; Set new value of gain
        shl     bx,1
;-----
; Store Data in Buffer:
TCS_g:  mov     tc_av[di],dx      ; Store new filter value in static array
        mov     cl,tc_filter
        shr     dx,cl            ; Truncate to 12 significant digits
        mov     cl,4
        ror     bx,cl            ; Move gain to high byte
        add     dx,bx            ; Insert gain into space
        or      dx,1000h        ; Set flag bit to show data taken
; Calculate row number:
        mov     bx,bp            ; bp contains counter
        and     bx,0FF00h        ; mask off low bits - branch,channel, ite
        mov     si,td_ct_hw      ; get high word
        mov     cx,4
        shr     si,1
        rcr     bx,1
        loop    TCS_h
        and     bx,tc_buf_size-1 ; Implement circular buffer
        shl     bx,1             ; Adjust for words
        mov     ax,@fardata
        mov     es,ax
        ASSUME es:@fardata
        mov     tc_buf[bx][di],dx ; Store data in buffer
TCS_i:  ret
TCStore ENDP
;-----
PUBLIC   TCStart
TCStart PROC
        mov     bx,bp            ; Handle TC Mux Channel now
        and     bx,00111100b    ; Identify TC Mux Channel
        shr     bx,1
        shr     bx,1
        mov     dx,base_adr+14  ; bx points for bytes now
        mov     al,bl            ; Point to pp control port
        out     dx,al            ; Load in TX mux address
        mov     cx,000Fh        ; Select TC mux channel
        loop    TCSt_a           ; Timing loop to allow
        mov     al,tc_gn[bx]    ; TC mux to settle
        or      al,128          ; Get gain from array
        ; Make control byte

```

```

mov     dx,base_adr+4      ; Select control port
out     dx,al              ; Write control byte
inc     dx                 ; Select channel port
mov     al,tc_mux_ch       ; Get channel number
out     dx,al              ; Write channel number
inc     dx                 ; Select start port
out     dx,al              ; Start conversion of data
ret
TCStart ENDP
=====
This routine is called by QuickBasic through a software interrupt
to retrieve the previously collected data.

in basic:
call GetTCCh (itime%,channel%,gain%,value%,error%)
GetTCCh > int A3h

Int A3h
-----
Call with
ax = channel%
bx = time displacement
-----
Returns
ax = ErrorCode    0 = no error
bx = Counts
cx = Gain Code 0 = Gain 1, 1 = Gain 10, 2 = Gain 100, 3 = Gain 500
es = Segment of ch_buf
dx = Offset of ch_buf
si = size of ch_buf_size

Nothing preserved
PUBLIC   GetTCCh
GetTCCh PROC
sti
mov     di,ax              ; ax is channel num
and     di,0Fh             ; mov channel num to di
shl     di,1              ; Restrict to 0 to 15
mov     ax,@data           ; Adjust for words
mov     ds,ax             ; Establish addressing

cmp     bx,(tc_buf_size/16)-1 ; Time displacement is in bx
jnl     GetTCCh_b         ; Test to see if it is in range
mov     cl,5              ; if not in range return error code
shl     bx,cl             ; Adjust for 16 channels + words
mov     ax,@fardata
mov     es,ax
ASSUME es:@fardata
mov     dx,tc_buf[bx][di] ; Read Data
mov     ax,dx
and     ax,1000h          ; Test Data Read flag
jz      GetTCCh_c
mov     ax,dx
and     ax,6000h          ; Separate out gain
mov     cl,3
rol     ax,cl
mov     cx,ax             ; Return gain
mov     bx,dx
and     bx,0FFFh          ; Separate out data
mov     ax,0              ; Return error code of zero

GetTCCh_a:
mov     si,(tc_buf_size - 1) SHL 1
mov     dx,OFFSET tc_buf
iret

GetTCCh_b: mov     ax,bx              ; Not in range, return error code

```

```

        jmp      GetTCCh_a
GetTCCh_c: mov     ax,1                ; = bad range
        jmp      GetTCCh_a           ;Return error code of 1
GetTCCh ENDP
;=====
;      This routine installs interrupt vector A4h to GetTCCh
;
;      CALL AttachGetTCCh
PUBLIC   AttachGetTCCh
AttachGetTCCh PROC
    push     ds
    mov      ax, @data
    mov      ds, ax
;      Chain in GetTCCh using MS-DOS function:
    mov      ax, cs
    mov      ds, ax
    mov      dx, OFFSET GetTCCh
    mov      ax, 25A4h
    int      21h
    pop      ds
    ret
AttachGetTCCh ENDP
;=====
;      This routine uninstalls interrupt vector A4h to GetTCCh
;
;      CALL DetachGetTCCh
PUBLIC   DetachGetTCCh
DetachGetTCCh PROC
    push     ds
;      Remove GetTCCh using MS-DOS function:
    mov      dx, 0
    mov      ds, dx
    mov      ax, 25A4h
    int      21h
    pop      ds
    ret
DetachGetTCCh ENDP
;=====
;      Procedure is called from TDControl to clear tc_buf.
PUBLIC   TCClear
TCClear PROC
    mov      ax, @fardata             ;Establish addressing
    mov      es, ax
    ASSUME   ES:@fardata
    mov      cx, tc_buf_size          ; Zero buffer
    mov      ax, 0
    mov      di, OFFSET tc_buf
    cld
    rep      stosw
TCClear ENDP
END

```

B.7.4 GCROUTIN.ASM

```

        TITLE    Gas Chromatograph Routines
;      Revision History
;      [1] RLD 12-22-87 Extract Routines from TakeData
;      [2] RLD 4-16-89 Implement GCStore, GetGC
;      [3] RLD 4-20-89 Revision to GCStart to add GC switch valve controller.
;      [4] RLD 4-25-89 Revision to GetGC to allow extraction of addressing
;                      and size info for gc_buf needed for storing to disk.
;      [5] RLD 4-26-89 Add GCClear to allow zeroing of gc_buf
;                      PAGE      ,132
;                      DOSSEG
;                      .MODEL    small

```

```

PUBLIC      gc_buf_size
base_adr    EQU    1808      ; Base address of Lab Master Board
gc_ch       EQU    15        ; A to D channel for Gas Chromatograph
gc_buf_size EQU    8000h
gc_filter   EQU    3
gc_fltr_mul EQU    1 SHL gc_filter
gc_inject   EQU    0FFh
gc_load     EQU    0

.DATA
EXTRN      td_ct_hw:WORD
gc_gn       DB      0
gc_cv       DW      0
gc_av       DW      0800h SHL gc_filter
gc_reload   DB      0
gc_valve_pos DB      gc_load
gc_switch   DW      0
gc_end_time DW      0
gc_counter  DW      0FFFFh

gc_buf DW    gc_buf_size DUP (0)

.CODE
PUBLIC      GCStore, GCStart
GCStore     PROC
    mov      gc_cv,ax      ; Store current value
;-----
; Digital Filtering:
    mov      dx,gc_av      ; Retrieve old avg
    mov      bx,dx
    mov      cl,gc_filter
    shr      bx,cl          ; Adjust it to 12 digits
    sub      ax,bx          ; Calculate correction to filter value
    add      dx,ax          ; gc_av new = gc_av old + new - ch_av/(2^gc_filter)
    mov      gc_av,dx      ; Store new filter value in static array
    mov      cl,gc_filter
    shr      dx,cl          ; Truncate to 12 significant digits
    mov      cl,4
    or       dx,1000h      ; Set flag bit to show data taken
    mov      bh,gc_valve_pos ; Retrieve valve pos
    and      bx,8000h      ; Mask to correct bits
    or       dx,bx         ; Store in data
; Calculate row number:
    mov      bx,bp         ; bp contains counter
    and      bx,0FFFFh     ; mask off low bits - branch, repts
    mov      si,td_ct_hw   ; get high word
    mov      cx,4          ; Shift double word over
GCS_h:      shr      si,1
    rcr      bx,1
    loop     GCS_h
    and      bx,gc_buf_size-1 ; Implement circular buffer
    shl      bx,1          ; Adjust for words
    mov      ax,@fardata
    mov      es,ax
    ASSUME   es:@fardata
    mov      gc_buf[bx],dx ; Store data in buffer
    ret
GCStore     ENDP
;-----
GCStart     PROC
    GC Sampling Valve controller:
    mov      cx,gc_counter
    cmp      cx,gc_end_time
    jb       GCStt_b        ; If counter < end time continue
    mov      al,gc_reload    ; else
    and      al,al
    jz       GCStt_a        ; if reload is no then jump
    mov      cx,0            ; else zero counter
    mov      dx,base_adr+12  ; address port A
    mov      al,gc_inject    ; new valve position = inject

```

```

        out      dx,al      ; switch valve position
        mov      gc_valve_pos, al ; record valve position
        jmp      GCStt_b    ; continue
GCStt_a: mov      cx,OFFFEh    ; Peg out counter
GCStt_b: mov      al,gc_valve_pos ; Retrieve valve pos
        cmp      al,gc_inject ; Is valve = inject?
        jnz      GCStt_c    ; If no then continue
        cmp      cx,gc_switch ; else Is counter < switch
        jb       GCStt_c    ; If counter < end time continue
        mov      dx,base_adr+12 ; address port A
        mov      al,gc_load    ; new valve position = load
        out      dx,al      ; switch valve position
        mov      gc_valve_pos, al ; record valve position
GCStt_c: inc      cx          ; increment counter
        mov      gc_counter,cx ; store it
; Trigger LabMaster:
        mov      al,gc_gn      ; Get gain from memory
        or       al,128        ; Make control byte
        mov      dx,base_adr+4 ; Select control port
        out      dx,al      ; Write control byte
        inc      dx          ; Select channel port
        mov      al,gc_ch      ; Get channel number
        out      dx,al      ; Write channel number
        inc      dx          ; Select start port
        out      dx,al      ; Start conversion of data
        ret
GCStart ENDP
=====
; This routine is called by QuickBasic through a software interrupt
; to control the GC switching valve cycle.
;
; Int A2h Control GC Sampling Valve Cycle
;-----
; Call with
; AX = 0 Don't start another run after this one
;      OFFh Do start another run after this one
; BX = End time of a run in GC counts : 9.1 counts / sec
; CX = Switch Time for run in GC counts
;-----
; Returns
; AX = 0 No error
; AX = ErrorCode:
;
PUBLIC CtrlGC
CtrlGC PROC
        push     ds
        push     ax
        mov      ax,@data
        mov      ds,ax
        pop      ax
        cli      ; Disable interrupts to prevent
        mov      gc_switch,cx ; inadvertently using an inconsistent
        mov      gc_end_time,bx ; set of values.
        mov      gc_reload,al
        sti
        sub      ax,ax ; This routine has no error detection
                        ; at this time.
        pop      ds
        iret
CtrlGC ENDP
=====
; This routine is called by QuickBasic through a software interrupt
; to retrieve the previously collected data.
;

```



```

in basic:
call GetGC (itime%,channel%,ValvePos%,value%,error%)
GetGC > int A5h

```

```

Int A5h

```

```

-----
Call with
ax = small steps, 1 sec intervals
bx = time displacement, 14 sec intervals
Note: This method is used to preserve symmetry with
other interrupt routines, although it does introduce some
overhead into the calling program
-----

```

```

Returns
ax = ErrorCode    0 = no error
bx = Counts
cx = Valve position,
es = SEGMENT of gc_buf
dx = OFFSET of gc_buf
si = size of gc_buf in bytes

```

```

Nothing Preserved

```

```

PUBLIC    GetGC

```

```

GetGC

```

```

PROC

```

```

sti

```

```

mov     di,ax                ; ax is small time displacement
and     di,0Fh               ; move to di
shl     di,1                 ; Restrict to 0 to 15
mov     ax,@data             ; Adjust for words
mov     ds,ax                ; Establish addressing

                                ; Time displacement is in bx
cmp     bx,(gc_buf_size/16)-1 ; Test to see if it is in range
jnl     GtGC_b               ; if not in range return error code
mov     cl,5
shl     bx,cl                ; Adjust for chunks of 16 words
mov     ax,@fardata
mov     es,ax
ASSUME  es:@fardata
mov     dx,gc_buf[bx][di]    ; Read Data
mov     ax,dx
and     ax,1000h              ; Test Data Read flag
jz      GtGC_c
mov     ax,dx
and     ax,8000h              ; Separate out valve position
mov     cl,3
rol     ax,cl
mov     cx,ax                ; Return valve position
mov     bx,dx
and     bx,0FFFh              ; Separate out data
mov     ax,0                  ; Return error code of zero

```

```

GtGC_a:

```

```

mov     dx, OFFSET gc_buf
mov     si, (gc_buf_size - 1) SHL 1

```

```

GtGC_b:

```

```

mov     ax,bx                ; Not in range, return error code
jmp     GtGC_a

```

```

GtGC_c:

```

```

mov     ax,1                  ; = bad range
jmp     GtGC_a                ; Return error code of 1

```

```

GetGC

```

```

ENDP

```

```

=====
This routine installs interrupt vector A5h to GetGC
and interrupt vector A2h to CtrlGC

```

```

CALL AttachGetGC
PUBLIC AttachGetGC

```

```

AttachGetGC PROC
    push    ds
    mov     ax, @data
    mov     ds, ax
;    Attach GetGC using MS-DOS function:
    mov     ax, cs
    mov     ds, ax
    mov     dx, OFFSET GetGC
    mov     ax, 25A5h
    int     21h
;    Attach CtrlGC using MS-DOS function:
    mov     ax, cs
    mov     ds, ax
    mov     dx, OFFSET CtrlGC
    mov     ax, 25A2h
    int     21h
    pop     ds
    ret
AttachGetGC ENDP
;=====
;    This routine uninstalls interrupt vector A5h to GetGC
;
;    CALL DetachGetGC
PUBLIC    DetachGetGC
DetachGetGC PROC
    push    ds
;    Remove GetGC using MS-DOS function:
    mov     dx, 0
    mov     ds, dx
    mov     ax, 25A5h
    int     21h
    pop     ds
    ret
DetachGetGC ENDP
;=====
;    Procedure is called from TDControl to clear gc_buf.
PUBLIC    GCClear
GCClear PROC
    mov     ax, @fardata      ;Establish addressing
    mov     es, ax
    ASSUME  ES:@fardata
    mov     cx, gc_buf_size   ; Zero buffer
    mov     ax, 0
    mov     di, OFFSET gc_buf
    cld
    rep     stosw
GCClear ENDP
END

```

APPENDIX C

DATA ANALYSIS PROGRAMS

C.1 Program FAT NIBBLE SHAPE ENTRY

```

' Language: QuickBasic for the MacIntosh, version 1.0 (Bad news!)
' Author: Rich Dobbs
' Written: 11/24 - 11/27
' Purpose: This program creates the data file that defines
' the shape of the channel and the "HotSpots"
' in the coal core for my experiments.
' This data file will be used by the flow calculation program.
'
' Features: This program is a Mouse and Menu driven program.
' It is fairly true to the MacIntosh User Interface
' although the dialog boxes and alerts have been implemented
' crudely.
'
' Specifics: The flow calculation will be done on a even mesh of 31 X 31 X 63
' (The mesh refinement scheme that I will use works best
' when the number of interior Nodes =  $(2^N - 1)$ ).
' The node spacing in dimensional units is 0.5 cm.
' Thus there are 60543 nodes. The permeability has to be
' defined at each node. The calculations will be done using
' a zoned media ( with Permeabilities constant in each material).
' Thus the material of each node must be specified.
' The data entry problem is massive. It is handled here
' using fat bit editing, a common approach to defining patterns
' on the MacIntosh personal computer. Several interesting
' twists arise in the program. First, instead of having the
' pattern be simply a bit pattern, it is a nibble pattern.
' A nibble is half a byte thus is 4 bits. This allows up to
' 16 different materials to be selected, although currently
' only 5 (Char, Shale, Void, Coal, and Concrete) are needed
' and implemented. To mark this nibble pattern, fill patterns
' are used instead of simply black or white fill. Secondly
' to conserve memory (not real needed) and disk space (very
' useful) for storing the pattern, the nibbles are packed
' into integers. Since this version of BASIC does not implement
' unsigned integers, various kludges had to be implemented.
' These kludges are confined to GetPerm and SetPerm.
'
' Glossary: HotSpots - Up to 16 specific points where flux calculations
' are requested.
' *****
' Version 1.11 Enhancements
'
' Date: 11/27
' Specifics:
' Implement Edit - Copy
' Implement Edit - Paste
' Implement Step Pointers
' *****
CLEAR ,1000000%
DIM SHARED FileNames$
DIM SHARED PermArray%(7,31,63)
DIM SHARED CurMatCode%, CurPattern%, ActiveView%, MouseMode%

```

```

DIM SHARED DefaultI%, DefaultJ%, DefaultK%, NCoorBias!
DIM SHARED HotSpots%(15,3)
DIM SHARED PatternCode%(15)
DIM SHARED Patterns%(64)
DIM SHARED DeltaX%, DeltaY%, rect%(3)
DIM SHARED ChangesSaved%, CopyMarkedK%
DIM SHARED True%, False%
'*****Start Up Code*****
False% = 0
True% = NOT False%
DeltaX% = 19
DeltaY% = 18
PatternCode%(0) = 14
PatternCode%(1) = 11 'Char
PatternCode%(2) = 10 'Shale
PatternCode%(3) = 15 'Void
PatternCode%(4) = 1 'Coal
PatternCode%(5) = 12 'Cement
CALL InitializePatterns
CALL InitializePermArray
CALL SetDefaultValues
CALL DefineMenu
'*****Main Program*****
WINDOW 5,FileName$, ,1
CALL ShowActiveView
InMainLoop% = True%
WHILE InMainLoop%
MenuO% = MENU(0)
MousO% = MOUSE(0)
KeyO$ = INKEY$
DialogO% = DIALOG(0)
IF KeyO$ <> "" THEN CALL HandleKey (KeyO$)
IF DialogO% <> 0 THEN CALL HandleDialog (DialogO%)
IF MousO% <> 0 THEN CALL HandleMouse (MousO%)
IF MenuO% <> 0 THEN CALL HandleMenu (MenuO%)
WEND
END
'*****Menuing Routines*****
SUB DefineMenu STATIC
MENU 1,0,1, "File"
MENU 1,1,1, "New"
MENU 1,2,1, "Open..." : CmdKey 1, 2, "O"
MENU 1,3,0, "-----"
MENU 1,4,1, "Save" : CmdKey 1,4, "S"
MENU 1,5,1, "Save as..."
MENU 1,6,0, "-----"
MENU 1,7, 1, "Quit" : CmdKey 1,7, "Q"
MENU 2, 0, 1, "Edit"
MENU 2,1, 0, "Copy" : CmdKey 2, 1, "C"
MENU 2, 2, 0, "Paste" : CmdKey 2, 2, "V"
MENU 3,0,1, "Material"
MENU 3,1,2, "Char"
MENU 3,2,1, "Shale"
MENU 3,3,1, "Void"
MENU 3,4,1, "Coal"
MENU 3,5,1, "Concrete"
MENU 4,0,1, "Window"
MENU 4,1,1, "W - E - T - B"
MENU 4,2,1, "E - W - T - B"
MENU 4,3,1, "S - N - W - E"
MENU 4,4,1, "S - N - T - B"
MENU 5,0, 1, "ViewPointers"
MENU 5,1,1, "West -> East = " + STR$(DefaultI%)
MENU 5,2,1, "Top -> Bottom = " + STR$(DefaultJ%)
MENU 5, 3, 1, "South -> North = " + STR$(DefaultK%)
MENU 6,0,1, "Tools"
MENU 6,1,1, "Mark Material"
MENU 6,2,1, "Mark Hotspots"

```

```

MENU 7,0,1, "Stop Pointer"
MENU 7, 1, 1, "East"
MENU 7, 2, 1, "Down"
MENU 7, 3, 1, "North"
MENU 7, 4, 1, "South"
END SUB
SUB HandleMenu (Menu0%) STATIC
SELECT CASE Menu0%
CASE 1 'File
CALL HandleFileMenu
CASE 2 'Edit
SELECT CASE MENU(1)
CASE 1 'Copy
CopyMarkedK% = DefaultK%
CASE 2 'Paste
IF CopyMarkedK% < 1 OR CopyMarkedK% > 63 THEN EXIT SUB
FOR i% = 1 TO 31
FOR j% = 1 TO 31
CALL GetPerm (i%, j%, CopyMarkedK%, Code%)
CALL SetPerm (i%, j%, DefaultK%, Code%)
NEXT
NEXT
CALL ShowActiveView
CASE ELSE
END SELECT
CASE 3 ' Current Material
MENU 3, CurMatCode%, 1
CurMatCode% = MENU(1)
CurPattern% = PatternCode%(CurMatCode%)
MENU 3, CurMatCode%, 2
CASE 4
CLS
MENU 4, ActiveView%, 1
ActiveView% = MENU(1)
MENU 4, ActiveView%, 2
CALL ShowActiveView
CASE 5
CALL HandleViewPointersMenu
CASE 6
MENU 6, MouseMode%, 1
MouseMode% = MENU(1)
MENU 6, MouseMode%, 2
IF MouseMode% = 1 THEN
CALL INITCURSOR
ELSE
ChangeCursor 3
END IF
CASE 7
CLS
SELECT CASE MENU(1)
CASE 1 'East
DefaultI% = DefaultI% + 1
IF DefaultI% = 32 THEN DefaultI% = 1
CASE 2 'Down
DefaultJ% = DefaultJ% + 1
IF DefaultJ% = 32 THEN DefaultJ% = 1
CASE 3 'North
DefaultK% = DefaultK% + 1
IF DefaultK% = 64 THEN DefaultK% = 1
CASE 4 'South
DefaultK% = DefaultK% - 1
IF DefaultK% = 0 THEN DefaultK% = 63
END SELECT
CALL ShowActiveView
CASE ELSE
END SELECT
MENU
END SUB

```

```

SUB HandleFileMenu STATIC
SELECT CASE MENU(1)
CASE 1 'File - New
CALL SaveChangesInquiry(Returncode%)
IF Returncode% = 0 THEN
CALL SetDefaultValues
CALL InitializePermArray
WINDOW 5,FileName$, ,1
CLS
CALL ShowActiveView
END IF
CASE 2 'File - Open
CALL SaveChangesInquiry (Returncode%)
IF Returncode% = 0 THEN
TrialFileName$ = FILES$(1,"RLD1")
IF TrialFileName$ <> "" THEN
FileName$ = TrialFileName$
CALL FileRead (Returncode%)
IF Returncode% = -1 THEN
WINDOW 4,,, -1
TS$ = "This file could not be read successfully."
rect%(0) = 20 : rect%(1) = 20
rect%(2) = 80 : rect%(3) = 180 : Just% = 1
TextBox TS$, rect%(0), Just%
BUTTON 1,1,"Clear Message",(50,100)-(150,120)
WHILE DIALOG(0) <> 1 : WEND
WINDOW CLOSE 4
FileName$ = "Untitled"
CALL SetDefaultValues
CALL InitializePermArray
END IF
END IF
WINDOW 5,FileName$, ,1
CLS
CALL ShowActiveView
END IF
CASE 4 'File - Save
CALL FileSave
ChangesSaved% = 0
CASE 5 ' File - Save As...
TrialFileName$ = FILES$(0,"Save data to disk as:")
IF TrialFileName$ <> "" THEN
FileName$ = TrialFileName$
CALL FileSave
ChangesSaved% = 0
END IF
WINDOW 5,FileName$, ,1
CLS
CALL ShowActiveView
CASE 7 '"Quit"
CALL SaveChangesInquiry (Returncode%)
IF Returncode% = 0 THEN
STOP
END IF
END SELECT
END SUB

SUB HandleViewPointersMenu STATIC
WINDOW 2,,(500,200 )-(800, 300),-2
SELECT CASE MENU(1)
CASE 1 'West East == i%
ExitCode% = -1
WHILE ExitCode% = -1
PRINT "Current value is East", DefaultI%
PRINT "Allowable Range: 1 - 31"
INPUT "Pointer for West - East Axis: ";i%
IF i% >= 1 AND i% <= 31 THEN
DefaultI% = i%
ExitCode% = 0

```

```

ELSE
BEEP
PRINT "You Dummy!"
END IF
WEND
ActiveView% = 4
CASE 2 'Top Bottom == j%
ExitCode% = -1
WHILE ExitCode% = -1
PRINT "Current value is Bottom", DefaultJ%
PRINT "Allowable Range: 1 - 31"
INPUT "Pointer for Top - Bottom Axis: "; j%
IF j% >= 1 AND j% <= 31 THEN
DefaultJ% = j%
ExitCode% = 0
ELSE
BEEP
PRINT "You Dummy!"
END IF
WEND
ActiveView% = 3
CASE 3 'South North == k%
ExitCode% = -1
WHILE ExitCode% = -1
PRINT "Current value is North", DefaultK%
PRINT "Allowable Range: 1 - 63"
INPUT "Pointer for South - North Axis"; k%
IF k% >= 1 AND k% <= 63 THEN
DefaultK% = k%
ExitCode% = 0
ELSE
BEEP
PRINT "You Dummy!"
END IF
WEND
ActiveView% = 1
END SELECT
WINDOW CLOSE 2
CLS
ShowActiveView
END SUB

*****Mousing Routines*****
SUB HandleMouse (Mous0%) STATIC
ChangesSaved% = -1
SELECT CASE MouseMode%
CASE 1 'Mark Materials
SELECT CASE ActiveView%
CASE 1 'WETB
CALL HandleMouseMarkMaterialsWETB
CASE 2 'EWTB
CALL HandleMouseMarkMaterialsEWTB
CASE 3 'SNWE
CALL HandleMouseMarkMaterialsSNWE
CASE 4 'SNTB
CALL HandleMouseMarkMaterialsSNTB
CASE ELSE
BEEP
END SELECT
CASE 2 'Mark HotSpots
SELECT CASE ActiveView%
CASE 1 'WETB
CALL HandleMouseMarkHotSpotsWETB (Mous0%)
CASE 2 'EWTB
CALL HandleMouseMarkHotSpotsEWTB (Mous0%)
CASE 3 'SNWE
CALL HandleMouseMarkHotSpotsSNWE (Mous0%)
CASE 4 'SNTB
CALL HandleMouseMarkHotSpotsSNTB (Mous0%)

```

```

CASE ELSE
BEEP
END SELECT
ShowActiveView
END SELECT
END SUB
'*****View Routines*****
'*****West - East - Top - Bottom *****
SUB HandleMouseMarkMaterialsWETB STATIC
WHILE MOUSE(0) = -1
x% = MOUSE(1)
y% = MOUSE(2)
CALL GetIndicesWETB (x%, y%, i%, j%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
CALL SetPerm (i%, j%, DefaultK%, CurMatCode%)
CALL IsHotSpotQuery (i%, j%, DefaultK%, HotSpotNum%)
IF HotSpotNum% = -1 THEN
CALL ToggleWETB (i%, j%, CurPattern%, -1)
ELSE
CALL ToggleWETB (i%, j%, CurPattern%, 0)
END IF
WEND
END SUB
SUB HandleMouseMarkHotSpotsWETB (Mous0%) STATIC
SELECT CASE Mous0%
CASE 1 'Single Click
x% = MOUSE(1)
y% = MOUSE(2)
CALL GetIndicesWETB (x%, y%, i%, j%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
CALL GetPerm (i%, j%, DefaultK%, SpotMatCode%)
SpotMatPattern% = PatternCode%(SpotMatCode%)
CALL IsHotSpotQuery (i%, j%, DefaultK%, HotSpotNum%)
IF HotSpotNum% = -1 THEN
CALL AddHotSpot (i%, j%, DefaultK%, Returncode%)
IF Returncode% = 0 THEN
CALL ToggleWETB (i%, j%, SpotMatPattern%, 0)
END IF
END IF
CASE 2 'DoubleClick
x% = MOUSE(1)
y% = MOUSE(2)
CALL GetIndicesWETB (x%, y%, i%, j%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
CALL GetPerm (i%, j%, DefaultK%, SpotMatCode%)
SpotMatPattern% = PatternCode%(SpotMatCode%)
CALL IsHotSpotQuery (i%, j%, DefaultK%, HotSpotNum%)
IF HotSpotNum% >= 0 THEN
CALL RemoveHotSpot (HotSpotNum%)
CALL ToggleWETB (i%, j%, SpotMatPattern%, -1)
END IF
CASE ELSE
'Do nothing
END SELECT
END SUB
SUB GetIndicesWETB(x%, y%, i%, j%, ErrorCode%) STATIC
'Warning: This must be kept compatible with ToggleWETB *****
j% = (y% - 75) \DeltaY%
i% = (x% - 75) \DeltaX%
IF i% < 1 OR i% > 31 OR j% < 1 OR j% > 31 THEN
ErrorCode% = -1
ELSE
ErrorCode% = 0
END IF
END SUB
SUB ToggleWETB (i%, j%, Code%, IsNotHotSpot%) STATIC
rect%(0) = 75 + j% * DeltaY%
rect%(1) = 75 + i% * DeltaX%

```



```

rect%(2) = rect%(0) + DeltaY%
rect%(3) = rect%(1) + DeltaX%
IF IsHotSpot% = -1 THEN
CALL FILLRECT(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
ELSE
CALL ERASERECT(VARPTR(rect%(0)))
CALL FILLOVAL(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
CALL FRAMEOVAL (VARPTR(rect%(0)))
END IF
END SUB
SUB ShowWETB (k%) STATIC
FOR j% = 1 TO 31
FOR i% = 1 TO 31
CALL GetPerm (i%, j%, k%, Code%)
CALL ToggleWETB (i%, j%, PatternCode%(Code%), -1)
NEXT
NEXT
FOR m% = 0 TO 15
IF HotSpots%(m%, 3) = k% AND HotSpots%(m%, 1) <> - 1 THEN
i% = HotSpots%(m%, 1)
j% = HotSpots%(m%, 2)
CALL GetPerm (i%, j%, k%, Code%)
CALL ToggleWETB (i%, j%, PatternCode%(Code%), 0)
END IF
NEXT
CALL MarkViewsWETB
CALL MakeLabelsWETB
CALL MakeLegend
END SUB
SUB MarkViewsWETB STATIC
'Draw Line along Default j - (X axis)
y% = 75 + DefaultJ% * DeltaY% + DeltaY% \2
LINE (75, y%) - (75 + 33 * DeltaX%, y%)
'Draw Line along Default i% - (Y axis)
x% = 75 + (DefaultI% ) * DeltaX% + DeltaX% \2
LINE (x%, 75) - (x%, 75 + 33 * DeltaY%)
END SUB
SUB MakeLabelsWETB STATIC
CALL TEXTFACE(1)
CALL TEXTSIZE (24)
LOCATE 2,2
TempDefaultK% = DefaultK%
CALL GetNCoor (TempDefaultK%, NCoor!)
PRINT USING "North Coordinate ###.## CM"; NCoor!
Just% = 0
TS$ = "West"
rec%(0) = 355 : rec%(1) = 5 : rec%(2) = 385 : rec%(3) = 75
TextBox TS$, rec%(0), Just%
TS$ = "East"
rec%(0) = 355 : rec%(1) = 685 : rec%(2) = 385 : rec%(3) = 750
TextBox TS$, rec%(0), Just%
TS$ = "Top"
rec%(0) = 60 : rec%(1) = 350 : rec%(2) = 80 : rec%(3) = 425
TextBox TS$, rec%(0), Just%
TS$ = "Bottom"
rec%(0) = 685 : rec%(1) = 320 : rec%(2) = 700 : rec%(3) = 460
TextBox TS$, rec%(0), Just%
END SUB
'***** East - West - Top Bottom *****
SUB HandleMouseMarkMaterialsEWTB STATIC
WHILE MOUSE(0) = -1
x% = MOUSE(1)
y% = MOUSE(2)
CALL GetIndicesEWTB (x%, y%, i%, j%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
CALL SetPerm (i%, j%, DefaultK%, CurMatCode%)
CALL IsHotSpotQuery (i%, j%, DefaultK%, HotSpotNum%)
IF HotSpotNum% = -1 THEN

```

```

CALL ToggleEWB (i%, j%, CurPattern%, -1)
ELSE
CALL ToggleEWB (i%, j%, CurPattern%, 0)
END IF
WEND
END SUB
SUB HandleMouseMarkHotSpotsEWB (MousO%) STATIC
SELECT CASE MousO%
CASE 1 'Single Click
x% = MOUSE(1)
y% = MOUSE(2)
CALL GetIndicesEWB (x%, y%, i%, j%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
CALL GetPerm (i%, j%, DefaultK%, SpotMatCode%)
SpotMatPattern% = PatternCode%(SpotMatCode%)
CALL IsHotSpotQuery (i%, j%, DefaultK%, HotSpotNum%)
IF HotSpotNum% = -1 THEN
CALL AddHotSpot (i%, j%, DefaultK%, Returncode%)
IF Returncode% = 0 THEN
CALL ToggleEWB (i%, j%, SpotMatPattern%, 0)
END IF
END IF
CASE 2 'DoubleClick
x% = MOUSE(1)
y% = MOUSE(2)
CALL GetIndicesEWB (x%, y%, i%, j%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
CALL GetPerm (i%, j%, DefaultK%, SpotMatCode%)
SpotMatPattern% = PatternCode%(SpotMatCode%)
CALL IsHotSpotQuery (i%, j%, DefaultK%, HotSpotNum%)
IF HotSpotNum% >= 0 THEN
CALL RemoveHotSpot (HotSpotNum%)
CALL ToggleEWB (i%, j%, SpotMatPattern%, -1)
END IF
CASE ELSE
'Do nothing
END SELECT
END SUB
SUB GetIndicesEWB(x%, y%, i%, j%, ErrorCode%) STATIC
'Warning: This must be kept compatible with ToggleEWB *****
j% = (y% - 75) \DeltaY%
i% = 31 - (x% - 75) \DeltaX%
IF i% < 1 OR i% > 31 OR j% < 1 OR j% > 31 THEN
ErrorCode% = -1
ELSE
ErrorCode% = 0
END IF
END SUB
SUB ToggleEWB (i%, j%, Code%, IsHotSpot%) STATIC
rect%(0) = 75 + j% * DeltaY%
rect%(1) = 75 + (31 - i%) * DeltaX%
rect%(2) = rect%(0) + DeltaY%
rect%(3) = rect%(1) + DeltaX%
IF IsHotSpot% = -1 THEN
CALL FILLRECT(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
ELSE
CALL ERASERECT(VARPTR(rect%(0)))
CALL FILLOVAL(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
CALL FRAMEOVAL (VARPTR(rect%(0)))
END IF
END SUB
SUB ShowEWB (k%) STATIC
FOR j% = 1 TO 31
FOR i% = 1 TO 31
CALL GetPerm (i%, j%, k%, Code%)
CALL ToggleEWB (i%, j%, PatternCode%(Code%), -1)
NEXT
NEXT

```

```

FOR m% = 0 TO 15
IF HotSpots%(m%, 3) = k% AND HotSpots%(m%, 1) <> - 1 THEN
i% = HotSpots%(m%, 1)
j% = HotSpots%(m%, 2)
CALL GetPerm (i%, j%, k%, Code%)
CALL ToggleEWTB (i%, j%, PatternCode%(Code%), 0)
END IF
NEXT
CALL MarkViewsEWTB
CALL MakeLabelsEWTB
CALL MakeLegend
END SUB
SUB MarkViewsEWTB STATIC
'Draw Line along Default j - (X axis)
y% = 75 + DefaultJ% * DeltaY% + DeltaY% \2
LINE (75-DeltaX%, y%) - (75 + 32 * DeltaX%, y%)
'Draw Line along Default i% - (YAxis)
x% = 75 + (31 - DefaultI%)* DeltaX% + DeltaX% \2
LINE (x%, 75) - (x%, 75 + 33 * DeltaY%)
END SUB
SUB MakeLabelsEWTB STATIC
CALL TEXTFACE(1)
CALL TEXTSIZE (24)
LOCATE 2,2
TempDefaultK% = DefaultK%
CALL GetNCoor (TempDefaultK%, NCoor!)
PRINT USING "North Coordinate ###.## CM"; NCoor!
Just% = 0
TS$ = "East"
rec%(0) = 355 : rec%(1) = 5 : rec%(2) = 385 : rec%(3) = 75
TextBox TS$, rec%(0), Just%
TS$ = "West"
rec%(0) = 355 : rec%(1) = 680 : rec%(2) = 385 : rec%(3) = 750
TextBox TS$, rec%(0), Just%
TS$ = "Top"
rec%(0) = 60 : rec%(1) = 350 : rec%(2) = 80 : rec%(3) = 425
TextBox TS$, rec%(0), Just%
TS$ = "Bottom"
rec%(0) = 665 : rec%(1) = 320 : rec%(2) = 700 : rec%(3) = 460
TextBox TS$, rec%(0), Just%
END SUB
'*****South North West East View*****
SUB HandleMouseMarkMaterialsSNWE STATIC
WHILE MOUSE(0) = -1
x% = MOUSE(1)
y% = MOUSE(2)
CALL GetIndicesSNWE (x%, y%, k%, i%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
CALL SetPerm (i%, DefaultJ%, k%, CurMatCode%)
CALL IsHotSpotQuery (i%, DefaultJ%, k%, HotSpotNum%)
IF HotSpotNum% = -1 THEN
CALL ToggleSNXX(k%, i%, CurPattern%, -1)
ELSE
CALL ToggleSNXX(k%, i%, CurPattern%, 0)
END IF
WEND
END SUB
SUB HandleMouseMarkHotSpotsSNWE (Mous0%) STATIC
SELECT CASE Mous0%
CASE 1 'Single Click
x% = MOUSE(1)
y% = MOUSE(2)
CALL GetIndicesSNWE (x%, y%, k%, i%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
CALL GetPerm (i%, DefaultJ%, k%, SpotMatCode%)
SpotMatPattern% = PatternCode%(SpotMatCode%)
CALL IsHotSpotQuery (i%, DefaultJ%, k%, HotSpotNum%)
IF HotSpotNum% = -1 THEN

```

```

CALL AddHotSpot (i%, DefaultJ%, k%, ReturnCode%)
IF ReturnCode% = 0 THEN
CALL ToggleSNXX (k%, i%, SpotMatPattern%, 0)
END IF
END IF
CASE 2 'DoubleClick
x% = MOUSE(1)
y% = MOUSE(2)
CALL GetIndicesSNWE (x%, y%, k%, i%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
CALL GetPerm (i%, DefaultJ%, k%, SpotMatCode%)
SpotMatPattern% = PatternCode%(SpotMatCode%)
CALL IsHotSpotQuery (i%, DefaultJ%, k%, HotSpotNum%)
IF HotSpotNum% >= 0 THEN
CALL RemoveHotSpot (HotSpotNum%)
CALL ToggleSNXX (k%, i%, SpotMatPattern%, -1)
END IF
CASE ELSE
'Do nothing
END SELECT
END SUB
SUB GetIndicesSNWE(x%, y%, k%, i%, ErrorCode%) STATIC
'Warning: This must be kept compatible with ToggleSNWE *****
i% = (y% - 75) \DeltaY%
k% = (x% - 10) \DeltaX%
IF k% < 1 OR k% > 63 OR i% < 1 OR i% > 31 THEN
ErrorCode% = -1
ELSE
ErrorCode% = 0
END IF
END SUB
SUB ToggleSNXX (x%, y%, Code%, IsNotHotSpot%) STATIC
rect%(0) = 75 + y% * DeltaY%
rect%(1) = 10 + x% * DeltaX%
rect%(2) = rect%(0) + DeltaY%
rect%(3) = rect%(1) + DeltaX%
IF IsNotHotSpot% = -1 THEN
CALL FILLRECT(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
ELSE
CALL ERASERECT(VARPTR(rect%(0)))
CALL FILLOVAL(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
CALL FRAMEOVAL (VARPTR(rect%(0)))
END IF
END SUB
SUB ShowsNWE (j%) STATIC
FOR k% = 1 TO 63
FOR i% = 1 TO 31
CALL GetPerm (i%, j%, k%, Code%)
CALL ToggleSNXX (k%, i%, PatternCode%(Code%), -1)
NEXT
NEXT
FOR m% = 0 TO 15
IF HotSpots%(m%, 2) = j% AND HotSpots%(m%, 1) <> - 1 THEN
i% = HotSpots%(m%, 1)
k% = HotSpots%(m%, 3)
CALL GetPerm (i%, j%, k%, Code%)
CALL ToggleSNXX (k%, i%, PatternCode%(Code%), 0)
END IF
NEXT
CALL MarkViewsSNWE
CALL MakeLabelsSNWE
CALL MakeLegend
END SUB
SUB MarkViewsSNWE STATIC
'Draw Line along Default I - (X axis)
y% = 75 + DefaultI% * DeltaY% + DeltaY% \2
LINE (10, y%) - (1240, y%)

```

```

'Draw Line along DefaultK% - (YAxis)
x% = 10 + DefaultK% * DeltaX% + DeltaX% \2
LINE (x%, 75) - (x%, 75 + 33 * DeltaY%)
END SUB
SUB GetTopCoor (j%, BotCoor!) STATIC
BotCoor! = -.5 *(j% - 16)
END SUB
SUB MakeLabelsSNWE STATIC
CALL TEXTFACE(1)
CALL TEXTSIZE (24)
LOCATE 2,4
CALL GetTopCoor (DefaultJ%, TopCoor!)
PRINT USING "Top Coordinate ###.## CM"; TopCoor!
Just% = 0
TS$ = "South"
rec%(0) = 325 : rec%(1) = 10 : rec%(2) = 500 : rec%(3) = 30
TextBox TS$, rec%(0), Just%
TS$ = "North"
rec%(0) = 325 : rec%(1) = 1235 : rec%(2) = 500 : rec%(3) = 1255
TextBox TS$, rec%(0), Just%
TS$ = "West"
rec%(0) = 60 : rec%(1) = 600 : rec%(2) = 80 : rec%(3) = 700
TextBox TS$, rec%(0), Just%
TS$ = "East"
rec%(0) = 665 : rec%(1) = 600 : rec%(2) = 700 : rec%(3) = 700
TextBox TS$, rec%(0), Just%
END SUB
'*****South North Top Bottom View *****
SUB HandleMouseMarkMaterialsSNTB STATIC
WHILE MOUSE(0) = -1
x% = MOUSE(1)
y% = MOUSE(2)
CALL GetIndicesSNTB (x%, y%, k%, j%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
CALL SetPerm (DefaultI%, j%, k%, CurMatCode%)
CALL IsHotSpotQuery (DefaultI%, j%, k%, HotSpotNum%)
IF HotSpotNum% = -1 THEN
CALL ToggleSNXX(k%, j%, CurPattern%, -1)
ELSE
CALL ToggleSNXX(k%, j%, CurPattern%, 0)
END IF
END IF
WEND
END SUB
SUB HandleMouseMarkHotSpotsSNTB (MousO%) STATIC
SELECT CASE MousO%
CASE 1 'Single Click
x% = MOUSE(1)
y% = MOUSE(2)
CALL GetIndicesSNTB (x%, y%, k%, j%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
CALL GetPerm (DefaultI%, j%, k%, SpotMatCode%)
SpotMatPattern% = PatternCode%(SpotMatCode%)
CALL IsHotSpotQuery (DefaultI%, j%, k%, HotSpotNum%)
IF HotSpotNum% = -1 THEN
CALL AddHotSpot (DefaultI%, j%, k%, Returncode%)
IF Returncode% = 0 THEN
CALL ToggleSNXX (k%, j%, SpotMatPattern%, 0)
END IF
END IF
CASE 2 'DoubleClick
x% = MOUSE(1)
y% = MOUSE(2)
CALL GetIndicesSNTB (x%, y%, k%, j%, ErrorCode%)
IF ErrorCode% <> 0 THEN EXIT SUB
CALL GetPerm (DefaultI%, j%, k%, SpotMatCode%)
SpotMatPattern% = PatternCode%(SpotMatCode%)
CALL IsHotSpotQuery (DefaultI%, j%, k%, HotSpotNum%)
IF HotSpotNum% >= 0 THEN

```

```

CALL RemoveHotSpot (HotSpotNum%).
CALL ToggleSNXX (k%, j%, SpotMatPattern%, -1)
END IF
CASE ELSE
'Do nothing
END SELECT
END SUB
SUB GetIndicesSNTB(x%, y%, k%, j%, ErrorCode%) STATIC
'Warning: This must be kept compatible with ToggleSNXX *****
j% = (y% - 75) \DeltaY%
k% = (x% - 10) \DeltaX%
IF k% < 1 OR k% > 63 OR j% < 1 OR j% > 31 THEN
ErrorCode% = -1
ELSE
ErrorCode% = 0
END IF
END SUB
SUB ShowSNTB (i%) STATIC
FOR k% = 1 TO 63
FOR j% = 1 TO 31
CALL GetPerm (i%, j%, k%, Code%)
CALL ToggleSNXX (k%, j%, PatternCode%(Code%), -1)
NEXT
NEXT
FOR m% = 0 TO 15
IF HotSpots%(m%, 1) = i% AND HotSpots%(m%, 1) <> - 1 THEN
k% = HotSpots%(m%, 3)
j% = HotSpots%(m%, 2)
CALL GetPerm (i%, j%, k%, Code%)
CALL ToggleSNXX (k%, j%, PatternCode%(Code%), 0)
END IF
NEXT
CALL MarkViewsSNTB
CALL MakeLabelsSNTB
CALL MakeLegend
END SUB
SUB MarkViewsSNTB STATIC
'Draw Line along DefaultJ% - (X axis)
y% = 75 + DefaultJ% * DeltaY% + DeltaY% \2
LINE (10, y%) - (1240, y%)
'Draw Line along DefaultK% - (YAxis)
x% = 10 + DefaultK% * DeltaX% + DeltaX% \2
LINE (x%, 75) - (x%, 75 + 33 * DeltaY%)
END SUB
SUB MakeLabelsSNTB STATIC
CALL TEXTFACE(1)
CALL TEXTSIZE (24)
LOCATE 2,4
CALL GetEastCoor (DefaultI%, EastCoor!)
PRINT USING "East Coordinate ###.## CM"; EastCoor!
Just% = 0
TS$ = "South"
rec%(0) = 325 : rec%(1) = 10 : rec%(2) = 500 : rec%(3) = 30
TextBox TS$, rec%(0), Just%
TS$ = "North"
rec%(0) = 325 : rec%(1) = 1235 : rec%(2) = 500 : rec%(3) = 1255
TextBox TS$, rec%(0), Just%
TS$ = "Top"
rec%(0) = 60 : rec%(1) = 600 : rec%(2) = 80 : rec%(3) = 700
TextBox TS$, rec%(0), Just%
TS$ = "Bottom"
rec%(0) = 665 : rec%(1) = 600 : rec%(2) = 700 : rec%(3) = 700
TextBox TS$, rec%(0), Just%
END SUB
SUB GetEastCoor (j%, EastCoor!) STATIC
EastCoor! = .5 *(j% - 16)
END SUB

```

```

'*****End of Code for Views *****
SUB MakeLegend STATIC
CALL TEXTFACE(1)
CALL TEXTSIZE (24)
Just% = 0
TS$ = "Char Shale Void Coal Con."
rec%(0) = 710
rec%(1) = 77
rec%(2) = 740
rec%(3) = 750
TextBox TS$, rec%(0), Just%
FOR i% = 1 TO 5
rect%(0) = 713
rect%(1) = 10 + 135 * i%
rect%(2) = rect%(0) + DeltaY%
rect%(3) = rect%(1) + DeltaX%
CALL FILLRECT(VARPTR(rect%(0)), VARPTR(Patterns%(PatternCode%(i%)*4)))
CALL FRAMERECT(VARPTR(rect%(0)))
NEXT
END SUB

'*****Lower Level Routines*****
SUB FileSave STATIC
'Note: Although this file is written sequentially,
'PermArray is stored as binary to
'reduce the size of the disk file.
CALL ChangeCursor (4)
OPEN FileName$ FOR OUTPUT AS #5 LEN = 32
MagicStatement$ = "This is a data file for the perm matrix RLD 11/26/89"
WRITE #5, MagicStatement$
WRITE #5, CurMatCode%, CurPattern%, ActiveView%, MouseMode%
WRITE #5, DefaultI%, DefaultJ%, DefaultK%, NCoorBias!
FOR m% = 0 TO 15
WRITE #5, HotSpots%(m%, 1), HotSpots%(m%, 2), HotSpots%(m, 3)
NEXT
FOR i% = 0 TO 7
FOR j% = 1 TO 31
FOR k% = 1 TO 63
PRINT #5, MKI$(PermArray%(i%, j%, k%));
NEXT
NEXT
NEXT
CLOSE #5
NAME FileName$ AS FileName$, "RLD1"
CALL INITCURSOR
END SUB

SUB FileRead (Returncode%) STATIC
Returncode% = -1
'Note: Although this file is read sequentially,
'PermArray is stored as binary to
'reduce the size of the disk file.
OPEN FileName$ FOR INPUT AS #5
MagicStatement$ = "This is a data file for the perm matrix RLD 11/26/89"
INPUT #5, MagicStatementCandidate$
IF MagicStatementCandidate$ <> MagicStatement$ THEN EXIT SUB
CALL ChangeCursor (4)
INPUT #5, CurMatCode%, CurPattern%, ActiveView%, MouseMode%
INPUT #5, DefaultI%, DefaultJ%, DefaultK%, NCoorBias!
FOR m% = 0 TO 15
INPUT #5, HotSpots%(m%, 1), HotSpots%(m%, 2), HotSpots%(m, 3)
NEXT
FOR i% = 0 TO 7
FOR j% = 1 TO 31
FOR k% = 1 TO 63
PermArray%(i%, j%, k%) = CVI(INPUT$(2, #5))
NEXT
NEXT
NEXT
CLOSE #5

```

```

CALL INITCURSOR
Returncode% = 0
END SUB
SUB SaveChangesInquiry (Returncode%) STATIC
IF ChangesSaved% = 0 THEN
Returncode% = 0
ELSE
WINDOW 3,,(400, 200)-(700, 330),-2
TS$ = "Changes to the active file have not been saved."
TS$ = TS$ + " Do you want TO SAVE changes before proceeding"
rect%(0) = 20 : rect%(1) = 20 : rect%(2) = 80 : rect%(3) = 280
Just% = 0
TextBox TS$, rect%(0), Just%
BUTTON 1, 1, "CANCEL", (20, 90) - (90, 110), 1
BUTTON 2, 1, "NO", (135, 90) - (185, 110), 1
BUTTON 3, 1, "YES", (230, 90) - (280, 110), 1
rect%(0) = 85 : rect%(1) = 225 : rect%(2) = 115 : rect%(3) = 285
CALL PENSIZE(3,3)
CALL FRAMEROUNRECT (VARPTR(rect%(0)),15,15)
CALL PENNORMAL
ExitCode% = -1
WHILE ExitCode% = -1
IF DIALOG(0) = 1 THEN
Selector% = DIALOG(1)
ExitCode% = 0
ELSEIF INKEY$ = CHR$(13) THEN
Selector% = 3
ExitCode% = 0
END IF
WEND
WINDOW CLOSE 3
SELECT CASE Selector%
CASE 1
WINDOW CLOSE 3
CALL ShowActiveView
Returncode% = -1
CASE 2
Returncode% = 0
CASE 3
CALL FileSave
ChangesSaved% = 0
Returncode% = 0
END SELECT
END IF
END SUB
SUB ShowActiveView STATIC
MENU 4, 1, 1
MENU 4, 2, 1
MENU 4, 3, 1
MENU 4, 4, 1
MENU 4, ActiveView%, 2
MENU 6, 1, 1
MENU 6, 2, 1
MENU 6, MouseMode%, 2
MENU 5,1,1,"West -> East = " + STR$(DefaultI%)
MENU 5,2,1,"Top -> Bottom = " + STR$(DefaultJ%)
MENU 5, 3, 1, "South -> North = " + STR$(DefaultK%)
SELECT CASE ActiveView%
CASE 1 ' W - E - T - B
MENU 2, 1, 1
MENU 2, 2, 1
CALL ShowWETB (DefaultK%)
CASE 2
CALL ShowEWTB (DefaultK%)
MENU 2,1,1
MENU 2, 2, 1
CASE 3
CALL ShowSNWE (DefaultJ%)

```



```

MENU 2, 1, 0
MENU 2, 2, 0
CASE 4
CALL ShowSMTB (DefaultI%)
MENU 2, 1, 0
MENU 2, 2, 0
END SELECT
END SUB
SUB HandleKey (KeyO$) STATIC
SELECT CASE KeyO$
CASE "1"
MENU 3, CurMatCode%, 1
CurMatCode% = 1
CurPattern% = PatternCode%(CurMatCode%)
MENU 3, CurMatCode%, 2
CASE "2"
MENU 3, CurMatCode%, 1
CurMatCode% = 2
CurPattern% = PatternCode%(CurMatCode%)
MENU 3, CurMatCode%, 2
CASE "3"
MENU 3, CurMatCode%, 1
CurMatCode% = 3
CurPattern% = PatternCode%(CurMatCode%)
MENU 3, CurMatCode%, 2
CASE "4"
MENU 3, CurMatCode%, 1
CurMatCode% = 4
CurPattern% = PatternCode%(CurMatCode%)
MENU 3, CurMatCode%, 2
CASE "5"
MENU 3, CurMatCode%, 1
CurMatCode% = 5
CurPattern% = PatternCode%(CurMatCode%)
MENU 3, CurMatCode%, 2
CASE "6"
ActiveView% = 1
ShowActiveView
CASE "7"
ActiveView% = 2
ShowActiveView
CASE "8"
ActiveView% = 3
ShowActiveView
CASE "9"
ActiveView% = 4
ShowActiveView
CASE ELSE
SELECT CASE ASC(KeyO$)
CASE 43, 29, 30
inc% = 1
HandleKeyMove inc%
CASE 45, 28, 31
inc% = -1
HandleKeyMove inc%
CASE ELSE
BEEP
LOCATE 1,1
PRINT " Length = "; LEN(KeyO$), " ASCII = "; ASC(KeyO$),
" String = "; KeyO$
inc% = 0
END SELECT
END SELECT
END SUB
SUB HandleKeyMove (inc%) STATIC
SELECT CASE ActiveView%
CASE 1, 2 ' W - E - T- B & E - W - T- B
DefaultK% = DefaultK% + inc%

```

```

IF DefaultK% > 63 THEN
DefaultK% = 1
ELSEIF DefaultK% < 1 THEN
DefaultK% = 63
END IF
CASE 3 'S - N - W - E
DefaultJ% = DefaultJ% + inc%
IF DefaultJ% > 31 THEN
DefaultJ% = 1
ELSEIF DefaultJ% < 1 THEN
DefaultJ% = 31
END IF
CASE 4 'S - N - T - B
DefaultI% = DefaultI% + inc%
IF DefaultI% > 31 THEN
DefaultI% = 1
ELSEIF DefaultI% < 1 THEN
DefaultI% = 31
END IF
END SELECT
ShowActiveView
END SUB
SUB HandleDialog (DialogO%) STATIC
END SUB
SUB SetDefaultValues STATIC
'Defaults
FileName$ = "Untitled"
ChangesSaved% = 0 'True
DefaultI% = 16
DefaultJ% = 16
DefaultK% = 1
CurMatCode% = 1
CurPattern% = PatternCode%(CurMatCode%)
ActiveView% = 1
MouseMode% = 1
FOR m% = 0 TO 15
HotSpots%(m%, 1) = -1
NEXT
CALL PromptForNCoorBias
END SUB
SUB PromptForNCoorBias STATIC
WINDOW 6,,(300,200)-(600,340),-2
ExitCode% = -1
WHILE ExitCode% = -1
PRINT
INPUT "Enter the starting North Coordinate: ", Trial!
IF Trial! < -10! OR Trial! > 10! THEN
PRINT "This value doesn't seem correct."
INPUT "Are you sure? ", a$
a$ = UCASE$(a$)
IF a$ = "Y" THEN ExitCode% = 0
ELSE
ExitCode% = 0
END IF
WEND
NCoorBias! = Trial!
WINDOW CLOSE 6
CALL SHOWCURSOR
END SUB
SUB GetNCoor (TempK%, NCoor!) STATIC
NCoor! = (TempK% - 1)/2! + NCoorBias!
END SUB
SUB IsHotSpotQuery (i%, j%, k%, HotSpotNum%) STATIC
HotSpotNum% = -1
FOR m% = 0 TO 15
IF i% = HotSpots%(m%, 1) THEN
IF j% = HotSpots%(m%, 2) THEN
IF k% = HotSpots%(m%, 3) THEN

```

```

HotSpotNum% = m%
EXIT SUB
END IF
END IF
END IF
NEXT
END SUB
SUB RemoveHotSpot (HotSpotNum%) STATIC
HotSpots%( HotSpotNum%, 1) = -1
END SUB
SUB AddHotSpot (i%, j%, k%, Returncode%) STATIC
Returncode% = -1
FOR m% = 0 TO 15
IF HotSpots%(m%, 1) = -1 THEN
HotSpots%(m%, 1) = i%
HotSpots%(m%, 2) = j%
HotSpots%(m%, 3) = k%
Returncode% = 0
EXIT SUB
END IF
NEXT
END SUB
SUB InitializePermArray STATIC
CALL ChangeCursor (4)
FOR j% = 1 TO 31
FOR i% = 1 TO 31
R! = .5* SQR(CSNG((j% - 16) * (j% - 16) + (i% - 16) * (i% - 16)))
IF R! > 7.6 THEN Code% = 5 ELSE Code% = 4
' If program is run from environment,
' then do minimal Initialization for testing,
' Otherwise do complete initialization.
IF SYSTEM(4) = 0 THEN KMax% = 1 ELSE KMax% = 63
FOR k% = 1 TO KMax%
CALL SetPerm (i%, j% , k%, Code%)
NEXT
NEXT
NEXT
CALL INTCURSOR
END SUB
SUB SetPerm(i%, j%, k%, Code%) STATIC
Segment% = i% \ 4 'Segment = 0 to 7
Offset% = i% MOD 4 'Offset = 0 to 3
Old& = PermArray%(Segment%, j%, k%)
IF Old& < 0 THEN Old& = Old& + 65536&
SELECT CASE Offset%
CASE 0
Mask& = &HF
Shift& = 1
CASE 1
Mask& = &HF0
Shift& = 16
CASE 2
Mask& = &HF00
Shift& = 256
CASE 3
Mask& = &HF000
Shift& = 4096
END SELECT
Old& = Old& AND NOT Mask& 'Blank out old value
value& = CLNG(Code%) * Shift&
Old& = Old& + value&
IF Old& > 32767 THEN Old& = Old& - 65536&
PermArray%(Segment%, j%, k%) = Old&
END SUB
SUB GetPerm (i%, j%, k%, Code%) STATIC
Segment% = i% \ 4 'Segment = 0 to 7
Offset% = i% MOD 4 'Offset = 0 to 3
Old& = PermArray%(Segment%, j%, k%)

```

```

IF Old% < 0 THEN Old% = Old% + 65536%
SELECT CASE Offset%
CASE 0
Mask% = &HF
Shift% = 1
CASE 1
Mask% = &HFO
Shift% = 16
CASE 2
Mask% = &HFO0
Shift% = 256
CASE 3
Mask% = &HFO00
Shift% = 4096
END SELECT
Code% = (Old% AND Mask%) \Shift%
END SUB
SUB InitializePatterns STATIC
CALL ChangeCursor (4)
Patterns%(0) = 0
Patterns%(1) = 0
Patterns%(2) = 0
Patterns%(3) = 0
Patterns%(4) = &H8000
Patterns%(5) = &H800
Patterns%(6) = &H4000
Patterns%(7) = &H400
Patterns%(8) = &H8800
Patterns%(9) = &H0
Patterns%(10) = &H8800
Patterns%(11) = &H0
Patterns%(12) = &H8800
Patterns%(13) = &H2200
Patterns%(14) = &H8800
Patterns%(15) = &H2200
Patterns%(16) = &H8010
Patterns%(17) = &H240
Patterns%(18) = &H810
Patterns%(19) = &H2004
Patterns%(20) = &H8421
Patterns%(21) = &H842
Patterns%(22) = &H1084
Patterns%(23) = &H2108
Patterns%(24) = &H8822
Patterns%(25) = &H8822
Patterns%(26) = &H8822
Patterns%(27) = &H8822
Patterns%(28) = &H9249
Patterns%(29) = &H2492
Patterns%(30) = &H4924
Patterns%(31) = &H9249
Patterns%(32) = &HAA55
Patterns%(33) = &HAA55
Patterns%(34) = &HAA55
Patterns%(35) = &HAA55
Patterns%(36) = &HDB6D
Patterns%(37) = &HB6DB
Patterns%(38) = &H6DB6
Patterns%(39) = &HDB6B
Patterns%(40) = &H77DD
Patterns%(41) = &H77DD
Patterns%(42) = &H77DD
Patterns%(43) = &H77DD
Patterns%(44) = &H77FF
Patterns%(45) = &HDDFF
Patterns%(46) = &H77FF
Patterns%(47) = &HDDFF
Patterns%(48) = &HFFFF

```

```

Patterns%(49) = &HFFFF
Patterns%(50) = &HFFFF
Patterns%(51) = &HFFFF
Patterns%(52) = &HFFFF
Patterns%(53) = &HFFFF
Patterns%(54) = &HFFFF
Patterns%(55) = &HFFFF
Patterns%(56) = &H0
Patterns%(57) = &H80
Patterns%(58) = &H0
Patterns%(59) = &H0
Patterns%(60) = &HFFFF
Patterns%(61) = &HC3C3
Patterns%(62) = &HC3C3
Patterns%(63) = &HFFFF
END SUB

```

C.2 Program COAL CORE FLUX CALC .BAS

```

' Language: QuickBasic for the Macintosh, version 1.0
' Author: Rich Dobbs
' Written: 11/28 - 12/9 (Also worked on Front Calculations Program)
' Purpose: This program calculates the flow field through the coal core.
           The shape of the core and channel is defined by the shape data
           file from the Fat Nibble Shape Entry editor.

' Glossary: HotSpots - Up to 16 specific points where flux calculations
           are requested.
'*****
' Features: Version 1.0 - Major features implemented
           Successive Over Relaxation Routine Implemented
           SNTB, SHWE, WETB, EWTB views of pressure, flux, residual implemented
           File Save and File Read Working
'*****
'*****Global Declarations*****
CLEAR ,1000000%
'*****Inherited Global Declarations*****
'Data from shape definition file
DIM SHARED PermCodeArray%(8, 32, 64)
DIM SHARED CurMatCode%, CurPattern%, ActiveView%, MouseMode%
DIM SHARED DefaultI%, DefaultJ%, DefaultK%, NCoorBias!
DIM SHARED HotSpots%(15,3), Patterns%(23)
'*****New Global Declarations for Coal Core Flux Calc .bas*****
' Data needed by Front calculations
DIM SHARED ShapeDefFileName$
DIM SHARED CalcFileName$
DIM SHARED MeshLevel%, StepI%, StepJ%, StepK%
DIM SHARED MatPerm!(15)
DIM SHARED P!(32, 32, 64) ' 280 KB
' Data Defining situation inside of Coal Core
DIM SHARED SOR!, Tol!, TotErr!, RelErr!, ItsAtLevel%
DIM SHARED FluxScale!, FatArrowLimit!
DIM SHARED WinOpen%(16), WinUpdate%(16)
DIM SHARED SNTBBitCode%, SHWEBitCode%, WETBBitCode%, EWTBBitCode%
' Data needing restoration to default values
DIM SHARED ChangesSaved%
DIM SHARED Flux!(6)
DIM SHARED StoredPollingInfo%(3)
DIM SHARED Halt%, TaskWaiting%, StoredNumOfIts%
' Constants need initialization only
DIM SHARED OffsetI%(6), OffsetJ%(6), OffsetK%(6)
DIM SHARED True%, False%
'Automatically reinitialized
DIM SHARED CalcShortFileName$

```

```

'*****Reverse Inheirited from Coal Core Surfaace calc
DIM SHARED AverageFluxAlongCore!(64), AutomaticPrint%
CALL CoreFlux
SUB CoreFlux STATIC
    CALL WindowInitialize
    CALL StartUpCode
    CALL DefineMenu(0)
    CALL InitializeBoundaryConditions
    CALL SetDefaultValues
    WHILE True%
        AllEventsHandled% = False%
        WHILE AllEventsHandled% = False%
            CALL Poll (Dialog0%, Mous0%, Menu0%)
            IF Dialog0% = 0 AND Mous0% = 0 AND Menu0% = 0 THEN
                AllEventsHandled% = True%
            IF Dialog0% <> 0 THEN CALL HandleDialog(Dialog0%)
            IF Mous0% <> 0 THEN CALL HandleMouse (Mous0%)
            IF Menu0% <> 0 THEN CALL HandleMenu (Menu0%)
        WEND
        CALL TaskDispatcher
    WEND
END SUB
SUB Poll(Dialog0%, Mous0%, Menu0%) STATIC
    IF StoredPollingInfo%(0) = True% THEN
        Dialog0% = StoredPollingInfo%(1)
        Mous0% = StoredPollingInfo%(2)
        Menu0% = StoredPollingInfo%(3)
        StoredPollingInfo%(0) = False%
    ELSE
        Dialog0% = DIALOG(0)
        Mous0% = MOUSE(0)
        Menu0% = MENU(0)
    END IF
END SUB
SUB UserInteractionManager (UserIntervention%) STATIC
    Dialog0% = DIALOG(0)
    Mous0% = MOUSE(0)
    Menu0% = MENU(0)
    IF Mous0% <> 0 OR Menu0% <> 0 OR Dialog0% <> 0 THEN
        UserIntervention% = True%
        StoredPollingInfo%(0) = True%
        StoredPollingInfo%(1) = Dialog0%
        StoredPollingInfo%(2) = Mous0%
        StoredPollingInfo%(3) = Menu0%
    END IF
END SUB
SUB HandleDialog (Dialog0%) STATIC
    SELECT CASE Dialog0%
        CASE 3
            Win% = DIALOG(3)
            WINDOW Win%
            'Window clicked to be made active
            'Make active, which causer a dialog event 5 which
            'actually draws the WINDOW
        CASE 4
            WinOpen%(DIALOG(4)) = False%
            'Window clicked closed
            'Record that a window has been closed
        CASE 5
            Win% = DIALOG(5)
            CALL Refreshwindow (Win%)
        CASE ELSE
    END SELECT
END SUB
'Mouse Code
SUB HandleMouse (Mous0%) STATIC
    SELECT CASE Mous0%
        CASE ELSE
            BEEP
    END SELECT
END SUB
'MenuCode
SUB DefineMenu (MenuNum%) STATIC

```

```

SELECT CASE MenuNum%
CASE 0 'Original Definition
  MENU 1, 0, 1, "File"
  MENU 1, 1, 1, "New"
  MENU 1, 2, 1, "Open"
  MENU 1, 3, 0, "-----"
  MENU 1, 4, 1, "Save" : CmdKey 1, 4, "S"
  MENU 1, 5, 1, "Save as..."
  MENU 1, 6, 0, "-----"
  MENU 1, 7, 1, "Print AW..."
  IF AutomaticPrint% = True% THEN
    MENU 1, 8, 2, "Print Auto..."
  ELSE
    MENU 1, 8, 1, "Print Auto..."
  END IF
  MENU 1, 9, 1, "Print Combo"
  MENU 1, 10, 0, "-----"
  MENU 1, 11, 1, "Quit" : CmdKey 1, 11, "Q"
  MENU 2, 0, 1, "Edit"
  MENU 2, 1, 1, "Cut" : CmdKey 2, 1, "X"
  MENU 2, 2, 1, "Copy" : CmdKey 2, 2, "C"
  MENU 2, 3, 1, "Paste" : CmdKey 2, 3, "V"
  MENU 2, 4, 1, "Copy AW"
  MENU 3, 0, 1, "Run"
  MENU 3, 1, 1, "Continuous..."
  MENU 3, 2, 1, "Several..."
  MENU 3, 3, 1, "Once"
  MENU 3, 4, 0, "-----"
  MENU 3, 5, 1, "Refine Mesh"
  MENU 3, 6, 1, "Change SOR..." : CmdKey 3, 6, "*"
  MENU 3, 7, 0, "Stop" : CmdKey 3, 7, "/"
  MENU 3, 8, 1, "Change TOL..."
  MENU 3, 9, 1, "Coarsen Mesh"
  MENU 3, 10, 1, "Calc. Avg. Flux"
  MENU 4, 0, 1, "Display"
  MENU 4, 1, 1, "General Info" : CmdKey 4, 1, "G"
  MENU 4, 2, 1, "SNTB..."
  MENU 4, 3, 1, "SNWE..."
  MENU 4, 4, 1, "WETB..."
  MENU 4, 5, 1, "EWTB..."
  MENU 4, 6, 1, "Set Flux Scale..."
  MENU 4, 7, 1, "Average Flux"
CASE 1 'User interaction menu
  MENU 1, 1, 0 'Can't start new
  MENU 1, 2, 0 'Can't open old file
  MENU 3, 1, 0 'Can't run CONTINUOUS without stoping current run
  MENU 3, 2, 0 'Or SEVERAL
  MENU 3, 3, 0 'Or ONCE
  MENU 3, 5, 0 'Or REFINE MESH
  MENU 3, 7, 1 'But can STOP current run
  MENU 3, 9, 0 'But not coarsen mesh
  MENU 3, 10, 0 'Can't Determine Average Flux during Run
CASE ELSE
END SELECT
END SUB
SUB HandleMenu (Menu0%) STATIC
  SELECT CASE Menu0%
  CASE 1 'File
    CALL HandleFileMenu
  CASE 2
    CALL HandleEditMenu
  CASE 3 'Run
    CALL HandleRunMenu
  CASE 4 'Display
    CALL HandleDisplayMenu
  CASE ELSE
    BEEP
  END SELECT
END SELECT

```

```

MENU
END SUB
'*****Edit Menu Code*****
' See new version at end of code reinherited from Coal Core Surface Calc. bas
'*****Run Menu Code*****
SUB HandleRunMenu STATIC
  ChangesSaved% = False%
  SELECT CASE MENU(1)
    CASE 1 'Continuously...
      CALL NumberOfItsPromptDialogBox
      TaskWaiting% = 2
    CASE 2 'Several...
      CALL NumberOfItsPromptDialogBox
      TaskWaiting% = 1
    CASE 3 'Once
      TaskWaiting% = 1
      StoredNumOfIts% = 1
    CASE 5
      ChangeCursor 4
      RefineMesh
      INITCURSOR
    CASE 6 'Change SOR
      ChangeSORDialogBox
    CASE 7 'Halt execution (Stop on menu)
      IF TaskWaiting% <> 0 THEN
        HaltAlert
        Halt% = True%
      END IF
    CASE 8 'Change Tolerance
      ChangeTolDialogBox
    CASE 9
      CoarsenMesh
    CASE 10 'Determine Average Flux Along Core"
      DetermineAverageFluxAlongCore
  END SELECT
END SUB
SUB NumberOfItsPromptDialogBox STATIC
  WINDOW 16,,(300,200)-(700,300),-2
  OldStoredNumOfIts% = StoredNumOfIts%
  Okay% = False%
  WHILE NOT Okay%
    PRINT "Current number of iterations:"
    PRINT OldStoredNumOfIts%
    PRINT "New number of iterations"
    INPUT StoredNumOfIts%
    IF StoredNumOfIts% > 0! THEN Okay% = True%
  WEND
  WINDOW CLOSE 16
END SUB
SUB HaltAlert STATIC
  WINDOW 16,,(300,200)-(575,280),2
  IF Halt% = False% THEN MessageTime$ = TIME$
  PRINT "Halt message received at "; MessageTime$;"."
  PRINT "Current Iteration being completed."
  IF SYSTEM(4) = 0 THEN
    EstTime! = 60! * CSNG(2^ MeshLevel%) / 16384!
  ELSE
    EstTime! = 3! * CSNG(2^ MeshLevel%) / 16384!
  END IF
  PRINT "Approximate time to complete "
  PRINT USING "last iteration is ###.## minutes."; EstTime!
END SUB
SUB ChangeSORDialogBox STATIC
  WINDOW 16,,(300,200)-(700,300),-2
  OldSOR! = SOR!
  Okay% = False%
  WHILE NOT Okay%
    PRINT "Current Value of Relaxation Parameter:"

```



```

PRINT OldSOR!
PRINT "New Value of Relaxation Parameter: (0 - 2)"
INPUT SOR!
IF SOR! < 2! AND SOR! > 0! THEN Okay% = True%
WEND
WINDOW CLOSE 16
IF WinOpen%(11) = True% THEN CALL RefreshWindow(11) 'Update General Info
END SUB
SUB ChangeTolDialogBox STATIC
WINDOW 16,,(300,200)-(700,300),-2
OldTol! = Tol!
Okay% = False%
WHILE NOT Okay%
PRINT "Current Value of Tolerance Parameter:"
PRINT OldTol!
PRINT "New Value of Tolerance Parameter: "
INPUT Tol!
IF (Tol! > 1E-16) AND (Tol! < .01) THEN
Okay% = True%
ELSE
PRINT "The appropriate range is 1.0e-16 to 1.0e-2"
PRINT
END IF
WEND
WINDOW CLOSE 16
IF WinOpen%(11) = True% THEN CALL RefreshWindow(11) 'Update General Info
END SUB
'*****File Menu Code*****
SUB HandleFileMenu STATIC
SELECT CASE MENU(1)
CASE 1 'File - New
HandleFileNew
CASE 2 'File - Open
HandleFileOpen
CASE 4 'File - Save
CALL CalcFileWrite
ChangesSaved% = True%
CASE 5 'File - Save As...
TrialFileName$ = FILES$(0,"Save data to disk as:")
IF TrialFileName$ <> "" THEN
CalcFileName$ = TrialFileName$
CALL CalcFileWrite
ChangesSaved% = True%
END IF
CASE 7 "'Print AW..."
PrintActiveWindow
EXIT SUB 'To avoid parsing the file name again
CASE 8 'Print Auto...
HandlePrintAuto
CASE 9 'Print Combo
PrintIntoPlotCombo
EXIT SUB
CASE 11 'Quit
CALL SaveChangesInquiry (ReturnCode%)
IF ReturnCode% = True% THEN
STOP
END IF
END SELECT
ParseFileName CalcFileName$, CalcShortFileName$
END SUB
SUB HandlePrintAuto STATIC
IF AutomaticPrint% = True% THEN
AutomaticPrint% = False%
MENU 1, 8, 1
ELSE
AutomaticPrint% = True%
MENU 1, 8, 2
END IF

```

```

END SUB
SUB HandleFileOpen STATIC
    CALL SaveChangesInquiry (ReturnCode%)
    IF ReturnCode% = True% THEN
        TrialFileName$ = FILES$(1,"RLD2")
        IF TrialFileName$ <> "" THEN
            CalcFileName$ = TrialFileName$
            CALL CalcFileRead (ReturnCode%)
            IF ReturnCode% = 0 THEN
                ShapeDefFileRead(ReturnCode%)
                IF ReturnCode% <> 0 THEN
                    FailedFileName$ = ShapeDefFileName$
                END IF
            ELSE
                FailedFileName$ = CalcFileName$
            END IF
            IF ReturnCode% <> 0 THEN
                WINDOW 16,,, -1
                TS$ = FailedFileName$ +
                    ": This file could not be read successfully."
                rect%(0) = 20 : rect%(1) = 20 : rect%(2) = 80
                rect%(3) = 180 : just% = 1
                TextBox TS$, rect%(0), just%
                BUTTON 1,1,"Clear Message",(50,100)-(150,120)
                WHILE DIALOG(0) <> 1 : WEND
                WINDOW CLOSE 16
                CalcFileName$ = "Untitled"
                ShapeDefFileName$ = "Untitled"
                CALL SetDefaultValues
            END IF
        END IF
    END SUB
SUB HandleFileNew STATIC
    CALL SaveChangesInquiry (ReturnCode%)
    IF ReturnCode% = True% THEN
        TrialFileName$ = FILES$(1,"RLD1")
        IF TrialFileName$ <> "" THEN
            ShapeDefFileName$ = TrialFileName$
            CALL SetDefaultValues
            CALL ShapeDefFileRead(ReturnCode%)
            IF ReturnCode% = -1 THEN
                WINDOW 4,,, -1
                TS$ = "This file could not be read successfully."
                rect%(0) = 20 : rect%(1) = 20
                rect%(2) = 80 : rect%(3) = 180 : just% = 1
                TextBox TS$, rect%(0), just%
                BUTTON 1,1,"Clear Message",(50,100)-(150,120)
                WHILE DIALOG(0) <> 1 : WEND
                WINDOW CLOSE 4
                ShapeDefFileName$ = "Untitled"
            ELSE
                f$ = UCASE$(ShapeDefFileName$)
                h% = 1
                Done% = False%
                WHILE Done% = False%
                    hn% = INSTR(h%, f$, ":")
                    IF hn% > 0 THEN
                        h% = hn% + 1
                    ELSE
                        Done% = True%
                    END IF
                WEND
                g% = INSTR(h%, f$, "SHAPE")
                IF g% > 0 THEN
                    f$ = LEFT$(f$, g% - 1) + " Flux Calcs"
                ELSE
                    f$ = ShapeDefFileName$ + " Flux Calcs"
                END IF
            END IF
        END IF
    END SUB

```

```

        END IF
        CalcFileName$ = f$
        RefineMesh
        RefineMesh
        RefineMesh
        RefineMesh
    END IF
END IF
END IF
END SUB
SUB ShapeDefFileRead (ReturnCode%) STATIC
    ReturnCode% = -1
    'Note: Although this file is read sequentially,
    'PermCodeArray0 stored as binary to
    'reduce the size of the disk file.
    OPEN ShapeDefFileName$ FOR INPUT AS #5
    MagicStatement$ = "This is a data file for the perm matrix   RLD 11/26/89"
    INPUT #5, MagicStatementCandidates$
    IF MagicStatementCandidates$ <> MagicStatement$ THEN EXIT SUB
    ChangeCursor 4
    INPUT #5, CurMatCode%, CurPattern%, ActiveView%, MouseMode%
    INPUT #5, DefaultIX%, DefaultJY%, DefaultKZ%, NCoorBias%
    FOR m% = 0 TO 15
        INPUT #5, HotSpots%(m%, 1), HotSpots%(m%, 2), HotSpots%(m, 3)
    NEXT
    FOR i% = 0 TO 7
        FOR j% = 1 TO 31
            FOR k% = 1 TO 63
                PermCodeArray%(i%, j%, k%) = CVI(INPUT$(2,#5))
            NEXT
        NEXT
    NEXT
    CALL FixUpPermCodeArray
    CLOSE #5
    INITCURSOR
    ReturnCode% = 0
END SUB
SUB CalcFileWrite STATIC
    ChangeCursor 4
    OPEN CalcFileName$ FOR OUTPUT AS #5 LEN = 32
    MagicStatement$ =
        "This is a data file for the calculation matrix   RLD 12/7/89"
    WRITE #5, MagicStatement$
    WRITE #5, ShapeDefFileName$
    WRITE #5, MeshLevel%, StepIX%, StepJY%, StepKZ%
    WRITE #5, SOR!, Tol!, TotErr!, RelErr!, ItsAtLevel%
    WRITE #5, FluxScale!, FatArrowLimit!
    WRITE #5, SNTBBitCode%, SNWEBitCode%, WETBBitCode%, EWTBBitCode%
    WRITE #5, "Window Information-MagicLine"
    FOR i% = 0 TO 16
        WRITE #5, WinOpen%(i%), WinUpdate%(i%)
    NEXT
    WRITE #5, "Material Permeabilities-MagicLine"
    FOR i% = 0 TO 15
        WRITE #5, MatPerm!(i%)
    NEXT
    WRITE #5, "Pressure Data stored in binary-MagicLine"
    FOR i% = 0 TO 32 STEP StepIX%
        FOR j% = 0 TO 32 STEP StepJY%
            FOR k% = 0 TO 64 STEP StepKZ%
                PRINT#5, MKS$(P!(i%, j%, k%));
            NEXT
        NEXT
    NEXT
    CLOSE #5
    NAME CalcFileName$ AS CalcFileName$, "RLD2"
    INITCURSOR
END SUB

```

```

SUB CalcFileRead( ExitCode%) STATIC
  ChangeCursor 4
  OPEN CalcFileName$ FOR INPUT AS #5 LEN = 32
  MagicLine$ = "This is a data file for the calculation matrix   RLD 12/7/89"
  SearchForMagic MagicLine$, Found%
  IF Found% = False% THEN
    ExitCode% = 1
    CLOSE #5
    EXIT SUB
  END IF
  INPUT #5, ShapeDefFileName$
  INPUT #5, MeshLevel%, StepI%, StepJ%, StepK%
  INPUT #5, SOR!, Tol!, TotErr!, RelErr!, ItsAtLevel%
  INPUT #5, FluxScale!, FatArrowLimit!
  INPUT #5, SWTBBitCode%, SNWEBitCode%, WETBBitCode%, EWTBBitCode%
  MagicLine$ = "Window Information-MagicLine"
  SearchForMagic MagicLine$, Found%
  IF Found% = False% THEN
    ExitCode% = 2
    CLOSE #5
    EXIT SUB
  END IF
  FOR i% = 0 TO 16
    INPUT #5, WinOpen%(i%), WinUpdate%(i%)
  NEXT
  MagicLine$ = "Material Permeabilities-MagicLine"
  SearchForMagic MagicLine$, Found%
  IF Found% = False% THEN
    ExitCode% = 3
    CLOSE #5
    EXIT SUB
  END IF
  FOR i% = 0 TO 15
    INPUT #5, MatPerm!(i%)
  NEXT
  MagicLine$ = "Pressure Data stored in binary-MagicLine"
  SearchForMagic MagicLine$, Found%
  IF Found% = False% THEN
    ExitCode% = 4
    CLOSE #5
    EXIT SUB
  END IF
  FOR i% = 0 TO 32 STEP StepI%
    FOR j% = 0 TO 32 STEP StepJ%
      FOR k% = 0 TO 64 STEP StepK%
        P!(i%, j%, k%) = CVS(INPUT$(4,*5))
      NEXT
    NEXT
  NEXT
  CLOSE #5
  'NAME CalcFileName$ AS CalcFileName$, "RLD2"
  CALL INITCURSOR
  ExitCode% = 0
END SUB

SUB SearchForMagic (MagicLine$, Found%) STATIC
  Found% = False%
  a$ = CHR$(34)
  WHILE NOT EOF(5) AND Found% = False%
    LINE INPUT #5, TestLine$
    g% = INSTR(TestLine$, a$)
    IF g% > 0 THEN
      h% = INSTR(TestLine$, MagicLine$)
      IF h% > 0 THEN
        Found% = True%
      END IF
    END IF
  WEND
END SUB

```

```

SUB SaveChangesInquiry (ReturnCode%) STATIC
IF ChangesSaved% = True% THEN
    ReturnCode% = True%
ELSE
    WINDOW 16,,(400, 200)-(700, 330),-2
    TS$ = "Changes to the active file have not been saved."
    TS$ = TS$ + " Do you want to save changes before proceeding"
    rect%(0) = 20 : rect%(1) = 20 : rect%(2) = 80 : rect%(3) = 280
    just% = 0
    TextBox TS$, rect%(0), just%
    BUTTON 1, 1, "CANCEL", (20, 90) - (80, 110), 1
    BUTTON 2, 1, "NO", (135, 90) - (185, 110), 1
    BUTTON 3, 1, "YES", (230, 90) - (280, 110), 1
    rect%(0) = 85 : rect%(1) = 225 : rect%(2) = 115 : rect%(3) = 285
    CALL PENSIZE(3,3)
    CALL FRAMEGUNDRECT (VAFPTR(rect%(0)),15,15)
    CALL PENNORMAL
    ExitCode% = False%
    WHILE ExitCode% = False%
        IF DIALOG(0) = 1 THEN
            Selector% = DIALOG(1)
            ExitCode% = True%
        ELSEIF INKEY$ = CHR$(13) THEN
            Selector% = 3
            ExitCode% = True%
        END IF
    WEND
    WINDOW CLOSE 16
    SELECT CASE Selector%
        CASE 1
            ReturnCode% = False%
        CASE 2
            ReturnCode% = True%
        CASE 3
            CALL CalcFileWrite
            ChangesSaved% = True%
            ReturnCode% = True%
    END SELECT
END IF
END SUB

'*****Display Menu Code*****
SUB HandleDisplayMenu STATIC
SELECT CASE MENU(1)
    CASE 1 'General Info
        WinOpen%(11) = True%
        WinUpdate%(11) = True%
        WINDOW 11
    CASE 2 'SNTB...
        Title$ = CalcShortFileName$ + ": South - North - Top - Bottom"
        OldBitCode% = SNTBBitCode%
        DisplayChoicesDialogBox Title$, SNTBBitCode%, DefaultI%, 31, Okay%
        IF Okay% THEN
            WinOpen%(7) = True%
            WinUpdate%(7) = True%
            WINDOW 7, Title$ + ", I = " + STR$(SNTBBitCode% AND 127)
        ELSE
            SNTBBitCode% = OldBitCode%
        END IF
    CASE 3 'SNWE...
        Title$ = CalcShortFileName$ + ": South - North - West - East"
        OldBitCode% = SNWEBitCode%
        DisplayChoicesDialogBox Title$, SNWEBitCode%, DefaultJ%, 31, Okay%
        IF Okay% THEN
            WinOpen%(8) = True%
            WinUpdate%(8) = True%
            WINDOW 8, Title$ + ", J = " + STR$(SNWEBitCode% AND 127)
        ELSE
            SNWEBitCode% = OldBitCode%
        END IF
END SELECT

```

```

END IF
CASE 4      'WETB...
Title$ = CalcShortFileName$ + ": West - East - Top - Bottom"
OldBitCode% = WETBBitCode%
DisplayChoicesDialogBox Title$, WETBBitCode%, DefaultK%, 63, Okay%
IF Okay% THEN
WinOpen%(9) = True%
WinUpdate%(9) = True%
WINDOW 9, Title$ + ", K = " + STR$(WETBBitCode% AND 127)
ELSE
WETBBitCode% = OldBitCode%
END IF
CASE 5      'EWTB...
Title$ = CalcShortFileName$ + ": East - West - Top - Bottom"
OldBitCode% = EWTBBitCode%
DisplayChoicesDialogBox Title$, EWTBBitCode%, DefaultK%, 63, Okay%
IF Okay% THEN
WinOpen%(10) = True%
WinUpdate%(10) = True%
WINDOW 10, Title$ + ", K = " + STR$(EWTBBitCode% AND 127)
ELSE
EWTBBitCode% = OldBitCode%
END IF
CASE 6      'Set Flux Scale...'
CALL SetFluxScaleDialogBox
CASE 7      'Plot Average Flux Along Core for Export'
WinOpen%(5) = True%
WinUpdate%(5) = True%
WINDOW 5
CASE ELSE
END SELECT
END SUB
SUB SetFluxScaleDialogBox STATIC
WINDOW 16,,(300,200)-(700,300),-2
OldFluxScale! = FluxScale!
Okay% = False%
WHILE NOT Okay%
PRINT "Current Value of Flux Scale Factor:"
PRINT OldFluxScale!
PRINT "New Value of Flux Scale Factor:"
INPUT FluxScale!
IF FluxScale! > 0! THEN Okay% = True%
WEND
OldFatArrowLimit! = FatArrowLimit!
Okay% = False%
WHILE NOT Okay%
PRINT "Current Value of Fat Arrow Scale Factor:"
PRINT OldFatArrowLimit!
PRINT "New Value of Fat Arrow Scale Factor:)"
INPUT FatArrowLimit!
IF FatArrowLimit! > 0! THEN Okay% = True%
WEND
WINDOW CLOSE 16
IF WinOpen%(11) = True% THEN CALL RefreshWindow(11) 'Update General Info
END SUB
SUB DisplayChoicesDialogBox (Title$, BitCode%, Default%, MaxNum%, Okay%) STATIC
IF MaxNum% > 63 OR MaxNum% < 1 THEN
Okay% = False%
ExitSub%
END IF
IF MaxNum% > 31 THEN
WINDOW 16,,(300,200)-(750,540),-2
ELSE
WINDOW 16,,(300,200)-(750,460),-2
END IF
END IF
LOCATE 2,4
TEXTFACE 1
PRINT Title$

```

```

'Define Buttons
ViewNum% = (BitCode% AND 127)
IF ViewNum% = 0 THEN ViewNum% = Default% : 'View is stored in 6 bits
a% = (BitCode% AND 128) \ 128 + 1 '7th bit = show core? 1 = yes
b% = (BitCode% AND 256) \ 256 + 1 '8th bit = show pressure? "" ""
c% = (BitCode% AND 512) \ 512 + 1 '9th bit = show flux? "" ""
d% = (BitCode% AND 1024) \ 1024 + 1 '10th bit = show residual?
e% = (BitCode% AND 2048) \ 2048 + 1 '11 bit = clear screen?
f% = (BitCode% AND 4096) \ 4096 '12 bit = use default view?
IF f% = 1 THEN
    ViewNum% = Default%
    t1% = 2
    t2% = 1
ELSE
    t1% = 1
    t2% = 2
END IF
BUTTON 64, a%, "Show Core Material", (20, 40)-(300, 60), 2
BUTTON 65, b%, "Show Pressure Field", (20, 60)-(300, 80), 2
BUTTON 66, c%, "Show Flux", (20, 80)-(300, 100), 2
BUTTON 67, d%, "Show Flux Discrepancy", (20, 100)-(300, 120), 2
BUTTON 68, e%, "Clear Screen Completely", (20, 120)-(300, 140), 2
BUTTON 69, t1%, "Use Default View", (20, 160)-(200, 180), 3
BUTTON 70, t2%, "Select View", (20, 180)-(200, 200), 3
BUTTON 80, 1, "Cancel", (300, 40)-(400, 60), 1
BUTTON 81, 2, "Okay", (300, 80)-(400, 100), 1
rect%(0) = 170 : rect%(1) = 210 : rect%(2) = 190 : rect%(3) = 400
ViewNumText$ = "Pointer: " + STR$(ViewNum%)
just% = 0
TextBox ViewNumText$, rect%(0), just%
IF MaxNum% = 31 THEN
    ScaleX% = 13
    ModFactor% = 2
ELSE
    ScaleX% = 6
    ModFactor% = 4
END IF
IF f% = 0 THEN IsOn% = 1 ELSE IsOn% = 0
FOR i% = 1 TO MaxNum%
    x% = 20 + ScaleX% * i%
    y% = 200 + ((i% + 3) MOD ModFactor%) * 24
    BUTTON i%, IsOn%, "", (x%, y%)-(x% + 20, y% + 20), 3
NEXT
IF f% = 0 THEN BUTTON ViewNum%, 2 'Show current view button
'Button Management Code:
DontExit% = True%
WHILE DontExit%
    Dialog0% = 0
    WHILE Dialog0% <> 1
        Dialog0% = DIALOG(0)
    WEND
    ButtonNum% = DIALOG(1)
    SELECT CASE ButtonNum%
        CASE 1 TO 63 'Radio Buttons which mark current view
            BUTTON ViewNum%, 1
            ViewNum% = ButtonNum%
            BUTTON ViewNum%, 2
            ViewNumText$ = "Pointer: " + STR$(ViewNum%)
            just% = 0
            TextBox ViewNumText$, rect%(0), just%
        CASE 64, 65, 66, 67, 68 'A Check Box has been toggle
            IF BUTTON(ButtonNum%) = 1 THEN
                BUTTON ButtonNum%, 2
            ELSE
                BUTTON ButtonNum%, 1
            END IF
        CASE 69 'Radio button for defaultview pushed
    
```

```

        BUTTON 69, 2
        BUTTON 70, 1
        FOR i% = 1 TO MaxNum%
            BUTTON i%, 0
        NEXT
        ViewNumText$ = "Pointer = " + STR$(Default%)
        just% = 0
        TextBox ViewNumText$, rect%(0), just%
CASE 70
    'Radio button for select view pushed
    BUTTON 69, 1
    BUTTON 70, 2
    FOR i% = 1 TO MaxNum%
        BUTTON i%, 1
    NEXT
    BUTTON ViewNum%, 2
    ViewNumText$ = "Pointer = " + STR$(ViewNum%)
    just% = 0
    TextBox ViewNumText$, rect%(0), just%
CASE 80
    'Cancel
    DontExit% = False%
    Okay% = False%
CASE 81
    DontExit% = False%
    Okay% = True%
CASE ELSE
END SELECT
WEND
IF Okay% THEN
    'Store value here
    IF BUTTON(64) = 2 THEN BitCode% = 128 ELSE BitCode% = 0
    IF BUTTON(65) = 2 THEN BitCode% = BitCode% + 256
    IF BUTTON(66) = 2 THEN BitCode% = BitCode% + 512
    IF BUTTON(67) = 2 THEN BitCode% = BitCode% + 1024
    IF BUTTON(68) = 2 THEN BitCode% = BitCode% + 2048
    IF BUTTON(69) = 2 THEN
        BitCode% = BitCode% + Default%
        BitCode% = BitCode% + 4096
    ELSE
        BitCode% = BitCode% + ViewNum%
    END IF
END IF
END IF
WINDOW CLOSE 16
END SUB
'Display Writing Code
SUB UpdateActiveDisplays STATIC
    FOR i% = 1 TO 16
        IF WinUpdate%(i%) = True% THEN
            IF WinOpen%(i%) = True% THEN
                CALL RefreshWindow(i%)
            END IF
        END IF
    NEXT
END SUB
SUB WindowInitialize STATIC
    WINDOW 1, "Coal Core Flux Calculation", (50,50)-(800,100), 7
    WINDOW 5, "Average Flux Along Core", (250,80)-(650,480), 7
    WINDOW 7, "South - North - Top - Bottom", (660, 80)-(1260, 480), 7
    WINDOW 8, "South - North - West - East", (660, 510)-(1260, 910), 7
    WINDOW 9, "West - East - Top - Bottom", (250,80)-(650,480), 7
    WINDOW 10, "East - West - Top - Bottom", (250, 510)-(650, 910), 7
    WINDOW 11, "General Info", (50, 80)-(240,360), 6
    'Window 16 is for dialog boxes that immediately disappear
    WINDOW CLOSE 11
    WINDOW CLOSE 10
    WINDOW CLOSE 9
    WINDOW CLOSE 8
    WINDOW CLOSE 7
    WINDOW CLOSE 5

```



```

END SUB
SUB RefreshWindow (i%) STATIC
IF WinOpen%(i%) = True% THEN
WINDOW OUTPUT i%
SELECT CASE i%
CASE 5
PlotAverageFluxAlongCoreForExport
CASE 7
DisplaySNTB
CASE 8
DisplaySNWE
CASE 9
DisplayWETB
CASE 10
DisplayEWTB
CASE 11
DisplayGeneralInfo
CASE ELSE
END SELECT
END IF
END SUB
'*****Window 11 Code*****
SUB DisplayGeneralInfo STATIC
CLS
TEXTFONT 0
TEXTSIZE (12)
'TEXTFACE 1
PRINT #22,
PRINT #22, " File Name: "
PRINT #22, " "; CalcShortFileName$
PRINT #22, USING " Mesh Level: ## of 14"; MeshLevel%
PRINT #22, USING " Step in I: ##"; StepI%
PRINT #22, USING " Step in J: ##"; StepJ%
PRINT #22, USING " Step in K: ##"; StepK%
PRINT #22, USING " Iterations: ####"; ItsAtLevel%
PRINT #22, USING " Relaxation: ##.##"; SOR!
PRINT #22, USING " Tolerance: ##.#####"; Tol!
PRINT #22, USING " Scaled Error: ##.#####"; RelErr!
PRINT #22, USING " Total Error: ##.#####"; TotErr!
PRINT #22, USING " Min XS Flux: ##.#####"; AverageFluxAlongCore!(0)
PRINT #22, USING " 1/4 XS Flux: ##.#####"; AverageFluxAlongCore!(16)
PRINT #22, USING " 1/2 XS Flux: ##.#####"; AverageFluxAlongCore!(32)
PRINT #22, USING " 3/4 XS Flux: ##.#####"; AverageFluxAlongCore!(48)
PRINT #22, USING " Max XS Flux: ##.#####"; AverageFluxAlongCore!(64)
END SUB
'*****SNTB - Window 7*****
SUB DisplaySNTB STATIC
WinWidth% = WINDOW(2)
WinHeight% = WINDOW(3)
IF WinWidth% <> OldWinWidth% OR WinHeight% <> OldWinHeight% THEN
CLS
OldWinWidth% = WinWidth%
OldWinHeight% = WinHeight%
END IF
i% = SNTBBitCode% AND 127
i% = (i% \ StepI%) * StepI% 'Find nearest active plane
IF j% = 0 THEN j% = StepJ%
IF i% = 32 THEN i% = 32 - StepI%
IF SNTBBitCode% AND 2048 THEN CLS
IF SNTBBitCode% AND 128 THEN CALL DisplayShapeSNTB (i%)
IF SNTBBitCode% AND 256 THEN CALL DisplayPresSNTB(i%)
IF SNTBBitCode% AND 512 THEN CALL DisplayFluxSNTB(i%)
IF SNTBBitCode% AND 1024 THEN CALL DisplayErrorFluxSNTB(i%)
END SUB
SUB DisplayPresSNTB (i%) STATIC
CALL GetWindowPositionScale (k0%, dk%, j0%, dj%)
FOR j% = 0 TO 32 STEP StepJ%
FOR k% = 0 TO 64 STEP StepK%

```

```

        PSET (k0% + 2 * k% * dk%, j0% + 2 * j% * dj%)
        r% = dj% * SQR((P!(i%, j%, k%) + 1))
        CIRCLE STEP (0,0),r%
    NEXT
NEXT
END SUB
SUB DisplayErrorFluxSNTB (i%) STATIC
    CALL GetWindowPositionScale (k0%, dk%, j0%, dj%)
    Headsize% = dj% \ 2
    maxlength% = CSNG(dj%) * FatArrowLimit!
    FOR j% = StepJ% TO 31 STEP StepJ%
        FOR k% = StepK% TO 63 STEP StepK%
            CALL GetPermCode(i%, j%, k%, Code%)
            IF Code% = 3 THEN
                'Don't display flux for void region
            ELSE
                PSET (k0% + 2 * k% * dk%, j0% + 2 * j% * dj%)
                CALL GetFlux(i%, j%, k%, Flux!())
                ErrorFlux! = Flux!(1) + Flux!(2) + Flux!(3) +
                    Flux!(4) + Flux!(5) + Flux!(6)
                r% = dj% * ABS(ErrorFlux!) * 100
                IF ErrorFlux! > 0 THEN
                    CIRCLE STEP (0, 0), r%
                    LINE STEP (0, -r%) -STEP (0, 2 * r%)
                ELSE
                    CIRCLE STEP (0, 0), r%
                    LINE STEP (-r%, 0) -STEP (2 * r%, 0)
                END IF
            END IF
        NEXT
    NEXT
END SUB
SUB DisplayFluxSNTB (i%) STATIC
    CALL GetWindowPositionScale (k0%, dk%, j0%, dj%)
    Headsize% = dj% \ 2
    maxlength% = CSNG(dj%) * FatArrowLimit!
    FOR j% = StepJ% TO 31 STEP StepJ%
        FOR k% = StepK% TO 63 STEP StepK%
            CALL GetPermCode(i%, j%, k%, Code%)
            IF Code% = 3 THEN
                'Don't display flux for void region
            ELSE
                MOVETO k0% + 2 * k% * dk%, j0% + 2 * j% * dj%
                CALL GetFlux(i%, j%, k%, Flux!())
                jFlux! = Flux!(3) - Flux!(4)
                kFlux! = Flux!(5) - Flux!(6)
                kk% = kFlux! * dk% * FluxScale!
                jj% = jFlux! * dj% * FluxScale!
                Arrow kk%, jj%, Headsize%, maxlength%
            END IF
        NEXT
    NEXT
END SUB
SUB DisplayShapeSNTB (i%) STATIC
    CALL GetWindowPositionScale (k0%, dk%, j0%, dj%)
    FOR j% = 1 TO 31
        FOR k% = 1 TO 63
            rect%(1) = k0% + k% * 2 * dk% - dk%
            rect%(0) = j0% + j% * 2 * dj% - dj%
            rect%(3) = k0% + k% * 2 * dk% + dk%
            rect%(2) = j0% + j% * 2 * dj% + dj%
            CALL GetPermCode(i%, j%, k%, Code%)
            CALL FILLRECT(VARPTR(rect%()), VARPTR(Patterns%(Code%*4)))
        NEXT
    NEXT
END SUB
'*****South - North - West - East*****
SUB DisplaySNWE STATIC

```

```

WinWidth% = WINDOW(2)
WinHeight% = WINDOW(3)
IF WinWidth% <> OldWinWidth% OR WinHeight% <> OldWinHeight% THEN
  CLS
  OldWinWidth% = WinWidth%
  OldWinHeight% = WinHeight%
END IF
j% = SNWEBitCode% AND 127
j% = (j% \ StepJ%) * StepJ% 'Find nearest active plane
IF j% = 32 THEN j% = 32 - StepJ%
IF j% = 0 THEN j% = StepJ%
IF SNWEBitCode% AND 2048 THEN CLS
IF SNWEBitCode% AND 128 THEN CALL DisplayShapeSNWE (j%)
IF SNWEBitCode% AND 256 THEN CALL DisplayPresSNWE(j%)
IF SNWEBitCode% AND 512 THEN CALL DisplayFluxSNWE(j%)
IF SNWEBitCode% AND 1024 THEN CALL DisplayErrorFluxSNWE(j%)
END SUB
SUB DisplayPresSNWE (j%) STATIC
  CALL GetWindowPositionScale (k0%, dk%, i0%, di%)
  FOR i% = 0 TO 32 STEP StepI%
    FOR k% = 0 TO 64 STEP StepK%
      PSET (k0% + 2 * k% * dk%, i0% + 2 * i% * di%)
      r% = di% * SQR((P!(i%, j%, k%) + 1))
      CIRCLE STEP (0,0),r%
    NEXT
  NEXT
END SUB
SUB DisplayErrorFluxSNWE (j%) STATIC
  CALL GetWindowPositionScale (k0%, dk%, i0%, di%)
  Headsize% = di% \ 2
  maxlength% = CSNG(di%) * FatArrowLimit!
  FOR i% = StepI% TO 31 STEP StepI%
    FOR k% = StepK% TO 63 STEP StepK%
      CALL GetPermCode(i%, j%, k%, Code%)
      IF Code% = 3 THEN
        'Don't display flux for void region
      ELSE
        PSET (k0% + 2 * k% * dk%, i0% + 2 * i% * di%)
        CALL GetFlux(i%, j%, k%, Flux!())
        ErrorFlux! = Flux!(1) + Flux!(2) + Flux!(3) +
          Flux!(4) + Flux!(5) + Flux!(6)
        r% = di% * ABS(ErrorFlux!) * 100
        IF ErrorFlux! > 0 THEN
          CIRCLE STEP (0, 0), r%
          LINE STEP (0, -r%) -STEP (0, 2 * r%)
        ELSE
          CIRCLE STEP (0, 0), r%
          LINE STEP (-r%, 0) -STEP (2 * r%, 0)
        END IF
      END IF
    NEXT
  NEXT
END SUB
SUB DisplayFluxSNWE (j%) STATIC
  CALL GetWindowPositionScale (k0%, dk%, i0%, di%)
  Headsize% = di% \ 2
  maxlength% = CSNG(di%) * FatArrowLimit!
  FOR i% = StepI% TO 31 STEP StepI%
    FOR k% = StepK% TO 63 STEP StepK%
      CALL GetPermCode(i%, j%, k%, Code%)
      IF Code% = 3 THEN
        'Don't display flux for void region
      ELSE
        MOVETO k0% + 2 * k% * dk%, i0% + 2 * i% * di%
        CALL GetFlux(i%, j%, k%, Flux!())
        iFlux! = Flux!(1) - Flux!(2)
        kFlux! = Flux!(5) - Flux!(6)
        kk% = kFlux! * dk% * FluxScale!
      END IF
    NEXT
  NEXT
END SUB

```

```

        ii% = iFlux! * di% * FluxScale!
        Arrow kk%, ii%, Headsize%, maxlength%
    END IF
NEXT
NEXT
END SUB
SUB DisplayShapeSNWE (j%) STATIC
    CALL GetWindowPositionScale (k0%, dk%, i0%, di%)
    FOR i% = 1 TO 31
        FOR k% = 1 TO 63
            rect%(1) = k0% + k% * 2 * dk% - dk%
            rect%(0) = i0% + i% * 2 * di% - di%
            rect%(3) = k0% + k% * 2 * dk% + dk%
            rect%(2) = i0% + i% * 2 * di% + di%
            CALL GetPermCode(i%, j%, k%, Code%)
            CALL FILLRECT(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
        NEXT
    NEXT
END SUB
'*****
SUB GetWindowPositionScale (k0%, dk%, j0%, dj%) STATIC
    WinWidth% = WINDOW(2)
    WinHeight% = WINDOW(3)
    j0% = WinHeight% \ 6
    SampleLen% = j0% * 8
    IF SampleLen% > WinWidth% THEN
        k0% = j0%
    ELSE
        k0% = (WinWidth% - SampleLen%) \ 2
        IF k0% > j0% THEN
            k0% = j0%
        ELSEIF k0% < 5 THEN
            k0% = 5
        END IF
    END IF
    dj% = j0% \ 16
    dk% = dj%
END SUB
'*****Display WETB Code*****
SUB DisplayWETB STATIC
    IF StepK% = 64 THEN EXIT SUB
    WinWidth% = WINDOW(2)
    WinHeight% = WINDOW(3)
    IF WinWidth% <> OldWinWidth% OR WinHeight% <> OldWinHeight% THEN
        CLS
        OldWinWidth% = WinWidth%
        OldWinHeight% = WinHeight%
    END IF
    k% = WETBBitCode% AND 127
    k% = (k% \ StepK%) * StepK% 'Find nearest active plane
    IF k% = 64 THEN k% = 64 - StepK%
    IF k% = 0 THEN k% = StepK%
    IF WETBBitCode% AND 2048 THEN CLS
    IF WETBBitCode% AND 128 THEN CALL DisplayShapeWETB (k%)
    IF WETBBitCode% AND 256 THEN CALL DisplayPresWETB(k%)
    IF WETBBitCode% AND 512 THEN CALL DisplayFluxWETB(k%)
    IF WETBBitCode% AND 1024 THEN CALL DisplayErrorFluxWETB(k%)
END SUB
SUB DisplayPresWETB (k%) STATIC
    CALL GetWindowPositionScale (i0%, di%, j0%, dj%)
    FOR j% = 0 TO 32 STEP StepJ%
        FOR i% = 0 TO 32 STEP StepI%
            PSET (i0% + 2 * i% * di%, j0% + 2 * j% * dj%)
            r% = dj% * SQR((P!(i%, j%, k%) + 1!))
            CIRCLE STEP (0,0), r%
        NEXT
    NEXT
END SUB

```

```

SUB DisplayErrorFluxWETB (k%) STATIC
CALL GetWindowPositionScaleS (i0%, di%, j0%, dj%)
Headsizes% = dj% \ 2
maxlength% = CSNG(dj%) * FatArrowLimit!
FOR j% = StepJ% TO 31 STEP StepJ%
  FOR i% = StepI% TO 31 STEP StepI%
    CALL GetPermCode(i%, j%, k%, Code%)
    IF Code% = 3 THEN
      'Don't display flux for void region
    ELSE
      PSET (i0% + 2 * i% * di%, j0% + 2 * j% * dj%)
      CALL GetFlux(i%, j%, k%, Flux!())
      ErrorFlux! = Flux!(1) + Flux!(2) + Flux!(3) +
        Flux!(4) + Flux!(5) + Flux!(6)
      r% = dj% * ABS(ErrorFlux!) * 100
      IF ErrorFlux! > 0 THEN
        CIRCLE STEP (0, 0), r%
        LINE STEP (0, -r%) -STEP (0, 2 * r%)
      ELSE
        CIRCLE STEP (0, 0), r%
        LINE STEP (-r%, 0) -STEP (2 * r%, 0)
      END IF
    END IF
  NEXT
NEXT
END SUB

SUB DisplayFluxWETB (k%) STATIC
CALL GetWindowPositionScaleS (i0%, di%, j0%, dj%)
Headsizes% = dj% \ 2
maxlength% = CSNG(dj%) * FatArrowLimit!
FOR j% = StepJ% TO 31 STEP StepJ%
  FOR i% = StepI% TO 31 STEP StepI%
    CALL GetPermCode(i%, j%, k%, Code%)
    IF Code% = 3 THEN
      'Don't display flux for void region
    ELSE
      MOVETO i0% + 2 * i% * di%, j0% + 2 * j% * dj%
      CALL GetFlux(i%, j%, k%, Flux!())
      jFlux! = Flux!(3) - Flux!(4)
      iFlux! = Flux!(1) - Flux!(2)
      ii% = iFlux! * di% * FluxScale!
      jj% = jFlux! * dj% * FluxScale!
      Arrow ii%, jj%, Headsizes%, maxlength%
    END IF
  NEXT
NEXT
END SUB

SUB DisplayShapeWETB (k%) STATIC
CALL GetWindowPositionScaleS (i0%, di%, j0%, dj%)
FOR j% = 1 TO 31
  FOR i% = 1 TO 31
    rect%(1) = i0% + i% * 2 * di% - di%
    rect%(0) = j0% + j% * 2 * dj% - dj%
    rect%(3) = i0% + i% * 2 * di% + di%
    rect%(2) = j0% + j% * 2 * dj% + dj%
    CALL GetPermCode(i%, j%, k%, Code%)
    CALL FILLRECT(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
  NEXT
NEXT
END SUB

'*****Display EWTB Code*****
SUB DisplayEWTB STATIC
IF StepK% = 64 THEN EXIT SUB
WinWidth% = WINDOW(2)
WinHeight% = WINDOW(3)
IF WinWidth% <> OldWinWidth% OR WinHeight% <> OldWinHeight% THEN
  CLS
  OldWinWidth% = WinWidth%

```

```

    OldWinHeight% = WinHeight%
END IF
k% = EWTBBitCode% AND 127
k% = (k% \ StepK%) * StepK% 'Find nearest active plane
IF k% = 64 THEN k% = 64 - StepK%
IF k% = 0 THEN k% = StepK%
IF EWTBBitCode% AND 2048 THEN CLS
IF EWTBBitCode% AND 128 THEN CALL DisplayShapeEWTB (k%)
IF EWTBBitCode% AND 256 THEN CALL DisplayPresEWTB(k%)
IF EWTBBitCode% AND 512 THEN CALL DisplayFluxEWTB(k%)
IF EWTBBitCode% AND 1024 THEN CALL DisplayErrorFluxEWTB(k%)
END SUB
SUB DisplayPresEWTB (k%) STATIC
CALL GetWindowPositionScaleS (i0%, di%, j0%, dj%)
FOR j% = 0 TO 32 STEP StepJ%
    FOR i% = 0 TO 32 STEP StepI%
        PSET (i0% + 2 * (32 - i%) * di%, j0% + 2 * j% * dj%)
        r% = dj% * SQR((P!(i%, j%, k%) + 1))
        CIRCLE STEP (0,0),r%
    NEXT
NEXT
END SUB
SUB DisplayErrorFluxEWTB (k%) STATIC
CALL GetWindowPositionScaleS (i0%, di%, j0%, dj%)
Headsize% = dj% \ 2
maxlength% = CSNG(dj%) * FatArrowLimit!
FOR j% = StepJ% TO 31 STEP StepJ%
    FOR i% = StepI% TO 31 STEP StepI%
        CALL GetPermCode(i%, j%, k%, Code%)
        IF Code% = 3 THEN
            'Don't display flux for void region
        ELSE
            PSET (i0% + 2 * (32 - i%) * di%, j0% + 2 * j% * dj%)
            CALL GetFlux(i%, j%, k%, Flux!())
            ErrorFlux! = Flux!(1) + Flux!(2) + Flux!(3) +
                        Flux!(4) + Flux!(5) + Flux!(6)
            r% = dj% * ABS(ErrorFlux!) * 100
            IF ErrorFlux! > 0 THEN
                CIRCLE STEP (0, 0), r%
                LINE STEP (0, -r%) -STEP (0, 2 * r%)
            ELSE
                CIRCLE STEP (0, 0), r%
                LINE STEP (-r%, 0) -STEP (2 * r%, 0)
            END IF
        END IF
    NEXT
NEXT
END SUB
SUB DisplayFluxEWTB (k%) STATIC
CALL GetWindowPositionScaleS (i0%, di%, j0%, dj%)
Headsize% = dj% \ 2
maxlength% = CSNG(dj%) * FatArrowLimit!
FOR j% = StepJ% TO 31 STEP StepJ%
    FOR i% = StepI% TO 31 STEP StepI%
        CALL GetPermCode(i%, j%, k%, Code%)
        IF Code% = 3 THEN
            'Don't display flux for void region
        ELSE
            MOVETO i0% + 2 * (32 - i%) * di%, j0% + 2 * j% * dj%
            CALL GetFlux(i%, j%, k%, Flux!())
            jFlux! = Flux!(3) - Flux!(4)
            iFlux! = Flux!(1) - Flux!(2)
            ii% = iFlux! * di% * FluxScale!
            jj% = jFlux! * dj% * FluxScale!
            Arrow ii%, jj%, Headsize%, maxlength%
        END IF
    NEXT
NEXT
END SUB

```

```

END SUB
SUB DisplayShapeEWTB (k%) STATIC
CALL GetWindowPositionScales (i0%, di%, j0%, dj%)
FOR j% = 1 TO 31
  FOR i% = 1 TO 31
    rect%(1) = i0% + (32 - i%) * 2 * di% - di%
    rect%(0) = j0% + j% * 2 * dj% - dj%
    rect%(3) = i0% + (32 - i%) * 2 * di% + di%
    rect%(2) = j0% + j% * 2 * dj% + dj%
    CALL GetPermCode(i%, j%, k%, Code%)
    CALL FILLRECT(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
  NEXT
NEXT
END SUB
'*****
SUB GetWindowPositionScales (k0%, dk%, j0%, dj%) STATIC
WinWidth% = WINDOW(2)
WinHeight% = WINDOW(3)
j0% = WinHeight% \ 8
SampleWidth% = j0% * 4
IF SampleWidth% > WinWidth% THEN
  k0% = j0%
ELSE
  k0% = (WinWidth% - SampleWidth%) \ 2
  IF k0% > j0% THEN
    k0% = j0%
  ELSEIF k0% < 5 THEN
    k0% = 5
  END IF
END IF
dj% = j0% \ 16
di% = dj%
END SUB
'Code for control of the numerical tasks
SUB TaskDispatcher STATIC
  SELECT CASE TaskWaiting%
    CASE 1 'Iterate a specified number of times
      ChangeCursor 4
      DefineMenu 1 'Define UserInteraction Menu
      MaxNumOfIts% = StoredNumOfIts% 'Retrieve a global variable
      Iterate MaxNumOfIts%, IterateExitCode%
      SELECT CASE IterateExitCode%
        CASE 1, 2, 3 'Normal, Max Its, or Calculation Halted by user
          TaskWaiting% = False%
          IF Halt% = True% THEN WINDOW CLOSE 16
          Halt% = False%
          DefineMenu 0
          INITCURSOR
          BEEP
        CASE -1 'Exit to handle event then reenter
      END SELECT
    CASE 2 'Iterate n times then refine mesh, repeat until done
      ChangeCursor 4
      DefineMenu 1 'Define UserInteraction Menu
      MaxNumOfIts% = StoredNumOfIts% 'Retrieve a global variable
      ExitCode% = 0
      WHILE ExitCode% = 0
        Iterate MaxNumOfIts%, IterateExitCode%
        SELECT CASE IterateExitCode%
          CASE 1, 2 'Converged or Max Num of Iterations
            CalcFileWrite
            DetermineAverageFluxAlongCore
            IF AutomaticPrint% = True% THEN CALL PrintInfoPlotCombo
            RefineMesh
          CASE 3 'Calculation Halted by user
            ExitCode% = IterateExitCode%
            TaskWaiting% = False%
            Halt% = False%

```

```

        DefineMenu 0
        INITCURSOR
        WINDOW CLOSE 16          'Clear halt message of screen
        BEEP
        CASE -1                  'Exit to handle event then reenter
            ExitCode% = IterateExitCode%
        END SELECT
    WEND
CASE ELSE
END SELECT
END SUB
SUB PrintInfoPlotCombo STATIC
    WinOpen%(5) = True%
    WINDOW 5
    PICTURE ON : SHOWPEN          'begin recording
        RefreshWindow 5          'make image
    PICTURE OFF                  'end recording
    ScreenImageOfAvgFluxPlot$ = PICTURE$
    WinOpen%(11) = True%
    WINDOW 11
    PICTURE ON : SHOWPEN          'begin recording
        RefreshWindow 11        'make image
    PICTURE OFF                  'end recording
    ScreenImageOfGeneralInfo$ = PICTURE$
    CLOSE #22
    OPEN "lpt1:" FOR OUTPUT AS #22
    WINDOW OUTPUT #22
    PICTURE (10,10)-(541,330),ScreenImageOfAvgFluxPlot$
    PICTURE (10,340),ScreenImageOfGeneralInfo$
    ScreenImageOfAvgFluxPlot$ = ""
    ScreenImageOfGeneralInfo$ = ""
    CLOSE # 22
    OPEN "SCRN:" FOR OUTPUT AS #22
    RefreshWindow 11
    RefreshWindow 5
END SUB
'Numerical Code
'*****Successive Over Relaxation Code*****
SUB Iterate (MaxNumOfIts%, ExitCode%) STATIC
    'Repeatedly conducts the SOR iterative solution for the pressure
    'field until either the solution is converged or the maximum
    'number of iterations are reached while allowing user interaction
    'On first entry k% = 0, It% = 0
    ExitCode% = 0
    WHILE ExitCode% = 0
        k% = k% + StepK%
        SubIterate k%, ErrorAccum!
        IF k% = 64 - StepK% THEN
            It% = It% + 1
            ItsAtLevel% = ItsAtLevel% + 1
            k% = 0
            TestForConvergence ErrorAccum!, IsConverged%
            IF Halt% = True% THEN
                'Semaphore set to halt calculation      'Must be highest priority!
                It% = 0
                ExitCode% = 3
            ELSEIF IsConverged% = True% THEN
                'Normal Exit
                It% = 0
                ExitCode% = 1
            ELSEIF It% >= MaxNumOfIts% THEN
                'Exit with MaxIts exceeded
                It% = 0
                ExitCode% = 2
            END IF
            ErrorAccum! = 0!
            DetermineAverageFluxAlongCoreAtQtrs
            UpdateActiveDisplays
        END WHILE

```



```

ELSE
  UserInteractionManager UserInteraction%
  IF UserInteraction% THEN
    'Exit to handle event, then reenter calculation
    ExitCode% = -1
  END IF
END IF
WEND
END SUB
SUB SubIterate(k%, ErrorAccum!) STATIC
  'Implements the SOR pass at a given k value.
  ' 7 Point (3-d) Conservative Operator
  FOR i% = StepI% TO 31 STEP StepI%          'StepI% is global shared
    FOR j% = StepJ% TO 31 STEP StepJ%        'StepJ% is global shared
      CALL GetPermCode(i%, j%, k%, Code%)
      IF Code% <> 3 THEN                      'Except for void calculate SOR iteration
        Perm0! = MatPerm!(Code%)
        Accum1! = 0!
        Accum2! = 0!
        FOR w% = 1 TO 6
          iz% = i% + OffsetI%(w%) * StepI%
          jz% = j% + OffsetJ%(w%) * StepJ%
          kz% = k% + OffsetK%(w%) * StepK%
          LenScale% = i% - iz% + j% - jz% + k% - kz%
          LSM2! = 1! / CSNG(LenScale% * LenScale%)
          CALL GetPermCode(iz%, jz%, kz%, Code%)
          Perm1! = MatPerm!(Code%)
          Perm! = (Perm1! * Perm0!) / (Perm1! + Perm0!)
          d! = Perm! * LSM2!
          Accum1! = Accum1! + P!(iz%, jz%, kz%) * d!
          Accum2! = Accum2! + d!
        NEXT
        'SOR is a global shared variable
        Pnew! = Accum1!/Accum2!
        ErrorAccum! = ErrorAccum! + ABS(Pnew! - P!(i%, j%, k%))
        P!(i%, j%, k%) = SOR! * Pnew! + (1! - SOR!) * P!(i%, j%, k%)
      ELSE
        IF k% < 20 THEN                      'For Void use no resistance
          P!(i%, j%, k%) = -1!
        ELSE
          P!(i%, j%, k%) = 1
        END IF
      END IF
    NEXT j%
  NEXT i%
NEXT
END SUB
'*****Mesh Management code*****
SUB CoarsenMesh STATIC
  SELECT CASE MeshLevel%
    CASE 2, 5, 8, 11, 14
      StepK% = 2 * StepK%
    CASE 3, 6, 9, 12
      StepJ% = 2 * StepJ%
    CASE 4, 7, 10, 13
      StepI% = 2 * StepI%
    CASE 1
      MeshLevel% = 2
    CASE ELSE
      STOP 'Error occurred
  END SELECT
  MeshLevel% = MeshLevel% -1
  ItsAtLevel% = 0
  CALL UpdateActiveDisplays          'Should reflect new mesh immediately
END SUB
SUB RefineMesh STATIC
  MeshLevel% = MeshLevel% +1
  SELECT CASE MeshLevel%
    CASE 1, 2, 5, 8, 11, 14

```

```

        CALL KMeshRefinement
        ItsAtLevel% = 0
    CASE 3, 6, 9, 12
        CALL JMeshRefinement
        ItsAtLevel% = 0
    CASE 4, 7, 10, 13
        CALL IMeshRefinement
        ItsAtLevel% = 0
    CASE 15
        MeshLevel% = 14
    CASE ELSE
        STOP 'Error occurred'
END SELECT
CALL UpdateActiveDisplays          'Should reflect new mesh immediately
END SUB
SUB IMeshRefinement STATIC
    'First, use linear interpolation (averaging) to determine pressure
    'at new points. The linear interpolation includes the effect
    'of different permeabilities.
    HalfStep% = StepI% \ 2
    FOR i% = 0 TO 31 STEP StepI%
        FOR j% = StepJ% TO 31 STEP StepJ%
            FOR k% = StepK% TO 63 STEP StepK%
                CALL GetPermCode(i% + HalfStep%, j%, k%, Code1%)
                IF Code1% = 3 THEN
                    IF k% > 20 THEN
                        P!(i% + HalfStep%, j%, k%) = 1
                    ELSE
                        P!(i% + HalfStep%, j%, k%) = -1
                    END IF
                ELSE
                    CALL GetPermCode(i%, j%, k%, Code0%)
                    CALL GetPermCode(i% + StepI%, j%, k%, Code2%)
                    K1! = MatPerm!(Code0%) * MatPerm!(Code1%)
                    K1! = K1! / (MatPerm!(Code0%) + MatPerm!(Code1%))
                    K2! = MatPerm!(Code1%) * MatPerm!(Code2%)
                    K2! = K2! / (MatPerm!(Code1%) + MatPerm!(Code2%))
                    Pt! = K1! * P!(i%, j%, k%) + K2! * P!(i% + StepI%, j, k%)
                    P!(i% + HalfStep%, j%, k%) = Pt! / (K1! + K2!)
                END IF
            NEXT k%
        NEXT j%
    NEXT i%
    StepI% = HalfStep%
END SUB
SUB JMeshRefinement STATIC
    'First, use linear interpolation (averaging) to determine pressure
    'at new points
    HalfStep% = StepJ% \ 2
    FOR i% = StepI% TO 31 STEP StepI%
        FOR j% = 0 TO 31 STEP StepJ%
            FOR k% = StepK% TO 63 STEP StepK%
                CALL GetPermCode(i%, j% + HalfStep%, k%, Code1%)
                IF Code1% = 3 THEN
                    IF k% > 20 THEN
                        P!(i%, j% + HalfStep%, k%) = 1
                    ELSE
                        P!(i%, j% + HalfStep%, k%) = -1
                    END IF
                ELSE
                    CALL GetPermCode(i%, j%, k%, Code0%)
                    CALL GetPermCode(i%, j% + StepJ%, k%, Code2%)
                    K1! = MatPerm!(Code0%) * MatPerm!(Code1%)
                    K1! = K1! / (MatPerm!(Code0%) + MatPerm!(Code1%))
                    K2! = MatPerm!(Code1%) * MatPerm!(Code2%)
                    K2! = K2! / (MatPerm!(Code1%) + MatPerm!(Code2%))
                    Pt! = K1! * P!(i%, j%, k%) + K2! * P!(i%, j% + StepJ%, k%)
                    P!(i%, j% + HalfStep%, k%) = Pt! / (K1! + K2!)
                END IF
            NEXT k%
        NEXT j%
    NEXT i%

```

```

        END IF
    NEXT
NEXT
NEXT
StepJ% = HalfStep%
END SUB
SUB KMeshRefinement STATIC
    HalfStep% = StepK% \ 2
    FOR i% = StepI% TO 31 STEP StepI%
        FOR j% = StepJ% TO 31 STEP StepJ%
            FOR k% = 0 TO 63 STEP StepK%
                CALL GetPermCode(i%, j%, k% + HalfStep%, Code1%)
                IF Code1% = 3 THEN
                    IF k% > 20 THEN
                        P!(i%, j%, k% + HalfStep%) = 1
                    ELSE
                        P!(i%, j%, k% + HalfStep%) = -1
                    END IF
                ELSE
                    CALL GetPermCode(i%, j%, k%, Code0%)
                    CALL GetPermCode(i%, j%, k% + StepK%, Code2%)
                    K1! = MatPerm!(Code0%) * MatPerm!(Code1%)
                    K1! = K1! / (MatPerm!(Code0%) + MatPerm!(Code1%))
                    K2! = MatPerm!(Code1%) * MatPerm!(Code2%)
                    K2! = K2! / (MatPerm!(Code1%) + MatPerm!(Code2%))
                    Pt! = K1! * P!(i%, j%, k%) + K2! * P!(i%, j%, k% + StepK%)
                    P!(i%, j%, k% + HalfStep%) = Pt! / (K1! + K2!)
                END IF
            NEXT
        NEXT
    NEXT
    StepK% = HalfStep%
END SUB
'*****
SUB TestForConvergence (ErrorAccum!, IsConverged%) STATIC
    'Tol! is a global variable
    Scale! = (64 \ StepK%) * (32 \ StepI%) * (32 \ StepJ%)
    TotErr! = ErrorAccum!
    RelErr! = ErrorAccum! / Scale!
    IF RelErr! < Tol! THEN IsConverged% = True% ELSE IsConverged% = False%
END SUB
'Utility Sub Routines
SUB ParseFileName (FullyValidatedFileName$, ShortFileName$) STATIC
    ShortFileName$ = FullyValidatedFileName$
    Done% = False%
    WHILE Done% = False%
        g% = INSTR(ShortFileName$, ":")
        IF g% > 0 THEN
            ShortFileName$ = MID$(ShortFileName$, g% + 1)
        ELSE
            Done% = True%
        END IF
    WEND
END SUB
SUB InitializeBoundaryConditions STATIC
    ChangeCursor 4
    FOR i% = 0 TO 32
        FOR j% = 0 TO 32
            P!(i%, j%, 0) = -1!
            P!(i%, j%, 64) = +1!
        NEXT
    NEXT
    FOR k% = 1 TO 63
        a! = CSNG(k% - 32) / 32
        FOR Ptr% = 0 TO 32
            P!(Ptr%, 0, k%) = a!
            P!(0, Ptr%, k%) = a!
            P!(Ptr%, 32, k%) = a!
        NEXT
    NEXT

```

```

    P!(32, Ptr%, k%) = a!
NEXT
NEXT
INITCURSOR
END SUB
SUB GetFlux(i%, j%, k%, FluxOut!()) STATIC
'Calculates the Flux at a given node
    CALL GetPermCode(i%, j%, k%, Code%)
    Perm0! = MatPerm!(Code%)
    Accum1! = 0!
    Accum2! = 0!
    FOR w% = 1 TO 6
        iz% = i% + OffsetI%(w%) * StepI%
        jz% = j% + OffsetJ%(w%) * StepJ%
        kz% = k% + OffsetK%(w%) * StepK%
        ls% = iz% + jz% + kz% - i% - j% - k%
        RL! = 1! / ABS(ls%)
        CALL GetPermCode(iz%, jz%, kz%, Code%)
        Perm1! = MatPerm!(Code%)
        Perm! = (Perm1! * Perm0!) / (Perm1! + Perm0!)
        FluxOut!(w%) = Perm! * (P!(iz%, jz%, kz%) - P!(i%, j%, k%)) * RL!
    NEXT
END SUB
SUB GetPermCode (i%, j%, k%, Code%) STATIC
    Segment% = i% \ 4
    Offset% = i% MOD 4
    IF Segment% > 8 THEN
        Code% = 999
        EXIT SUB
    END IF
    Old% = PermCodeArray%(Segment%, j%, k%)
    IF Old% < 0 THEN Old% = Old% + 65536%
    SELECT CASE Offset%
        CASE 0
            Mask% = &HF
            Shift% = 1
        CASE 1
            Mask% = &HFO
            Shift% = 16
        CASE 2
            Mask% = &HF00
            Shift% = 256
        CASE 3
            Mask% = &HF000
            Shift% = 4096
    END SELECT
    Code% = (Old% AND Mask%) \ Shift%
END SUB
'Start up, Misc Code
SUB SetDefaultValues STATIC
'Defaults
    CalcFileName$ = "Untitled - Calc"
    ChangesSaved% = True%
    StepI% = 16
    StepJ% = 16
    StepK% = 64
    SOR! = 1.5
    Tol! = .000001
    MeshLevel% = 0
    FluxScale! = 25!
    FatArrowLimit! = 2!
    MatPerm!(0) = .00001 'Boundary
    MatPerm!(1) = 10! 'Char
    MatPerm!(2) = .1 'Shale
    MatPerm!(3) = 10! 'Void
    MatPerm!(4) = 1! 'Coal
    MatPerm!(5) = .05 'Concrete
    MatPerm!(6) = 1

```

```

MatPerm!(7) = 1
MatPerm!(8) = 1
MatPerm!(9) = 1
MatPerm!(10) = 1
MatPerm!(11) = 1
MatPerm!(12) = 1
MatPerm!(13) = 1
MatPerm!(14) = 1
MatPerm!(15) = 1
DefaultI% = 16
DefaultJ% = 16
DefaultK% = 32
END SUB
SUB StartUpCode STATIC
False% = 0 : True% = NOT False%
SetArray OffsetI%(1), -1, 1, 0, 0, 0, 0
SetArray OffsetJ%(1), 0, 0, -1, 1, 0, 0
SetArray OffsetK%(1), 0, 0, 0, 0, -1, 1
SetArray Patterns%(4), &H800, &H20, &H200, &H0
SetArray Patterns%(8), &H8822, &H8822, &H8822, &H8822
SetArray Patterns%(12), &H0, &H0, &H0, &H0
SetArray Patterns%(16), &H9002, &H48, &H220, &H8400
SetArray Patterns%(20), &HAA55, &HAA55, &HAA55, &HAA55
OPEN "SCRN:" FOR OUTPUT AS #22
END SUB
SUB FixUpPermCodeArray STATIC
FOR i% = 1 TO 7
  FOR j% = 1 TO 31
    PermCodeArray%(i%, j%, 0) = PermCodeArray%(i%, j%, 1)
    PermCodeArray%(i%, j%, 64) = PermCodeArray%(i%, j%, 63)
  NEXT
NEXT
END SUB
SUB Arrow (dx%, dy%, size%, maxlength%) STATIC
dx& = dx%
dy& = dy%
LineLength& = SQR(dx& * dx& + dy& * dy&)
IF LineLength& < maxlength% THEN
  PENSIZE 1,1
ELSEIF LineLength& \ 4 < maxlength% THEN
  dx% = dx% \ 4
  dy% = dy% \ 4
  LineLength& = LineLength& \ 4
  PENSIZE 2,2
ELSEIF LineLength& \ 9 < maxlength% THEN
  dx% = dx% \ 9
  dy% = dy% \ 9
  LineLength& = LineLength& \ 9
  PENSIZE 3,3
ELSE
  dx% = dx% \ 16
  dy% = dy% \ 16
  LineLength& = LineLength& \ 16
  PENSIZE 4,4
END IF
IF LineLength& < size% THEN LineLength& = size%
Adx% = (size% * dx%) \ LineLength&
Ady% = (size% * dy%) \ LineLength&
CALL LINE (dx%, dy%)
CALL LINE (-Adx% + Ady% \ 2, -Ady% - Adx% \ 2)
CALL MOVE (-Ady%, Adx%)
CALL LINE (+Adx% + Ady% \ 2, + Ady% - Adx% \ 2)
END SUB
'*****Routine for Average Flux*****
SUB PlotAverageFluxAlongCore STATIC
IF AverageFluxAlongCore!(0) > 0! THEN yb! = 0! ELSE _

```

```

'Char
'Shale
'Void
'Coal
'Cement

```

```

        yb! = AverageFluxAlongCore!(0)
IF AverageFluxAlongCore!(64) < 0 THEN yt! = 0! ELSE _
        yt! = AverageFluxAlongCore!(64)
IF yt! = yb! THEN EXIT SUB
GetWindowPositionScale k0%, dk%, j0%, dj%
'Draw Box
xMin% = k0%
xMax% = k0% + 2 * 64 * dk%
yMax% = j0% + 2 * 34 * dj%
yMin% = j0% + 2 * 40 * dj%
LINE (xMin%, yMin% + dj%) - (xMax%, yMax% - dj%), , B
IF AverageFluxAlongCore!(0) > 0! THEN yb! = 0! ELSE _
        yb! = AverageFluxAlongCore!(0)
IF AverageFluxAlongCore!(64) < 0 THEN yt! = 0! ELSE _
        yt! = AverageFluxAlongCore!(64)
Slope! = CSNG(yMax% - yMin%) / (yt! - yb!)
'Draw the zero flux line
y0% = yMin% - yb! * Slope!
PENSIZE 2,2
MOVETO xMin%, y0%
LINETO xMax%, y0%
x% = k0% + 2 * dk%
y% = yMin% + (AverageFluxAlongCore!(1) - yb!) * Slope!
MOVETO x%, y%
FOR k% = 2 TO 63
    'Map x%
    x% = k0% + 2 * k% * dk%
    'Map y%
    y% = yMin% + (AverageFluxAlongCore!(k%) - yb!) * Slope!
    LINETO x%, y%
NEXT
PENNORMAL
END SUB
SUB PlotAverageFluxAlongCoreForExport STATIC
CLS
IF AverageFluxAlongCore!(0) > 0! THEN yb! = 0! ELSE _
        yb! = AverageFluxAlongCore!(0)
IF AverageFluxAlongCore!(64) < 0 THEN yt! = 0! ELSE _
        yt! = AverageFluxAlongCore!(64)
IF yt! = yb! THEN EXIT SUB
'Draw Box
xMin% = 30
xMax% = WINDOW(2) - 30
yMax% = 30
yMin% = WINDOW(3) - 30
LINE (xMin%, yMin% + 10) - (xMax%, yMax% - 10), , B
IF AverageFluxAlongCore!(0) > 0! THEN yb! = 0! ELSE _
        yb! = AverageFluxAlongCore!(0)
IF AverageFluxAlongCore!(64) < 0 THEN yt! = 0! ELSE _
        yt! = AverageFluxAlongCore!(64)
SlopeY! = CSNG(yMax% - yMin%) / (yt! - yb!)
SlopeX! = CSNG(xMax% - xMin%) / (64! - 0!)
'Draw the zero flux line
y0% = yMin% - yb! * SlopeY!
PENSIZE 2,2
MOVETO xMin%, y0%
LINETO xMax%, y0%
x% = xMin% + (StepK%) * SlopeX!
y% = yMin% + (AverageFluxAlongCore!(StepK%) - yb!) * SlopeY!
MOVETO x%, y%
FOR k% = 2 * StepK% TO 63 STEP StepK%
    x% = xMin% + k% * SlopeX!
    IF AverageFluxAlongCore!(k%) <> 0 THEN
        y% = yMin% + (AverageFluxAlongCore!(k%) - yb!) * SlopeY!
        LINETO x%, y%
    END IF
NEXT
PENNORMAL

```

```

END SUB
SUB DetermineAverageFluxAlongCore STATIC
  ChangeCursor 4
  CrossSectionArea! = 31!*31! * 3.1414 /4! 'Exclude the surrounding concrete
  AreaPerNode! = CSNG(StepI%) * CSNG(StepJ%)
  MinAFAC! = 1E+23
  MaxAFAC! = -1E+23
  FOR k% = StepK% TO 63 STEP StepK%
    Sum! = 0!
    FOR i% = StepI% TO 31 STEP StepI%
      FOR j% = StepJ% TO 31 STEP StepJ%
        GetFlux i%, j%, k%, Flux! ( )
        Sum! = Sum! + (Flux!(5) - Flux!(6))
      NEXT j%
    NEXT i%
    AverageFluxAlongCore!(k%) = Sum! * AreaPerNode! / CrossSectionArea!
    IF MinAFAC! > AverageFluxAlongCore!(k%) THEN
      MinAFAC! = AverageFluxAlongCore!(k%)
    IF MaxAFAC! < AverageFluxAlongCore!(k%) THEN
      MaxAFAC! = AverageFluxAlongCore!(k%)
  NEXT k%
  AverageFluxAlongCore!(0) = MinAFAC!
  AverageFluxAlongCore!(64) = MaxAFAC!
  INITCURSOR
  BEEP
END SUB

SUB DetermineAverageFluxAlongCoreAtQtrs STATIC
  ChangeCursor 4
  CrossSectionArea! = 31!*31! * 3.1414 /4! 'Exclude the surrounding concrete
  AreaPerNode! = CSNG(StepI%) * CSNG(StepJ%)
  FOR k% = 16 TO 48 STEP 16
    Sum! = 0!
    FOR i% = StepI% TO 31 STEP StepI%
      FOR j% = StepJ% TO 31 STEP StepJ%
        GetFlux i%, j%, k%, Flux! ( )
        Sum! = Sum! + (Flux!(5) - Flux!(6))
      NEXT j%
    NEXT i%
    AverageFluxAlongCore!(k%) = Sum! * AreaPerNode! / CrossSectionArea!
  NEXT k%
END SUB

SUB HandleEditMenu STATIC
  SELECT CASE MENU(1)
    CASE 4 'Edit - Copy Active Window to Clipboard
      BEEP : BEEP
      ActiveWin% = WINDOW(0)
      SELECT CASE ActiveWin%
        CASE 5, 7, 8, 9, 10 'Graphics Windows
          'WINDOW 7, "South - North - Top - Bottom"
          'WINDOW 8, "South - North - West - East"
          'WINDOW 9, "West - East - Top - Bottom"
          'WINDOW 10, "East - West - Top - Bottom"
          PICTURE ON : SHOWPEN 'begin recording
          RefreshWindow ActiveWin% 'make image
          PICTURE OFF 'end recording
          ScreenImage$ = PICTURE$
          CLOSE #22
          OPEN "clip:picture" FOR OUTPUT AS #22
          PRINT #22, ScreenImage$
          CLOSE # 22
          OPEN "SCRN:" FOR OUTPUT AS #22
          PICTURE ON : PICTURE OFF
        CASE 11 'Textwindow
          'WINDOW 11, "General Info"
          CLOSE #22
          OPEN "clip:text" FOR OUTPUT AS #22
          RefreshWindow ActiveWin%
          CLOSE #22

```

```

        OPEN "SCRN:" FOR OUTPUT AS #22
    CASE ELSE
    END SELECT
CASE ELSE
END SELECT
END SUB
SUB PrintActiveWindow STATIC
    BEEP : BEEP
    ActiveWin% = WINDOW(0)
    SELECT CASE ActiveWin%
        CASE 5, 7, 8, 9, 10
            'Graphics Windows
            'WINDOW 7, "South - North - Top - Bottom"
            'WINDOW 8, "South - North - West - East"
            'WINDOW 9, "West - East - Top - Bottom"
            'WINDOW 10, "East - West - Top - Bottom"
            PICTURE ON : SHOWPEN
                RefreshWindow ActiveWin%
                'begin recording
            PICTURE OFF
                'make image
                'end recording
            ScreenImage$ = PICTURE$
            CLOSE #22
            OPEN "lpt1:prompt" FOR OUTPUT AS #22
            WINDOW OUTPUT #22
            PICTURE ,ScreenImage$
            CLOSE # 22
            OPEN "SCRN:" FOR OUTPUT AS #22
            PICTURE ON : PICTURE OFF
        CASE 11
            'Textwindow
            'WINDOW 11, "General Info"
            CLOSE #22
            OPEN "lpt1:prompt" FOR OUTPUT AS #22
            RefreshWindow ActiveWin%
            CLOSE #22
            OPEN "SCRN:" FOR OUTPUT AS #22
            RefreshWindow ActiveWin%
    CASE ELSE
    END SELECT
END SUB

```

C.3 Program COAL CORE SURFACE CALC .BAS

```

' *****Coal Core Channel Surface Flux Calculations*****
'
' Author:      Rich Dobbs
' Date:        12/12/89 - 12/20/89
'              12/27/89
'              (Minor Fixes to Average Flux Determination)
' Purpose:     This is a post processor for the flow calculation program
'              Coal Core Flux Calculations. Transforms the raw data
'              from the pressure field to a normal flux at select points
'              on the channel surface. It also reports various derivatives of this
'              flux.
'
' *****Subroutine Names*****
'
' MainProgramForFrontCalcs
' HandleDialog
' DefineMenu
' HandleMenu
' DisplayTipInfo
' DisplaySurfaceNormFlux
' DisplaySurfaceDNormFlux
' WriteOutStatusWordInfo
' FreeTipNumber
' DetermineActiveView

```



```

' HandleTipGroupMenu
' TipGroupNameDialogBox
' *****Mouse Tool Routines
' HandleMouse
' SelectTheTip
' HandleGetNormalFluxes
' HandleGetNormalFluxesSNTB
' FindNormal
' FindIntersection
' HandleReshapeCurve
' GetCurve
' DrawCurve
' LaGrange
' DefineConstants
' GetTipStatusWord
' SetTipStatusWord
' *****Window Management Routines
' HandleDisplayMenu
' SetFluxScaleDialogBox
' DisplayChoicesDialogBox
' UpdateActiveDisplays
' WindowInitialize
' RefreshWindow
' *****Display Writing Routines
' DisplaySNTB
' DisplayPresSNTB
' DisplayErrorFluxSNTB
' DisplayFluxSNTB
' DisplayShapeSNTB
' DisplaySNWE
' DisplayPresSNWE
' DisplayErrorFluxSNWE
' DisplayFluxSNWE
' DisplayShapeSNWE
' GetWindowPositionScale
' DisplayWETB
' DisplayPresWETB
' DisplayErrorFluxWETB
' DisplayFluxWETB
' DisplayShapeWETB
' DisplayEWTB
' DisplayPresEWTB
' DisplayErrorFluxEWTB
' DisplayFluxEWTB
' DisplayShapeEWTB
' GetWindowPositionScaleS
' *****Misc Routines
' GetFlux
' GetPermCode
' SetDefaultValues
' StartUpCode
' fixUpPermCodeArray
' Arrow
' *****File Management Routine
' HandleFileMenu
' FileFailureAlertBox
' ParseFileName
' MakeNewName
' ShapeDefFileRead
' CalcFileRead
' SearchForMagic
' SaveChangesInquiry
' *****
CLEAR , 600000&
' *****Global Constants*****
DIM SHARED True%, False%, Defined%, NotDefined%
DIM SHARED SNTB%, SNWE%, WETB%, EWTB%

```

```

DIM SHARED NoPoints%
DIM SHARED Curve%, Fluxes%, TipPoint%, Calculation%
'*****Global Declarations*****
'*****Inheirited Declarations*****
'Global Declarations for Data from shape definition file
DIM SHARED PermCodeArray%(8, 32, 64)
DIM SHARED CurMatCode%, CurPattern%, ActiveView%, MouseMode%
DIM SHARED DefaultI%, DefaultJ%, DefaultK%, NCoorBias!
DIM SHARED HotSpots%(15,3), Patterns%(23)
' Global Declarations from Coal Core Flux Calc .bas
' Data needed by Front calculations
DIM SHARED ShapeDefFileName$
DIM SHARED CalcFileName$
DIM SHARED MeshLevel%, StepI%, StepJ%, StepK%
DIM SHARED MatPerm!(15)
DIM SHARED P!(32, 32, 64) ' 280 KB
DIM SHARED Flux!(6)
DIM SHARED OffsetI%(6), OffsetJ%(6), OffsetK%(6)
' Data Defining situation inside of Coal Core
DIM SHARED SUR!, Tol!, TotErr!, RelErr!, ItsAtLevel%
' Data for Display Manager
DIM SHARED FluxScale!, FatArrowLimit!
DIM SHARED WinOpen%(16), WinUpdate%(16)
DIM SHARED SNTBBitCode%, SNWEBitCode%, WETBBitCode%, EWTBBitCode%
DIM SHARED rect%(4)
'*****
DIM SHARED MouseTool%
DIM SHARED ChangesSaved%
DIM SHARED sTau!(5)
DIM SHARED AverageFluxAlongCore!(64)
'*****Temporary Storage Data Structure for Tip Info*****
DIM SHARED tsX!(5), tsY!(5)
DIM SHARED tftau!(10), tffluxNorm!(10), tffluxTang!(10), tfIndex(10)
'*****Data Structure for Tip Group info*****
DIM SHARED TipGroupNum%
' can be 1 to 24 ' maps to 4 to 99 for CurTipNum%
DIM SHARED TipGroupDirectory%(20), TipGroupName$(200), TipGroupComment$(20)
DIM SHARED TipGroup2DLaplacian!(20), TipGroupFluxNorm!(20)
DIM SHARED TipGroup2DLaplacianUnCert!(20), TipGroupFluxNormUnCert!(20)
'*****Data Structure for Tips
DIM SHARED CurTipNum%
DIM SHARED TipStatusWord%(100), TipFixedCoor%(100)
DIM SHARED TipTau!(100), TipFluxNorm!(100), TipD2FluxNorm!(100)
DIM SHARED TipdX!(100), TipdY!(100), TipJac!(100), TipX!(100), TipY!(100)
DIM SHARED sX!(100, 5), sY!(100, 5) ' TipNum, 5 points define shape
DIM SHARED LastFluxPtr%(100)
DIM SHARED ftau!(100, 10), fsX!(100,10), fsY!(100,10)
DIM SHARED ffluxNorm!(100, 10), ffluxTang!(100, 10)
DIM SHARED fiCoor%(100, 10), fjCoor%(100, 10), fkCoor%(100, 10)
'*****
DIM SHARED FrontCalcShortFileName$, FrontCalcFileName$
DIM SHARED SortKey!(20), SortIndex%(20), TempArray%(20), TempArray!(20)
TipGroupNum% = 0
False% = 0
True% = NOT False%
MouseTool% = 1
sTau!(1) = -2! : sTau!(2) = -1! : sTau!(3) = 0! : sTau!(4) = 1! : sTau!(5) = 2!
OPEN "SCRW:" FOR OUTPUT AS #22
CALL MainProgramforFrontCalcs
END
SUB DetermineAverageFluxAlongCore STATIC
ChangeCursor 4
CrossSectionArea! = 31!*31! * 3.14159 /4!
'Exclude the surrounding concrete
AreaPerNode! = CSNG(StepI%) * CSNG(StepJ%)
MinAFAC! = 1E+23
MaxAFAC! = -1E+23
FOR k% = StepK% TO 63 STEP StepK%

```

```

Sum! = 0!
FOR i% = StepI% TO 31 STEP StepI%
  FOR j% = StepJ% TO 31 STEP StepJ%
    GetFlux i%, j%, k%, Flux! ( )
    Sum! = Sum! + (Flux!(5) - Flux!(6))
  NEXT
NEXT
NEXT
AverageFluxAlongCore!(k%) = Sum! * AreaPerNode! / CrossSectionArea!
IF MinAFAC! > AverageFluxAlongCore!(k%)
  THEN MinAFAC! = AverageFluxAlongCore!(k%)
IF MaxAFAC! < AverageFluxAlongCore!(k%)
  THEN MaxAFAC! = AverageFluxAlongCore!(k%)
NEXT
AverageFluxAlongCore!(0) = MinAFAC!
AverageFluxAlongCore!(64) = MaxAFAC!
INITCURSOR
BEEP
END SUB
SUB DetermineAverageFluxAlongCoreAtQtrs STATIC
  ChangeCursor 4
  CrossSectionArea! = 31!*31! * 3.1414 /4!
  'Exclude the surrounding concrete
  AreaPerNode! = CSNG(StepI%) * CSNG(StepJ%)
  FOR k% = 16 TO 48 STEP 16
    Sum! = 0!
    FOR i% = StepI% TO 31 STEP StepI%
      FOR j% = StepJ% TO 31 STEP StepJ%
        GetFlux i%, j%, k%, Flux! ( )
        Sum! = Sum! + (Flux!(5) - Flux!(6))
      NEXT
    NEXT
    AverageFluxAlongCore!(k%) = Sum! * AreaPerNode! / CrossSectionArea!
  NEXT
  INITCURSOR
END SUB
SUB MainProgramforFrontCalcs STATIC
  CALL DefineConstants
  CALL WindowInitialize
  CALL StartUpCode
  CALL DefineMenu
  CALL SetDefaultValues
  BEEP
  WHILE True%
    Dialog0% = DIALOG(0)
    Mous0% = MOUSE(0)
    Menu0% = MENU(0)
    IF Menu0% <> 0 THEN CALL HandleMenu (Menu0%)
    IF Mous0% <> 0 THEN CALL HandleMouse (Mous0%)
    IF Dialog0% <> 0 THEN CALL HandleDialog (Dialog0%)
  WEND
END SUB
SUB HandleDialog (Dialog0%) STATIC
  SELECT CASE Dialog0%
    CASE 3
      Win% = DIALOG(3)
      'Make active, which causes a dialog event 5 which
      'actually draws the WINDOW
      WINDOW Win%
      IF Win% >=7 AND Win% <=10 THEN
        IF TipGroupNum% > 0 THEN
          DetermineActiveView TheView%, TheFixedCoordinate%
          IF TheView% <> -1 THEN
            GetTipNum TipGroupNum%, TheView%, CurTipNum%
            LoadTemporaryArrays
            RefreshWindow 12
            RefreshWindow 6
          END IF
        END IF
      END IF
    END IF
  END IF
END SUB

```

```

CASE 4
    WinOpen%(DIALOG(4)) = False% 'Window clicked closed
    'Record that a window has been closed
CASE 5
    'Window has been overwritten
    Win% = DIALOG(5)
    CALL RefreshWindow (Win%)
CASE ELSE
END SELECT
END SUB
SUB DefineMenu STATIC
MENU 1, 0, 1, "File"
    MENU 1, 1, 1, "New"
    MENU 1, 2, 0, "Open"
    MENU 1, 3, 0, "-----"
    MENU 1, 4, 0, "Save" : CmdKey 1, 4, "S"
    MENU 1, 5, 0, "Save As..."
    MENU 1, 6, 0, "-----"
    MENU 1, 7, 1, "Print AW..."
    MENU 1, 8, 0, "-----"
    MENU 1, 9, 1, "Quit" : CmdKey 1, 9, "Q"
MENU 2, 0, 1, "Edit"
    MENU 2, 1, 1, "Cut" : CmdKey 2, 1, "X"
    MENU 2, 2, 1, "Copy" : CmdKey 2, 2, "C"
    MENU 2, 3, 1, "Paste" : CmdKey 2, 3, "V"
    MENU 2, 4, 1, "Copy AW"
MENU 3, 0, 0, "Tools"
    MENU 3, 1, 2, "Draw Front"
    MENU 3, 2, 0, "Reshape Front"
    MENU 3, 3, 0, "Select Nodes"
    MENU 3, 4, 0, "Override Tip Selection"
MENU 4, 0, 1, "TipGroup"
    MENU 4, 1, 1, "New"
    MENU 4, 2, 1, "Old.."
    MENU 4, 3, 0, "-----"
    MENU 4, 4, 1, "Name..."
    MENU 4, 5, 0, "-----"
    MENU 4, 6, 1, "Delete"
MENU 5, 0, 1, "Display"
    MENU 5, 1, 1, "General Info" : CmdKey 5, 1, "G"
    MENU 5, 2, 1, "SNTB..."
    MENU 5, 3, 1, "SNWE..."
    MENU 5, 4, 1, "WETB..."
    MENU 5, 5, 1, "EWTB..."
    MENU 5, 6, 1, "Set Flux Scale..."
    MENU 5, 7, 1, "Tip Info" : CmdKey 5, 7, "I"
    MENU 5, 8, 1, "Tip Plots"
    MENU 5, 9, 1, "Tip Group Info"
    MENU 5, 10, 1, "Clear Screen"
    MENU 5, 11, 1, "Redraw Screen" : CmdKey 5, 11, "R"
    MENU 5, 12, 1, "Average Flux"
MENU 6, 0, 1, "Debug"
    MENU 6, 1, 1, "Sort FluxInfo"
    MENU 6, 2, 1, "Determine Average Flux Along Core"
    MENU 6, 3, 1, "Plot Average Flux Along Core"
END SUB
SUB HandleMenu (Menu0%) STATIC
SELECT CASE Menu0%
CASE 1
    'File
    HandleFileMenu
CASE 2
    'Edit
    HandleEditMenu
CASE 3
    MENU 3, MouseTool%, 1
    MouseTool% = MENU(1)
    MENU 3, MouseTool%, 2
CASE 4
    HandleTipGroupMenu
CASE 5
    HandleDisplayMenu

```

```

CASE 6      'Debug"
  HandleDebugMenu
CASE ELSE
END SELECT
MENU
END SUB
'*****Edit Menu Code*****
SUB HandleEditMenu STATIC
  SELECT CASE MENU(1)
    CASE 4      'Edit - Copy Active Window to Clipboard
      BEEP : BEEP
      ActiveWin% = WINDOW(0)
      SELECT CASE ActiveWin%
        CASE 5, 6, 7, 8, 9, 10      'Graphics Windows
          'WINDOW 6, "Tip Plots"
          'WINDOW 7, "South - North - Top - Bottom"
          'WINDOW 8, "South - North - West - East"
          'WINDOW 9, "West - East - Top - Bottom"
          'WINDOW 10, "East - West - Top - Bottom"
          PICTURE ON : SHOWPEN                      'begin recording
          RefreshWindow ActiveWin%                  'make image
          PICTURE OFF                                'end recording
          ScreenImage$ = PICTURE$
          CLOSE #22
          OPEN "clip:picture" FOR OUTPUT AS #22
          PRINT #22, ScreenImage$
          CLOSE # 22
          OPEN "SCRN:" FOR OUTPUT AS #22
          PICTURE ON : PICTURE OFF
        CASE 11, 12, 13      'Textwindow
          'WINDOW 11, "General Info"
          'WINDOW 12, "Tip Information"
          'WINDOW 13, "Tip Group Information"
          CLOSE #22
          OPEN "clip:text" FOR OUTPUT AS #22
          RefreshWindow ActiveWin%
          CLOSE #22
          OPEN "SCRN:" FOR OUTPUT AS #22
        CASE ELSE
      END SELECT
    CASE ELSE
  END SELECT
END SUB
SUB PrintActiveWindow STATIC
  BEEP : BEEP
  ActiveWin% = WINDOW(0)
  SELECT CASE ActiveWin%
    CASE 5, 6, 7, 8, 9, 10      'Graphics Windows
      'WINDOW 6, "Tip Plots"
      'WINDOW 7, "South - North - Top - Bottom"
      'WINDOW 8, "South - North - West - East"
      'WINDOW 9, "West - East - Top - Bottom"
      'WINDOW 10, "East - West - Top - Bottom"
      PICTURE ON : SHOWPEN                      'begin recording
      RefreshWindow ActiveWin%                  'make image
      PICTURE OFF                                'end recording
      ScreenImage$ = PICTURE$
      CLOSE #22
      OPEN "lpt1:prompt" FOR OUTPUT AS #22
      WINDOW OUTPUT #22
      PICTURE ,ScreenImage$
      CLOSE # 22
      OPEN "SCRN:" FOR OUTPUT AS #22
      PICTURE ON : PICTURE OFF
    CASE 11, 12, 13      'Textwindow
      'WINDOW 11, "General Info"
      'WINDOW 12, "Tip Information"
      'WINDOW 13, "Tip Group Information"

```

```

        CLOSE #22
        OPEN "lpt1:" FOR OUTPUT AS #22
        RefreshWindow ActiveWin%
        CLOSE #22
        OPEN "SCRN:" FOR OUTPUT AS #22
        RefreshWindow ActiveWin%
    CASE ELSE
    END SELECT
END SUB
SUB HandleDebugMenu STATIC
    SELECT CASE MENU(1)
    CASE 1
        SortFluxInfo CurTipNum%
        RefreshWindow 12
    CASE 2
        DetermineAverageFluxAlongCore
    CASE 3
        PlotAverageFluxAlongCore
    CASE ELSE
    END SELECT
END SUB
SUB DisplayTipInfo STATIC
    CLS
    TEXTFACE 1
    LOCATE 2,1
    PRINT #22, "   File Name:           ", FrontCalcShortFileName$
    PRINT #22, "   Tip Group Number:   ", TipGroupNum%
    PRINT #22, "   Tip Group Name:     ", TipGroupName$(TipGroupNum%)
    PRINT #22, "   Tip Group Comment:  ", TipGroupComment$(TipGroupNum%)
    PRINT #22, "   Current Tip Number: ", CurTipNum%
    WriteOutStatusWordInfo
    tn% = CurTipNum%
    PRINT #22,
    PRINT #22, "   Tip Fixed Coordinate: ", TipFixedCoor%(tn%)
    PRINT #22,
    PRINT #22, "   Front Shape Definition"
    PRINT #22, "           Tau           X           Y"
    Format1$ = "   ###.##           ###.##           ###.##"
    Format2$ = "   ###.##   #####.####   #####.#### "
    Format3$ = "           ###   ###   ###"
    FOR i% = 1 TO 5
        PRINT #22, USING Format1$; sTau!(i%), sX!(tn%,i%), sY!(tn%,i%)
    NEXT
    PRINT #22,
    PRINT #22, "   Last Flux Pointer : ", LastFluxPtr%(CurTipNum%)
    PRINT #22, "   Flux to Front Selection"
    PRINT #22,
    PRINT #22, "           Tau           Flux(Normal)           Flux(Tang.)           i           j           k"
    FOR i% = 1 TO LastFluxPtr%(CurTipNum%)
        PRINT #22,
        USING Format2$; fTau(tn%, i%), fFluxNorm!(tn%, i%), fFluxTang!(tn%, i%);
        PRINT #22,
        USING Format3$; fiCoor%(tn%, i%), fjCoor%(tn%, i%), fkCoor%(tn%, i%)
    NEXT
END SUB
SUB WriteOutStatusWordInfo STATIC
    PRINT #22, "   Status Word:           ", HEX$(TipStatusWord%(CurTipNum%))
    GetTipStatusWord Curve%, TheCurve%
    IF TheCurve% = Defined% THEN
        PRINT #22, "   Curve:           ", "Defined"
    ELSEIF TheCurve% = NotDefined% THEN
        PRINT #22, "   Curve:           ", "Not Defined"
    END IF
    GetTipStatusWord Fluxes%, TheFluxes%
    IF TheFluxes% = Defined% THEN
        PRINT #22, "   Fluxes:           ", "Defined"
    ELSEIF TheFluxes% = NotDefined% THEN
        PRINT #22, "   Fluxes:           ", "Not Defined"
    END IF

```

```

END IF
GetTipStatusWord TipPoint%, TheTipPoint%
IF TheTipPoint% = Defined% THEN
    PRINT #22, " Tip Point:                                "," Defined"
    PRINT #22, USING
    " Tau at Tip:                                          ###.## "; TipTau!(CurTipNum%)
    PRINT #22, USING
    " X at Tip:                                           ###.## "; TipX!(CurTipNum%)
    PRINT #22, USING
    " Y at Tip:                                           ###.## "; TipY!(CurTipNum%)
    PRINT #22, USING
    " Flux at Tip:                                         ###.###~"; TipFluxNorm!(CurTipNum%)
    PRINT #22, USING
    " d2(Normal Flux)/dS2 at Tip: ###.###~"; TipL2FluxNorm!(CurTipNum%)
ELSEIF TheTipPoint% = NotDefined% THEN
    PRINT #22, " Tip Point:                                ","Not Defined"
END IF
END SUB
SUB DisplayTipPlots STATIC
CLS
IF LastFluxPtr%(CurTipNum%) > 2 THEN
    LoadTemporaryArrays
    yMin% = 10 'Current y coordinate in window
    yMax% = (WINDOW(3) * .5) - 10 'Height of window less a small amount
    xMin% = 10
    xMax% = WINDOW(2) - 10 'Width of window less a small amount
    DisplaySurfaceNormFlux xMin%, yMin%, xMax%, yMax%
    yMin% = yMax% + 10
    yMax% = WINDOW(3) - 10
    DisplaySurfaceDNormFlux xMin%, yMin%, xMax%, yMax%
    LOCATE 1, 1
    PRINT #22, "Current Tip Number: " + STR$(CurTipNum%)
END IF
END SUB
SUB FreeTipGroupNum (TrialTipGroupNum%) STATIC
FOR i% = 1 TO 20
    IF TipGroupDirectory%(i%) = NotDefined% THEN
        TrialTipGroupNum% = i%
        EXIT SUB
    END IF
NEXT
TrialTipGroupNum% = 0
END SUB
SUB DetermineActiveView (TheView%, TheFixedCoordinate%) STATIC
SELECT CASE WINDOW(0)
CASE 7
    'South - North - Top - Bottom
    TheView% = SNTB%
    TheFixedCoordinate% = SNTBBitCode% AND 127
CASE 8
    'South - North - West - East
    TheView% = SNWE%
    TheFixedCoordinate% = SNWEBitCode% AND 127
CASE 9
    'West - East - Top - Bottom
    TheView% = WETB%
    TheFixedCoordinate% = WETBBitCode% AND 127
CASE 10
    'East - West - Top - Bottom
    TheView% = EWTB%
    TheFixedCoordinate% = EWTBBitCode% AND 127
CASE ELSE
    TheView% = -1
END SELECT
END SUB
SUB SetActiveView (TheView%, TheFixedCoordinate%) STATIC
SELECT CASE TheView%
CASE 2
    'South - North - Top - Bottom

```

```

WinOpen%(7) = True%
WinUpdate%(7) = True%
SNTBBitCode% = (SNTBBitCode% AND &HFF80) OR TheFixedCoordinate%
WINDOW 7, Title$ + ", I = " + STR$(SNTBBitCode% AND 127)
CASE 4 'South - North - West - East
WinOpen%(8) = True%
WinUpdate%(8) = True%
SNWEBitCode% = (SNWEBitCode% AND &HFF80) OR TheFixedCoordinate%
WINDOW 8, Title$ + ", J = " + STR$(SNWEBitCode% AND 127)
CASE 6 'West - East - Top - Bottom
WinOpen%(9) = True%
WinUpdate%(9) = True%
WETBBitCode% = (WETBBitCode% AND &HFF80) OR TheFixedCoordinate%
WINDOW 9, Title$ + ", K = " + STR$(WETBBitCode% AND 127)
CASE 0 'East - West - Top - Bottom
WinOpen%(10) = True%
WinUpdate%(10) = True%
EWTBBitCode% = (EWTBBitCode% AND &HFF80) OR TheFixedCoordinate%
WINDOW 10, Title$ + ", K = " + STR$(EWTBBitCode% AND 127)
CASE ELSE
END SELECT
END SUB
SUB HandleTipGroupMenu STATIC
SELECT CASE MENU(1)
CASE 1 'Tip Group - New
FreeTipGroupNum TrialTipGroupNum%
IF TrialTipGroupNum% > 0 THEN
TipGroupNum% = TrialTipGroupNum%
TipGroupDirectory%(TipGroupNum%) = Defined%
TipGroupName$(TipGroupNum%) = ""
TipGroupComment$(TipGroupNum%) = ""
MENU 3, 0, 1
MENU 3, 1, 2
MENU 3, 2, 0
MENU 3, 3, 0
MENU 3, 4, 0
MouseTool% = 1
END IF
CASE 2 'Tip Group - Old
TipGroupOldDialogBox NewTipGroupNum%
IF NewTipGroupNum% > 0 THEN
TipGroupNum% = NewTipNum
END IF
CASE 4 'Tip - Name...
TipGroupNameDialogBox
CASE 5 'Tip - Delete
TipGroupDeletionDialogBox IsConfirmed%
IF IsConfirmed% = True% THEN
TipGroupDirectory%(TipGroupNum%) = NotDefined%
END IF
CASE ELSE
END SELECT
END SUB
SUB TipGroupDeletionDialogBox (IsConfirmed%) STATIC
WINDOW 16,,(300,200)-(610,270),-2
TEXTFACE 1
MOVETO 20,20
DrawText "Do you really want to delete this tip?"
BUTTON 1, 1, "Ok", (10,40)-(100, 60)
BUTTON 2, 1, "Cancel", (200, 40)-(300,60)
WHILE DIALOG(0) <> 1
WEND
IF DIALOG(1) = 1 THEN IsConfirmed% = True% ELSE IsConfirmed% = False%
END SUB
SUB TipGroupNameDialogBox STATIC
WINDOW 16,,,-2
LINE INPUT "Name: "; TipGroupName$(TipGroupNum%)
LINE INPUT "Comment: "; TipGroupComment$(TipGroupNum%)

```



```

WINDOW CLOSE 16
END SUB
SUB SetCurTipNum (TheTipGroupNum%, TheView%) STATIC
IF TheView% = SNWE% THEN
CurTipNum% = TheTipGroupNum% * 4
ELSEIF TheView% = SNTB% THEN
CurTipNum% = TheTipGroupNum% * 4 + 1
ELSEIF TheView% = WETB% THEN
CurTipNum% = TheTipGroupNum% * 4 + 2
ELSEIF TheView% = EWTB% THEN
CurTipNum% = TheTipGroupNum% * 4 + 3
END IF
END SUB
SUB GetTipNum (TheTipGroupNum%, TheView%, TipNum%) STATIC
IF TheView% = SNWE% THEN
TipNum% = TheTipGroupNum% * 4
ELSEIF TheView% = SNTB% THEN
TipNum% = TheTipGroupNum% * 4 + 1
ELSEIF TheView% = WETB% THEN
TipNum% = TheTipGroupNum% * 4 + 2
ELSEIF TheView% = EWTB% THEN
TipNum% = TheTipGroupNum% * 4 + 3
END IF
END SUB
SUB HandleMouse (Mous0%) STATIC
IF MouseTool% = 0 THEN EXIT SUB
IF TipGroupNum% = 0 THEN EXIT SUB
DetermineActiveView TheView%, TheFixedCoordinate%
IF TheView% = -1 THEN EXIT SUB
GetTipNum TipGroupNum%, TheView%, CurTipNum%
WINDOW OUTPUT WINDOW(0) 'Make the active window the output window
SELECT CASE MouseTool%
CASE 1 'Draw Curve
IF Mous0% = 1 THEN
GetTipStatusWord Curve%, TheCurve%
IF TheCurve% = Defined% AND
TheFixedCoordinate% = TipFixedCoor%(CurTipNum%) THEN
DrawCurve False%, TheView%
END IF
GetCurve TheView%
TipFixedCoor%(CurTipNum%) = TheFixedCoordinate%
DrawCurve True%, TheView%
RefreshWindow 12
END IF
CASE 2 'Reshape Curve
GetTipStatusWord Curve%, TheCurve%
IF TheCurve% = Defined% AND
TheFixedCoordinate% = TipFixedCoor%(CurTipNum%) THEN
HandleReshapeCurve Mous0%, TheView%
RefreshWindow 12
END IF
CASE 3 'Define Normal Fluxes
IF Mous0% = 1 THEN
GetTipStatusWord Curve%, TheCurve%
IF TheCurve% = Defined% AND
TheFixedCoordinate% = TipFixedCoor%(CurTipNum%) THEN
HandleGetNormalFluxes TheView%
IF LastFluxPtr%(CurTipNum%) >= 3 THEN
SelectTipAutomaticallyAndDoCalculations
GetTipNum TipGroupNum%, SNWE%, TipNumSNWE%
GetTipNum TipGroupNum%, SNTB%, TipNumSNTB%
IF LastFluxPtr%(TipNumSNWE%) >= 3 AND
LastFluxPtr%(TipNumSNTB%) >= 3 THEN
TipGroupCalculation TipNumSNWE%, TipNumSNTB%
END IF
END IF
RefreshWindow 12
RefreshWindow 6

```

```

        END IF
    END IF
    CASE ELSE
    END SELECT
END SUB
SUB TipGroupCalculation (T1%, T2%) STATIC
'NonDimensional Units at this point
TipGroupFluxNorm!(TipGroupNum%) = .5 * (TipFluxNorm!(T1%) + TipFluxNorm!(T2%))
TipGroup2DLaplacian!(TipGroupNum%) = TipD2FluxNorm!(T1%) + TipD2FluxNorm!(T2%)
'Uncertainty Analysis
TipGroupFluxNormUnCert!(TipGroupNum%) =
    .5 * (TipFluxNorm!(T1%) - TipFluxNorm!(T2%))
CosTheta! = ABS((TipdX!(T1%) * TipdX!(T2%) ) / (TipJac!(T1%) * TipJac!(T2%)))
TipGroup2DLaplacianUnCert!(TipGroupNum%) =
    TipGroup2DLaplacian!(TipGroupNum%) * CosTheta!
END SUB
SUB LoadTemporaryArrays STATIC
FOR i% = 1 TO 5
    tsX!(i%) = sX!(CurTipNum%, i%)
    tsY!(i%) = sY!(CurTipNum%, i%)
NEXT
FOR i% = 1 TO 10
    tfTau!(i%) = fTau!(CurTipNum%, i%)
    tfFluxNorm!(i%) = fFluxNorm!(CurTipNum%, i%)
    tfFluxTang!(i%) = fFluxTang!(CurTipNum%, i%)
NEXT
END SUB
SUB SelectTheTipManually (Mous0%) STATIC
STOP 'This routine must be updated before use
IF Mous0% <> 1 THEN EXIT SUB 'Capture only releases not drags
ChangeCursor 4
x0! = MOUSE(3)
y0! = MOUSE(4)
d2Best! = 1E+15
FOR TauQ! = -2.5 TO 2.5 STEP .1
    LaGrange 5, sTau!( ), tsX!( ), TauQ!, xt!, ErrCode%
    LaGrange 5, sTau!( ), tsY!( ), TauQ!, yt!, ErrCode%
    d2! = (x0! - xt!) * (x0! - xt!) + (y0! - yt!) * (y0! - yt!)
    IF d2! < d2Best! THEN
        d2Best! = d2!
        TauBest! = TauQ!
        xBest! = xt!
        yBest! = yt!
    END IF
NEXT
FOR TauQQ! = TauQ! -.1 TO TauQ! + .1 STEP .005
    LaGrange 5, sTau!( ), tsX!( ), TauQQ!, xt!, ErrCode%
    LaGrange 5, sTau!( ), tsY!( ), TauQQ!, yt!, ErrCode%
    d2! = (x0! - xt!) * (x0! - xt!) + (y0! - yt!) * (y0! - yt!)
    IF d2! < d2Best! THEN
        d2Best! = d2!
        TauBest! = TauQQ!
        xBest! = xt!
        yBest! = yt!
    END IF
NEXT
TipTau!(CurTipNum%) = TauBest!
SetTipStatusWord TipPoint%, Defined%
CIRCLE (xBest! + 1, yBest!), 3
CIRCLE (xBest! - 1, yBest!), 3
CIRCLE (xBest!, yBest! + 1), 3
CIRCLE (xBest!, yBest! - 1), 3
INITCURSOR
END SUB
SUB HandleGetNormalFluxes (TheView%) STATIC
IF TheView% = SWTB% OR TheView% = SWWE% THEN
    HandleGetNormalFluxesSW TheView%
END IF

```

```

END SUB
SUB HandleGetNormalFluxesSN (TheView%) STATIC
'Have one routine for both SNTB and SNWE
ChangeCursor 4
GetWindowPositionScale k0%, dk%, j0%, dj%
xl% = MOUSE (3)
yl% = MOUSE(4)
xl% = INT(.5 * CSNG(xl% - k0%)/CSNG(dk%) + .5)
yl% = INT(.5 * CSNG(yl% - j0%)/CSNG(dj%) + .5)
'Find node coordinates i, j, k :
IF TheView% = SNTB% THEN '<+--+--+--+--+--+--+--+
    i% = TipFixedCoor%(CurTipNum%)
    j% = yl%
ELSE
    j% = TipFixedCoor%(CurTipNum%)
    i% = yl%
END IF
k% = xl%
'Check to see if this node is new:
IsNewNode% = True%
FOR g% = 1 TO LastFluxPtr%(CurTipNum%)
    IF i% = fiCoor%(CurTipNum%, g%) THEN
        IF j% = fjCoor%(CurTipNum%, g%) AND k% = fkCoor%(CurTipNum%, g%) THEN
            IsNewNode% = False%
            f% = g%
        END IF
    END IF
NEXT
IF IsNewNode% = True% AND LastFluxPtr%(CurTipNum%) < 10 THEN
    'Draw Circle
    PSET (k0% + 2 * xl% * dk%, j0% + 2 * yl% * dj%)
    CIRCLE STEP (0,0), dk% \ 2
    LastFluxPtr%(CurTipNum%) = LastFluxPtr%(CurTipNum%) + 1
    h% = LastFluxPtr%(CurTipNum%)
    fiCoor%(CurTipNum%, h%) = i%
    fjCoor%(CurTipNum%, h%) = j%
    fkCoor%(CurTipNum%, h%) = k%
    'Then calculate x0, y0 for i%, j% , k%
    x0! = xl%
    y0! = yl%
    CALL GetFlux(i%, j%, k%, Flux!())
    IF TheView% = SNTB% THEN '<+--+--+--+--+--+--+--+
        FluxY! = Flux!(3) - Flux!(4)
    ELSE
        FluxY! = Flux!(1) - Flux!(2)
    END IF
    FluxX! = Flux!(5) - Flux!(6)
    FindIntersection x0!, y0!, FluxX!, FluxY!, fTau!(CurTipNum%, h%), XS!, yS!
    fsX!(CurTipNum%, h%) = XS!
    fsY!(CurTipNum%, h%) = yS!
    PSET (k0% + 2 * XS! * dk%, j0% + 2 * yS! * dj%)
    CIRCLE STEP (0, 0), 5
    CIRCLE STEP (0, 0), 3
    FindNormal fTau!(TipNum%, h%), nx!, ny!
    'Then calculate the normal & tangential component of the Flux at that point
    fFluxNorm!(CurTipNum%, h%) = nx! * FluxX! + ny! * FluxY!
    fFluxTang!(CurTipNum%, h%) = -ny! * FluxX! + nx! * FluxY!
    SetTipStatusWord Fluxes%, Defined%
ELSEIF IsNewNode% = False% THEN
    'Erase Circle for point
    PRESET (k0% + 2 * xl% * dk%, j0% + 2 * yl% * dj%)
    CIRCLE STEP (0,0), dk% \ 2, 30
    'Erase Intersection Marker
    XS! = fsX!(CurTipNum%, f%)
    yS! = fsY!(CurTipNum%, f%)
    PRESET (k0% + 2 * XS! * dk%, j0% + 2 * yS! * dj%)
    CIRCLE STEP (0, 0), 5, 30
    CIRCLE STEP (0, 0), 3, 30

```

```

    CompressFLists f%
END IF
SortFluxInfo CurTipNum%
INITCURSOR
END SUB
SUB CompressFLists (f%) STATIC
    'Delete from list by moving last item into slot,
    ' then reducing size of list by one
    h% = LastFluxPtr%(CurTipNum%)
    LastFluxPtr%(CurTipNum%) = LastFluxPtr%(CurTipNum%) - 1
    fiCoor%(CurTipNum%, f%) = fiCoor%(CurTipNum%, h%)
    fjCoor%(CurTipNum%, f%) = fjCoor%(CurTipNum%, h%)
    fkCoor%(CurTipNum%, f%) = fkCoor%(CurTipNum%, h%)
    fTau!(CurTipNum%, f%) = fTau!(CurTipNum%, h%)
    fsX!(CurTipNum%, f%) = fsX!(CurTipNum%, h%)
    fsY!(CurTipNum%, f%) = fsY!(CurTipNum%, h%)
    fFluxNorm!(CurTipNum%, f%) = fFluxNorm!(CurTipNum%, h%)
    fFluxTang!(CurTipNum%, f%) = fFluxTang!(CurTipNum%, h%)
END SUB
SUB DisplaySurfaceNormFlux (xMin%, yMin%, xMax%, yMax%) STATIC
    BEEP
    PENSIZE 2,2
    LINE (xMin%, yMin%)-(xMax%, yMax%),,B
    ax! = tftau!(LastFluxPtr%(CurTipNum%)) - tftau!(1)
    IF ax! = 0! THEN EXIT SUB
    FluxNormMin! = 10000!
    FluxNormMax! = -10000!
    FOR x! = tftau!(1) TO tftau!(LastFluxPtr%(CurTipNum%)) STEP ax! * .05
        LaGrange LastFluxPtr%(CurTipNum%), tftau!( ),
            tftFluxNorm!( ), x!, y!, ErrCode%
        IF FluxNormMin! > y! THEN FluxNormMin! = y!
        IF FluxNormMax! < y! THEN FluxNormMax! = y!
    NEXT
    ay! = FluxNormMax! - FluxNormMin!
    IF ay! = 0! THEN EXIT SUB
    cx! = tftau!(1)
    dx% = (xMax% - xMin%) * .9
    dy% = (yMax% - yMin%) * .8
    cy! = FluxNormMin!
    FOR i% = 1 TO LastFluxPtr%(CurTipNum%)
        x% = xMin% + dx% \ 18 + ((tftau!(i%) - cx!) / ax!) * dx%
        y% = yMax% - dy% \ 16 - ((tftFluxNorm!(i%) - cy!) / ay!) * dy%
        PSET (x%, y%)
        CIRCLE STEP (0,0), 7
        CIRCLE STEP (0,0), 8
    NEXT
    First% = True%
    FOR x! = tftau!(1) TO
        tftau!(LastFluxPtr%(CurTipNum%)) + ax! * .025 STEP ax! * .025
        LaGrange LastFluxPtr%(CurTipNum%), tftau!( ),
            tftFluxNorm!( ), x!, y!, ErrCode%
        x% = xMin% + dx% \ 18 + ((x! - cx!) / ax!) * dx%
        y% = yMax% - dy% \ 16 - ((y! - cy!) / ay!) * dy%
        IF First% THEN
            MOVETO x%, y%
            First% = False%
        ELSE
            LINETO x%, y%
        END IF
    NEXT
    x% = xMin% + dx% \ 18 + ((TipTau!(CurTipNum%) - cx!) / ax!) * dx%
    LINE (x%, yMin%)-(x%, yMax%)
    x% = xMin% + dx% \ 18 + ((0 - cx!) / ax!) * dx%
    LINE (x%, yMin%)-(x%, yMin% + (yMax% - yMin%) \ 8 )
    LINE (x%, yMax% - (yMax% - yMin%) \ 8)-(x%, yMax%)
    PENNORMAL
END SUB

```

```

SUB DisplaySurfaceDNormFlux (xMin%, yMin%, xMax%, yMax%) STATIC
  PENSIZE 2,2
  LINE (xMin%, yMin%)-(xMax%, yMax%),,B
  ax! = tftau!(LastFluxPtr%(CurTipNum%)) - tftau!(1)
  IF ax! = 0! THEN EXIT SUB
  dFluxNormMin! = 1000000!
  dFluxNormMax! = -1000000!
  First% = True%
  FOR x! = tftau!(1) TO tftau!(LastFluxPtr%(CurTipNum%)) STEP ax! * .025
    LaGrange LastFluxPtr%(CurTipNum%), tftau!( ),
    tftfluxNorm!( ), x!, y!, ErrCode%
    IF First% THEN
      xOld! = x!
      yOld! = y!
      First% = False%
    ELSE
      dFluxNorm! = (y! - yOld!) / (x! - xOld!)
      xOld! = x!
      yOld! = y!
      IF dFluxNormMin! > dFluxNorm! THEN dFluxNormMin! = dFluxNorm!
      IF dFluxNormMax! < dFluxNorm! THEN dFluxNormMax! = dFluxNorm!
    END IF
  NEXT
  ay! = dFluxNormMax! - dFluxNormMin!
  IF ay! = 0! THEN EXIT SUB
  cx! = tftau!(1)
  dx% = (xMax% - xMin%) * .9
  dy% = (yMax% - yMin%) * .8
  cy! = dFluxNormMin!
  xOld! = tftau!(1)
  yOld! = tftfluxNorm!(1)
  First% = True%
  FOR x! = tftau!(1) + .025 * ax! TO
    tftau!(LastFluxPtr%(CurTipNum%)) + ax! * .025 STEP ax! * .025
    LaGrange LastFluxPtr%(CurTipNum%), tftau!( ),
    tftfluxNorm!( ), x!, y!, ErrCode%
    dFluxNorm! = (y! - yOld!) / (x! - xOld!)
    xp! = (xOld! + x!) * .5
    xOld! = x!
    yOld! = y!
    x% = xMin% + dx% \ 18 + ((xp! - cx!) / ax!) * dx%
    y% = yMax% - dy% \ 16 - ((dFluxNorm! - cy!) / ay!) * dy%
    IF First% THEN
      MOVETO x%, y%
      First% = False%
    ELSE
      LINETO x%, y%
    END IF
  NEXT
  y% = yMax% - dy% \ 16 - ((0 - cy!) / ay!) * dy%
  LINE (xMin%, y%) - (xMax%, y%)
  x% = xMin% + dx% \ 18 + ((TipTau!(CurTipNum%) - cx!) / ax!) * dx%
  LINE (x%, yMin%) - (x%, yMax%)
  x% = xMin% + dx% \ 18 + ((0 - cx!) / ax!) * dx%
  LINE (x%, yMin%) - (x%, yMin% + (yMax% - yMin%) \ 8 )
  LINE (x%, yMax% - (yMax% - yMin%) \ 8) - (x%, yMax%)
  PENNORMAL
END SUB

SUB SelectTipAutomaticallyAndDoCalculations STATIC
  Ptr% = LastFluxPtr%(CurTipNum%)
  aTau! = tftau!(Ptr%) - tftau!(1)
  TauOld! = tftau!(1)
  FlxOld! = tftfluxNorm!(1)
  MinAbsValDFluxNorm! = 1E+23
  FOR Tau! = tftau!(1) + .01 * aTau! TO
    tftau!(Ptr%) + aTau! * .025 STEP aTau! * .01
    LaGrange Ptr%, tftau!( ), tftfluxNorm!( ), Tau!, Flx!, ErrCode%

```

```

dFluxNorm! = (Flx! - FlxOld!) / (Tau! - TauOld!)
'Central Difference formula
Tau! = (TauOld! + Tau!) * .5
IF MinAbsValDFluxNorm! > ABS(dFluxNorm!) THEN
    MinAbsValDFluxNorm! = ABS(dFluxNorm!)
    TauBest! = Tau!
END IF
TauOld! = Tau!
FlxOld! = Flx!
NEXT
MENU 6, 2, 1, STR$(MinAbsValDFluxNorm!)
TipTau!(CurTipNum%) = TauBest!
LaGrange 5, sTau!( ), tsX!( ), TauBest!, TipX!(CurTipNum%), ErrCode%
LaGrange 5, sTau!( ), tsY!( ), TauBest!, TipY!(CurTipNum%), ErrCode%
TauMinus! = TauBest! - .01 * aTau!
TauPlus! = TauBest! + .01 * aTau!
dTau! = (TauPlus! - TauMinus!)
LaGrange 5, sTau!( ), tsX!( ), TauMinus!, xMinus!, ErrCode%
LaGrange 5, sTau!( ), tsX!( ), TauPlus!, xPlus!, ErrCode%
dX! = (xPlus! - xMinus!) / (dTau! + dTau!)
TipdX!(CurTipNum%) = dX!
LaGrange 5, sTau!( ), tsY!( ), TauMinus!, yMinus!, ErrCode%
LaGrange 5, sTau!( ), tsY!( ), TauPlus!, yPlus!, ErrCode%
dY! = (yPlus! - yMinus!) / (dTau! + dTau!)
TipdY!(CurTipNum%) = dY!
dSdTau! = SQR(dX! * dX! + dY! * dY!)
TipJac!(CurTipNum%) = dSdTau!
LaGrange Ptr%, tftau!( ), tfflaxnorm!( ), TauBest!, FlxBest!, ErrCode%
LaGrange Ptr%, tftau!( ), tfflaxnorm!( ), TauMinus!, FlxMinus!, ErrCode%
LaGrange Ptr%, tftau!( ), tfflaxnorm!( ), TauPlus!, FlxPlus!, ErrCode%
TipFluxNorm!(CurTipNum%) = FlxBest!
Denom! = (dTau! * dTau!) * (dSdTau! * dSdTau!)
TipD2FluxNorm!(CurTipNum%) = (FlxMinus! - 2! * FlxBest! + FlxPlus!) / Denom!
SetTipStatusWord TipPoint%, Defined%
INITCURSOR
END SUB
SUB FindNormal (Tau0!, nx, ny!) STATIC
'For simplicity, calculate using finite differences.
TauPlus! = Tau0! + .02
TauMinus! = Tau0! - .02
LaGrange 5, sTau!( ), tsX!( ), TauPlus!, xPlus!, ErrCode%
LaGrange 5, sTau!( ), tsY!( ), TauPlus!, yPlus!, ErrCode%
LaGrange 5, sTau!( ), tsX!( ), TauMinus!, xMinus!, ErrCode%
LaGrange 5, sTau!( ), tsY!( ), TauMinus!, yMinus!, ErrCode%
dX! = xPlus! - xMinus!
dY! = yPlus! - yMinus!
Denom! = SQR(dX! * dX! + dY! * dY!)
nx! = dY! / Denom!
ny! = -dX! / Denom!
END SUB
SUB FindIntersection (x0!, y0!, FluxX!, FluxY!, TauS!, XS!, yS!) STATIC
d2Best! = 1E+15
'Generate initial old value
Tau! = -2.5
LaGrange 5, sTau!( ), tsX!( ), Tau!, xt!, ErrCode%
LaGrange 5, sTau!( ), tsY!( ), Tau!, yt!, ErrCode%
Rold! = FluxY! * (xt! - x0!) - FluxX! * (yt! - y0!)
FOR Tau! = -2.51 TO 2.5 STEP .1
    LaGrange 5, sTau!( ), tsX!( ), Tau!, xt!, ErrCode%
    LaGrange 5, sTau!( ), tsY!( ), Tau!, yt!, ErrCode%
    Rnew! = FluxY! * (xt! - x0!) - FluxX! * (yt! - y0!)
    IF Rnew! * Rold! <= 0! THEN
        RoldQ! = Rold!
        FOR TauQ = Tau! - .1 TO Tau! STEP .005
            LaGrange 5, sTau!( ), tsX!( ), TauQ!, xt!, ErrCode%
            LaGrange 5, sTau!( ), tsY!( ), TauQ!, yt!, ErrCode%
            RnewQ! = FluxY! * (xt! - x0!) - FluxX! * (yt! - y0!)
            IF RnewQ! * RoldQ! <= 0 THEN

```

```

        d2! = (x0! - xt!) * (x0! - xt!) + (y0! - yt!) * (y0! - yt!)
        IF d2! < d2Best! THEN
            d2Best! = d2!
            TauS! = TauQ!
            XS! = xt!
            yS! = yt!
        END IF
    END IF
    RoldQ! = RnewQ!
NEXT
END IF
Rold! = Rnew!
NEXT
END SUB
SUB SortFluxInfo (TheTipNum%) STATIC
    TheEnd% = LastFluxPtr%(TheTipNum%)
    'Make Index
    FOR i% = 1 TO TheEnd%
        SortKey!(i%) = fTau!(TheTipNum%, i%)
        SortIndex%(i%) = i%
    NEXT
    NoSwaps% = False%
    WHILE NoSwaps% = False%
        NoSwaps% = True%
        FOR i% = 1 TO TheEnd% - 1
            IF SortKey!(i%) > SortKey!(i% + 1) THEN
                NoSwaps% = False%
                Temp! = SortKey!(i%)
                SortKey!(i%) = SortKey!(i% + 1)
                SortKey!(i% + 1) = Temp!
                Temp% = SortIndex%(i%)
                SortIndex%(i%) = SortIndex%(i% + 1)
                SortIndex%(i% + 1) = Temp%
            END IF
        NEXT
    WEND
    ReOrderFluxReal SortIndex%(), fTau!(), TheTipNum%, TheEnd%
    ReOrderFluxReal SortIndex%(), fsX!(), TheTipNum%, TheEnd%
    ReOrderFluxReal SortIndex%(), fsY!(), TheTipNum%, TheEnd%
    ReOrderFluxReal SortIndex%(), fFluxNorm!(), TheTipNum%, TheEnd%
    ReOrderFluxReal SortIndex%(), fFluxTang!(), TheTipNum%, TheEnd%
    ReOrderFluxInteger SortIndex%(), fiCoor%(), TheTipNum%, TheEnd%
    ReOrderFluxInteger SortIndex%(), fjCoor%(), TheTipNum%, TheEnd%
    ReOrderFluxInteger SortIndex%(), fkCoor%(), TheTipNum%, TheEnd%
    LoadTemporaryArrays
END SUB
SUB ReOrderFluxReal (LocalSortIndex%(), Values!(), ArrayPtr%, TheEnd%) STATIC
    FOR i% = 1 TO TheEnd%
        TempArray!(i%) = Values!(ArrayPtr%, LocalSortIndex%(i%))
    NEXT
    FOR i% = 1 TO TheEnd%
        Values!(ArrayPtr%, i%) = TempArray!(i%)
    NEXT
END SUB
SUB ReOrderFluxInteger (LocalSortIndex%(), Values%(), ArrayPtr%, TheEnd%) STATIC
    FOR i% = 1 TO TheEnd%
        TempArray%(i%) = Values%(ArrayPtr%, LocalSortIndex%(i%))
    NEXT
    FOR i% = 1 TO TheEnd%
        Values%(ArrayPtr%, i%) = TempArray%(i%)
    NEXT
END SUB
SUB HandleReshapeCurve (Mous0%, TheView%) STATIC
    IF TheView% = SNWE% OR TheView% = SNTB% THEN
        GetWindowPositionScale k0%, dk%, j0%, dj%
    ELSE
        GetWindowPositionScales k0%, dk%, j0%, dj%
    END IF
END IF

```

```

WAVE 0
SELECT CASE Mous0%
CASE 1
  x0% = MOUSE(3)
  y0% = MOUSE(4)
  m% = False%
  FOR i% = 1 TO 5
    xl% = k0% + (dk% + dk%) * tsX!(i%)
    yl% = j0% + (dj% + dj%) * tsY!(i%)
    IF (ABS(xl% - x0%) + ABS(yl% - y0%) < 5) THEN m% = i%
  NEXT
  IF m% <> False% THEN
    DrawCurve False%, TheView%
    xl% = MOUSE(5)
    yl% = MOUSE(6)
    tsX!(m%) = .5 * CSNG(xl% - k0%)/CSNG(dk%)
    tsY!(m%) = .5 * CSNG(yl% - j0%)/CSNG(dj%)
    sX!(CurTipNum%, m%) = tsX!(m%)
    sY!(CurTipNum%, m%) = tsY!(m%)
    DrawCurve True%, TheView%
    SetTipStatusWord Curve%, Defined%
  END IF
CASE -1
  x0% = MOUSE(3)
  y0% = MOUSE(4)
  m% = False%
  FOR i% = 1 TO 5
    xl% = k0% + (dk% + dk%) * tsX!(i%)
    yl% = j0% + (dj% + dj%) * tsY!(i%)
    IF (ABS(xl% - x0%) + ABS(yl% - y0%) < 5) THEN m% = i%
  NEXT
  IF m% <> False% THEN
    Mous0% = MOUSE(0)
    xl% = MOUSE(5)
    yl% = MOUSE(6)
    WHILE Mous0% <> 1
      IF MOUSE(5) <> xl% OR MOUSE(6) <> yl% THEN
        DrawCurve False%, TheView%
        xl% = MOUSE(5)
        yl% = MOUSE(6)
        tsX!(m%) = .5 * CSNG(xl% - k0%)/CSNG(dk%)
        tsY!(m%) = .5 * CSNG(yl% - j0%)/CSNG(dj%)
        DrawCurve True%, TheView%
      END IF
      Mous0% = MOUSE(0)
    WEND
    DrawCurve False%, TheView%
    xl% = MOUSE(5)
    yl% = MOUSE(6)
    tsX!(m%) = .5 * CSNG(xl% - k0%)/CSNG(dk%)
    tsY!(m%) = .5 * CSNG(yl% - j0%)/CSNG(dj%)
    sX!(CurTipNum%, m%) = tsX!(m%)
    sY!(CurTipNum%, m%) = tsY!(m%)
    DrawCurve True%, TheView%
    SetTipStatusWord Curve%, Defined%
  END IF
CASE ELSE
  SOUND 80, 1, 1, 0
END SELECT
END SUB
SUB GetCurve (TheView%) STATIC
  IF TheView% = SNWE% OR TheView% = SNTB% THEN
    GetWindowPositionScale k0%, dk%, j0%, dj%
  ELSE
    GetWindowPositionScaleS k0%, dk%, j0%, dj%
  END IF
  xl% = MOUSE(3)
  yl% = MOUSE(4)

```



```

tsX!(1) = .5 * CSNG(x1% - k0%)/CSNG(dk%) : sX!(CurTipNum%, 1) = tsX!(1)
tsY!(1) = .5 * CSNG(y1% - j0%)/CSNG(dj%) : sY!(CurTipNum%, 1) = tsY!(1)
CIRCLE (x1%, y1%), 3
FOR i% = 2 TO 5
    Mous0% = 0
    WHILE Mous0% <> 1
        Mous0% = MOUSE(0)
    WEND
    x1% = MOUSE(3)
    y1% = MOUSE(4)
    tsX!(i%) = .5 * CSNG(x1% - k0%)/CSNG(dk%) : sX!(CurTipNum%, i%) = tsX!(i%)
    tsY!(i%) = .5 * CSNG(y1% - j0%)/CSNG(dj%) : sY!(CurTipNum%, i%) = tsY!(i%)
    CIRCLE (x1%, y1%), 3
NEXT
SetTipStatusWord Curve%, Defined%
END SUB
SUB DrawCurve (Show%, TheView%) STATIC
IF TheView% = SNWE% OR TheView% = SMTB% THEN
    GetWindowPositionScale k0%, dk%, j0%, dj%
ELSE
    GetWindowPositionScaleS k0%, dk%, j0%, dj%
END IF
PENSIZE 3,3
IF Show% = True% THEN
    FOR i% = 1 TO 5
        x1% = k0% + (dk% + dk%) * tsX!(i%)
        y1% = j0% + (dj% + dj%) * tsY!(i%)
        CIRCLE (x1%, y1%), 3
    NEXT
ELSEIF Show% <> NoPoints% THEN
    FOR i% = 1 TO 5
        x1% = k0% + (dk% + dk%) * tsX!(i%)
        y1% = j0% + (dj% + dj%) * tsY!(i%)
        CIRCLE (x1%, y1%), 3, 30
    NEXT
END IF
TauStep! = .03
First% = True%
FOR Tau! = -2.1 TO 2.1 STEP TauStep!
    LaGrange 5, sTau!( ), tsX!( ), Tau!, x0!, ErrCode%
    IF ErrCode% = True% THEN
        EXIT SUB
    END IF
    LaGrange 5, sTau!( ), tsY!( ), Tau!, y0!, ErrCode%
    IF ErrCode% = True% THEN
        EXIT SUB
    END IF
    IF Show% = False% THEN CALL PENMODE (11) ELSE CALL PENMODE (8)
    x1% = k0% + (dk% + dk%) * x0!
    y1% = j0% + (dj% + dj%) * y0!
    IF First% THEN
        MOVE TO x1%, y1%
        First% = False%
    ELSE
        LINE TO x1%, y1%
    END IF
NEXT
PENNORMAL
END SUB
SUB LaGrange (n%, xk!( ), yk!( ), x!, y!, ErrCode%) STATIC
' This is a simple implementation of LaGrange interpolation.
' No attempts have been made to make it efficient.
' On normal return ErrCode% = 0, else ErrCode% = -1
y! = 0
FOR j% = 1 TO n%
    L! = 1!
    FOR i% = 1 TO n%
        IF i% <> j% THEN

```

```

    d! = (xk!(j%) - xk!(i%))
    IF d! <> 0 THEN
        L! = L! * (x! - xk!(i%)) / d!
    ELSE
        ErrCode% = True%
        BEEP
        EXIT SUB
    END IF
END IF
NEXT
y! = y! + L! * yk!(j%)
NEXT
ErrCode% = False%
END SUB
SUB DefineConstants STATIC
    TipView% = 6
    TipName% = 8
    TipComment% = 16
    ,
    ,
    ,
    ,
    Curve% = 256
    Fluxes% = 512
    TipPoint% = 1024
    Calculation% = 2048
    ,
    ,
    Defined% = 1
    NotDefined% = 0
    SNTB% = 2
    STWE% = 4
    WETB% = 6
    EWTB% = 0
    NoPoints% = 7
END SUB
SUB GetTipStatusWord (Operation%, Value%) STATIC
    TSW% = TipStatusWord%(CurTipNum%)
    SELECT CASE Operation%
        CASE 256 'Curve
            Value% = (TSW% AND 256) \ 256
        CASE 512 'Fluxes%
            Value% = (TSW% AND 512) \ 512
        CASE 1024 'Tip Point
            Value% = (TSW% AND 1024) \ 1024
        CASE 2048 'Tip Calculations
            Value% = (TSW% AND 2048) \ 2048
        CASE ELSE
            WINDOW 16
            PRINT "GetTipStatusWord
            PRINT "Operation :"; Operation%
            PRINT "Value: "; Value%
            PRINT "Not Defined"
            PRINT "Press a key to exit"
            WHILE INKEY$ = ""
            WEND
            STOP
        END SELECT
END SUB
SUB SetTipStatusWord (Operation%, Value%) STATIC
    TSW% = TipStatusWord%(CurTipNum%)
    SELECT CASE Operation%
        CASE 256 'Curve
            TSWM% = TSW% AND &HFOFF
            IF Value% = Defined% THEN
                TSW% = TSWM% + &H100
            MENU 3, 2, 1
            MENU 3, 3, 1
            MENU 3, 4, 0

```

```

ELSEIF Value% = Undefined% THEN
    TSW% = TSWM%
    MENU 3, 2, 0
    MENU 3, 3, 0
    MENU 3, 4, 0
END IF
LastFluxPtr%(CurTipNum%) = 0
CASE 512 'Fluxes%
    TSWM% = TSW% AND &HF1FF
    IF Value% = Defined% THEN
        TSW% = TSWM% + &H200
        IF LastFluxPtr%(CurTipNum%) > 2 THEN MENU 3, 4, 1
    ELSEIF Value% = Undefined% THEN
        TSW% = TSWM%
    END IF
CASE 1024 'Tip Point
    TSWM% = TSW% AND &HF3FF
    IF Value% = Defined% THEN
        TSW% = TSWM% + &H400
    ELSEIF Value% = Undefined% THEN
        TSW% = TSWM%
    END IF
CASE 2048 'Tip Calculations
    TSWM% = TSW% AND &HF7FF
    IF Value% = Defined% THEN
        TSW% = TSWM% + &H800
    ELSEIF Value% = Undefined% THEN
        TSW% = TSWM%
    END IF
CASE ELSE
    WINDOW 16
    PRINT "SetTipStatusWord
    PRINT "Operation :"; Operation%
    PRINT "Value: "; Value%
    PRINT "Not Defined"
    PRINT "Press a key to exit"
    WHILE INKEY$ = ""
    WEND
    STOP
END SELECT
TipStatusWord%(CurTipNum%) = TSW%
END SUB

'*****Routines imported from Coal Core Flux Calculation *****
'Display Routines
'Read Calc File
' GetFlux
'Window Cordinate
'Screen Management
'*****Display Menu Code*****
SUB HandleDisplayMenu STATIC
    SELECT CASE MENU(1)
        CASE 7 'Display Tip Info
            WinOpen%(12) = True%
            WinUpdate%(12) = True%
            WINDOW 12
        CASE 8 'Tip Plots
            WINDOW 6, "Tip Plots"
            WinOpen%(6) = True%
            WinUpdate%(6) = True%
            WINDOW 6
        CASE 9 'Tip Group Info
            WINDOW 13, "Tip Group Information"
            WinOpen%(13) = True%
            WinUpdate%(13) = True%
            WINDOW 13
        CASE 10
            CLS
        CASE 11 'Redraw Screen
    
```

```

Win% = WINDOW(0)      'Invalidate current window to force redraw
WINDOW CLOSE Win%
WINDOW Win%
CASE 12                'Plot Average Flux Along Core for Export"
WinOpen%(5) = True%
WinUpdate%(5) = True%
WINDOW 5
CASE 1                  'General Info
WinOpen%(11) = True%
WinUpdate%(11) = True%
WINDOW 11
CASE 2                  'SNTB...
Title$ = FrontCalcShortFileName$ + ": South - North - Top - Bottom"
OldBitCode% = SNTBBitCode%
DisplayChoicesDialogBox Title$, SNTBBitCode%, DefaultI%, 31, Okay%
IF Okay% THEN
WinOpen%(7) = True%
WinUpdate%(7) = True%
WINDOW 7, Title$ + ", I = " + STR$(SNTBBitCode% AND 127)
ELSE
SNTBBitCode% = OldBitCode%
END IF
CASE 3                  'SNWE...
Title$ = FrontCalcShortFileName$ + ": South - North - West - East"
OldBitCode% = SNWEBitCode%
DisplayChoicesDialogBox Title$, SNWEBitCode%, DefaultJ%, 31, Okay%
IF Okay% THEN
WinOpen%(8) = True%
WinUpdate%(8) = True%
WINDOW 8, Title$ + ", J = " + STR$(SNWEBitCode% AND 127)
ELSE
SNWEBitCode% = OldBitCode%
END IF
CASE 4                  'WETB...
Title$ = FrontCalcShortFileName$ + ": West - East - Top - Bottom"
OldBitCode% = WETBBitCode%
DisplayChoicesDialogBox Title$, WETBBitCode%, DefaultK%, 63, Okay%
IF Okay% THEN
WinOpen%(9) = True%
WinUpdate%(9) = True%
WINDOW 9, Title$ + ", K = " + STR$(WETBBitCode% AND 127)
ELSE
WETBBitCode% = OldBitCode%
END IF
CASE 5                  'EWTB...
Title$ = FrontCalcShortFileName$ + ": East - West - Top - Bottom"
OldBitCode% = EWTBBitCode%
DisplayChoicesDialogBox Title$, EWTBBitCode%, DefaultK%, 63, Okay%
IF Okay% THEN
WinOpen%(10) = True%
WinUpdate%(10) = True%
WINDOW 10, Title$ + ", K = " + STR$(EWTBBitCode% AND 127)
ELSE
EWTBBitCode% = OldBitCode%
END IF
CASE 6                  'Set Flux Scale...'
CALL SetFluxScaleDialogBox
CASE ELSE
END SELECT
END SUB
SUB SetFluxScaleDialogBox STATIC
WINDOW 16,,(300,200)-(700,300),-2
OldFluxScale! = FluxScale!
Okay% = False%
WHILE NOT Okay%
PRINT "Current Value of Flux Scale Factor:"
PRINT OldFluxScale!
PRINT "New Value of Flux Scale Factor:"

```

```

    INPUT FluxScale!
    IF FluxScale! > 0! THEN Okay% = True%
WEND
OldFatArrowLimit! = FatArrowLimit!
Okay% = False%
WHILE NOT Okay%
    PRINT "Current Value of Fat Arrow Scale Factor:"
    PRINT OldFatArrowLimit!
    PRINT "New Value of Fat Arrow Scale Factor:)"
    INPUT FatArrowLimit!
    IF FatArrowLimit! > 0! THEN Okay% = True%
WEND
WINDOW CLOSE 16
IF WinOpen%(11) = True% THEN CALL RefreshWindow(11) 'Update General Info
END SUB
SUB DisplayChoicesDialogBox (Title$, BitCode%, Default%, MaxNum%, Okay%) STATIC
    IF MaxNum% > 63 OR MaxNum% < 1 THEN
        Okay% = False%
        ExitSub%
    END IF
    IF MaxNum% > 31 THEN
        WINDOW 16,,(300,200)-(750,540),-2
    ELSE
        WINDOW 16,,(300,200)-(750,460),-2
    END IF
    LOCATE 2,4
    TEXTFACE 1
    PRINT Title$
    'Define Buttons
    ViewNum% = (BitCode% AND 127)
    IF ViewNum% = 0 THEN ViewNum% = Default% : IF ViewNum% = 0 THEN STOP
    a% = (BitCode% AND 128) \ 128 + 1 '7th bit = show core? 1 = yes
    B% = (BitCode% AND 256) \ 256 + 1 '8th bit = show pressure?
    c% = (BitCode% AND 512) \ 512 + 1 '9th bit = show flux?
    d% = (BitCode% AND 1024) \ 1024 + 1 '10th bit = show residual?
    E% = (BitCode% AND 2048) \ 2048 + 1 '11 bit = clear screen?
    f% = (BitCode% AND 4096) \ 4096 '12 bit = use default view?
    IF f% = 1 THEN
        ViewNum% = Default%
        T1% = 2
        T2% = 1
    ELSE
        T1% = 1
        T2% = 2
    END IF
    BUTTON 64, a%, "Show Core Material", (20, 40)-(300, 60), 2
    BUTTON 65, B%, "Show Pressure Field", (20, 60)-(300, 80), 2
    BUTTON 66, c%, "Show Flux", (20, 80)-(300, 100), 2
    BUTTON 67, d%, "Show Flux Discrepancy", (20, 100)-(300, 120), 2
    BUTTON 68, E%, "Clear Screen Completely", (20, 120)-(300, 140), 2
    BUTTON 69, T1%, "Use Default View", (20, 160)-(200, 180), 3
    BUTTON 70, T2%, "Select View", (20, 180)-(200, 200), 3
    BUTTON 80, 1, "Cancel", (300, 40)-(400, 60), 1
    BUTTON 81, 2, "Okay", (300, 80)-(400, 100), 1
    rect%(0) = 170 : rect%(1) = 210 : rect%(2) = 190 : rect%(3) = 400
    ViewNumText$ = "Pointer: " + STR$(ViewNum%)
    just% = 0
    TextBox ViewNumText$, rect%(0), just%
    IF MaxNum% = 31 THEN
        ScaleI% = 13
        ModFactor% = 2
    ELSE
        ScaleI% = 6
        ModFactor% = 4
    END IF
    IF f% = 0 THEN IsOn% = 1 ELSE IsOn% = 0
    FOR i% = 1 TO MaxNum%

```

```

x% = 20 + ScaleX%* i%
y% = 200 + ((i% + 3) MOD ModFactor%) * 24
BUTTON i%, IsOn%, "", (x%, y%)-(x% + 20, y% + 20), 3
NEXT
IF f% = 0 THEN BUTTON ViewNum%, 2 'Show current view button
'Button Management Code:
DontExit% = True%
WHILE DontExit%
  Dialog0% = 0
  WHILE Dialog0% <> 1
    Dialog0% = DIALOG(0)
  WEND
  ButtonNum% = DIALOG(1)
  SELECT CASE ButtonNum%
    CASE 1 TO 63 'Radio Buttons which mark current view
      BUTTON ViewNum%, 1
      ViewNum% = ButtonNum%
      BUTTON ViewNum%, 2
      ViewNumText$ = "Pointer: " + STR$(ViewNum%)
      just% = 0
      TextBox ViewNumText$, rect%(0), just%
    CASE 64, 65, 66, 67, 68 'A Check Box has been toggle
      IF BUTTON(ButtonNum%) = 1 THEN
        BUTTON ButtonNum%, 2
      ELSE
        BUTTON ButtonNum%, 1
      END IF
    CASE 69 'Radio button for defaultview pushed
      BUTTON 69, 2
      BUTTON 70, 1
      FOR i% = 1 TO MaxNum%
        BUTTON i%, 0
      NEXT
      ViewNumText$ = "Pointer = " + STR$(Default%)
      just% = 0
      TextBox ViewNumText$, rect%(0), just%
    CASE 70 'Radio button for select view pushed
      BUTTON 69, 1
      BUTTON 70, 2
      FOR i% = 1 TO MaxNum%
        BUTTON i%, 1
      NEXT
      BUTTON ViewNum%, 2
      ViewNumText$ = "Pointer = " + STR$(ViewNum%)
      just% = 0
      TextBox ViewNumText$, rect%(0), just%
    CASE 80 'Cancel
      DontExit% = False%
      Okay% = False%
    CASE 81
      DontExit% = False%
      Okay% = True%
    CASE ELSE
      END SELECT
  WEND
  IF Okay% THEN
    'Store value here
    IF BUTTON(64) = 2 THEN BitCode% = 128 ELSE BitCode% = 0
    IF BUTTON(65) = 2 THEN BitCode% = BitCode% + 256
    IF BUTTON(66) = 2 THEN BitCode% = BitCode% + 512
    IF BUTTON(67) = 2 THEN BitCode% = BitCode% + 1024
    IF BUTTON(68) = 2 THEN BitCode% = BitCode% + 2048
    IF BUTTON(69) = 2 THEN
      BitCode% = BitCode% + Default%
      BitCode% = BitCode% + 4096
    ELSE
      BitCode% = BitCode% + ViewNum%
    END IF
  END IF

```

```

END IF
WINDOW CLOSE 16
END SUB
'Display Writing Code
SUB UpdateActiveDisplays STATIC
FOR i% = 1 TO 16
    IF WinUpdate%(i%) = True% THEN
        IF WinOpen%(i%) = True% THEN
            CALL RefreshWindow(i%)
        END IF
    END IF
NEXT
END SUB
SUB WindowInitialize STATIC
WINDOW 1, "Coal Core Surface Calculations", (50,50)-(800,100), 7
WINDOW 5, "Average Flux along Core", (250,80)-(650,480), 7
WINDOW 6, "Tip Plots", (250, 510)-(650, 910), 7
WINDOW 7, "South - North - Top - Bottom", (660, 80)-(1280, 480), 7
WINDOW 8, "South - North - West - East", (660, 510)-(1280, 910), 7
WINDOW 9, "West - East - Top - Bottom", (250,80)-(650,480), 7
WINDOW 10, "East - West - Top - Bottom", (250, 510)-(650, 910), 7
WINDOW 11, "General Info", (50, 80)-(240,400), 6
WINDOW 12, "Tip Information", (50, 50) - (500, 800), 6
WINDOW 13, "Tip Group Information", (50, 50) - (500, 400), 6
'Window 16 is for dialog boxes that immediately disappear
WINDOW CLOSE 13
WINDOW CLOSE 12
WINDOW CLOSE 11
WINDOW CLOSE 10
WINDOW CLOSE 9
WINDOW CLOSE 8
WINDOW CLOSE 7
WINDOW CLOSE 6
WINDOW CLOSE 5
END SUB
SUB RefreshWindow (i%) STATIC
IF WinOpen%(i%) = True% THEN
    WINDOW OUTPUT i%
    SELECT CASE i%
        CASE 5
            PlotAverageFluxAlongCoreForExport
        CASE 6
            DisplayTipPlots
        CASE 7
            DisplaySNTB
        CASE 8
            DisplaySEWE
        CASE 9
            DisplayWETB
        CASE 10
            DisplayEWTB
        CASE 11
            DisplayGeneralInfo
        CASE 12
            DisplayTipInfo
        CASE 13
            DisplayTipGroupInfo
        CASE ELSE
            '
    END SELECT
END IF
END SUB
SUB DisplayTipGroupInfo STATIC
CLS
TEXTFONT 0
TEXTSIZE (12)
LOCATE 2,1
PRINT #22, "   File Name:                "; FrontCalcShortFileName$
PRINT #22, "   Tip Group Number:            "; TipGroupNum%

```

```

PRINT #22, " Tip Group Name: "; TipGroupName$(TipGroupNum%)
PRINT #22, " Tip Group Comment: "; TipGroupComment$(TipGroupNum%)
PRINT #22,
  USING " Tip Normal Flux: ###.###- - - - "; TipGroupFluxNorm!(TipGroupNum%)
PRINT #22,
  USING " Uncertainty: ###.###- - - - ";
  TipGroupFluxNormUnCert!(TipGroupNum%)
PRINT #22,
  USING " 2d Laplacian: ###.###- - - - ";
  TipGroup2DLaplacian!(TipGroupNum%)
PRINT #22,
  USING " Uncertainty: ###.###- - - - ";
  TipGroup2DLaplacianUnCert!(TipGroupNum%)
PRINT #22, USING " Average SN Flux: ###.###- - - - ";
  AverageFluxAlongCore!(32)

PRINT #22,
PRINT #22,
PRINT #22, " All results are reported in internal units."
PRINT #22, " Internal length scale = 0.5 cm. Fluxes"
PRINT #22, " must be scaled with Average Flux internal "
PRINT #22, " and Average Dimensional Flux (based on the"
PRINT #22, " coal XS area)."
END SUB
SUB DisplayGeneralInfo STATIC
CLS
TEXTFONT 0
TEXTSIZE (12)
PRINT #22,
PRINT #22, " File Name: "
PRINT #22, " "; FrontCalcShortFileName$
PRINT #22, USING " Mesh Level: ## of 14"; MeshLevel%
PRINT #22, USING " Step in I: ##"; StepI%
PRINT #22, USING " Step in J: ##"; StepJ%
PRINT #22, USING " Step in K: ##"; StepK%
PRINT #22, USING " Iterations: ###"; ItsAtLevel%
PRINT #22, USING " Relaxation: ###.###"; SOR!
PRINT #22, USING " Tolerance: ###.###- - - -"; Tol!
PRINT #22, USING " Scaled Error: ###.###- - - -"; RelErr!
PRINT #22, USING " Total Error: ###.###- - - -"; TotErr!
PRINT #22, USING " Min XS Flux: ###.###- - - -"; AverageFluxAlongCore!(0)
PRINT #22, USING " 1/4 XS Flux: ###.###- - - -"; AverageFluxAlongCore!(16)
PRINT #22, USING " 1/2 XS Flux: ###.###- - - -"; AverageFluxAlongCore!(32)
PRINT #22, USING " 3/4 XS Flux: ###.###- - - -"; AverageFluxAlongCore!(48)
PRINT #22, USING " Max XS Flux: ###.###- - - -"; AverageFluxAlongCore!(64)
END SUB
'*****SNTB - Window 7*****
SUB DisplaySNTB STATIC
WinWidth% = WINDOW(2)
WinHeight% = WINDOW(3)
IF WinWidth% <> OldWinWidth% OR WinHeight% <> OldWinHeight% THEN
  CLS
  OldWinWidth% = WinWidth%
  OldWinHeight% = WinHeight%
END IF
i% = SNTBBitCode% AND 127
i% = (i% \ StepI%) * StepI% 'Find nearest active plane
IF j% = 0 THEN j% = StepJ%
IF i% = 32 THEN i% = 32 - StepI%
IF SNTBBitCode% AND 2048 THEN CLS
IF SNTBBitCode% AND 128 THEN CALL DisplayShapeSNTB (i%)
IF SNTBBitCode% AND 256 THEN CALL DisplayPresSNTB(i%)
IF SNTBBitCode% AND 512 THEN CALL DisplayFluxSNTB(i%)
IF SNTBBitCode% AND 1024 THEN CALL DisplayErrorFluxSNTB(i%)
DisplayFrontCurveInfoSW SNTB%, i%
END SUB
SUB DisplayFrontCurveInfoSW (TheView%, TheFixedCoor%) STATIC
PENNORMAL
IF TipGroupNum% <> 0 THEN

```



```

GetTipNum TipGroupNum%, TheView%, CurTipNum%
GetTipStatusWord Curve%, TheCurve%
IF TheCurve% = Defined% AND TheFixedCoor% = TipFixedCoor%(CurTipNum%) THEN
    LoadTemporaryArrays
    SELECT CASE MouseTool%
        CASE 1, 2 'Draw Curve or Reshape Curve
            DrawCurve True%, TheView%
        CASE 3 'Define Normal Fluxes
            DrawCurve NoPoints%, TheView%
            CALL GetWindowPositionScale (k0%, dk%, j0%, dj%)
            FOR f% = 1 TO LastFluxPtr%(CurTipNum%)
                xl% = fkCoor%(CurTipNum%, f%)
                IF TheView% = SNTB% THEN
                    yl% = fjCoor%(CurTipNum%, f%)
                ELSE
                    yl% = fiCoor%(CurTipNum%, f%)
                END IF
                PSET (k0% + 2 * xl% * dk%, j0% + 2 * yl% * dj%)
                CIRCLE STEP (0,0), dk% \ 2
                XS! = fsX!(CurTipNum%, f%)
                yS! = fsY!(CurTipNum%, f%)
                PSET (k0% + 2 * XS! * dk%, j0% + 2 * yS! * dj%)
                CIRCLE STEP (0, 0), 5
                CIRCLE STEP (0, 0), 3
            NEXT
        CASE ELSE
            END SELECT
    END IF
END IF
END SUB
SUB DisplayPresSNTB (i%) STATIC
    CALL GetWindowPositionScale (k0%, dk%, j0%, dj%)
    FOR j% = 0 TO 32 STEP StepJ%
        FOR k% = 0 TO 64 STEP StepK%
            PSET (k0% + 2 * k% * dk%, j0% + 2 * j% * dj%)
            r% = dj% * SQR((P!(i%, j%, k%) + 1))
            CIRCLE STEP (0,0),r%
        NEXT
    NEXT
END SUB
SUB DisplayErrorFluxSNTB (i%) STATIC
    CALL GetWindowPositionScale (k0%, dk%, j0%, dj%)
    FOR j% = StepJ% TO 31 STEP StepJ%
        FOR k% = StepK% TO 63 STEP StepK%
            CALL GetPermCode(i%, j%, k%, Code%)
            IF Code% = 3 THEN
                'Don't display flux for void region
            ELSE
                PSET (k0% + 2 * k% * dk%, j0% + 2 * j% * dj%)
                CALL GetFlux(i%, j%, k%, Flux!())
                ErrorFlux! = Flux!(1) + Flux!(2) + Flux!(3) +
                    Flux!(4) + Flux!(5) + Flux!(6)
                r% = dj% * ABS(ErrorFlux!) * 100
                IF ErrorFlux! > 0 THEN
                    CIRCLE STEP (0, 0), r%
                    LINE STEP (0, -r%) -STEP (0, 2 * r%)
                ELSE
                    CIRCLE STEP (0, 0), r%
                    LINE STEP (-r%, 0) -STEP (2 * r%, 0)
                END IF
            END IF
        NEXT
    NEXT
END SUB
SUB DisplayFluxSNTB (i%) STATIC
    CALL GetWindowPositionScale (k0%, dk%, j0%, dj%)
    Headsize% = dj% \ 2
    maxlength% = CSNG(dj%) * FatArrowLimit!

```

```

FOR j% = StepJ% TO 31 STEP StepJ%
  FOR k% = StepK% TO 63 STEP StepK%
    CALL GetPermCode(i%, j%, k%, Code%)
    IF Code% = 3 THEN
      'Don't display flux for void region
    ELSE
      MOVETO k0% + 2 * k% * dk%, j0% + 2 * j% * dj%
      CALL GetFlux(i%, j%, k%, Flux!())
      jFlux! = Flux!(3) - Flux!(4)
      kFlux! = Flux!(5) - Flux!(6)
      kk% = kFlux! * dk% * FluxScale!
      jj% = jFlux! * dj% * FluxScale!
      Arrow kk%, jj%, Headsize%, maxlength%
    END IF
  NEXT
NEXT
END SUB
SUB DisplayShapeSWTB (i%) STATIC
  CALL GetWindowPositionScale (k0%, dk%, j0%, dj%)
  FOR j% = 1 TO 31
    FOR k% = 1 TO 63
      rect%(1) = k0% + k% * 2 * dk% - dk%
      rect%(0) = j0% + j% * 2 * dj% - dj%
      rect%(3) = k0% + k% * 2 * dk% + dk%
      rect%(2) = j0% + j% * 2 * dj% + dj%
      CALL GetPermCode(i%, j%, k%, Code%)
      CALL FILLRECT(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
    NEXT
  NEXT
END SUB
'*****South - North - West - East*****
SUB DisplaySNWE STATIC
  WinWidth% = WINDOW(2)
  WinHeight% = WINDOW(3)
  IF WinWidth% <> OldWinWidth% OR WinHeight% <> OldWinHeight% THEN
    CLS
    OldWinWidth% = WinWidth%
    OldWinHeight% = WinHeight%
  END IF
  j% = SNWEBitCode% AND 127
  j% = (j% \ StepJ%) * StepJ% 'Find nearest active plane
  IF j% = 32 THEN j% = 32 - StepJ%
  IF j% = 0 THEN j% = StepJ%
  IF SNWEBitCode% AND 2048 THEN CLS
  IF SNWEBitCode% AND 128 THEN CALL DisplayShapeSNWE (j%)
  IF SNWEBitCode% AND 256 THEN CALL DisplayPresSNWE(j%)
  IF SNWEBitCode% AND 512 THEN CALL DisplayFluxSNWE(j%)
  IF SNWEBitCode% AND 1024 THEN CALL DisplayErrorFluxSNWE(j%)
  DisplayFrontCurveInfoSN SNWE%, j%
  PlotAverageFluxAlongCore
END SUB
SUB DisplayPresSNWE (j%) STATIC
  CALL GetWindowPositionScale (k0%, dk%, i0%, di%)
  FOR i% = 0 TO 32 STEP StepI%
    FOR k% = 0 TO 64 STEP StepK%
      PSET (k0% + 2 * k% * dk%, i0% + 2 * i% * di%)
      r% = di% * SQR((P!(i%, j%, k%) + 1!))
      CIRCLE STEP (0,0),r%
    NEXT
  NEXT
END SUB
SUB DisplayErrorFluxSNWE (j%) STATIC
  CALL GetWindowPositionScale (k0%, dk%, i0%, di%)
  Headsize% = di% \ 2
  maxlength% = CSNG(di%) * FatArrowLimit!
  FOR i% = StepI% TO 31 STEP StepI%
    FOR k% = StepK% TO 63 STEP StepK%
      CALL GetPermCode(i%, j%, k%, Code%)

```

```

IF Code% = 3 THEN
  'Don't display flux for void region
ELSE
  PSET (k0% + 2 * k% * dk%, i0% + 2 * i% * di%)
  CALL GetFlux(i%, j%, k%, Flux!())
  ErrorFlux! = Flux!(1) + Flux!(2) +
    Flux!(3) + Flux!(4) + Flux!(5) + Flux!(6)
  r% = di% * ABS(ErrorFlux!) * 100
  IF ErrorFlux! > 0 THEN
    CIRCLE STEP (0, 0), r%
    LINE STEP (0, -r%) -STEP (0, 2 * r%)
  ELSE
    CIRCLE STEP (0, 0), r%
    LINE STEP (-r%, 0) -STEP (2 * r%, 0)
  END IF
END IF
NEXT
NEXT
END SUB
SUB DisplayFluxSNWE (j%) STATIC
  CALL GetWindowPositionScale (k0%, dk%, i0%, di%)
  Headsize% = di% \ 2
  maxlength% = CSNG(di%) * FatArrowLimit!
  FOR i% = StepI% TO 31 STEP StepI%
    FOR k% = StepK% TO 63 STEP StepK%
      CALL GetPermCode(i%, j%, k%, Code%)
      IF Code% = 3 THEN
        'Don't display flux for void region
      ELSE
        MOVETO k0% + 2 * k% * dk%, i0% + 2 * i% * di%
        CALL GetFlux(i%, j%, k%, Flux!())
        iFlux! = Flux!(1) - Flux!(2)
        kFlux! = Flux!(5) - Flux!(6)
        kk% = kFlux! * dk% * FluxScale!
        ii% = iFlux! * di% * FluxScale!
        Arrow kk%, ii%, Headsize%, maxlength%
      END IF
    NEXT
  NEXT
NEXT
END SUB
SUB DisplayShapeSNWE (j%) STATIC
  CALL GetWindowPositionScale (k0%, dk%, i0%, di%)
  FOR i% = 1 TO 31
    FOR k% = 1 TO 63
      rect%(1) = k0% + k% * 2 * dk% - dk%
      rect%(0) = i0% + i% * 2 * di% - di%
      rect%(3) = k0% + k% * 2 * dk% + dk%
      rect%(2) = i0% + i% * 2 * di% + di%
      CALL GetPermCode(i%, j%, k%, Code%)
      CALL FILLRECT(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
    NEXT
  NEXT
NEXT
END SUB
*****
SUB GetWindowPositionScale (k0%, dk%, j0%, dj%) STATIC
  WinWidth% = WINDOW(2)
  WinHeight% = WINDOW(3)
  j0% = WinHeight% \ 6
  SampleLen% = j0% * 8
  IF SampleLen% > WinWidth% THEN
    k0% = j0%
  ELSE
    k0% = (WinWidth% - SampleLen%) \ 2
    IF k0% > j0% THEN
      k0% = j0%
    ELSEIF k0% < 5 THEN
      k0% = 5
    END IF
  END IF

```

```

END IF
dj% = j0% \ 16
dk% = dj%
END SUB
'*****Display WETB Code*****
SUB DisplayWETB STATIC
IF StepK% = 64 THEN EXIT SUB
WinWidth% = WINDOW(2)
WinHeight% = WINDOW(3)
IF WinWidth% <> OldWinWidth% OR WinHeight% <> OldWinHeight% THEN
CLS
OldWinWidth% = WinWidth%
OldWinHeight% = WinHeight%
END IF
k% = WETBBitCode% AND 127
k% = (k% \ StepK%) * StepK% 'Find nearest active plane
IF k% = 64 THEN k% = 64 - StepK%
IF k% = 0 THEN k% = StepK%
IF WETBBitCode% AND 2048 THEN CLS
IF WETBBitCode% AND 128 THEN CALL DisplayShapeWETB (k%)
IF WETBBitCode% AND 256 THEN CALL DisplayPresWETB(k%)
IF WETBBitCode% AND 512 THEN CALL DisplayFluxWETB(k%)
IF WETBBitCode% AND 1024 THEN CALL DisplayErrorFluxWETB(k%)
DisplayFrontCurveInfoWETB(k%)
END SUB
SUB DisplayFrontCurveInfoWETB(k%) STATIC
IF TipGroupNum% <> 0 THEN
TheView% = WETB%
GetTipNum TipGroupNum%, TheView%, CurTipNum%
GetTipStatusWord Curve%, TheCurve%
IF TheCurve% = Defined% AND k% = TipFixedCoor%(CurTipNum%) THEN
LoadTemporaryArrays
DrawCurve True%, TheView%
END IF
END IF
END SUB
SUB DisplayPresWETB (k%) STATIC
CALL GetWindowPositionScales (i0%, di%, j0%, dj%)
FOR j% = 0 TO 32 STEP StepJ%
FOR i% = 0 TO 32 STEP StepI%
PSET (i0% + 2 * i% * di%, j0% + 2 * j% * dj%)
r% = dj% * SQR((P!(i%, j%, k%) + 1!))
CIRCLE STEP (0,0), r%
NEXT
NEXT
END SUB
SUB DisplayErrorFluxWETB (k%) STATIC
CALL GetWindowPositionScales (i0%, di%, j0%, dj%)
Headsize% = dj% \ 2
maxlength% = CSNG(dj%) * FatArrowLimit!
FOR j% = StepJ% TO 31 STEP StepJ%
FOR i% = StepI% TO 31 STEP StepI%
CALL GetPermCode(i%, j%, k%, Code%)
IF Code% = 3 THEN
'Don't display flux for void region
ELSE
PSET (i0% + 2 * i% * di%, j0% + 2 * j% * dj%)
CALL GetFlux(i%, j%, k%, Flux!())
ErrorFlux! = Flux!(1) + Flux!(2) + Flux!(3) +
Flux!(4) + Flux!(5) + Flux!(6)
r% = dj% * ABS(ErrorFlux!) * 100
IF ErrorFlux! > 0 THEN
CIRCLE STEP (0, 0), r%
LINE STEP (0, -r%) -STEP (0, 2 * r%)
ELSE
CIRCLE STEP (0, 0), r%
LINE STEP (-r%, 0) -STEP (2 * r%, 0)
END IF
END IF
END IF

```

```

    END IF
  NEXT
NEXT
END SUB
SUB DisplayFluxWETB (k%) STATIC
  CALL GetWindowPositionScaleS (i0%, di%, j0%, dj%)
  Headsize% = dj% \ 2
  maxlength% = CSNG(dj%) * FatArrowLimit!
  FOR j% = StepJ% TO 31 STEP StepJ%
    FOR i% = StepI% TO 31 STEP StepI%
      CALL GetPermCode(i%, j%, k%, Code%)
      IF Code% = 3 THEN
        'Don't display flux for void region
      ELSE
        MOVETO i0% + 2 * i% * di%, j0% + 2 * j% * dj%
        CALL GetFlux(i%, j%, k%, Flux!())
        jFlux! = Flux!(3) - Flux!(4)
        iFlux! = Flux!(1) - Flux!(2)
        ii% = iFlux! * di% * FluxScale!
        jj% = jFlux! * dj% * FluxScale!
        Arrow ii%, jj%, Headsize%, maxlength%
      END IF
    NEXT
  NEXT
NEXT
END SUB
SUB DisplayShapeWETB (k%) STATIC
  CALL GetWindowPositionScaleS (i0%, di%, j0%, dj%)
  FOR j% = 1 TO 31
    FOR i% = 1 TO 31
      rect%(1) = i0% + i% * 2 * di% - di%
      rect%(0) = j0% + j% * 2 * dj% - dj%
      rect%(3) = i0% + i% * 2 * di% + di%
      rect%(2) = j0% + j% * 2 * dj% + dj%
      CALL GetPermCode(i%, j%, k%, Code%)
      CALL FILLRECT(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
    NEXT
  NEXT
NEXT
END SUB
'*****Display EWTB Code*****
SUB DisplayEWTB STATIC
  IF StepK% = 64 THEN EXIT SUB
  WinWidth% = WINDOW(2)
  WinHeight% = WINDOW(3)
  IF WinWidth% <> OldWinWidth% OR WinHeight% <> OldWinHeight% THEN
    CLS
    OldWinWidth% = WinWidth%
    OldWinHeight% = WinHeight%
  END IF
  k% = EWTBBitCode% AND 127
  k% = (k% \ StepK%) * StepK% 'Find nearest active plane
  IF k% = 64 THEN k% = 64 - StepK%
  IF k% = 0 THEN k% = StepK%
  IF EWTBBitCode% AND 2048 THEN CLS
  IF EWTBBitCode% AND 128 THEN CALL DisplayShapeEWTB (k%)
  IF EWTBBitCode% AND 256 THEN CALL DisplayPresEWTB(k%)
  IF EWTBBitCode% AND 512 THEN CALL DisplayFluxEWTB(k%)
  IF EWTBBitCode% AND 1024 THEN CALL DisplayErrorFluxEWTB(k%)
  DisplayFrontCurveInfoEWTB(k%)
END SUB
SUB DisplayFrontCurveInfoEWTB(k%) STATIC
  IF TipGroupNum% <> 0 THEN
    TheView% = EWTB%
    GetTipNum TipGroupNum%, TheView%, CurTipNum%
    GetTipStatusWord Curve%, TheCurve%
    IF TheCurve% = Defined% AND k% = TipFixedCoor%(CurTipNum%) THEN
      LoadTemporaryArrays
      DrawCurve True%, TheView%
    END IF
  END IF

```

```

END IF
END SUB
SUB DisplayPresEWTB (k%) STATIC
CALL GetWindowPositionScaleS (i0%, di%, j0%, dj%)
FOR j% = 0 TO 32 STEP StepJ%
  FOR i% = 0 TO 32 STEP StepI%
    PSET (i0% + 2 * (32 - i%) * di%, j0% + 2 * j% * dj%)
    r% = dj% * SQR((P!(i%, j%, k%) + 1))
    CIRCLE STEP (0,0),r%
  NEXT
NEXT
END SUB
SUB DisplayErrorFluxEWTB (k%) STATIC
CALL GetWindowPositionScaleS (i0%, di%, j0%, dj%)
Headsizes% = dj% \ 2
maxlength% = CSNG(dj%) * FatArrowLimit!
FOR j% = StepJ% TO 31 STEP StepJ%
  FOR i% = StepI% TO 31 STEP StepI%
    CALL GetPermCode(i%, j%, k%, Code%)
    IF Code% = 3 THEN
      'Don't display flux for void region
    ELSE
      PSET (i0% + 2 * (32 - i%) * di%, j0% + 2 * j% * dj%)
      CALL GetFlux(i%, j%, k%, Flux!())
      ErrorFlux! = Flux!(1) + Flux!(2) + Flux!(3) +
                  Flux!(4) + Flux!(5) + Flux!(6)
      r% = dj% * ABS(ErrorFlux!) * 100
      IF ErrorFlux! > 0 THEN
        CIRCLE STEP (0, 0), r%
        LINE STEP (0, -r%) -STEP (0, 2 * r%)
      ELSE
        CIRCLE STEP (0, 0), r%
        LINE STEP (-r%, 0) -STEP (2 * r%, 0)
      END IF
    END IF
  NEXT
NEXT
END SUB
SUB DisplayFluxEWTB (k%) STATIC
CALL GetWindowPositionScaleS (i0%, di%, j0%, dj%)
Headsizes% = dj% \ 2
maxlength% = CSNG(dj%) * FatArrowLimit!
FOR j% = StepJ% TO 31 STEP StepJ%
  FOR i% = StepI% TO 31 STEP StepI%
    CALL GetPermCode(i%, j%, k%, Code%)
    IF Code% = 3 THEN
      'Don't display flux for void region
    ELSE
      MOVETO i0% + 2 * (32 - i%) * di%, j0% + 2 * j% * dj%
      CALL GetFlux(i%, j%, k%, Flux!())
      jFlux! = Flux!(3) - Flux!(4)
      iFlux! = Flux!(1) - Flux!(2)
      ii% = iFlux! * di% * FluxScale!
      jj% = jFlux! * dj% * FluxScale!
      Arrow ii%, jj%, Headsizes%, maxlength%
    END IF
  NEXT
NEXT
END SUB
SUB DisplayShapeEWTB (k%) STATIC
CALL GetWindowPositionScaleS (i0%, di%, j0%, dj%)
FOR j% = 1 TO 31
  FOR i% = 1 TO 31
    rect%(1) = i0% + (32 - i%) * 2 * di% - di%
    rect%(0) = j0% + j% * 2 * dj% - dj%
    rect%(3) = i0% + (32 - i%) * 2 * di% + di%
    rect%(2) = j0% + j% * 2 * dj% + dj%
    CALL GetPermCode(i%, j%, k%, Code%)
  NEXT
NEXT
END SUB

```

```

        CALL FILLRECT(VARPTR(rect%(0)), VARPTR(Patterns%(Code%*4)))
    NEXT
NEXT
END SUB
*****
SUB GetWindowPositionScales (k0%, dk%, j0%, dj%) STATIC
    WinWidth% = WINDOW(2)
    WinHeight% = WINDOW(3)
    j0% = WinHeight% \ 6
    SampleWidth% = j0% * 4
    IF SampleWidth% > WinWidth% THEN
        k0% = j0%
    ELSE
        k0% = (WinWidth% - SampleWidth%) \ 2
        IF k0% > j0% THEN
            k0% = j0%
        ELSEIF k0% < 5 THEN
            k0% = 5
        END IF
    END IF
    dj% = j0% \ 16
    dk% = dj%
END SUB
*****
SUB PlotAverageFluxAlongCore STATIC
    IF AverageFluxAlongCore!(0) > 0! THEN
        yb! = 0! ELSE yb! = AverageFluxAlongCore!(0)
    IF AverageFluxAlongCore!(64) < 0
        THEN yt! = 0! ELSE yt! = AverageFluxAlongCore!(64)
    IF yt! = yb! THEN EXIT SUB
    GetWindowPositionScale k0%, dk%, j0%, dj%
    'Draw Box
    xMin% = k0%
    xMax% = k0% + 2 * 64 * dk%
    yMax% = j0% + 2 * 34 * dj%
    yMin% = j0% + 2 * 40 * dj%
    LINE (xMin%, yMin% + dj%) - (xMax%, yMax% - dj%), , B
    IF AverageFluxAlongCore!(0) > 0! THEN
        yb! = 0! ELSE yb! = AverageFluxAlongCore!(0)
    IF AverageFluxAlongCore!(64) < 0 THEN
        yt! = 0! ELSE yt! = AverageFluxAlongCore!(64)
    Slope! = CSNG(yMax% - yMin%) / (yt! - yb!)
    'Draw the zero flux line
    y0% = yMin% - yb! * Slope!
    PENSIZE 2,2
    MOVETO xMin%, y0%
    LINETO xMax%, y0%
    x% = k0% + 2 * dk%
    y% = yMin% + (AverageFluxAlongCore!(1) - yb!) * Slope!
    MOVETO x%, y%
    FOR k% = 2 TO 63
        'Map x%
        x% = k0% + 2 * k% * dk%
        'Map y%
        y% = yMin% + (AverageFluxAlongCore!(k%) - yb!) * Slope!
        LINETO x%, y%
    NEXT
    PENNORMAL
END SUB
SUB PlotAverageFluxAlongCoreForExport STATIC
    CLS
    IF AverageFluxAlongCore!(0) > 0! THEN
        yb! = 0! ELSE yb! = AverageFluxAlongCore!(0)
    IF AverageFluxAlongCore!(64) < 0 THEN
        yt! = 0! ELSE yt! = AverageFluxAlongCore!(64)
    IF yt! = yb! THEN EXIT SUB
    'Draw Box
    xMin% = 10

```

```

xMax% = WINDOW(2) - 10
yMax% = 10
yMin% = WINDOW(3) - 20
LINE (xMin%, yMin% + 10) - (xMax%, yMax% - 10), , B
IF AverageFluxAlongCore!(0) > 0! THEN
    yb! = 0! ELSE yb! = AverageFluxAlongCore!(0)
IF AverageFluxAlongCore!(64) < 0 THEN
    yt! = 0! ELSE yt! = AverageFluxAlongCore!(64)
SlopeY! = CSNG(yMax% - yMin%) / (yt! - yb!)
SlopeX! = CSNG(xMax% - xMin%) / (64! - 0!)
'Draw the zero flux line
y0% = yMin% - yb! * SlopeY!
PENSIZE 2,2
MOVETO xMin%, y0%
LINETO xMax%, y0%
x% = xMin% + (1) * SlopeX!
y% = yMin% + (AverageFluxAlongCore!(1) - yb!) * SlopeY!
MOVETO x%, y%
FOR k% = 2 TO 63
    'Map x%
    x% = xMin% + k% * SlopeX!
    'Map y%
    y% = yMin% + (AverageFluxAlongCore!(k%) - yb!) * SlopeY!
    LINETO x%, y%
NEXT
PENNORMAL
END SUB
'Utility Sub Routines
SUB GetFlux(i%, j%, k%, FluxOut!()) STATIC
    'Calculates the Flux at a given node
    CALL GetPermCode(i%, j%, k%, Code%)
    Perm0! = MatPerm!(Code%)
    Accum1! = 0!
    Accum2! = 0!
    FOR w% = 1 TO 6
        iz% = i% + OffsetI%(w%) * StepI%
        jz% = j% + OffsetJ%(w%) * StepJ%
        kz% = k% + OffsetK%(w%) * StepK%
        ls% = iz% + jz% + kz% - i% - j% - k%
        RL! = 1! / ABS(ls%)
        CALL GetPermCode(iz%, jz%, kz%, Code%)
        Perm1! = MatPerm!(Code%)
        Perm! = (Perm1! * Perm0!) / (Perm1! + Perm0!)
        FluxOut!(w%) = Perm! * (P!(iz%, jz%, kz%) - P!(i%, j%, k%)) * RL!
    NEXT
END SUB
SUB GetPermCode (i%, j%, k%, Code%) STATIC
    Segment% = i% \ 4
    Offset% = i% MOD 4
    'Segment = 0 to 8
    'Offset = 0 to 3
    IF Segment% > 8 THEN
        Code% = 999
    EXIT SUB
END IF
Old% = PermCodeArray%(Segment%, j%, k%)
IF Old% < 0 THEN Old% = Old% + 65536%
SELECT CASE Offset%
CASE 0
    Mask% = &HF
    Shift% = 1
CASE 1
    Mask% = &HFO
    Shift% = 16
CASE 2
    Mask% = &HF00
    Shift% = 256
CASE 3
    Mask% = &HF000
    Shift% = 4096

```



```

END SELECT
Code% = (Old% AND Mask%) \ Shift%
END SUB
'Start up, Misc Code
SUB SetDefaultValues STATIC
'Defaults
CalcFileName$ = "Untitled - Calc"
ChangesSaved% = True%
StepI% = 16
StepJ% = 16
StepK% = 64
SOR! = 1.5
Tol! = .000001
MeshLevel% = 0
FluxScale! = 25!
FatArrowLimit! = 2!
MatPerm!(0) = .00001 'Boundary
MatPerm!(1) = 10! 'Char
MatPerm!(2) = .1 'Shale
MatPerm!(3) = 10! 'Void
MatPerm!(4) = 1! 'Coal
MatPerm!(5) = .05 'Concrete
MatPerm!(6) = 1
MatPerm!(7) = 1
MatPerm!(8) = 1
MatPerm!(9) = 1
MatPerm!(10) = 1
MatPerm!(11) = 1
MatPerm!(12) = 1
MatPerm!(13) = 1
MatPerm!(14) = 1
MatPerm!(15) = 1
DefaultI% = 16
DefaultJ% = 16
DefaultK% = 32
END SUB
SUB StartUpCode STATIC
False% = 0 : True% = NOT False%
SetArray OffsetI%(1), -1, 1, 0, 0, 0, 0
SetArray OffsetJ%(1), 0, 0, -1, 1, 0, 0
SetArray OffsetK%(1), 0, 0, 0, 0, -1, 1
SetArray Patterns%(4), &H800, &H20, &H200, &H0 'Char
SetArray Patterns%(8), &H8822, &H8822, &H8822, &H8822 'Shale
SetArray Patterns%(12), &H0, &H0, &H0, &H0 'Void
SetArray Patterns%(16), &H9002, &H48, &H220, &H8400 'Coal
SetArray Patterns%(20), &HAA55, &HAA55, &HAA55, &HAA55 'Cement
END SUB
SUB FixUpPermCodeArray STATIC
FOR i% = 1 TO 7
FOR j% = 1 TO 31
PermCodeArray%(i%, j%, 0) = PermCodeArray%(i%, j%, 1)
PermCodeArray%(i%, j%, 64) = PermCodeArray%(i%, j%, 63)
NEXT j%
NEXT i%
END SUB
SUB Arrow (dX%, dY%, size%, maxlength%) STATIC
dX% = dX%
dY% = dY%
LineLength% = SQR(dX% * dX% + dY% * dY%)
IF LineLength% < maxlength% THEN
PENSIZE 1,1
ELSEIF LineLength% \ 4 < maxlength% THEN
dX% = dX% \ 4
dY% = dY% \ 4
LineLength% = LineLength% \ 4
PENSIZE 2,2
ELSEIF LineLength% \ 9 < maxlength% THEN

```

```

    dX% = dX% \ 9
    dY% = dY% \ 9
    LineLength% = LineLength% \ 9
    PENSIZE 3,3
ELSE
    dX% = dX% \ 16
    dY% = dY% \ 16
    LineLength% = LineLength% \ 16
    PENSIZE 4,4
END IF
IF LineLength% < size% THEN LineLength% = size%
Adr% = (size% * dX%) \ LineLength%
Ady% = (size% * dY%) \ LineLength%
CALL LINE (dX%, dY%)
CALL LINE (-Adr% + Ady% \ 2, -Ady% - Adr% \ 2)
CALL MOVE ( -Ady%, Adr%)
CALL LINE (+Adr% + Ady% \ 2, +Ady% - Adr% \ 2)
END SUB
'*****File Management Code*****
SUB HandleFileMenu STATIC
SELECT CASE MENU(1)
CASE 1 'File - New
    FailedFileName$ = ""
    SaveChangesInquiry ReturnCode%
    IF ReturnCode% = True% THEN StillOkay% = True% ELSE StillOkay% = False%
    IF StillOkay% THEN
        TrialFileName$ = FILES$(1,"RLD2")
        IF TrialFileName$ <> "" THEN
            StillOkay% = True%
            CalcFileName$ = TrialFileName$
        ELSE
            StillOkay% = False%
        END IF
    END IF
    IF StillOkay% THEN
        CalcFileRead ReturnCode%
        IF ReturnCode% = 0 THEN
            StillOkay% = True%
        ELSE
            StillOkay% = False%
            FailedFileName$ = CalcFileName$
        END IF
    END IF
    IF StillOkay% THEN
        ShapeDefFileRead ReturnCode%
        IF ReturnCode% = 0 THEN
            StillOkay% = True%
        ELSE
            StillOkay% = False%
            FailedFileName$ = ShapeDefFileName$
        END IF
    END IF
    IF (NOT StillOkay%) AND (FailedFileName$ <> "") THEN
        FileFailureAlertBox FailedFileName$
        FrontCalcFileName$ = "Untitled"
        CalcFileName$ = "Untitled"
        ShapeDefFileName$ = "Untitled"
        SetDefaultValues
    END IF
    IF StillOkay% THEN
        MakeNewName CalcFileName$, "Flux", FrontCalcFileName$, "Front Calcs"
        DetermineAverageFluxAlongCoreAtQtrs
    END IF
CASE 2 'File - Open
    FailedFileName$ = ""
    SaveChangesInquiry ReturnCode%
    IF ReturnCode% = True% THEN StillOkay% = True% ELSE StillOkay% = False%

```

```

IF StillOkay% THEN
    TrialFileName$ = FILES$(1,"RLD3")
    IF TrialFileName$ <> "" THEN
        StillOkay% = True% ELSE StillOkay% = False%
    END IF
IF StillOkay% THEN
    FrontCalcFileName$ = TrialFileName$
    FrontCalcFileRead ReturnCode%
    IF ReturnCode% = 0 THEN
        StillOkay% = True%
    ELSE
        StillOkay% = False%
        FailedFileName$ = FrontCalcFileName$
    END IF
END IF
IF StillOkay% THEN
    CalcFileRead ReturnCode%
    IF ReturnCode% = 0 THEN
        StillOkay% = True%
    ELSE
        StillOkay% = False%
        FailedFileName$ = CalcFileName$
    END IF
END IF
IF StillOkay% THEN
    ShapeDefFileRead ReturnCode%
    IF ReturnCode% = 0 THEN
        StillOkay% = True%
    ELSE
        StillOkay% = False%
        FailedFileName$ = ShapeDefFileName$
    END IF
END IF
IF (NOT StillOkay%) AND (FailedFileName$ <> "") THEN
    FileFailureAlertBox FailedFileName$
    FrontCalcFileName$ = "Untitled"
    CalcFileName$ = "Untitled"
    ShapeDefFileName$ = "Untitled"
    SetDefaultValues
END IF
CASE 4 'File - Save
    CALL FrontCalcFileWrite
    ChangesSaved% = True%
CASE 5 'File - Save As...
    TrialFileName$ = FILES$(0,"Save data to disk as:")
    IF TrialFileName$ <> "" THEN
        FrontCalcFileName$ = TrialFileName$
        CALL FrontCalcFileWrite
        ChangesSaved% = True%
    END IF
CASE 7 'Print Active Window...'
    PrintActiveWindow
EXIT SUB ' To avoid parsing the file name again
CASE 9 '"Quit"
    CALL SaveChangesInquiry (ReturnCode%)
    IF ReturnCode% = True% THEN
        STOP
    END IF
END SELECT
ParseFileName FrontCalcFileName$, FrontCalcShortFileName$
BEEP
END SUB
SUB FileFailureAlertBox (FailedFileName$) STATIC
    WINDOW 16,,, -1
    BEEP
    TS$ = FailedFileName$ + ": This file could not be read successfully."
    rect%(0) = 20 : rect%(1) = 20 : rect%(2) = 80 : rect%(3) = 180 : just% = 1
    TextBox TS$, rect%(0), just%

```

```

    BUTTON 1,1,"Clear Message",(50,100)-(150,120)
    WHILE DIALOG(0) <> 1 : WEND
    WINDOW CLOSE 16
END SUB
SUB ParseFileName (FullyValidatedFileName$, ShortFileName$) STATIC
    ShortFileName$ = FullyValidatedFileName$
    Done% = False%
    WHILE Done% = False%
        g% = INSTR(ShortFileName$, ":")
        IF g% > 0 THEN
            ShortFileName$ = MID$(ShortFileName$, g% + 1)
        ELSE
            Done% = True%
        END IF
    WEND
END SUB
SUB MakeNewName (AncestorFileName$, AncestorKey$,
                HeirFileName$, HeirKey$) STATIC
    f$ = UCASE$(AncestorFileName$)
    ak$ = UCASE$(AncestorKey$)
    h% = 1
    Done% = False%
    WHILE Done% = False%
        hn% = INSTR(h%, f$, ":")
        IF hn% > 0 THEN
            h% = hn% + 1
        ELSE
            Done% = True%
        END IF
    WEND
    g% = INSTR(h%, f$, ak$)
    IF g% > 0 THEN
        HeirFileName$ = LEFT$(AncestorFileName$, g% - 1) + HeirKey$
    ELSE
        HeirFileName$ = AncestorFileName$ + HeirKey$
    END IF
END SUB
SUB ShapeDefFileRead (ReturnCode%) STATIC
    ReturnCode% = -1
    'Note: Although this file is read sequentially,
    'PermCodeArray0 stored as binary to
    'reduce the size of the disk file.
    ParseFileName ShapeDefFileName$, ShortShapeDefFileName$
    ShapeDefFileName$ = ShortShapeDefFileName$
    IF SYSTEM(4) = 0 THEN
        WINDOW 16,,(300,25)-(900,400),4
        CLS
        PRINT "Please show the poor program where to find"
        PRINT ShortShapeDefFileName$
        PRINT "Perhaps some day MicroSoft will fix this bug"
        ShortShapeDefFileName$ = FILES$(1, "RLD1")
        WINDOW CLOSE 16
    END IF
    OPEN ShortShapeDefFileName$ FOR INPUT AS #5
    MagicStatement$ = "This is a data file for the perm matrix   RLD 11/26/89"
    INPUT #5, MagicStatementCandidate$
    IF MagicStatementCandidate$ <> MagicStatement$ THEN EXIT SUB
    ChangeCursor 4
    INPUT #5, CurMatCode%, CurPattern%, ActiveView%, MouseMode%
    INPUT #5, DefaultI%, DefaultJ%, DefaultK%, MCoorBias!
    FOR m% = 0 TO 15
        INPUT #5, HotSpots%(m%, 1), HotSpots%(m%, 2), HotSpots%(m, 3)
    NEXT
    FOR i% = 0 TO 7
        FOR j% = 1 TO 31
            FOR k% = 1 TO 63
                PermCodeArray%(i%, j%, k%) = CVI(INPUT$(2,#5))
            NEXT
        NEXT
    NEXT

```

```

    NEXT
NEXT
CALL FixUpPermCodeArray
CLOSE #5
BEEP
INITCURSOR
ReturnCode% = 0
END SUB
SUB CalcFileRead( ExitCode%) STATIC
    ChangeCursor 4
    OPEN CalcFileName$ FOR INPUT AS #5 LEN = 32
    MagicStatement$ =
        "This is a data file for the calculation matrix   RLD 12/7/89"
    SearchForMagic MagicStatement$, Found%
    IF Found% = False% THEN
        ExitCode% = 1
        CLOSE #5
        EXIT SUB
    END IF
    INPUT #5, ShapeDefFileName$
    INPUT #5, MeshLevel%, StepI%, StepJ%, StepK%
    INPUT #5, SOR!, Tol!, TotErr!, RelErr!, ItsAtLevel%
    INPUT #5, FluxScale!, FatArrowLimit!
    INPUT #5, SNTBBitCode%, SNWEBitCode%, WETBBitCode%, EWTBBitCode%
    MagicLine$ = "Window Information-MagicLine"
    SearchForMagic MagicLine$, Found%
    IF Found% = False% THEN
        ExitCode% = 2
        CLOSE #5
        EXIT SUB
    END IF
    FOR i% = 0 TO 16
        INPUT #5, WinOpen%(i%), WinUpdate%(i%)
    NEXT
    MagicLine$ = "Material Permeabilities-MagicLine"
    SearchForMagic MagicLine$, Found%
    IF Found% = False% THEN
        ExitCode% = 3
        CLOSE #5
        EXIT SUB
    END IF
    FOR i% = 0 TO 15
        INPUT #5, MatPerm!(i%)
    NEXT
    MagicLine$ = "Pressure Data stored in binary-MagicLine"
    SearchForMagic MagicLine$, Found%
    IF Found% = False% THEN
        ExitCode% = 4
        CLOSE #5
        EXIT SUB
    END IF
    FOR i% = 0 TO 32 STEP StepI%
        FOR j% = 0 TO 32 STEP StepJ%
            FOR k% = 0 TO 64 STEP StepK%
                P!(i%, j%, k%) = CVS(INPUT$(4,#5))
            NEXT
        NEXT
    NEXT
    CLOSE #5
    BEEP
    CALL INITCURSOR
    ExitCode% = 0
END SUB
SUB SearchForMagic (MagicLine$, Found%) STATIC
    Found% = False%
    a$ = CHR$(34)
    WHILE NOT EOF(5) AND Found% = False%
        LINE INPUT #5, TestLine$

```

```

g% = INSTR(TestLine$, a$)
IF g% > 0 THEN
    h% = INSTR(TestLine$, MagicLine$)
    IF h% > 0 THEN
        Found% = True%
    END IF
END IF
WEND
END SUB
SUB SaveChangesInquiry (ReturnCode%) STATIC
    IF ChangesSaved% = True% THEN
        ReturnCode% = True%
    ELSE
        WINDOW 16,,(400, 200)-(700, 330),-2
        TS$ = "Changes to the active file have not been saved."
        TS$ = TS$ + " Do you want to save changes before proceeding"
        rect%(0) = 20 : rect%(1) = 20 : rect%(2) = 80 : rect%(3) = 280
        just% = 0
        TextBox TS$, rect%(0), just%
        BUTTON 1, 1, "CANCEL", (20, 90) -(90, 110), 1
        BUTTON 2, 1, "NO", (135, 90) -(185, 110), 1
        BUTTON 3, 1, "YES", (230, 90) -(280, 110), 1
        rect%(0) = 85 : rect%(1) = 225 : rect%(2) = 115 : rect%(3) = 285
        CALL PENSIZE(3,3)
        CALL FRAMEROUNDRECT (VARPTR(rect%(0)),15,15)
        CALL PENNORMAL
        ExitCode% = False%
        WHILE ExitCode% = False%
            IF DIALOG(0) = 1 THEN
                Selector% = DIALOG(1)
                ExitCode% = True%
            ELSEIF INKEY$ = CHR$(13) THEN
                Selector% = 3
                ExitCode% = True%
            END IF
        WEND
        WINDOW CLOSE 16
        SELECT CASE Selector%
            CASE 1
                ReturnCode% = False%
            CASE 2
                ReturnCode% = True%
            CASE 3
                CALL FrontCalcFileWrite
                ChangesSaved% = True%
                ReturnCode% = True%
        END SELECT
    END IF
END SUB

```

APPENDIX D

COMPUTER PROGRAM FROM CHAPTER 7

D.1 Program SMOOTH

```

SUBROUTINE Smooth (Propagation Model, Propagation Model Info)
INCLUDE 'GLOBAL.CMN'
INCLUDE 'Gauss.cmn'          ! Gauss (integration) common block
EXTERNAL Propagation Model, Propagation Model Info
REAL Time
INTEGER unmInitShap, unmCurShap, unmCumShap, unmStartUpInfo
INTEGER unmFinalShap
INTEGER i
REAL PictX1, PictY1, PictX2, PictY2,
*      ScaleNew, EndTime, TimeStep, Spacing, ActualSpacing
INTEGER iIncrementsNew, iActiveSurface, iNodePlacementStrategy
INTEGER iRemeshIntv, iRecordIntv
LOGICAL lWaitForKey, lRemeshAtStartUp
CHARACTER flnmInitShap*64, flnmStartUpInfo*64
NAMELIST /StartUpInfo/
*      PictX1, PictY1, PictX2, PictY2,
*      ScaleNew, iIncrementsNew,
*      flnmInitShap, iActiveSurface, EndTime, TimeStep,
*      iRemeshIntv, iNodePlacementStrategy, Spacing, lWaitForKey,
*      lRemeshAtStartUp, iRecordIntv
REAL RWKSP(90000)
COMMON /WORKSP/ RWKSP
CALL IWKIN(90000)
unmInitShap = 12
unmCurShap = 13
unmCumShap = 11
unmStartUpInfo = 10
unmFinalShap = 9
C *** Default values for StartUpInfo NAMELIST
PictX1 = -1.
PictY1 = -1.
PictX2 = 1.
PictY2 = 1.
ScaleNew = .5
iIncrementsNew = 20
flnmInitShap = ' '
iActiveSurface = 1
EndTime = 10.
TimeStep = 1.
iRemeshIntv = 10
iNodePlacementStrategy = 10
Spacing = 0.0
lWaitForKey = .FALSE.
lRemeshAtStartUp = .TRUE.
iRecordIntv = 5
C *** Data Input:
WRITE (*, *) 'Start Up Information File:'
READ (*,*) flnmStartUpInfo
OPEN (unmStartUpInfo, FILE = flnmStartUpInfo, STATUS='OLD')
WRITE (*,*) 'Thank You'
READ (unmStartUpInfo, StartUpInfo)
CLOSE (unmStartUpInfo)
OPEN (unmInitShap, File=flnmInitShap, STATUS='OLD')

```

```

      CALL Read Input Data Set (unmInitShap, Time)
      CLOSE (unmInitShap)
C *** General Start Up activities:
      OPEN (unmCurShap, FILE='Current.shp', STATUS='UNKNOWN')
      OPEN (unmFinalShap, FILE='Final.shp', STATUS='UNKNOWN')
      IF (lRemeshAtStartUp) THEN
        CALL SmRemesh
        &      (iActiveSurface, iNodePlacementStrategy,
        &      Spacing, ActualSpacing)
      END IF
      CALL Write Input Data Set (unmCurShap, Time)
      WRITE (*,*) CHAR(07), ' '
C *** Draw all surfaces initially : (PC version:)
      WRITE(*,*) chCaseName
C *** Initialization Complete -
      CALL Smooth Main Loop
      &      (Time, EndTime, TimeStep,
      &      Propagation Model, Propagation Model Info,
      &      unmCurShap, iActiveSurface,
      &      iRemeshIntv, iNodePlacementStrategy, Spacing,
      &      unmFinalShap, iRecordIntv)
      IF (lWaitForKey) THEN
        WRITE (*,*) CHAR(07), ' '
        READ (*,*)
      END IF
      RETURN
      END
      SUBROUTINE Smooth Main Loop
      &      (Time, EndTime, TimeStep,
      &      Propagation Model, Propagation Model Info,
      &      unmCurShap, iActiveSurface,
      &      iRemeshIntv, iNodePlacementStrategy, Spacing,
      &      unmFinalShap, iRecordIntv)
      REAL Time, EndTime, TimeStep, Spacing
      INTEGER unmCurShap, iActiveSurface, iNodePlacementStrategy
      INTEGER unmFinalShap
      INTEGER iRemeshCtr, iRemeshIntv, iRecordIntv, iRecordCtr
      EXTERNAL Propagation Model, Propagation Model Info
      REAL t
      DO t = Time, EndTime, TimeStep
        CALL OneTimeStep
        &      (TimeStep, Propagation Model, Propagation Model Info)
        Time = Time + TimeStep
        iRecordCtr = iRecordCtr + 1
        IF (MOD(iRecordCtr, iRecordIntv) .eq. 0) THEN
          CALL Write Input Data Set (unmCurShap, Time)
        END IF
        iRemeshCtr = iRemeshCtr + 1
        IF (MOD(iRemeshCtr, iRemeshIntv) .eq. 0) THEN
          CALL SmRemesh
          &      (iActiveSurface, iNodePlacementStrategy,
          &      Spacing, ActualSpacing)
          IF (ActualSpacing .gt. 1.1 * Spacing) THEN
            CALL Write Input Data Set (unmFinalShap, Time)
          STOP
        END IF
      END DO
      CALL Write Input Data Set (unmFinalShap, Time)
      RETURN
      END
      SUBROUTINE OneTimeStep
      &      (TimeStep, Propagation Model, Propagation Model Info)
      REAL TimeStep
      EXTERNAL Propagation Model, Propagation Model Info
      INTEGER Inner, Outer
      PARAMETER (Inner = 1, Outer = 2)
      REAL VelMultNew, VelMultOld

```



```

C *** Predictor Step:
C   CALL Set Up Flow Problem (Outer)
C   CALL Calc Flux at Front
C   CALL Set Up Flow Problem (Inner)
C   CALL Calc Flux at Front
C   VelMultOld = 0.0
C   VelMultNew = 1.0
C   CALL Sm Propagation Controller
C   & (Propagation Model, Propagation Model Info,
C   & TimeStep, VelMultOld, VelMultNew)
C *** Corrector Step:
C   CALL Set Up Flow Problem (Outer)
C   CALL Calc Flux at Front
C   CALL Set Up Flow Problem (Inner)
C   CALL Calc Flux at Front
C   VelMultOld = 0.0
C   VelMultNew = 0.5
C   CALL Sm Propagation Controller
C   & (Propagation Model, Propagation Model Info,
C   & TimeStep, VelMultOld, VelMultNew)
C   RETURN
C   END

```

D.1.1 Common Block GLOBAL

```

C   Global.cmn
C
C   This file defines the variables in the /SmoothGlobal/ common
C   block
C
C   Major Revision:   January 5, 1990
C   Changes made to support eqn and dof ordering.
C   Eliminate use of segments. They are replaced with
C   the concept of surfaces. The connectivity information
C   in segments is now stored as nodal information as
C   iNextNode.
C
C   Array Size Parameters:
C   INTEGER MaxNodes, MaxEqns, MaxSurfs, MaxRegions
C   INTEGER MaxDOFGroups, MaxDOFs
C   PARAMETER (MaxNodes = 150)
C   PARAMETER (MaxDOFs = 4 * MaxNodes)
C   PARAMETER (MaxEqns = 300)
C   PARAMETER (MaxRegions = 40)
C   PARAMETER (MaxSurfs = 20)
C   PARAMETER (MaxDOFGroups = MaxSurfs * 2)
C
C   Encoding for Derivative State:
C   INTEGER iVarIsValue, iVarIsTang
C   PARAMETER (iVarIsValue = 1, iVarIsTang = 2)
C
C   Encoding for variable type:
C   INTEGER iVarIsPres, iVarIsFlux
C   PARAMETER (iVarIsPres = 1, iVarIsFlux = 2)
C
C   Encoding for DOF types:
C   INTEGER iHermPres, iHermFlux, iHermdPres, iHermdFlux
C   PARAMETER (iHermPres = 1, iHermFlux = 2)
C   PARAMETER (iHermdPres = 3, iHermdFlux = 4)
C
C   Encoding for Equation types:
C   INTEGER iFixedPressureEqn, iFixedFluxEqn, iHermEqn
C   PARAMETER (iFixedPressureEqn = 1, iFixedFluxEqn = 2)
C   PARAMETER (iHermEqn = 3)
C
C   Misc:

```

```

      REAL pi
      PARAMETER (pi = 3.141592654)
      CHARACTER*64 chCaseName
      INTEGER      iNumRegions, iNumDOFGroups, iNumSurfaces,
&                iNumNodes, iNumDOFs
C
C      Regional data:
      REAL Permeability(MaxRegions)
      INTEGER iPerm(MaxRegions)
C
C      DOF Group data
      INTEGER iSurfaceOrder(MaxDOFGroups)
      INTEGER iDerivStateOrder(MaxDOFGroups)
C
C      Surface data:
      INTEGER iFirstNode(MaxSurfs)
      INTEGER iRegion(MaxSurfs)
      INTEGER iEqnType(MaxSurfs)
      INTEGER iVarType(MaxSurfs)
      INTEGER iCirculation(MaxSurfs)
C
C      Nodal data:
      INTEGER iNextNode(MaxNodes)
      INTEGER iDOFIndex(MaxNodes,2,2)
      REAL dXiPlus(MaxNodes), dXiMinus(MaxNodes)
      REAL X(MaxNodes), Y(MaxNodes), dX(MaxNodes), dY(MaxNodes)
      REAL Pres(MaxNodes), dPres(MaxNodes)
      REAL Flux(MaxNodes), dFlux(MaxNodes)
C
C      Equation Ordering Information
      INTEGER iEqnTypeIndex(MaxDOFs)
      INTEGER iRegionIndex(MaxDOFs)
C
C      Degree of Freedom Data:
      INTEGER iNodeIndex(MaxDOFs)
      INTEGER iDerivIndex(MaxDOFs)
      INTEGER iVarTypeIndex(MaxDOFs)
      INTEGER iCircIndex(MaxDOFs)
      INTEGER iLastActiveDOF
C
      COMMON /SmoothGlobal/
& chCaseName,
& iNumRegions, iNumDOFGroups, iNumSurfaces,
& iNumNodes, iNumDOFs,
&
& Permeability, iPerm,
&
& iSurfaceOrder, iDerivStateOrder,
&
& iFirstNode, iRegion, iEqnType, iVarType, iCirculation,
&
& iNextNode, iDOFIndex,
& dXiPlus, dXiMinus,
& X, Y, dX, dY,
& Pres, dPres, Flux, dFlux,
&
& iEqnTypeIndex, iRegionIndex, iNodeIndex,
&
& iDerivIndex, iVarTypeIndex, iCircIndex, iLastActiveDOF

```

D.2 Subroutines for SM PROPAG

```

SUBROUTINE Sm Propagation Controller
*   (Propagation Model, Propagation Model Info,
*   TimeStep, VelMultOld, VelMultNew)
INCLUDE 'SmPropag.cmn'
EXTERNAL Propagation Model
EXTERNAL Propagation Model Info
CALL Sm SetUp For Model
*   (Propagation Model Info, VelMultOld, VelMultNew)
CALL Sm Smooth Fluxes
CALL Propagation Model (iNumKnots)
CALL Sm Propagate Front (TimeStep)
RETURN
END
SUBROUTINE Sm SetUp For Model
*   (Propagation Model Info, VelMultOld, VelMultNew)
INCLUDE 'Global.cmn'
INCLUDE 'SmPropag.cmn'
REAL Scales(10), VelMultOld, VelMultNew
C *** Through adjustment of the Velocity multipliers
C   for old and new values, a single routine
C   implements both the prediction step and the
C   the corrector step.
C   Predictor: Explicit Forward Euler
C   (VelMultOld = 0., VelMultNew = 1.)
C   Corrector: Trapezoid rule
C   (VelMultOld = -0.5., VelMultNew = 0.5)
C
StoredVelMultOld = VelMultOld
StoredVelMultNew = VelMultNew
CALL Propagation Model Info (iFrontSurface, Scales)
iNodeBasePtr = iFirstNode(iFrontSurface)
FScale = Scales(1)
PScale = Scales(2)
LScale = Scales(3)
VScale = Scales(4)
RETURN
END
SUBROUTINE Sm Smooth Fluxes
INCLUDE 'SmPropag.cmn'
INCLUDE 'Global.cmn'
INTEGER iCurNode, iKnot
C
CALL SmArcLenIntegration
*   (iFrontSurface, iNumFrontNodes, ArcLenArray)
iNumFrontNodes = iNumFrontNodes - 1
iNumKnots = iNumFrontNodes + 2
C   Get Gas Flux, Pressure, Scaling to dimensional quantities
iCurNode = iNodeBasePtr
DO iKnot = 1, iNumFrontNodes
CALL SmExtractFrontFluxPres (iKnot, iCurNode)
iCurNode = iNextNode(iCurNode)
END DO
C   Smooth Gas Flux, Pressure (Use a hermetian spline to
C   give a continous 2nd tangential derivative
CALL CSHER
*   (iNumFrontNodes, ArcLenArray, FluxNew, dFluxNew,
*   Break, FluxCSCoef)
CALL CSHER
*   (iNumFrontNodes, ArcLenArray, PresNew, dPresNew,
*   Break, PresCSCoef)
iNumFluxIntervals = iNumFrontNodes - 1
RETURN
END

```

```

SUBROUTINE SmExtractFrontFluxPres (iKnot, iCurNode)
C This routine extracts Flux and Pressure data on the
C front from the /Smooth Global/ common block and puts it
C into arrays in the /Smooth Propag/ common block in
C preparation for hermetian cubic spline smoothing.
INCLUDE 'SmPropag.cmn'
INCLUDE 'Global.cmn'
REAL dXi, dXdXi, dYdXi, Jac
IF (iCirculation(iFrontSurface) .eq. 1) THEN
    dXi = dXiPlus(iCurNode)
ELSE
    dXi = -dXiMinus(iCurNode)
ENDIF
dXdXi = dX(iCurNode) / dXi
dYdXi = dY(iCurNode) / dXi
Jac = sqrt (dXdXi * dXdXi + dYdXi * dYdXi)
PresNew(iKnot) = Pres(iCurNode)
FluxNew(iKnot) = Flux(iCurNode)
dPresNew(iKnot) = ((dPres(iCurNode) / dXi) / Jac)
dFluxNew(iKnot) = ((dFlux(iCurNode) / dXi) / Jac)
RETURN
END
SUBROUTINE Sm Propagate Front (TimeStep)
INCLUDE 'SmPropag.cmn'
INCLUDE 'Global.cmn'
INTEGER iCurNode, iKnot, i, iNumIntervals
REAL Zetah, CSDer
EXTERNAL CSDer
C Move points normal a distance given in the vector Vel,
C results are stored in vectors Xnew, Ynew vs Zeta
iCurNode = iNodeBasePtr
DO iKnot = 1, iNumFrontNodes * 2, 2
    Zetah = float(iKnot)
    CALL SmMoveSegment (iKnot, Zetah, iCurNode, TimeStep)
    iCurNode = iNextNode(iCurNode)
END DO
iKnot = iNumFrontNodes * 2 + 1
Xnew(iKnot) = Xnew(1)
Ynew(iKnot) = Ynew(1)
Zeta(iKnot) = Zeta(iKnot-1) + 1.
C Fit a Cubic Spline to these points (periodic)
CALL CSPer(2 * iNumFrontNodes + 1, Zeta, Xnew, BreakHS, XCSCoef)
CALL CSPer(2 * iNumFrontNodes + 1, Zeta, Ynew, BreakHS, YCSCoef)
C Evaluate value and derivatives and store in Global
iNumIntervals = 2 * iNumFrontNodes
iCurNode = iNodeBasePtr
DO i = 1, iNumFrontNodes * 2, 2
    Zetah = FLOAT(i)
    C Evalate and store X, Y, dX, dY
    X(iCurNode) =
    & CSDer (0, Zetah, iNumIntervals, BreakHS, XCSCoef)
    Y(iCurNode) =
    & CSDer (0, Zetah, iNumIntervals, BreakHS, YCSCoef)
    dX(iCurNode) =
    & CSDer (1, Zetah, iNumIntervals, BreakHS, XCSCoef)
    dY(iCurNode) =
    & CSDer (1, Zetah, iNumIntervals, BreakHS, YCSCoef)
    iCurNode = iNextNode(iCurNode)
END DO
RETURN
END
SUBROUTINE SmMoveSegment (iKnot, Zetah, iCurNode, TimeStep)
INCLUDE 'SmPropag.cmn'
INCLUDE 'Global.cmn'
INTEGER i, i1, i2, iCurNode, iSurf, iKnot, iKnotPtr
REAL X1, X2, Y1, Y2
REAL dXi1, dXi2, dX1, dX2, dY1, dY2
REAL Normi, Normj, NormalOrientation

```

```

REAL Xi, X1, CubVal, Y1, dX1, CubTan, dY1, Jac
EXTERNAL CubVal, CubTan
iSurf = iFrontSurface
IF (iCirculation(iSurf) .eq. 1) THEN
  NormalOrientation = 1.
ELSE
  NormalOrientation = -1.
END IF
i1 = iCurNode
i2 = iNextNode(iCurNode)
dXi1 = dXiPlus(i1)
dXi2 = dXiMinus(i2)
X1 = X(i1)
X2 = X(i2)
Y1 = Y(i1)
Y2 = Y(i2)
dX1 = dX(i1) / dXi1
dX2 = dX(i2) / dXi2
dY1 = dY(i1) / dXi1
dY2 = dY(i2) / dXi2
DO i = -1, 0
  iKnotPtr = iKnot + i + 1
  Xi = Float(i)
  X1 = CubVal(Xi, X1, dX1, X2, dX2)
  Y1 = CubVal(Xi, Y1, dY1, Y2, dY2)
  dX1 = CubTan(Xi, X1, dX1, X2, dX2)
  dY1 = CubTan(Xi, Y1, dY1, Y2, dY2)
  Jac = SQRT(dX1 * dX1 + dY1 * dY1)
  Normi = - dY1 / Jac * NormalOrientation
  Normj = + dX1 / Jac * NormalOrientation
  Xnew(iKnotPtr) = X1 + Normi * Vel(iKnotPtr) * TimeStep
  Ynew(iKnotPtr) = Y1 + Normj * Vel(iKnotPtr) * TimeStep
  Zeta(iKnotPtr) = Zeta + Xi + 1.
END DO
RETURN
END

```

```

C Service Functions For Model
C These provide a uniform method to access data, regardless
C of the underlining interpolation scheme:
C Cubic Hermetian or Cubic Spline
FUNCTION GetArcLen (iKnotPtr, iNumIncrements, iIncrement)
  INCLUDE 'SmPropag.cmn'
  REAL Frac, ArcLen, GetArcLen
  INTEGER iNode, iKnotPtr, iNumIncrements, iIncrement
  iNode = (iKnotPtr + 1) / 2
  Frac = (FLOAT (MOD (iKnotPtr + 1, 2)) +
  *   FLOAT (iIncrement) / FLOAT (iNumIncrements)) / 2.
  ArcLen = ArcLenArray(iNode) +
  *   Frac * (ArcLenArray(iNode + 1) - ArcLenArray(iNode))
  GetArcLen = ArcLen * LScale
  RETURN
END
FUNCTION GetFluxDer
  *   (iDeriv, iKnotPtr, iNumIncrements, iIncrement)
C Fluxes are stored as cubic spline coefficients
  INCLUDE 'SmPropag.cmn'
  REAL Frac, ArcLen, FluxDer
  INTEGER iNode, iKnotPtr, iNumIncrements, iIncrement
  REAL CSDer
  EXTERNAL CSDer
  iNode = (iKnotPtr + 1) / 2
  Frac = (FLOAT (MOD (iKnotPtr + 1, 2)) +
  *   FLOAT (iIncrement) / FLOAT (iNumIncrements)) / 2.
  ArcLen = ArcLenArray(iNode) +
  *   Frac * (ArcLenArray(iNode + 1) - ArcLenArray(iNode))
  IF (iDeriv .lt. 2) THEN
    FluxDer = CSDer
  *   (iDeriv, ArcLen, iNumFluxIntervals, Break, FluxCSCoef)

```

```

ELSEIF (iDeriv .eq. 2) THEN
  DeltaArcLen =
  & ((ArcLenArray(iNode+1) - ArcLenArray(iNode)))/4.
  ArcLenP = ArcLen + DeltaArcLen
  Fp = CSDer
  & (0, ArcLenP, iNumFluxIntervals, Break, FluxCSCoef)
  FO = CSDer
  & (0, ArcLen, iNumFluxIntervals, Break, FluxCSCoef)
  ArcLenM = ArcLen - DeltaArcLen
  Fm = CSDer
  & (0, ArcLenM, iNumFluxIntervals, Break, FluxCSCoef)
  FluxDer = (Fp - 2. * FO + Fm) / (DeltaArcLen * DeltaArcLen)
ELSE
  FluxDer = 0.
END IF
GetFluxDer = FluxDer * FScale / (LScale ** iDeriv)
RETURN
END
FUNCTION GetPresDer
  & (iDeriv, iKnotPtr, iNumIncrements, iIncrement)
  Pressures stored as cubic spline coefficients
  INCLUDE 'SmPropag.cmn'
  REAL Frac, ArcLen, GetPresDer, PresDer
  INTEGER iNode, iKnotPtr, iNumIncrements, iIncrement
  REAL CSDer
  EXTERNAL CSDer
  iNode = (iKnotPtr + 1) / 2
  Frac = (FLOAT (MOD (iKnotPtr + 1, 2)) +
  & FLOAT (iIncrement) / FLOAT (iNumIncrements)) / 2.
  ArcLen = ArcLenArray(iNode) +
  & Frac * (ArcLenArray(iNode + 1) - ArcLenArray(iNode))
  PresDer = CSDer
  & (iDeriv, ArcLen, iNumFluxIntervals, Break, PresCSCoef)
  GetPresDer = PresDer * PScale / (LScale ** iDeriv)
RETURN
END
SUBROUTINE SmStoreVelocity (iKnotPtr, Velocity)
  To prevent the Front propagation model from contaminating
  this common block, we provide this interface routine
  rather than including this common block in the RC model.

  By altering the values stored velocity multipliers
  a predictor-corrector propagation routine is
  implemented.
  INCLUDE 'SmPropag.cmn'
  IF ((iKnotPtr .gt. 0) .and. (iKnotPtr .le. iNumKnots)) THEN
    Vel(iKnotPtr) = StoredVelMultOld * Vel(iKnotPtr)
    & + StoredVelMultNew * Velocity * VScale
  END IF
RETURN
END

```

D.2.1 Common Block SM PROPAG

```

C   SmPropag.cmn
C
C   This file defines the variables in the /SmoothPropag/ common
C   block
C   Array Size Parameters:
C     INTEGER MFN, HSMFN
C     PARAMETER (MFN = 300)
C     PARAMETER (HSMFN = 2 * MFN)
C     INTEGER iNumFrontNodes, iNodeBasePtr, iFrontSurface, iNumKnots
C     INTEGER iNumFluxIntervals
C     REAL ArcLenArray(MFN), PresNew(MFN), dPresNew(MFN)
C     REAL FluxNew(MFN), dFluxNew(MFN)

```

```

REAL PresCSCoef(4, MFN), FluxCSCoef(4, MFN), Break(MFN)
REAL Zeta(HSMFN), Vel(HSMFN), Xnew(HSMFN), Ynew(HSMFN)
REAL XCSCoef(4, HSMFN), YCSCoef(4, HSMFN), BreakHS(HSMFN)
REAL FScale, PScale, LScale, VScale
REAL StoredVelMultOld, StoredVelMultNew
COMMON /SmoothPropag/
*   iNumFrontNodes, iNodeBasePtr, iFrontSurface, iNumKnots,
*   iNumFluxIntervals,
*
*   ArcLenArray, PresNew, dPresNew,
*   FluxNew, dFluxNew,
*   PresCSCoef, FluxCSCoef, Break,
*
*   Zeta, Vel, Xnew, Ynew,
*   XCSCoef, YCSCoef, BreakHS,
*
*   FScale, PScale, LScale, VScale,
*   StoredVelMultOld, StoredVelMultNew

```

D.3 Subroutines for SM REMESH

```

SUBROUTINE SmRemesh
*   (iSurf, iNodePlacementStrategy, Spacing, ArcLenIncr)
INTEGER iSurf, iNodePlacementStrategy
REAL Spacing
INCLUDE 'SmRemesh.cmn'
INTEGER iNumOfPoints, iDesiredNumOfNodes
REAL ArcLenArray(200), ArcLenIncr
iNewNode = 0
CALL SmArcLenIntegration
*   (iSurf, iNumOfPoints, ArcLenArray)
IF (iNodePlacementStrategy .gt. 0) THEN
    iDesiredNumOfNodes = iNodePlacementStrategy
    ArcLenIncr = ArcLenArray(iNumOfPoints)
    / FLOAT (iDesiredNumOfNodes)
ELSE
    iDesiredNumOfNodes = ArcLenArray(iNumOfPoints) / Spacing
    IF (iDesiredNumOfNodes .gt. -iNodePlacementStrategy) THEN
        iDesiredNumOfNodes = -iNodePlacementStrategy
    END IF
    ArcLenIncr = ArcLenArray(iNumOfPoints)
    / FLOAT (iDesiredNumOfNodes)
END IF
CALL SmRemesher
*   (iSurf, ArcLenArray, ArcLenIncr )
CALL SmRemeshUpdateDataFile (iSurf, iDesiredNumOfNodes)
RETURN
END
SUBROUTINE SmRemesher
*   (iSurf, ArcLenArray, ArcLenIncr )
INCLUDE 'Global.cmn'
INTEGER iSurf, iFN, iCurNode, iArrayPtr
LOGICAL lInitEntry
REAL ArcLenArray(*), ArcLenTarget
ArcLenTarget = 0.0
C Surface traversal: (loop through all points on the surface).
iFN = iFirstNode(iSurf)
iCurNode = iFN
lInitEntry = .TRUE.
iArrayPtr = 0
ArcLenArray(1) = 0.
DO WHILE ((iCurNode .ne. iFN) .or. (lInitEntry .eqv. .TRUE.))
    lInitEntry = .FALSE.
    iArrayPtr = iArrayPtr + 1
    IF (ArcLenArray(iArrayPtr + 1) .ge. ArcLenTarget) THEN
        CALL SmRemesherSub
    END IF
    iCurNode = iCurNode + 1
END DO

```

```

*      (iCurNode,
*      ArcLenArray(iArrayPtr), ArcLenArray(iArrayPtr+ 1),
*      ArcLenIncr, ArcLenTarget)
      END IF
      iCurNode = iNextNode(iCurNode)
END DO
RETURN
END
SUBROUTINE SmRemesherSub
*      (iCurNode, ArcLenBeg, ArcLenEnd,
*      ArcLenIncr, ArcLenTarget)
  INCLUDE 'Global.cmn' ! SmGlobal common block
  INTEGER iCurNode, i1, i2
  REAL ArcLenBeg, ArcLenEnd, ArcLenIncr, ArcLenTarget
  REAL ArcLenCur
  REAL X1, X2, dX1, dX2, Y1, Y2, dY1, dY2, P1, P2, dP1, dP2
  REAL F1, F2, dF1, dF2
  REAL Jac, Jac1, Jac2
  REAL Xi, dXi, dXxi, dYxi
  EXTERNAL CubVal, CubTan
  i1 = iCurNode
  i2 = iNextNode(iCurNode)
  X1 = X(i1)
  X2 = X(i2)
  Y1 = Y(i1)
  Y2 = Y(i2)
  P1 = Pres(i1)
  P2 = Pres(i2)
  F1 = Flux(i1)
  F2 = Flux(i2)
  dX1 = dX(i1) / dXiPlus(i1)
  dX2 = dX(i2) / dXiMinus(i2)
  dY1 = dY(i1) / dXiPlus(i1)
  dY2 = dY(i2) / dXiMinus(i2)
  dP1 = dPres(i1) / dXiPlus(i1)
  dP2 = dPres(i2) / dXiMinus(i2)
  dF1 = dFlux(i1) / dXiPlus(i1)
  dF2 = dFlux(i2) / dXiMinus(i2)
  Jac1 = sqrt(dX1 * dX1 + dY1 * dY1)
  Jac2 = sqrt(dX2 * dX2 + dY2 * dY2)
  dXi = Min(.02 * ArcLenIncr / (Jac1 + Jac2), .1)
  Xi = -1.
  ArcLenCur = ArcLenBeg
  DO WHILE (ArcLenEnd .gt. ArcLenTarget)
    IF (ArcLenCur .ge. ArcLenTarget) THEN
      CALL SmRemeshRecordNewNode
*      (Xi, ArcLenIncr,
*      X1, X2, Y1, Y2, P1, P2, F1, F2,
*      dX1, dX2, dY1, dY2, dP1, dP2, dF1, dF2)
      ArcLenTarget = ArcLenTarget + ArcLenIncr
    END IF
    Xi = Xi + dXi
    dXxi = CubTan(Xi, X1, dX1, X2, dX2)
    dYxi = CubTan(Xi, Y1, dY1, Y2, dY2)
    Jac = sqrt(dXi * dXxi + dYxi * dYxi)
    ArcLenCur = ArcLenCur + Jac * dXi
  END DO
  RETURN
END
SUBROUTINE SmRemeshRecordNewNode
*      (Xi, ArcLenIncr,
*      X1, X2, Y1, Y2, P1, P2, F1, F2,
*      dX1, dX2, dY1, dY2, dP1, dP2, dF1, dF2)
  INCLUDE 'SmRemesh.cmn' ! SmRemesh common block
  REAL ArcLenIncr
  REAL X1, X2, Y1, Y2, P1, P2, F1, F2
  REAL dX1, dX2, dY1, dY2, dP1, dP2, dF1, dF2
  REAL Xi, dXxi, dYxi, Jac, ScaleFactor

```



```

REAL CubVal, CubTan
EXTERNAL CubVal, CubTan
iNewNode = iNewNode + 1
dXxi = CubTan(Xi, X1, dX1, X2, dX2)
dYxi = CubTan(Xi, Y1, dY1, Y2, dY2)
Jac = SQRT(dXxi * dXxi + dYxi * dYxi)
ScaleFactor = ArcLenIncr / Jac / 2.
dXiPlusNew(iNewNode) = 1.
dXiMinusNew(iNewNode) = 1.
XNew(iNewNode) = CubVal(Xi, X1, dX1, X2, dX2)
YNew(iNewNode) = CubVal(Xi, Y1, dY1, Y2, dY2)
PresNew(iNewNode) = CubVal(Xi, P1, dP1, P2, dP2)
FluxNew(iNewNode) = CubVal(Xi, F1, dF1, F2, dF2)
dXNew(iNewNode) = dXxi * ScaleFactor
dYNew(iNewNode) = dYxi * ScaleFactor
dPresNew(iNewNode) = CubTan(Xi, P1, dP1, P2, dP2) * ScaleFactor
dFluxNew(iNewNode) = CubTan(Xi, F1, dF1, F2, dF2) * ScaleFactor
RETURN
END
SUBROUTINE SmRemeshUpdateDataFile (iNewSurf, iDesiredNumOfNodes)
INCLUDE 'Global.cmn'
INTEGER unmscratch, iNewNodePtr, iSurf, iTestSurf
I/TEGER iFirstNodeCrossRef(MaxSurfs), iNewFN(MaxSurfs)
unmscratch = 12
OPEN (unmscratch, STATUS='UNKNOWN')
C Generate a cross reference index for the other surface that
C has the same first node
DO iSurf = 1, iNumSurfaces
  DO iTestSurf = 1, iNumSurfaces
    IF (iSurf .ne. iTestSurf) THEN
      IF (iFirstNode(iSurf) .eq. iFirstNode(iTestSurf)) THEN
        iFirstNodeCrossRef(iSurf) = iTestSurf
        GOTO 10
      END IF
    END IF
  END DO
  10 CONTINUE
END DO
iNewNodePtr = 0
DO iSurf = 1, iNumSurfaces
  IF (iVarType(iSurf) .eq. iVarIsPres) THEN
    iNewFN(iSurf) = iNewNodePtr + 1
    iNewFN(iFirstNodeCrossRef(iSurf)) = iNewFN(iSurf)
    IF ((iSurf .eq. iNewSurf) .or.
      & (iFirstNodeCrossRef(iSurf) .eq. iNewSurf)) THEN
      CALL SmRemeshWriteNew
      & (iNewNodePtr, unmscratch, iDesiredNumOfNodes)
    ELSE
      CALL SmRemeshWriteOld (iSurf, iNewNodePtr, unmscratch)
    END IF
  END IF
END DO
C Now update the iFirstNode table by replacing it with the
C iNewFN table
DO iSurf = 1, iNumSurfaces
  iFirstNode(iSurf) = iNewFN(iSurf)
END DO
C Now read in the nodal values in the new, sorted order:
CALL SmRemeshReadNew (unmscratch, iNewNodePtr)
CLOSE (unmscratch)
RETURN
END
SUBROUTINE SmRemeshWriteOld (iSurf, iNewNodePtr, unmsOutput)
INCLUDE 'Global.cmn'
INCLUDE 'SmRemesh.cmn'
INTEGER unmsOutput, iFN, iNewFN, iCurNode
INTEGER iNewNodePtr, iSurf, iNewNextNode
LOGICAL lInitEntry

```

```

iFN = iFirstNode(iSurf)
iNewFN = iNewNodePtr + 1
iCurNode = iFN
iInitEntry = .TRUE.
DO WHILE
* ((iCurNode .ne. iFN) .or. (iInitEntry .eqv. .TRUE.))
  iInitEntry = .FALSE.
  iNewNodePtr = iNewNodePtr + 1 ! Ptr to record to be written
  IF (iNextNode(iCurNode) .eq. iFN) THEN
    iNewNextNode = iNewFN
  ELSE
    iNewNextNode = iNewNodePtr + 1
  END IF
  WRITE (unmOutput, '(1(i6,2x),2(F14.7,2x))')
  * iNewNextNode,
  * dXiPlus(iCurNode), dXiMinus(iCurNode)
  WRITE (unmOutput, '(5x,4(F14.7,2x))')
  * X(iCurNode), Y(iCurNode),
  * dX(iCurNode), dY(iCurNode)
  WRITE (unmOutput, '(10x,4(F14.7,2x))')
  * Pres(iCurNode), Flux(iCurNode),
  * dPres(iCurNode), dFlux(iCurNode)
  iCurNode = iNextNode(iCurNode)
END DO
RETURN
END
SUBROUTINE SmRemeshWriteNew
* (iNewNodePtr, unmOutput, iDesiredNumOfNodes)
INCLUDE 'SmRemesh.cmn'
INTEGER unmOutput, iNewNextNode, iNewCtr, iFN, iNewNodePtr
INTEGER iDesiredNumOfNodes
iFN = iNewNodePtr + 1
iNewNodePtr = iNewNodePtr + iDesiredNumOfNodes
DO iNewCtr = iFN, iNewNodePtr
  IF (iNewCtr .eq. iNewNodePtr) THEN
    iNewNextNode = iFN
  ELSE
    iNewNextNode = iNewCtr + 1
  END IF
  WRITE (unmOutput, '(1(i6,2x),2(F14.7,2x))')
  * iNewNextNode,
  * dXiPlusNew(iNewCtr), dXiMinusNew(iNewCtr)
  WRITE (unmOutput, '(5x,4(F14.7,2x))')
  * XNew(iNewCtr), YNew(iNewCtr),
  * dXNew(iNewCtr), dYNew(iNewCtr)
  WRITE (unmOutput, '(10x,4(F14.7,2x))')
  * PresNew(iNewCtr), FluxNew(iNewCtr),
  * dPresNew(iNewCtr), dFluxNew(iNewCtr)
END DO
RETURN
END
SUBROUTINE SmRemeshReadNew (unmInput, iNewNodePtr)
INCLUDE 'Global.cmn'
INTEGER unminput, iNewNodePtr
INTEGER j
REWIND (unmInput)
iNumNodes = iNewNodePtr
DO j = 1, iNumNodes
  READ (unmInput,*)
  * iNextNode(j), dXiPlus(j), dXiMinus(j)
  READ (unmInput,*)
  * X(j), Y(j), dX(j), dY(j)
  READ (unmInput,*)
  * Pres(j), Flux(j), dPres(j), dFlux(j)
END DO
RETURN
END
SUBROUTINE SmArcLenIntegration

```

```

      &      (iSurf, iNumOfPoints, ArcLenArray)
C
      INCLUDE 'Global.cmn'
      INTEGER iSurf, iFW, iCurNode, iArrayPtr
      LOGICAL lInitEntry
      REAL ArcLenArray(*)
      REAL SmArcLenforSeg
      EXTERNAL SmArcLenforSeg
C
      Surface traversal: (loop through all points on the surface).
      iFW = iFirstNode(iSurf)
      iCurNode = iFW
      lInitEntry = .TRUE.
      iArrayPtr = 1
      ArcLenArray(1) = 0.
      DO WHILE ((iCurNode .ne. iFW) .or. (lInitEntry .eqv. .TRUE.))
          lInitEntry = .FALSE.
          iArrayPtr = iArrayPtr + 1
          ArcLenArray(iArrayPtr) = ArcLenArray(iArrayPtr - 1)
      &      + SmArcLenforSeg(iCurNode)
          iCurNode = iNextNode(iCurNode)
      END DO
      iNumOfPoints = iArrayPtr
      RETURN
      END
      FUNCTION SmArcLenforSeg(iCurNode)
      INCLUDE 'Global.cmn'          ! SmGlobal common block
      INCLUDE 'Gauss.cmn'          ! Gauss (integration) common block
      INTEGER iCurNode, iNGauss, I, i1, i2
      REAL X1, X2, dX1, dX2, Y1, Y2, dY1, dY2
      REAL Jac, Om, ArcLen
      REAL Xi, dXxi, dYxi
      EXTERNAL CubVal, CubTan
      i1 = iCurNode
      i2 = iNextNode(iCurNode)
      X1 = X(i1)
      X2 = X(i2)
      Y1 = Y(i1)
      Y2 = Y(i2)
      dX1 = dX(i1) / dXiPlus(i1)
      dY1 = dY(i1) / dXiPlus(i1)
      dX2 = dX(i2) / dXiMinus(i2)
      dY2 = dY(i2) / dXiMinus(i2)
      iNGauss = 7
      ArcLen = 0.
      DO I = 1, iNGauss
          Xi = Coord(I, iNGauss)
          Om = Weight(I, iNGauss)
          dXxi = CubTan(Xi, X1, dX1, X2, dX2)
          dYxi = CubTan(Xi, Y1, dY1, Y2, dY2)
          Jac = sqrt(dXxi * dXxi + dYxi * dYxi)
          ArcLen = ArcLen + Jac * Om
      END DO
      SmArcLenforSeg = ArcLen
      RETURN
      END
      SUBROUTINE SmFluxIntegration
      &      (iSurf, SmTotalFlow)
C
      INCLUDE 'Global.cmn'
      INTEGER iSurf, iFW, iCurNode, iArrayPtr
      REAL SmTotalFlow
      LOGICAL lInitEntry
      EXTERNAL SmFlowforSeg
C
      Surface traversal: (loop through all points on the surface).
      iFW = iFirstNode(iSurf)
      iCurNode = iFW
      lInitEntry = .TRUE.
      SmTotalFlow = 0.

```

```

DO WHILE ((iCurNode .ne. iFN) .or. (iInitEntry .eqv. .TRUE.))
  iInitEntry = .FALSE.
  iArrayPtr = iArrayPtr + 1
  SmTotalFlow = SmTotalFlow + SmFlowforSeg(iCurNode)
  iCurNode = iNextNode(iCurNode)
END DO
RETURN
END
FUNCTION SmFlowforSeg(iCurNode)
  INCLUDE 'Global.cmn' ! SmGlobal common block
  INCLUDE 'Gauss.cmn' ! Gauss (integration) common block
  INTEGER iCurNode, iNGauss, I, i1, i2
  REAL X1, X2, dX1, dX2, Y1, Y2, dY1, dY2
  REAL Jac, Om
  REAL Xi, dXxi, dYxi
  REAL F1, F2, dF1, dF2, Flow, Fluxxi
  EXTERNAL CubVal, CubTan
  i1 = iCurNode
  i2 = iNextNode(iCurNode)
  X1 = X(i1)
  X2 = X(i2)
  Y1 = Y(i1)
  Y2 = Y(i2)
  F1 = Flux(i1)
  F2 = Flux(i2)
  dX1 = dX(i1) / dXiPlus(i1)
  dY1 = dY(i1) / dXiPlus(i1)
  dX2 = dX(i2) / dXiMinus(i2)
  dY2 = dY(i2) / dXiMinus(i2)
  dF1 = Flux(i1) / dXiMinus(i2)
  dF2 = Flux(i2) / dXiMinus(i2)
  iNGauss = 7
  Flow = 0.
  DO I = 1, iNGauss
    Xi = Coord(I, iNGauss)
    Om = Weight(I, iNGauss)
    dXxi = CubTan(Xi, X1, dX1, X2, dX2)
    dYxi = CubTan(Xi, Y1, dY1, Y2, dY2)
    Jac = sqrt(dXxi * dXxi + dYxi * dYxi)
    Fluxxi = CubVal(Xi, F1, dF1, F2, dF2)
    Flow = Flow + Jac * Om * Fluxxi
  END DO
  SmFlowforSeg = Flow
  RETURN
END

```

D.3.1 CommonBlock SM REMESH

```

INTEGER iMaxLenNew
PARAMETER (iMaxLenNew = 800)
REAL dXiPlusNew(iMaxLenNew), dXiMinusNew(iMaxLenNew)
REAL XNew(iMaxLenNew), dXNew(iMaxLenNew)
REAL YNew(iMaxLenNew), dYNew(iMaxLenNew)
REAL PresNew(iMaxLenNew), dPresNew(iMaxLenNew)
REAL FluxNew(iMaxLenNew), dFluxNew(iMaxLenNew)
INTEGER iNewNode
COMMON /SmReMeshCommon/
& dXiPlusNew, dXiMinusNew, XNew, dXNew,
& YNew, dYNew, PresNew, dPresNew, FluxNew, dFluxNew,
& iNewNode
C ArcLen, ArcLenTarget, ArcLenIncr,
C REAL ArcLen, ArcLenTarget, ArcLenIncr

```

D.4 Subroutines for SMIO

```

C      SUBROUTINE ReadInputDataSet (unmInput, Time)
C
C      This routine reads in the input data set
C
C      INCLUDE 'Global.cmn'
C
C      INTEGER unmInput, i, j, k
C      REAL Time
C
C      READ (unmInput, '(A)') chCaseName
C
C      READ (unmInput,*) iNumRegions
C      DO k = 1, iNumRegions
C        READ (unmInput,*) iPerm(k), Permeability(k)
C      END DO
C
C      READ (unmInput,*) iNumDOFGroups
C      DO k = 1, iNumDOFGroups
C        READ (unmInput,*) iSurfaceOrder(k), iDerivStateOrder(k)
C      END DO
C
C      READ (unmInput,*) iNumSurfaces
C      DO i = 1, iNumSurfaces
C        READ (unmInput,*) iFirstNode(i), iRegion(i), iEqnType(i),
C      &          iVarType(i), iCirculation(i)
C      END DO
C
C      READ (unmInput,*) iNumNodes
C      DO j = 1, iNumNodes
C        READ (unmInput,*)
C      &          iNextNode(j), dXiPlus(j), dXiMinus(j)
C        READ (unmInput,*)
C      &          X(j), Y(j), dX(j), dY(j)
C        READ (unmInput,*)
C      &          Pres(j), Flux(j), dPres(j), dFlux(j)
C      END DO
C
C      READ (unmInput,*) Time
C
C      RETURN
C      END
C
C      SUBROUTINE WriteInputDataSet (unmOutput, Time)
C
C      This routine write out the input data set
C
C      INCLUDE 'Global.cmn'
C
C      INTEGER unmOutput, i, j, k
C      REAL Time
C
C      WRITE (unmOutput, '(A)') chCaseName
C
C      WRITE (unmOutput,'(i6)') iNumRegions
C      DO k = 1, iNumRegions
C        WRITE (unmOutput,'(i6, 2x, F14.7)')
C      &          iPerm(k), Permeability(k)
C      END DO
C
C      WRITE (unmOutput,'(i6)') iNumDOFGroups
C      DO k = 1, iNumDOFGroups
C        WRITE (unmOutput,*) iSurfaceOrder(k), iDerivStateOrder(k)
C      END DO
C
C      WRITE (unmOutput,'(i6)') iNumSurfaces
C      DO i = 1, iNumSurfaces
C        WRITE (unmOutput,'(6(i6,2x))')
C      &          iFirstNode(i), iRegion(i), iEqnType(i),
C      &          iVarType(i), iCirculation(i)

```



```

C      STOP 'A singularity has been encountered'
C      ENDIF
C
C      CALL Store (xsol)
C
C      RETURN
C      END
C      SUBROUTINE GenerateDOFIndices
C
C      This subroutine generates the indices used to control the
C      Equation and Degree of Freedom ordering.
C
C      INCLUDE 'global.cmn'
C      LOGICAL lInitEntry
C      INTEGER iDOFPtr, i, iSurf, iDeriv
C      INTEGER iET, iVT, iEqnPtr, iRG, iFN, iCurNode, iCirc
C
C      iDOFPtr = 0
C
C      DO i = 1, iNumDOFGroups
C
C          iSurf = iSurfaceOrder(i)
C          iDeriv = iDerivStateOrder(i)
C
C          iET = iEqnType(iSurf)
C          iVT = iVarType(iSurf)
C          iRG = iRegion(iSurf)
C          iCirc = iCirculation(iSurf)
C          iFN = iFirstNode(iSurf)
C
C          iCurNode = iFN
C          lInitEntry = .TRUE.
C          DO WHILE ((iCurNode .ne. iFN) .or. (lInitEntry .eqv. .TRUE.))
C              lInitEntry = .FALSE.
C              iDOFPtr = iDOFPtr + 1
C              iEqnPtr = iDOFPtr
C
C              iDOFIndex(iCurNode, iVT, iDeriv) = iDOFPtr
C
C              Equation Ordering Indices
C              iEqnPtrIndex(iEqnPtr) = iET
C              IF (iET .eq. 3.) THEN
C                  iLastActiveDOF = iDOFPtr
C              END IF
C              iRegionIndex(iEqnPtr) = iRG
C
C              Node Index is both equation ordering and DOF ordering
C              iNodeIndex(iDOFPtr) = iCurNode
C
C              Degree of Freedom Ordering Indices
C              iDerivIndex(iDOFPtr) = iDeriv
C              iVarTypeIndex(iDOFPtr) = iVT
C              iCircIndex(iDOFPtr) = iCirc
C
C              iCurNode = iNextNode(iCurNode)
C
C          END DO
C      END DO
C
C      iNumDOFs = iDOFPtr
C
C      RETURN
C      END
C      FUNCTION PermData(iRegCod)
C      IMPLICIT REAL (a - h, o - z)
C
C      INCLUDE 'global.cmn'

```

```

C      INTEGER i
C
C      DO 10 i = 1, iNumRegions
C        IF (iRegCod .eq. iPerm(i)) THEN
C          PermData = Permeability(i)
C          RETURN
C        ENDIF
C      10 CONTINUE
C      PermData = 1.0
C      WRITE (*,*) 'Warning: Default value of permeability used'
C      RETURN
C      END
C      SUBROUTINE Store (b)
C
C      Transfers data from DOF vector to nodal tables
C
C      IMPLICIT REAL (a - h, o - z)
C      INCLUDE 'global.cmn'
C
C      INTEGER iDOFPtr, iCurNode, iVT, iDeriv, iSelectCase
C      REAL b(*)
C
C      DO iDOFPtr = 1, iLastActiveDOF
C        iCurNode = iNodeIndex(iDOFPtr)
C        iVT = iVarTypeIndex(iDOFPtr)
C        iDeriv = iDerivIndex(iDOFPtr)
C        iSelectCase = 2 * (iDeriv - 1) + iVT
C        IF (iSelectCase .eq. iHermPres) THEN
C          Pres(iCurNode) = b(iDOFPtr)
C        ELSEIF (iSelectCase .eq. iHermdPres) THEN
C          dPres(iCurNode) = b(iDOFPtr)
C        ELSEIF (iSelectCase .eq. iHermFlux) THEN
C          Flux(iCurNode) = b(iDOFPtr)
C        ELSEIF (iSelectCase .eq. iHermdFlux) THEN
C          dFlux(iCurNode) = b(iDOFPtr)
C        ELSE
C          STOP 'Failure in Store'
C        END IF
C      END DO
C
C      RETURN
C      END
C      SUBROUTINE GenBndIntEqnMatrix (A, b, iEqn)
C
C      This subroutine forms the matrix equation for the Boundary
C      Element Method. The equation ordering algorithm is
C      implemented here
C
C      INCLUDE 'global.cmn'
C
C      REAL A(MaxEqns, MaxEqns), b(MaxEqns)
C      INTEGER iEqn
C      INTEGER iEqnPtr, iMainNode, iRegCod, iEqnPair(2), iET, iDOFPtr
C      INTEGER iSelectCase
C      REAL dXiMainNode
C
C      DO iEqnPtr = 1, iLastActiveDOF
C        iMainNode = iNodeIndex(iEqnPtr)
C        IF (iDerivIndex(iEqnPtr) .eq. iVarIsValue) THEN
C          iRegCod = iRegionIndex(iEqnPtr)
C          iEqnPair(1) = iEqnPtr
C          iEqnPair(2) = iDOFIndex(iNodeIndex(iEqnPtr),
C                                iVarTypeIndex(iEqnPtr),
C                                iVarIsTang)
C          iET = iEqnTypeIndex(iEqnPtr)
C
C          iSelectCase = iET
C          IF (iSelectCase .eq. iFixedPressureEqn) THEN

```



```

WRITE (*,*) 'Data set not efficiently ordered'
WRITE (*,*) 'Fixed boundary condition surfaces should'
WRITE (*,*) 'follow active surfaces.'
C Write fixed pressure boundary condition:
iDOFPtr = iDOFIndex(iMainNode,iVarIsPres, iVarIsValue)
A(iEqnPair(1), iDOFPtr) = 1.
b(iEqnPair(1)) = Pres(iMainNode)
iDOFPtr = iDOFIndex(iMainNode,iVarIsPres, iVarIsTang)
A(iEqnPair(2), iDOFPtr) = 1.
b(iEqnPair(2)) = dPres(iMainNode)
C
ELSEIF (iSelectCase .eq. iFixedFluxEqn) THEN
WRITE (*,*) 'Data set not efficiently ordered'
WRITE (*,*) 'Fixed boundary condition surfaces should'
WRITE (*,*) 'follow active surfaces.'
C Write fixed flux boundary condition:
iDOFPtr = iDOFIndex(iMainNode,iVarIsFlux, iVarIsValue)
A(iEqnPair(1), iDOFPtr) = 1.
b(iEqnPair(1)) = Flux(iMainNode)
iDOFPtr = iDOFIndex(iMainNode,iVarIsFlux, iVarIsTang)
A(iEqnPair(2), iDOFPtr) = 1.
b(iEqnPair(2)) = dFlux(iMainNode)
C
ELSEIF (iSelectCase .eq. iHermEqn) THEN
IF (iCircIndex(iEqnPtr) .eq. 1) THEN
dXiMainNode = dXiPlus(iMainNode)
ELSE
dXiMainNode = -dXiMinus(iMainNode)
END IF
CALL HermNodeBIEs
& (iMainNode, dXiMainNode, iRegCod, iEqnPair, A, b)
C
ELSE
STOP 'Equation Type not supported in GenBndIntEqnMatrix'
C
END IF
END IF
END DO
C
iEqn = iLastActiveDOF
C
RETURN
END
SUBROUTINE HermNodeBIEs
& (iMainNode, dXiMainNode, iRegCod, iEqnPair, A, b)
C
C Generates the two boundary integral equations for
C with Laplace's equation given
C a hermetian node and an isotropic region.
C
INCLUDE 'global.cmn'
C
INTEGER iMainNode, iRegCod, iEqnPair(2)
REAL A(MaxEqns,MaxEqns), b(MaxEqns)
REAL Perm, PermData
INTEGER iSurf, iCirc, iFW, iCurNode, iDOFPtr
LOGICAL lInitEntry
EXTERNAL PermData
C
b(iEqnPair(1)) = 0.
b(iEqnPair(2)) = 0.
C
Perm = PermData(iRegCod)
DO iSurf = 1, iNumSurfaces
C Determine whether surface is on region:
IF (iRegion(iSurf) .eq. iRegCod) THEN
C
iCirc = iCirculation(iSurf)

```

```

C
C      Surface traversal: (loop through all points on the surface).
      iFN = iFirstNode(iSurf)
      iCurNode = iFN
      lInitEntry = .TRUE.
      DO WHILE
C          &      ((iCurNode .ne. iFN) .or. (lInitEntry .eqv. .TRUE.))
C
          lInitEntry = .FALSE.
          CALL HermNodeHermElem
C          &      (iMainNode, dXiMainNode, iCurNode,
C          &      iEqnPair, iCirc,
C          &      A, Perm, b)
          iCurNode = iNextNode(iCurNode)
      END DO
      END IF
      END DO
C
C      Subtract pi to coefficient for second equation
      iDOFPtr = iDOFIndex(iMainNode, iVarIsPres, iVarIsTang)
      IF (iDOFPtr .le. iLastActiveDOF) THEN
C          &      A(iEqnPair(2), iDOFPtr) = A(iEqnPair(2), iDOFPtr)
C          &      - pi / dXiMainNode
      ELSE
C          &      b(iEqnPair(2)) = b(iEqnPair(2)) -
C          &      (- pi / dXiMainNode) * dPres(iMainNode)
      END IF
C
      RETURN
      END
      SUBROUTINE HermNodeHermElem
C          &      (iMainNode, dXiMainNode, iTheNode, iEqnPair, iCirc,
C          &      A, Perm, b)
C
C      This routine generates the coefficients associated with
C      a hermetian main node, a hermetian element, an isotropic
C      region. This routine handles regular elements and also
C      those including the Main Node.
C
C      Factor (equation, pres or flux, node, value or tan deriv.)
      INCLUDE 'global.cmn'
C
      REAL A(MaxEqns,MaxEqns), b(MaxEqns), Factor(2,2,2,2)
      REAL Perm
      INTEGER iMainNode, iTheNode, iEqnPair(2)
      INTEGER iNodePair(2), iSingular, iNGauss
      REAL dXiEPair(2), Xa, X1, X2, dXa, dX1, dX2
      REAL Ya, Y1, Y2, dYa, dY1, dY2
C
      IF (iCirc .eq. 1) THEN
          iNodePair(1) = iTheNode
          iNodePair(2) = iNextNode(iTheNode)
          dXiEPair(1) = dXiPlus(iNodePair(1))
          dXiEPair(2) = dXiMinus(iNodePair(2))
      ELSE
          iNodePair(2) = iTheNode
          iNodePair(1) = iNextNode(iTheNode)
          dXiEPair(1) = -dXiMinus(iNodePair(1))
          dXiEPair(2) = -dXiPlus(iNodePair(2))
      ENDIF
C
      Xa = X(iMainNode)
      Ya = Y(iMainNode)
      dXa = dX(iMainNode) / dXiMainNode
      dYa = dY(iMainNode) / dXiMainNode
      X1 = X(iNodePair(1))
      Y1 = Y(iNodePair(1))
      dX1 = dX(iNodePair(1)) / dXiEPair(1)

```

```

dY1 = dY(iNodePair(1)) / dXiEPair(1)
X2 = X(iNodePair(2))
Y2 = Y(iNodePair(2))
dX2 = dX(iNodePair(2)) / dXiEPair(2)
dY2 = dY(iNodePair(2)) / dXiEPair(2)

C
C Here we identify the singular elements that need special
C treatment.
IF (iNodePair(1) .eq. iMainNode) THEN
  IF (dXa * dXa + dYa * dYa .gt. 1.e-15) iSingular = 1
  iNGauss = 7
ELSE IF (iNodePair(2) .eq. iMainNode) THEN
  IF (dXa * dXa + dYa * dYa .gt. 1.e-15) iSingular = -1
  iNGauss = 7
ELSE
  iSingular = 0
  iNGauss = 7
END IF

C
CALL HermNodeHermElemInteg
* (Xa, X1, X2, Ya, Y1, Y2, dXa, dX1, dX2, dYa, dY1, dY2,
* iSingular, iNGauss, Factor)

C
CALL HermNodeHermElemTranCoef
* (iEqnPair, iNodePair, dXiEPair,
* iMainNode, Factor,
* Perm, A, iCirc, b)

C
RETURN
END
SUBROUTINE HermNodeHermElemTranCoef
* (iEqnPair, iNodePair, dXiEPair,
* iMainNode, Factor,
* Perm, A, iCirc, b)

C
C Transfers the Partial Coefficients into the Coefficient
C Matrix
C
INCLUDE 'global.cmn'

C
REAL A(MaxEqns,MaxEqns), b(MaxEqns), Factor(2,2,2,2), Perm
REAL dXiEPair(2), rSign
INTEGER iEqnPair(2), iNodePair(2), iMainNode
INTEGER J, K, L, M, iDOFPtr
DO J = 1, 2
C
C DO K = 1, 2
C
C DO L = 1, 2
C
C WRITE (*,'(2(IX,4(i4),1x,F10.4))')
C
C * (J,K,L,M,Factor(J,K,L,M), M = 1,2)
C
C END DO
C
C END DO
C
C END DO
C
Factor (equation, pres or flux, node, value or tan. deriv.)
C
DO J = 1, 2
  DO L = 1, 2
    iDOFPtr = iDOFIndex(iNodePair(L),iVarIsPres, iVarIsValue)
    IF (iDOFPtr .le. iLastActiveDOF) THEN
      A(iEqnPair(J), iDOFPtr) =
C
C * A(iEqnPair(J), iDOFPtr)
C
C + Factor(J, iVarIsPres, L, iVarIsValue)
      ELSE
        b(iEqnPair(J)) = b(iEqnPair(J)) -
C
C * Pres(iNodePair(L)) *
C
C * Factor(J, iVarIsPres, L, iVarIsValue)
      END IF
    END IF
  END DO
END DO

```

```

iDOFPtr = iDOFIndex(iMainNode,iVarIsPres, iVarIsValue)
IF (iDOFPtr .le. iLastActiveDOF) THEN
  A(iEqnPair(J), iDOFPtr) =
  A(iEqnPair(J), iDOFPtr)
  *
  - Factor(J, iVarIsPres, L, iVarIsValue)
ELSE
  b(iEqnPair(J)) = b(iEqnPair(J)) -
  Pres(iMainNode) *
  ( - Factor(J, iVarIsPres, L, iVarIsValue) )
END IF

```

C

```

iDOFPtr = iDOFIndex(iNodePair(L),iVarIsPres, iVarIsTang)
IF (iDOFPtr .le. iLastActiveDOF) THEN
  A(iEqnPair(J), iDOFPtr) =
  A(iEqnPair(J), iDOFPtr)
  + Factor(J, iVarIsPres, L, iVarIsTang)
  / dXiEPair(L)
ELSE
  b(iEqnPair(J)) = b(iEqnPair(J)) -
  dPres(iNodePair(L)) *
  Factor(J, iVarIsPres, L, iVarIsTang)
  / dXiEPair(L)
END IF

```

C

```

iDOFPtr = iDOFIndex(iNodePair(L),iVarIsFlux, iVarIsValue)
rSign = SIGN(1.,FLOAT(iCirc))
IF (iDOFPtr .le. iLastActiveDOF) THEN
  A(iEqnPair(J), iDOFPtr) =
  A(iEqnPair(J), iDOFPtr)
  + rSign * Factor(J, iVarIsFlux, L, iVarIsValue)
  / (-Perm)
ELSE
  b(iEqnPair(J)) = b(iEqnPair(J)) -
  Flux(iNodePair(L)) *
  rSign * Factor(J, iVarIsFlux, L, iVarIsValue)
  / (-Perm)
END IF

```

C

```

iDOFPtr = iDOFIndex(iNodePair(L),iVarIsFlux, iVarIsTang)
IF (iDOFPtr .le. iLastActiveDOF) THEN
  A(iEqnPair(J), iDOFPtr) =
  A(iEqnPair(J), iDOFPtr)
  + rSign * Factor(J, iVarIsFlux, L, iVarIsTang)
  / (-Perm) / dXiEPair(L)
ELSE
  b(iEqnPair(J)) = b(iEqnPair(J)) -
  dFlux(iNodePair(L)) *
  rSign * Factor(J, iVarIsFlux, L, iVarIsTang)
  / (-Perm) / dXiEPair(L)
END IF
END DO
END DO

```

C

```

RETURN
END
SUBROUTINE HermNodeHermElemInteg
  (Xa, X1, X2, Ya, Y1, Y2, dXa, dX1, dX2, dYa, dY1, dY2,
  iSingular, iNGauss, Factor)

```

C

C

C

C

C

C

Performs evenly-weighted Gauss integration of kernels for
 Hermetian Elements for hermetian Main Node. Singular integrals
 are handled by singularity subtraction, rather than by using
 specially weighted Gauss integration.

```

INTEGER I, J, K, L, M, iSingNode
REAL Factor(2, 2, 2, 2), Shape(2, 2), Kern (2, 2), Om, Xi
REAL Ixi, Yxi, CubVal, CubTan, dXxi, dYxi, Jac0
REAL SingularIntegral

```

```

REAL log, float, sqrt
INTRINSIC log, float, sqrt

C
C Structure of arrays:
C Factor (equation, pres or flux, plus or minus, value or tan. deriv.)
C Shape (plus or minus, value or tan. deriv.)
C Kern (equation, pres or flux)
C
C Encoding for Derivative State:
  INTEGER iVarIsValue, iVarIsTang
  PARAMETER (iVarIsValue = 1, iVarIsTang = 2)
C
C Encoding for variable type:
  INTEGER iVarIsPres, iVarIsFlux
  PARAMETER (iVarIsPres = 1, iVarIsFlux = 2)
C
C See BLOCK DATA GaussInitialization for definition:
C INCLUDE 'Gauss.cmn'
C
C Zero out Factors:
DO 10 J = 1, 2
  DO 20 K = 1, 2
    DO 30 L = 1, 2
      DO 40 M = 1, 2
        Factor(J, K, L, M) = 0.0
40      CONTINUE
30      CONTINUE
20      CONTINUE
10     CONTINUE
  IF (iSingular .eq. -1) THEN
    Jac0 = sqrt(2Xa * dXa + dYa * dYa)
    iSingNode = 2
  ELSE IF (iSingular .eq. +1) THEN
    Jac0 = sqrt(dXa * dXa + dYa * dYa)
    iSingNode = 1
  END IF
C Gaussian Integration:
DO 50 I = 1, iNGauss
  Xi = Coord(I, iNGauss)
  Om = Weight(I, iNGauss)
  Shape(1,1) = 0.25 * (1. - Xi) * (1. - Xi) * (2. + Xi)
  Shape(1,2) = 0.25 * (1. - Xi) * (1. - Xi) * (1. + Xi)
  Shape(2,1) = 0.25 * (1. + Xi) * (1. + Xi) * (2. - Xi)
  Shape(2,2) = -0.25 * (1. + Xi) * (1. + Xi) * (1. - Xi)
  Xxi = CubVal(Xi, X1, dX1, X2, dX2)
  Yxi = CubVal(Xi, Y1, dY1, Y2, dY2)
  dXxi = CubTan(Xi, X1, dX1, X2, dX2)
  dYxi = CubTan(Xi, Y1, dY1, Y2, dY2)
  CALL HermKernels (Xa, Ya, dXa, dYa, Xxi, Yxi, dXxi, dYxi,
    * Kern)
  DO 60 J = 1, 2      ! Equation
    DO 70 K = 1, 2      ! Pressure / Flux
      DO 80 L = 1, 2      ! Mode (Minus or Plus)
        DO 90 M = 1, 2      ! Value or Tang Deriv
          Factor(J, K, L, M) = Factor(J, K, L, M)
    * + Kern(J, K) * Shape (L, M) * Om
90      CONTINUE
80      CONTINUE
70      CONTINUE
60      CONTINUE
  IF (iSingular .ne. 0) THEN
    Factor (1, iVarIsFlux, iSingNode, iVarIsValue) =
    * Factor(1, iVarIsFlux, iSingNode, iVarIsValue)
    * + log((1. - float(iSingular) * Xi) * Jac0 * Jac0 * Om
    * Factor (2, iVarIsFlux, iSingNode, iVarIsValue) =
    * Factor(2, iVarIsFlux, iSingNode, iVarIsValue)
    * + Om / (-float(iSingular) + Xi)
  END IF

```

```

50 CONTINUE
  IF (iSingular .ne. 0) THEN
    SingularIntegral = 2. * Jac0 * (1 - log(2.) - log(Jac0))
    Factor (1, iVarIsFlux, iSingNode, iVarIsValue) =
      * Factor(1, iVarIsFlux, iSingNode, iVarIsValue)
      * + SingularIntegral
  END IF

C
  RETURN
END
SUBROUTINE HermKernels (Xa, Ya, dXa, dYa, Xxi, Yxi,
*                      dXxi, dYxi, Kern)

C
C   This subroutine calculates the unchanging parts of
C   the kernels of the integrals associated with hermetian
C   nodes.
C

  REAL Normi, Normj, Kern(2,2), Jac, R, dRdt
  REAL GradRi, GradRj, dGradRi, dGradRj, dGradRRi, dGradRRj
  REAL sqrt, log
  INTRINSIC sqrt, log
  Encoding for variable type:
    INTEGER iVarIsPres, iVarIsFlux
    PARAMETER (iVarIsPres = 1, iVarIsFlux = 2)

C
C
  R = sqrt((Xxi - Xa) * (Xxi - Xa) + (Yxi - Ya) * (Yxi - Ya))
  Jac = sqrt(dXxi * dXxi + dYxi * dYxi)
  GradRi = (Xxi - Xa) / R
  GradRj = (Yxi - Ya) / R
  dRdt = ((Xa - Xxi) * dXa + (Ya - Yxi) * dYa) / R
  dGradRi = - dRdt * (Xxi - Xa) / (R * R) - dXa / R
  dGradRj = - dRdt * (Yxi - Ya) / (R * R) - dYa / R
  dGradRRi = dGradRi / R - dRdt * GradRi / (R * R)
  dGradRRj = dGradRj / R - dRdt * GradRj / (R * R)

C
  Normi = - dYxi / Jac
  Normj = + dXxi / Jac

C
  Kern(1,iVarIsPres) =
    * (GradRi * Normi + GradRj * Normj) * Jac / R
  Kern(1,iVarIsFlux) =
    * - log(R) * Jac
  Kern(2,iVarIsPres) =
    * (dGradRRi * Normi + dGradRRj * Normj) * Jac
  Kern(2,iVarIsFlux) =
    * - dRdt * Jac / R

C
  RETURN
END
FUNCTION CubVal ( Xi, Gminus, dGminus, Gplus, dGplus)

C
C   Returns the cubic hermetian interpolant given nodal values
C
  REAL rMplus, rMminus, rNplus, rNminus

  rMplus = 0.25 * (1. + Xi) * (1. + Xi) * (2. - Xi)
  rNplus = - 0.25 * (1. + Xi) * (1. + Xi) * (1. - Xi)
  rMminus = 0.25 * (1. - Xi) * (1. - Xi) * (2. + Xi)
  rNminus = 0.25 * (1. - Xi) * (1. - Xi) * (1. + Xi)
  CubVal = (rMplus * Gplus) + (rMminus * Gminus)
  * + (rNplus * dGplus) + (rNminus * dGminus)

C
  RETURN
END
FUNCTION CubTan ( Xi, Gminus, dGminus, Gplus, dGplus)

C
C   Returns the first tangential derivative of the cubic hermetian

```

```

C      interpolant given nodal values.
C
C      REAL dMplus, dMminus, dNplus, dNminus
C
C      dMplus = 0.75 * (1. - (Xi * Xi))
C      dMminus = - dMplus
C      dNplus = 0.25 * (- 1. + (2. * Xi) + (3.* Xi * Xi))
C      dNminus = - 0.25 * ( 1. + (2. * Xi) - (3.* Xi * Xi))
C      CubTan = (dMplus * Gplus) + (dMminus * Gminus)
C      *      + (dNplus * dGplus) + (dNminus * dGminus)
C
C      RETURN
C      END
C      FUNCTION CubTan2 (Xi, Gminus, dGminus, Gplus, dGplus)
C
C      Returns the second tangential derivative of the cubic hermetian
C      interpolant given nodal values.
C
C      REAL d2Mplus, d2Mminus, d2Nplus, d2Nminus
C
C      d2Mplus = - 1.5 * Xi
C      d2Mminus = 1.5 * Xi
C      d2Nplus = 0.5 + 1.5 * Xi
C      d2Nminus = - 0.5 + 1.5 * Xi
C      CubTan2 = (d2Mplus * Gplus) + (d2Mminus * Gminus)
C      *      + (d2Nplus * dGplus) + (d2Nminus * dGminus)
C
C      RETURN
C      END
C      FUNCTION CubTan3 (Xi, Gminus, dGminus, Gplus, dGplus)
C
C      Returns the third tangential derivative of the cubic hermetian
C      interpolant given nodal values.
C
C      REAL d3Mplus, d3Mminus, d3Nplus, d3Nminus
C
C      d3Mplus = - 1.5
C      d3Mminus = 1.5
C      d3Nplus = 1.5
C      d3Nminus = 1.5
C      Xi = Xi
C      CubTan3 = (d3Mplus * Gplus) + (d3Mminus * Gminus)
C      *      + (d3Nplus * dGplus) + (d3Nminus * dGminus)
C
C      RETURN
C      END
C      BLOCK DATA GaussInitialization
C
C      Initializes abscissas and weights for evenly and log weighted
C      Gauss integration.
C
C      INCLUDE 'Gauss.cmn'
C      Real Coord(16,16), Weight(16,16)
C      COMMON /Gauss/ Coord, Weight
C
C      DATA Coord(1, 1) / 0.0 /
C      DATA Weight(1, 1) / 2.0 /
C
C      DATA Coord(1, 2) / 0.5773502692 /
C      DATA Coord(2, 2) / -0.5773502692 /
C      DATA Weight(1, 2) / 1.0 /
C      DATA Weight(2, 2) / 1.0 /
C
C      DATA Coord(1, 3) / 0.0 /
C      DATA Coord(2, 3) / 0.7745966692 /
C      DATA Coord(3, 3) / 0.7745966692 /
C      DATA Weight(1, 3) / 0.8888888888 /
C      DATA Weight(2, 3) / 0.5555555556 /

```

DATA Weight(3, 3) / 0.5555555556 /

C

DATA Coord(1, 4) / 0.86113631 /
 DATA Coord(2, 4) / -0.86113631 /
 DATA Coord(3, 4) / 0.33998104 /
 DATA Coord(4, 4) / -0.33998104 /
 DATA Weight(1, 4) / 0.34785485 /
 DATA Weight(2, 4) / 0.34785485 /
 DATA Weight(3, 4) / 0.65214515 /
 DATA Weight(4, 4) / 0.65214515 /

C

DATA Coord(1, 5) / 0.000000000000000 /
 DATA Coord(2, 5) / -0.538469310105683 /
 DATA Coord(3, 5) / 0.538469310105683 /
 DATA Coord(4, 5) / -0.906179845938664 /
 DATA Coord(5, 5) / 0.906179845938664 /
 DATA Weight(1, 5) / 0.568888888888889 /
 DATA Weight(2, 5) / 0.478628670499366 /
 DATA Weight(3, 5) / 0.478628670499366 /
 DATA Weight(4, 5) / 0.236926885056189 /
 DATA Weight(5, 5) / 0.236926885056189 /

C

DATA Coord(1, 6) / 0.238619186083197 /
 DATA Coord(2, 6) / -0.238619186083197 /
 DATA Coord(3, 6) / 0.661209386466265 /
 DATA Coord(4, 6) / -0.661209386466265 /
 DATA Coord(5, 6) / 0.932469514203152 /
 DATA Coord(6, 6) / -0.932469514203152 /
 DATA Weight(1, 6) / 0.467913934572691 /
 DATA Weight(2, 6) / 0.467913934572691 /
 DATA Weight(3, 6) / 0.360761573048139 /
 DATA Weight(4, 6) / 0.360761573048139 /
 DATA Weight(5, 6) / 0.171324492379170 /
 DATA Weight(6, 6) / 0.171324492379170 /

C

DATA Coord(1, 7) / 0.000000000000000 /
 DATA Coord(2, 7) / -0.405845151377397 /
 DATA Coord(3, 7) / 0.405845151377397 /
 DATA Coord(4, 7) / -0.741531185599394 /
 DATA Coord(5, 7) / 0.741531185599394 /
 DATA Coord(6, 7) / -0.949107912343759 /
 DATA Coord(7, 7) / 0.949107912343759 /
 DATA Weight(1, 7) / 0.417959183673469 /
 DATA Weight(2, 7) / 0.381830050505119 /
 DATA Weight(3, 7) / 0.381830050505119 /
 DATA Weight(4, 7) / 0.279705391489277 /
 DATA Weight(5, 7) / 0.279705391489277 /
 DATA Weight(6, 7) / 0.129484966168870 /
 DATA Weight(7, 7) / 0.129484966168870 /

C

DATA Coord(1, 9) / 0.000000000000000 /
 DATA Coord(2, 9) / -0.324253423403809 /
 DATA Coord(3, 9) / 0.324253423403809 /
 DATA Coord(4, 9) / -0.613371432700590 /
 DATA Coord(5, 9) / 0.613371432700590 /
 DATA Coord(6, 9) / -0.836031107326636 /
 DATA Coord(7, 9) / 0.836031107326636 /
 DATA Coord(8, 9) / -0.968160239507626 /
 DATA Coord(9, 9) / 0.968160239507626 /
 DATA Weight(1, 9) / 0.330239355001260 /
 DATA Weight(2, 9) / 0.312347077040003 /
 DATA Weight(3, 9) / 0.312347077040003 /
 DATA Weight(4, 9) / 0.260610696402935 /
 DATA Weight(5, 9) / 0.260610696402935 /
 DATA Weight(6, 9) / 0.180648160694857 /
 DATA Weight(7, 9) / 0.180648160694857 /
 DATA Weight(8, 9) / 0.081274388361574 /


```

DATA Weight(9, 9) / 0.081274388361574 /
DATA Coord(1, 16) / - 0.095012509837637441085 /
DATA Coord(2, 16) / 0.095012509837637441085 /
DATA Coord(3, 16) / - 0.281803550779258913230 /
DATA Coord(4, 16) / 0.281803550779258913230 /
DATA Coord(5, 16) / - 0.458016777657227386342 /
DATA Coord(6, 16) / 0.458016777657227386342 /
DATA Coord(7, 16) / - 0.617876244402643748447 /
DATA Coord(8, 16) / 0.617876244402643748447 /
DATA Coord(9, 16) / - 0.755404408355003033895 /
DATA Coord(10, 16) / 0.755404408355003033895 /
DATA Coord(11, 16) / - 0.865631202387831743880 /
DATA Coord(12, 16) / 0.865631202387831743880 /
DATA Coord(13, 16) / - 0.944575023073232576078 /
DATA Coord(14, 16) / 0.944575023073232576078 /
DATA Coord(15, 16) / - 0.989400934991649932596 /
DATA Coord(16, 16) / 0.989400934991649932596 /
DATA Weight(1, 16) / 0.189450610455068496285 /
DATA Weight(2, 16) / 0.189450610455068496285 /
DATA Weight(3, 16) / 0.182603415044923588867 /
DATA Weight(4, 16) / 0.182603415044923588867 /
DATA Weight(5, 16) / 0.169156519395002538189 /
DATA Weight(6, 16) / 0.169156519395002538189 /
DATA Weight(7, 16) / 0.149595988816576732081 /
DATA Weight(8, 16) / 0.149595988816576732081 /
DATA Weight(9, 16) / 0.124628971255533872052 /
DATA Weight(10, 16) / 0.124628971255533872052 /
DATA Weight(11, 16) / 0.095158511682492784810 /
DATA Weight(12, 16) / 0.095158511682492784810 /
DATA Weight(13, 16) / 0.062253523938647892863 /
DATA Weight(14, 16) / 0.062253523938647892863 /
DATA Weight(15, 16) / 0.027152459411754094852 /
DATA Weight(16, 16) / 0.027152459411754094852 /
END

```

D.5.1 Common Block GAUSS

```

C      INCLUDE 'Gauss.cmn'
      Real Coord(16,16), Weight(16,16)
      COMMON /Gauss/ Coord, Weight

```

D.6 Subroutines for SMOOTH GRAPH

```

PROGRAM Smooth Graph
INCLUDE 'GLOBAL.CMN'
INCLUDE 'Gauss.cmn'          ! Gauss (integration) common block
REAL Time
INTEGER unmShapeData, unmStartUpInfo
REAL PictX1, PictY1, PictX2, PictY2,
&      ScaleNew, EndTime, TimeStep, Spacing
INTEGER iIncrementsNew, iActiveSurface, iNodePlacementStrategy
INTEGER iRemeshIntv, iRecordIntv
LOGICAL lWaitForKey, lRemeshAtStartup
CHARACTER flnmShapeData*64, flnmStartUpInfo*64, flnmInitShap*64
NAMELIST /StartUpInfo/
&      PictX1, PictY1, PictX2, PictY2,
&      ScaleNew, iIncrementsNew,
&      flnmInitShap, iActiveSurface, EndTime, TimeStep,
&      iRemeshIntv, iNodePlacementStrategy, Spacing, lWaitForKey,
&      lRemeshAtStartup, iRecordIntv
unmShapeDataFile = 23
unmStartUpInfo = 10
C *** Default values for StartUpInfo NAMELIST
PictX1 = -1.

```

```

PictY1 = -1.
PictX2 = 1.
PictY2 = 1.
ScaleNew = .5
iIncrementsNew = 20
flnmInitShap = ' '
iActiveSurface = 1
EndTime = 10.
TimeStep = 1.
iRemeshIntv = 10
iNodePlacementStrategy = 10
Spacing = 0.0
lWaitForKey = .FALSE.
lRemeshAtStartUp = .TRUE.
C *** Data Input:
WRITE (*, *) 'Start Up Information File:'
READ (*, *) flnmStartUpInfo
OPEN (unmStartUpInfo, FILE = flnmStartUpInfo, STATUS='OLD')
WRITE (*, *) 'Thank You'
READ (unmStartUpInfo, StartUpInfo)
CLOSE (unmStartUpInfo)
WRITE (*, *) 'Shape File Data:'
READ (*, *) flnmShapeData
OPEN (unmShapeData, File=flnmShapeData, STATUS='OLD')
CALL Read Input Data Set (unmShapeData, Time)
CALL SetFluxScale (ScaleNew, iIncrementsNew)
C *** Ready for pictures:
CALL StartGraphics (PictX1, PictY1, PictX2, PictY2)
WRITE (*, *) CHAR(07), ' '
C *** Draw all surfaces initially
WRITE(*, *) chCaseName
DO i = 1, iNumSurfaces
  CALL DrawSurface (i)
END DO
C *** Initialization Complete -
C Draw active surface for all time steps
DO WHILE (Time .LT. 1000.)
  CALL Read Input Data Set (unmShapeData, Time)
  IF (Time .lt. 1000.) THEN
    CALL DrawSurface (iActiveSurface)
  END IF
END DO
IF (lWaitForKey) THEN
  WRITE (*, *) CHAR(07), ' '
  READ (*, *)
END IF
CALL EndGraphics
END
C+++++
SUBROUTINE StartGraphics (Xmin, Ymin, Xmax, Ymax)
C
REAL Xmin, Ymin, Xmax, Ymax
CALL jbegin
CALL jdinit (1)
CALL jdevon (1)
c call jwindo (umin,umax,vmin,vmax) defines world coordinate system (clipping
CALL jwindo (Xmin, Xmax, Ymin, Ymax)
CALL jopen
RETURN
END
SUBROUTINE GenericMoveToRoutine (X, Y)
REAL X, Y
CALL jMove (X, Y)
RETURN
END
SUBROUTINE GenericLineToRoutine (X, Y)
REAL X, Y
CALL jDraw(X, Y)

```

```

        RETURN
      END
      SUBROUTINE EndGraphics ()
        CALL jclose
        CALL jdevar(1)
        CALL jdend(1)
        CALL jend
        RETURN
      END
C+++++
C      SUBROUTINE ReadInputDataSet (unmInput, Time)
C
C      This routine reads in the input data set
C
C      INCLUDE 'Global.cmn'
C
C      INTEGER unmInput, i, j, k, iErrOccurred
C      REAL Time
C
C      READ (unmInput, '(A)', IOSTAT = iErrOccurred, ERR=100) chCaseName
C
C      READ (unmInput,*) iNumRegions
C      DO k = 1, iNumRegions
C        READ (unmInput,*) iPerm(k), Permeability(k)
C      END DO
C
C      READ (unmInput,*) iNumDOFGroups
C      DO k = 1, iNumDOFGroups
C        READ (unmInput,*) iSurfaceOrder(k), iDerivStateOrder(k)
C      END DO
C
C      READ (unmInput,*) iNumSurfaces
C      DO i = 1, iNumSurfaces
C        READ (unmInput,*) iFirstNode(i), iRegion(i), iEqnType(i),
C        * iVarType(i), iCirculation(i)
C      END DO
C
C      READ (unmInput,*) iNumNodes
C      DO j = 1, iNumNodes
C        READ (unmInput,*)
C        * iNextNode(j), dXiPlus(j), dXiMinus(j)
C        READ (unmInput,*)
C        * X(j), Y(j), dX(j), dY(j)
C        READ (unmInput,*)
C        * Pres(j), Flux(j), dPres(j), dFlux(j)
C      END DO
C
C      READ (unmInput,*) Time
C
C      100 CONTINUE
C      IF (iErrOccurred .ne. 0) THEN
C        Time = 99999.
C      END IF
C      RETURN
C      END
C      SUBROUTINE WriteInputDataSet (unmOutput, Time)
C
C      This routine write out the input data set
C
C      INCLUDE 'Global.cmn'
C
C      INTEGER unmOutput, i, j, k
C      REAL Time
C
C      WRITE (unmOutput, '(A)') chCaseName
C
C      WRITE (unmOutput, '(i6)') iNumRegions
C      DO k = 1, iNumRegions

```

```

      WRITE (unmOutput,'(i6, 2x, F14.7)')
      *      iPerm(k), Permeability(k)
      END DO
C
      WRITE (unmOutput,'(i6)') iNumDOFGroups
      DO k = 1, iNumDOFGroups
        WRITE (unmOutput,*) iSurfaceOrder(k), iDerivStateOrder(k)
      END DO
C
      WRITE (unmOutput,'(i6)') iNumSurfaces
      DO i = 1, iNumSurfaces
        WRITE (unmOutput,'(6(i6,2x))')
        *      iFirstNode(i), iRegion(i), iEqnType(i),
        *      iVarType(i), iCirculation(i)
      END DO
C
      WRITE (unmOutput,'(i6)') iNumNodes
      DO j = 1, iNumNodes
        WRITE (unmOutput,'(i(i6,2x),2(F14.7,2x))')
        *      iNextNode(j),
        *      dXiPlus(j), dXiMinus(j)
        WRITE (unmOutput,'(5x,4(F14.7,2x))')
        *      X(j), Y(j), dX(j), dY(j)
        WRITE (unmOutput,'(10x,4(F14.7,2x))')
        *      Pres(j), Flux(j), dPres(j), dFlux(j)
      END DO
C
      WRITE (unmOutput,'(f12.4)') Time
C
      RETURN
      END
C+++++
C
C      SmVisual.for - Almost Device independent graphics support
C                      for the smooth routines
C
C      Calls:
C
C      CALL DrawSurface (iSurf)
C      CALL DrawFlux (iSurf)
C      CALL DrawNormal (iSurf)
C      CALL SetFluxScale (ScaleNew, iIncrementsNew)
C      CALL SurfaceTraversal (iSurf, AppliedSubroutine)
C      SUBROUTINE DrawSurface (iSurf)
C      INTEGER iSurf
C      EXTERNAL DrawSurfaceTool
C      CALL SurfaceTraversal (iSurf, DrawSurfaceTool)
C      RETURN
C      END
C      SUBROUTINE DrawSurfaceTool (iCurNode, iSurf)
C      INCLUDE 'Global.cmn'
C      INTEGER iCurNode, iNodePair(2), iCirc
C      REAL dXiEPair(2), X1, X2, dX1, dX2, Y1, Y2, dY1, dY2
C      REAL Xi, Xxi, Yxi
C      EXTERNAL CubVal
C      iSurf = iSurf
C      iCirc = 1
C      IF ( iCirc .eq. 1) THEN
C        iNodePair(1) = iCurNode
C        iNodePair(2) = iNextNode(iCurNode)
C        dXiEPair(1) = dXiPlus(iNodePair(1))
C        dXiEPair(2) = dXiMinus(iNodePair(2))
C      ELSE
C        iNodePair(2) = iCurNode
C        iNodePair(1) = iNextNode(iCurNode)
C        dXiEPair(1) = dXiMinus(iNodePair(1))
C        dXiEPair(2) = dXiPlus(iNodePair(2))

```

```

ENDIF
X1 = X(iNodePair(1))
Y1 = Y(iNodePair(1))
dX1 = dX(iNodePair(1)) / dXiEPair(1)
dY1 = dY(iNodePair(1)) / dXiEPair(1)
X2 = X(iNodePair(2))
Y2 = Y(iNodePair(2))
dX2 = dX(iNodePair(2)) / dXiEPair(2)
dY2 = dY(iNodePair(2)) / dXiEPair(2)
Xxi = CubVal(-1., X1, dX1, X2, dX2)
Yxi = CubVal(-1., Y1, dY1, Y2, dY2)
CALL GenericMoveToRoutine (Xxi, Yxi) ! Different for Vax /MS-DOS
DO Xi = -1., 1., .09999
  Xxi = CubVal(Xi, X1, dX1, X2, dX2)
  Yxi = CubVal(Xi, Y1, dY1, Y2, dY2)
  CALL GenericLineToRoutine (Xxi, Yxi) ! Different for Vax / MS-DOS
END DO
RETURN
END
BLOCK DATA
INTEGER iIncrements
REAL Scale
COMMON /SmVisualBackDoor/ Scale, iIncrements
DATA Scale / 1./, iIncrements / 5 /
END
SUBROUTINE DrawFlux (iSurf)
INTEGER iSurf
EXTERNAL DrawFluxTool
CALL SurfaceTraversal (iSurf, DrawFluxTool)
RETURN
END
SUBROUTINE SetFluxScale (ScaleNew, iIncrementsNew)
REAL ScaleNew
INTEGER iIncrementsNew
INTEGER iIncrements
REAL Scale
COMMON /SmVisualBackDoor/ Scale, iIncrements
Scale = ScaleNew
iIncrements = iIncrementsNew
RETURN
END
SUBROUTINE DrawFluxTool (iCurNode, iSurf)
INCLUDE 'Global.cmn'
INTEGER i, iNP(2), iCurNode, iSurf
REAL Normi, Normj, dXiE(2), Xh(2), Yh(2), dXh(2), dYh(2)
REAL Fluxh(2), dFluxh(2), Presh(2), dPresh(2)
REAL Xi, Xxi, CubVal, Yxi, dXxi, CubTan, dYxi
REAL Jac, FluxxiNorm, dPresXi, FluxTanXi, FluxXii, FluxXij
REAL Tangj, Tangi, FluxDir
REAL Perm, PermData
EXTERNAL CubVal, CubTan, PermData
INTEGER iIncrements
REAL Scale
COMMON /SmVisualBackDoor/ Scale, iIncrements
Perm = PermData(iRegion(iSurf))
IF (iCirculation(iSurf) .eq. 1) THEN
  iNP(1) = iCurNode
  iNP(2) = iNextNode(iCurNode)
  dXiE(1) = dXiPlus(iNP(1))
  dXiE(2) = dXiMinus(iNP(2))
  FluxDir = 1.
ELSE
  iNP(2) = iCurNode
  iNP(1) = iNextNode(iCurNode)
  dXiE(1) = -dXiMinus(iNP(1))
  dXiE(2) = -dXiPlus(iNP(2))
  FluxDir = -1.
ENDIF
ENDIF

```

```

DO i = 1,2
  Xh(i) = X(iNP(i))
  Yh(i) = Y(iNP(i))
  dXh(i) = dX(iNP(i)) / dXiE(i)
  dYh(i) = dY(iNP(i)) / dXiE(i)
  Fluxh(i) = Flux(iNP(i))
  dFluxh(i) = dFlux(iNP(i)) / dXiE(i)
  Presh(i) = Pres(iNP(i))
  dPresh(i) = dPres(iNP(i)) / dXiE(i)
END DO
DO Xi = -1., 1., 2./FLOAT(iIncrements)
  Xxi = CubVal(Xi, Xh(1), dXh(1), Xh(2), dXh(2))
  Yxi = CubVal(Xi, Yh(1), dYh(1), Yh(2), dYh(2))
  dXxi = CubTan(Xi, Xh(1), dXh(1), Xh(2), dXh(2))
  dYxi = CubTan(Xi, Yh(1), dYh(1), Yh(2), dYh(2))
  Jac = SQRT(dXxi * dXxi + dYxi * dYxi)
  IF (Jac .eq. 0.0) THEN
    WRITE (*,*) 'Surface : ', iSurf, 'Current Node: ', iCurNode
    STOP 'Fatal Error in DrawFluxTool'
  END IF
  Tangi = dXxi / Jac
  Tangj = dYxi / Jac
  Normi = - dYxi / Jac
  Normj = + dXxi / Jac
  FluxXiNorm =
  * CubVal(Xi, Fluxh(1), dFluxh(1), Fluxh(2), dFluxh(2))
  dPresXi =
  * CubTan(Xi, Presh(1), dPresh(1), Presh(2), dPresh(2))
  FluxTanXi = -dPresXi * Perm / Jac
  FluxXii = Normi * FluxDir * FluxXiNorm + Tangi * FluxTanXi
  FluxXij = Normj * FluxDir * FluxXiNorm + Tangj * FluxTanXi
  CALL GenericMoveToRoutine (Xxi, Yxi)
  CALL GenericLineToRoutine
  * (Xxi + FluxXii * Scale, Yxi + FluxXij * Scale)
END DO
RETURN
END
SUBROUTINE DrawNormal (iSurf)
  INTEGER iSurf
  EXTERNAL DrawNormalTool
  CALL SurfaceTraversal (iSurf, DrawNormalTool)
  RETURN
END
SUBROUTINE DrawNormalTool (iCurNode, iSurf)
  INCLUDE 'Global.cmn'
  INTEGER i, iNP(2), iCurNode, iSurf
  REAL Normi, Normj, dXiE(2), Xh(2), Yh(2), dXh(2), dYh(2)
  REAL Xi, Xxi, CubVal, Yxi, dXxi, CubTan, dYxi, Jac
  EXTERNAL CubVal, CubTan
  INTEGER iIncrements
  REAL Scale
  COMMON /SmVisualBackDoor/ Scale, iIncrements
  IF (iCirculation(iSurf) .eq. 1) THEN
    iNP(1) = iCurNode
    iNP(2) = iNextNode(iCurNode)
    dXiE(1) = dXiPlus(iNP(1))
    dXiE(2) = dXiMinus(iNP(2))
  ELSE
    iNP(2) = iCurNode
    iNP(1) = iNextNode(iCurNode)
    dXiE(1) = -dXiMinus(iNP(1))
    dXiE(2) = -dXiPlus(iNP(2))
  ENDIF
DO i = 1,2
  Xh(i) = X(iNP(i))
  Yh(i) = Y(iNP(i))
  dXh(i) = dX(iNP(i)) / dXiE(i)
  dYh(i) = dY(iNP(i)) / dXiE(i)

```

```

END DO
DO Xi = -1., 1., 2./FLOAT(iIncrements)
  Xxi = CubVal(Xi, Xh(1), dXh(1), Xh(2), dXh(2))
  Yxi = CubVal(Xi, Yh(1), dYh(1), Yh(2), dYh(2))
  dXxi = CubTan(Xi, Xh(1), dXh(1), Xh(2), dXh(2))
  dYxi = CubTan(Xi, Yh(1), dYh(1), Yh(2), dYh(2))
  Jac = SQRT(dXxi * dXxi + dYxi * dYxi)
  Normi = - dYxi / Jac
  Normj = + dXxi / Jac
  CALL GenericMoveToRoutine (Xxi, Yxi)
  CALL GenericLineToRoutine
  & (Xxi + Normi * Scale, Yxi + Normj * Scale)
END DO
RETURN
END
SUBROUTINE SurfaceTraversal (iSurf, AppliedSubroutine)
C This is a general purpose subroutine for visiting all of
C the Nodes that make up a surface.
C
  INCLUDE 'Global.cmn'
  INTEGER iSurf, iFW, iCurNode
  LOGICAL lInitEntry
  EXTERNAL AppliedSubroutine
C Surface traversal: (loop through all points on the surface).
  iFW = iFirstNode(iSurf)
  iCurNode = iFW
  lInitEntry = .TRUE.
  DO WHILE ((iCurNode .ne. iFW) .or. (lInitEntry .eqv. .TRUE.))
    lInitEntry = .FALSE.
    CALL AppliedSubroutine(iCurNode, iSurf)
    iCurNode = iNextNode(iCurNode)
  END DO
  RETURN
END
C+++++=====
FUNCTION CubVal ( Xi, Gminus, dGminus, Gplus, dGplus)
C
C Returns the cubic hermetian interpolant given nodal values
C
  REAL rMplus, rMminus, rNplus, rNminus
C
  rMplus = 0.25 * (1. + Xi) * (1. + Xi) * (2. - Xi)
  rNplus = - 0.25 * (1. + Xi) * (1. + Xi) * (1. - Xi)
  rMminus = 0.25 * (1. - Xi) * (1. - Xi) * (2. + Xi)
  rNminus = 0.25 * (1. - Xi) * (1. - Xi) * (1. + Xi)
  CubVal = (rMplus * Gplus) + (rMminus * Gminus)
  & + (rNplus * dGplus) + (rNminus * dGminus)
C
  RETURN
END
FUNCTION CubTan ( Xi, Gminus, dGminus, Gplus, dGplus)
C
C Returns the first tangential derivative of the cubic hermetian
C interpolant given nodal values.
C
  REAL dMplus, dMminus, dNplus, dNminus
C
  dMplus = 0.75 * (1. - (Xi * Xi))
  dMminus = - dMplus
  dNplus = 0.25 * (- 1. + (2. * Xi) + (3. * Xi * Xi))
  dNminus = - 0.25 * ( 1. + (2. * Xi) - (3. * Xi * Xi))
  CubTan = (dMplus * Gplus) + (dMminus * Gminus)
  & + (dNplus * dGplus) + (dNminus * dGminus)
C
  RETURN
END
FUNCTION CubTan2 (Xi, Gminus, dGminus, Gplus, dGplus)
C

```

```

C   Returns the second tangential derivative of the cubic hermetian
C   interpolant given nodal values.
C
C   REAL d2Mplus, d2Mminus, d2Nplus, d2Nminus
C
C   d2Mplus = - 1.5 * Xi
C   d2Mminus = 1.5 * Xi
C   d2Nplus = 0.5 + 1.5 * Xi
C   d2Nminus = - 0.5 + 1.5 * Xi
C   CubTan2 = (d2Mplus * Gplus) + (d2Mminus * Gminus)
C   &      + (d2Nplus * dGplus) + (d2Nminus * dGminus)
C
C   RETURN
C   END
C   FUNCTION CubTan3 (Xi, Gminus, dGminus, Gplus, dGplus)
C
C   Returns the third tangential derivative of the cubic hermetian
C   interpolant given nodal values.
C
C   REAL d3Mplus, d3Mminus, d3Nplus, d3Nminus
C
C   d3Mplus = - 1.5
C   d3Mminus = 1.5
C   d3Nplus = 1.5
C   d3Nminus = 1.5
C   Xi = Xi
C   CubTan3 = (d3Mplus * Gplus) + (d3Mminus * Gminus)
C   &      + (d3Nplus * dGplus) + (d3Nminus * dGminus)
C
C   RETURN
C   END
C   FUNCTION PermData(iRegCod)
C   IMPLICIT REAL (a - h, o - z)
C
C   INCLUDE 'global.cmn'
C
C   INTEGER i
C
C   DO 10 i = 1, iNumRegions
C       IF (iRegCod .eq. iPerm(i)) THEN
C           PermData = Permeability(i)
C           RETURN
C       ENDIF
C   10 CONTINUE
C   PermData = 1.0
C   WRITE (*,*) 'Warning: Default value of permeability used'
C   RETURN
C   END

```


APPENDIX E

COMPUTER PROGRAM FROM CHAPTER 8

E.1 Program WOODMODEL

```

C      PROGRAM WoodMdl
C
C      Usage : WoodMdl [CR] 'case.stp' [CR] 'ase.stp' [CR]
C
C      EXTERNAL Wood Model, Wood Model Info
C      INTEGER unmcConfigData
C      CHARACTER flnmConfigData*64
C
C      unmcConfigData = 17
C
C      WRITE (*,*) 'Configurational Data File:'
C      READ (*,*) flnmConfigData
C      OPEN (unmcConfigData, FILE = flnmConfigData, STATUS = 'OLD')
C      WRITE (*,*) 'Thank You'
C      CALL Wood Scales Setup (unmcConfigData)
C      CLOSE (unmcConfigData)
C
C      CALL Smooth (Wood Model, Wood Model Info)
C
C      END
C
C      SUBROUTINE Wood Scales Setup (unmcConfigData)
C
C      INTEGER unmcConfigData
C
C      REAL VelocityScale, PressureScale, LengthScale, FluxScale
C      REAL Perm, HeatTransferCoeff, TotalFlow, FluxMin
C      INTEGER iInnerRegionCode, iActiveFrontSurface, iTotalFlowSurface
C
C      COMMON /WoodScalesCmn/
C      & VelocityScale, PressureScale, LengthScale,
C      & FluxScale, Perm, iInnerRegionCode, iActiveFrontSurface,
C      & HeatTransferCoeff, iTotalFlowSurface, TotalFlow, FluxMin
C
C      NAMELIST /WoodScales/
C      & VelocityScale, PressureScale, LengthScale,
C      & FluxScale, Perm, iInnerRegionCode, iActiveFrontSurface,
C      & HeatTransferCoeff, iTotalFlowSurface, TotalFlow, FluxMin
C
C      READ (unmcConfigData, WoodScales)
C
C      RETURN
C      END
C
C      SUBROUTINE Wood Model Info (iFrontSurface, Scales)
C
C      INTEGER iFrontSurface

```

```

C      REAL Scales(*)
C
C      REAL VelocityScale, PressureScale, LengthScale, FluxScale
C      REAL Perm, HeatTransferCoeff, TotalFlow, FluxMin
C      INTEGER iInnerRegionCode, iActiveFrontSurface, iTotFlowSurface
C
C      COMMON /WoodScalesCmn/
C      * VelocityScale, PressureScale, LengthScale,
C      * FluxScale, Perm, iInnerRegionCode, iActiveFrontSurface,
C      * HeatTransferCoeff, iTotFlowSurface, TotalFlow, FluxMin
C
C      REAL SmTotalFlow
C
C      iFrontSurface = iActiveFrontSurface
C      IF (iTotFlowSurface .ne. 0) THEN
C          CALL SmFluxIntegration (iTotFlowSurface, SmTotalFlow)
C          Scales(i) = TotalFlow / SmTotalFlow
C      ELSE
C          Scales(1) = FluxScale ! FScale
C      END IF
C
C      Scales(2) = PressureScale      ! PScale
C      Scales(3) = LengthScale       ! LScale
C      Scales(4) = VelocityScale     ! VScale
C
C      RETURN
C      END
C
C
C
C      SUBROUTINE Wood Model (iNumKnots)
C
C      INTEGER iNumKnots, iKnotPtr
C      REAL Flux, d2FluxdS2
C      REAL Velocity
C      REAL GetFluxDer, GetPresDer, RC Front Velocity
C      EXTERNAL GetFluxDer, GetPresDer, RC Front Velocity
C      INTEGER iSeed
C      REAL RanNum
C
C      REAL VelocityScale, PressureScale, LengthScale, FluxScale
C      REAL Perm, HeatTransferCoeff, TotalFlow, FluxMin
C      INTEGER iInnerRegionCode, iActiveFrontSurface, iTotFlowSurface
C
C      COMMON /WoodScalesCmn/
C      * VelocityScale, PressureScale, LengthScale,
C      * FluxScale, Perm, iInnerRegionCode, iActiveFrontSurface,
C      * HeatTransferCoeff, iTotFlowSurface, TotalFlow, FluxMin
C
C      REAL ScriptQ, ScriptQ2, T0, T1, Sdbyg, Deter, Vdmlss, Tf
C      PARAMETER (ScriptQ = 2.5, ScriptQ2 = 6.25)
C      PARAMETER (T0 = 1.8, T1 = 0.4)
C
C      DO iKnotPtr = 1, iNumKnots
C
C          Flux = GetFluxDer (0, iKnotPtr, 1, 0)
C          IF (Flux .gt. FluxMin) THEN
C              d2FluxdS2 = GetFluxDer (2, iKnotPtr, 1, 0)
C              Tf = T0 + T1 * Flux
C              Sdbyg = d2FluxdS2 * HeatTransferCoeff / (Flux * Flux)
C              Deter = ScriptQ2 + 4. * (Tf - 1.) * Sdbyg
C              IF (Deter .gt. 0.) THEN
C                  Vdmlss = (ScriptQ + SQRT(Deter)) / (2. * (Tf - 1.)) - 1.
C                  ExtinctFactor = 1. -
C                      * exp(-(Flux - FluxMin)/(FluxMin + 0.0001))
C                  Velocity = Vdmlss * Flux * ExtinctFactor
C              ELSE
C                  Velocity = 0.0
C          END IF
C      END DO

```

```

        END IF
      ELSE
        Velocity = 0.0
      END IF
      IF (Velocity .lt. 0.) THEN
        Velocity = 0.
      END IF
      CALL SmStoreVelocity (iKnotPtr, Velocity)
C
    END DO
C
  RETURN
END

```

E.2 Sample Data Sets

E.2.1 SLAB3.STP

```

&WoodScales
  VelocityScale = 1.
  PressureScale = 1.
  LengthScale = 1.
  FluxScale = 0.25
  Perm = 0.
  iInnerRegionCode = 1
  iActiveFrontSurface = 1
  iTotFlowSurface = 0
  HeatTransferCoeff = 0.005
  FluxMin = 0.05
&end
&StartupInfo
  PictX1 = -5.
  PictY1 = -4.
  PictX2 = 3.0
  PictY2 = 4.
  ScaleNew = .5
  iIncrementsNew = 5
  flnmInitShap = 'slab3.dat'
  iActiveSurface = 1
  EndTime = 12.00
  TimeStep = .05001
  iRemeshIntv = 1
  iNodePlacementStrategy = -120
  Spacing = 0.20
  lWaitForKey = .FALSE.
  lRemeshAtStartup = .TRUE.
  iRecordIntv = 10
&end

```

E.2.2 SLAB3.DAT

```

Slab Geometry Particle Board Experiments
3
1      1.0000000
2      1.0000000
3      1.0000000
8
      1      1
      3      1
      1      2
      3      2

```

	2		1		
	4		1		
	2		2		
	4		2		
4					
2	1	3	2	1	
2	2	1	1	-1	
5	1	3	1	-1	
5	3	1	2	1	
16					
2	1.0000000		1.0000000		
	-2.2500000		.0000000	.0000000	-.5
	1.00		.0000	.0000000	.00000
3	1.0000000		1.0000000		
	-4.0000000		-.7500000	-1.0	0.00
	1.00		.0000	.0000000	.00000
4	1.00000		1.0000		
	-5.7500000		.0000000	.0000000	+1.5
	1.00		.0000	.0000000	.00000
1	1.0000000		1.0000000		
	-4.0000000		.750000	1.0	0.00
	1.00		.0000	.0000000	.00000
6	1.0000000		1.0000000		
	10.0	.000	0.00	-2.0	
	.00	1.0	.0000000		.00000
7	1.0000000		1.0000000		
	10.	-2.	0.00	-1.	
	.00	1.0	.0000000		.000
8	1.000	1.000			
	9.0	-3.0	-1.	.00	
	.00	.00	.0000000		.000
9	1.0000000		1.0000000		
	.0000000	-3.0	-4.	.0	
	.00	.00	.0000000		.000
10	1.0000000		1.0000000		
	-9.0	-3.0	-1.	.00	
	.00	.00	.0000000		.000
11	1.0000000		1.0000000		
	-10.0	-2.	0.0		+1.
	.00	-1.00	.0000000		.000
12	1.0000000		1.0000000		
	-10.00000		.0000000	.0000000	+2.0
	.00	-1.00	.0000000		.00000
13	1.0000000		1.0000000		
	-10.00000		2.0	.0000000	+1.
	.00	-1.00	.0000000		.00000
14	1.0000000		1.0000000		
	-9.00000	3.0	1.	.00	
	.00	.00	.0000000		.00000
15	1.000	1.000			
	.0000	3.0	4.00		.00
	.00	.00	.0000000		.000
16	1.000	1.000			
	9.000	3.0	1.	.00	
	.00	.00	.0000000		.000
5	1.000	1.000			
	10.000	2.	.000	-1.	
	.00	1.00	.0000000		.000
	.0000				

E.2.3 CIRCLE1.STP

```

&WoodScales
VelocityScale = 1.
PressureScale = 1.
LengthScale = 1.

```

```

FluxScale = 1.
Perm = 0.
iInnerRegionCode = 1
iActiveFrontSurface = 1
iTotalFlowSurface = 3
HeatTransferCoeff = 0.0075
TotalFlow = 2.35
FluxMin = 0.15
&END
&StartUpInfo
PictX1 = -4.
PictY1 = -4.
PictX2 = 4.
PictY2 = 4.
ScaleNew = .5
iIncrementsNew = 5
flnmInitShap = 'circle.dat'
iActiveSurface = 1
iRecordIntv = 10
EndTime = 6.0
TimeStep = .05
iRemeshIntv = 1
iNodePlacementStrategy = -120
Spacing = 0.20
lWaitForKey = .FALSE.
lRemeshAtStartUp = .TRUE.
&END

```

E.2.4 CIRCLE.DAT

Radial Geometry Particle Board Experiments

```

3
1      1.0000000
2      1.0000000
3      1.0000000
8
      1      1
      3      1
      1      2
      3      2
      2      1
      4      1
      2      2
      4      2
4
1      1      3      2      1
1      2      1      1      -1
5      1      3      2      -1
5      3      1      1      1
8
2      1.0000000      1.0000000
      .75000000      .0000000      .0000000      -.6
      1.00      .0000      .0000000      .00000
3      1.0000000      1.0000000
      .00000000      -.7500000      -.6      0.00
      1.00      .0000      .0000000      .00000
4      1.00000      1.0000      .0000000      .00000
      -.75000000      .0000000      .0000000      +.6
      1.00      .0000      .0000000      .00000
1      1.0000000      1.0000000
      .00000000      .750000      +0.6      0.00
      1.00      .0000      .0000000      .00000
6      1.0000000      1.0000000
      3.8000000      .0000000      .0000000      -3.04
      .00      .00      .0000000      .00000
7      1.0000000      1.0000000

```

	.00000000	-3.800000	-3.04	.00
	.00	.00	.00000000	.000
8	1.00000000	1.00000000		
	-3.80000000	.00000000	.00000000	+3.04
	.00	.00	.00000000	.00000
5	1.000	1.000		
	.0000	3.80000	3.04000	.00
	.00	.00	.00000000	.000
	.0000			

E.2.5 TOWWELLS1.STP

```

&WoodScales
  VelocityScale = 1.
  PressureScale = 1.
  LengthScale = 1.
  FluxScale = 1.
  Perm = 0.
  iInnerRegionCode = 1
  iActiveFrontSurface = 1
  iTotFlowSurface = 5
  HeatTransferCoeff = 0.0075
  TotalFlow = -2.35
  FluxMin = 0.15
&END
&StartUpInfo
  PictX1 = -5.
  PictY1 = -4.
  PictX2 = 3.
  PictY2 = 4.
  ScaleNew = .5
  iIncrementsNew = 5
  flnmInitShap = 'twowells.dat'
  iActiveSurface = 1
  iRecordIntv = 10
  EndTime = 12.01
  TimeStep = .05001
  iRemeshIntv = 1
  iNodePlacementStrategy = -120
  Spacing = 0.20
  lWaitForKey = .FALSE.
  lRemeshAtStartUp = .TRUE.
&END

```

E.2.6 TOWWELLS.DAT

```

Two Wells (4pt Circ,ff(4ptCirc, fp)(4 Pt Circ, fp))
5
1      1.0010000
2      1.0020000
3      1.0030000
4      1.0040000
5      1.0050000
12
1      1
3      1
5      1
1      2
3      2
5      2
2      1
4      1
6      1
2      2
4      2

```

6	6	2			
1	2	3	2	1	
1	3	1	1	-1	
5	2	3	1	-1	
5	4	2	2	1	
9	2	3	2	1	
9	5	1	1	-1	
12					
2	1.0000000	1.0000000			
	.750000000	.0000000	.0000000	-.6	
	.5000000	.0000	.0000000	.000000	
3	1.0000000	1.0000000			
	.00000000	-.750000	-.6000	.00	
	.5000000	.0000	.0000000	.000	
4	1.0000000	1.0000000			
	-.7500000	.0000000	.0000000	+.6	
	.5000000	.0000	.0000000	.00000	
1	1.000	1.000			
	.0000	.750000	.6000	.00	
	.500	.0000	.0000000	.000	
6	1.0000000	1.0000000			
	13.000000	.0000000	.0000000	-18.0	
	.000000	.0000	.0000000	.00000	
7	1.0000000	1.0000000			
	-7.0000000	-20.00000	-18.00000	0.00	
	.000000	.0000	.0000000	.00000	
8	1.0000000	1.0000000			
	-27.00000	.0000000	.0000000	18.0	
	.000000	.0000	.0000000	.00000	
5	1.0000000	1.0000000			
	-7.0000000	20.00000	+18.00000	0.00	
	.000000	.0000	.0000000	.00000	
10	1.0000000	1.0000000			
	-13.7500000	.0000000	.0000000	-.675	
	.000000	.0000	.0000000	.00000	
11	1.0000000	1.0000000			
	-14.5000000	-.7500000	-.675	0.00	
	.000000	.0000	.0000000	.00000	
12	1.0000000	1.0000000			
	-15.2500000	.0000000	.0000000	.675	
	.000000	.0000	.0000000	.00000	
9	1.0000000	1.0000000			
	-14.5000000	.7500000	+.6750	0.00	
	.000000	.0000	.0000000	.00000	
	.0000				

END

DATE FILMED

04 / 10 / 91